Clemson University
TigerPrints

All Theses | Theses

12-2018

Hardware Obfuscation for Finite Field Algorithms

Ankur A. Sharma
Clemson University, ankuratsh15@gmail.com

Follow this and additional works at: https://tigerprints.clemson.edu/all_theses

Recommended Citation
Sharma, Ankur A., "Hardware Obfuscation for Finite Field Algorithms" (2018). *All Theses*. 3013.
https://tigerprints.clemson.edu/all_theses/3013

This Thesis is brought to you for free and open access by the Theses at TigerPrints. It has been accepted for inclusion in All Theses by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

Hardware Obfuscation for Finite Field Algorithms

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Engineering

---

by
Ankur A Sharma
December 2018

Accepted by:
Dr. Yingjie Lao, Committee Chair
Dr. Richard R Brooks
Dr. Adam Hoover

Abstract

With the rise of computing devices, the security robustness of the devices has become of utmost importance. Companies invest huge sums of money, time and effort in security analysis and vulnerability testing of their software products. Bug bounty programs are held which incentivize security researchers for finding security holes in software. Once holes are found, software firms release security patches for their products.

The semiconductor industry has flourished with accelerated innovation. Fabless manufacturing has reduced the time-to-market and lowered the cost of production of devices. Fabless paradigm has introduced trust issues among the hardware designers and manufacturers. Increasing dependence on computing devices in personal applications as well as in critical infrastructure has given a rise to hardware attacks on the devices in the last decade. Reverse engineering and IP theft are major challenges that have emerged for the electronics industry.

Integrated circuit design companies experience a loss of billions of dollars because of malicious acts by untrustworthy parties involved in the design and fabrication process, and because of attacks by adversaries on the electronic devices in which the chips are embedded.

To counter these attacks, researchers have been working extensively towards finding strong countermeasures. Hardware obfuscation techniques make the reverse

engineering of device design and functionality difficult for the adversary. The goal is to conceal or lock the underlying intellectual property of the integrated circuit. Obfuscation in hardware circuits can be implemented to hide the gate-level design, layout and the IP cores.

Our work presents a novel hardware obfuscation design through reconfigurable finite field arithmetic units, which can be employed in various error correction and cryptographic algorithms. The effectiveness and efficiency of the proposed methods are verified by an obfuscated Reformulated Inversion-less Berlekamp-Massey (RiBM) architecture based Reed-Solomon decoder. Our experimental results show the hardware implementation of RiBM based Reed-Solomon decoder built using reconfigurable field multiplier designs. The proposed design provides only very low overhead with improved security by obfuscating the functionality and the outputs. The design proposed in our work can also be implemented in hardware designs of other algorithms that are based on finite field arithmetic. However, our main motivation was to target encryption and decryption circuits which store and process sensitive data and are used in critical applications.

Dedication

To my parents and my sister for their love and support, and to my advisor for his patience and faith in me.

Acknowledgments

I would first like to thank my advisor Dr. Yingjie Lao of the Holcombe Department of Electrical and Computer Engineering at Clemson University. Professor Lao always had his door open for me whenever I ran into a trouble or faced any major or minor obstacle in my research. He consistently allowed me to work on this research as my own work, providing me the necessary independence but at the same time also guided me and steered me in the right direction whenever he felt the need to do so. I would also like to thank Dr Xinmiao Zhang of the Department of Electrical and Computer Engineering at the Ohio State University who guided me in understanding some of the difficult concepts and pointed out the mistakes I did while progressing in my research.

I would also like to acknowledge Dr. Richard R Brooks and Dr Adam Hoover of the Holcombe Department of Electrical and Computer Engineering at Clemson University who were involved in the validation of this research project and were the second reader of this thesis. I am gratefully indebted to both of them for their valuable comments on this thesis.

Finally, I express my profound gratitude to my parents and the people around me for providing me with unfailing support and continuous encouragement throughout my years of education and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

For a long time, cybersecurity study and research was focused on information and software security concerning the protection of the integrity of data and the confidentiality of data while on storage for computation or in transmission. Software-based security research has brought our technology forward by providing us with various forms of encryption and decryption methods, passwords and bio-metric identification codes, digital signatures, anti-virus tools, etc. It also has helped us develop methodologies and tools to identify and analyze malicious codes that leverage properties of the system's hardware.

However, security management of modern electronic systems can no longer be based on the contestable and sometimes naive assumption that the underlying hardware is trusted and secure. The hardware includes the processor, the chips, integrated circuits, motherboard and other electronic systems that allow us to run the software. With the rise of the Internet of Things (IoT) and wearable technology, the technology is no more limited to a black box kept seemingly secured in one's apartment or office premise. The data being provided to big data companies by third-party corporations, governments as well as the general public has become more

sensitive. Thus, the potential ramifications of a single data breach can be far-reaching with disastrous consequences for all the parties involved. Therefore, the security of the data centres and portable devices, as well as the secure transmission of all such data, has become if great importance. For this purpose, not only software but also hardware needs to be evaluated for security and tested for any unauthorized modifications.

In recent years itself, many hardware vulnerabilities have been discovered in the integrated circuits and processor chips that are popularly being used in consumer electronics, computers, data centres, government offices, military equipment and the portions of critical infrastructure. For instance, modern processors from leading processor manufacturers were affected by Meltdown [30] and Spectre [24] vulnerabilities, from which the attackers could gain access to the data being processed on the hardware. Processors were in use in almost all computing systems. Though fixes for these vulnerabilities were released, they reduced the performance of the processor by up to 25%. There have also been discoveries of back-doors in ICs know as Hardware Trojans which can stealthily cause an IC malfunction. These are just two examples of malicious hardware tampering. There are other major concerns in today's hardware manufacturing supply chain such as IC counterfeiting and IP piracy.

1.1 Hardware Attacks

In today's semiconductor industry, a device usually involves a long supply chain from design to final packaging as shown in Figure 1.1. Due to the high cost of foundries and high design complexity of new chips, the supply chain spreads over multiple countries, companies and third-party vendors. Every stage of the supply chain may involve global suppliers who may or may not be trustworthy.

This might apparently seem beneficial from the monetary perspective as it

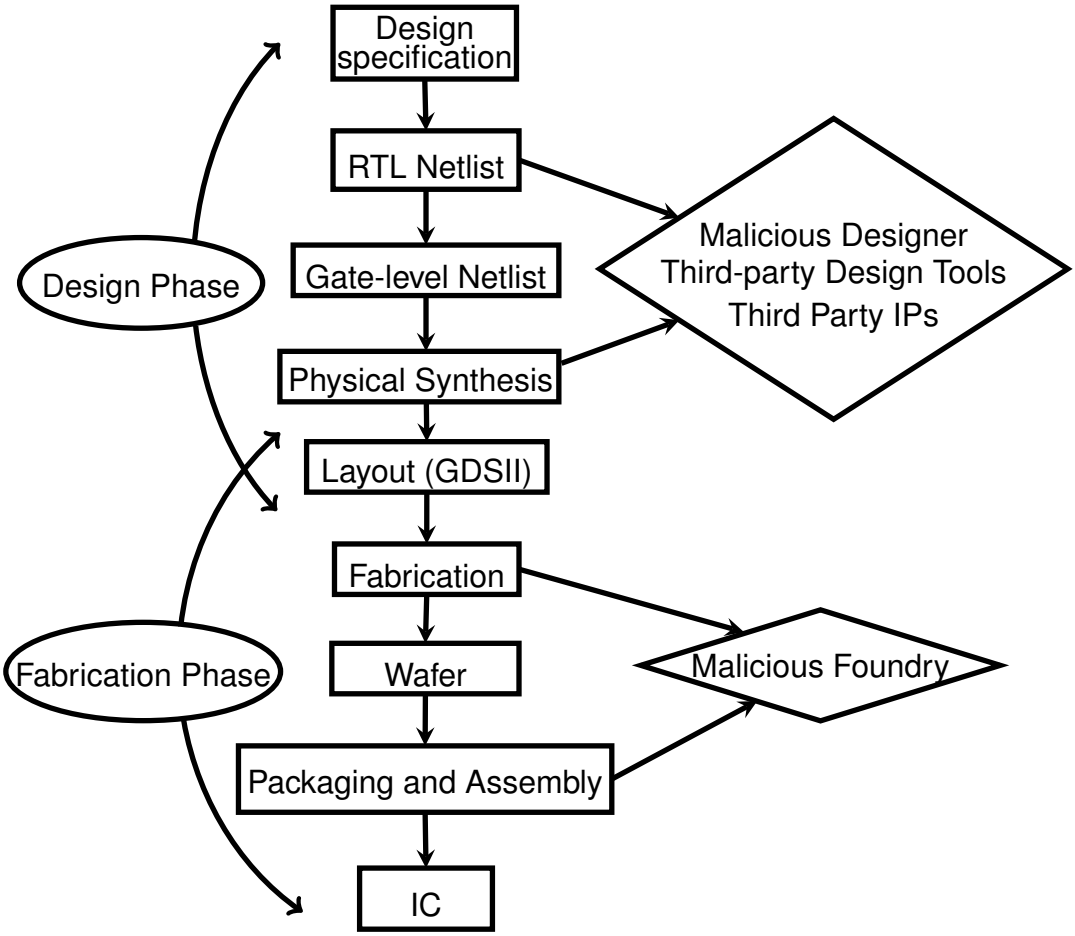


Figure 1.1: IC supply chain

reduces the time to market of the electronic products, as well as saves the cost. However, it also introduces new security challenges through the supply chain which may cause the organization to incur losses later on. Tampering hardware design and pirating the design by any of the third party vendors can result in huge losses to the IC design company in terms of profits as well as in terms of consumer trust.

Hardware Trojans may be inserted into the chips by the fabrication house. Contrary to the expectation of the IP provider, the final product may contain malicious logic and flaws that an attacker can exploit after the chips are embedded into the devices. Post-deployment, reverse engineering against ICs on the other hand may

also provide adversaries with hardware vulnerabilities or valuable IP information.

1.1.1 Attacks through the IC supply chain

There are two major stages in the semiconductor manufacturing process.

- **Design Stage:** Using a register transfer level language (RTL) such as VHDL or (System) Verilog, a design specification is described. Thereafter a synthesis tool is used to synthesize the design into a gate-level netlist. This netlist file is then used to generate a layout file which contains the information regarding the location and shapes of the cells. The layout file also has information about the routing paths between the gates. This file is used for chip fabrication.

- **Fabrication Stage:** For fabrication of the chips, masks such as photo-masks are required for lithography. These masks are produced using the (GDSII) layout file as a reference. The final product has the chip packaged and assembled onto a printed circuit board. To accelerate the delivery and lower the costs, most IC companies outsource the fabrication to off-shore foundries which work autonomously without any direct monitoring by the IP providing company.

Both of these stages have potential for attacks.

1.1.1.1 Design Stage Attacks

In the design stage, three of the potential security threats are a malicious design party, an untrusted IP from a third party, and the design tools bought from a third party. An employee in the project team who has full access to the chip design, if gone rogue, can modify the design to include malicious logic or backdoors. This might not be even discovered by the team and the company until and even after the product has been launched in the market.

Additionally, a rogue employee can sell the design's IP to a competitor firm which can then claim ownership of the design or can sabotage the market of the original designer firm by producing the chips based on the design illegally. These categories of attacks are commonly referred to as IP piracy and IP counterfeiting. Untrusted IP vendors may provide IPs with built-in backdoors, sabotaging the entire hardware design. This can allow attackers who know about the IP design to cause the device to malfunction under target conditions.

Designers and hardware testers often rely on third-party design tools for design optimization and verification of the design. However, these tools also pose a potential threat to the design as they might tamper the results produced. These kinds of attacks are referred to as Hardware Trojan attacks.

1.1.1.2 Fabrication Stage Attacks

Outsourcing of IC designs by the IC design companies to fabrication foundry houses is a common trend today. This lowers the gross cost of production of the final product as well as reduces the time-to-market of the product. However, the foundries, if not trustworthy, can pirate or tamper the designs by the actions of a single unethical employee of the foundry. By reverse engineering the design obtained from the contracting IC design company, the foundry can obtain gate-level design knowledge of the IP and sell the information to competing design firms. This comes under IP piracy.

In addition to this, the foundry can overproduce the chip for their own profit, spooning out profit from the IC designer. Since the foundry doesn't own the design, the overproduced chips may not go through proper testing and may be of much lower quality than the ones being sold by the IC designer. This IC overproduction by the foundry may result in reputation and profit loss for the IC designer company.

Foundries can also try to gain unauthorized profits by selling old out-of-spec chips by relabeling them as new chips. Counterfeiting of chips includes recycled chips as well as fake chips. Modification of the layout by an employee in a foundry may also cause a hardware Trojan to be inserted into the design leading to the production of a huge number of infected chips, trusted by the IC designer and its clients.

1.1.2 Attacks on hardware post-production

With increasing widespread adoption of computing systems in our daily lives and in the corporate industry, in terms of embedded systems, wearable technologies, portable devices, IoT devices and data centre firms for cloud services, attacks on hardware post-deployment to the client are also becoming major concerns. Recently, there were reports on evidence of security backdoors in the chips used in weaponry systems, nuclear power plants and transportation systems used by the military. This clearly indicates the urgency of hardware protection. Post-deployment attacks can be mainly categorized into Reverse engineering, Fault injection attacks and Side channel attacks. We discuss two of these categories which are more relevant to our work.

- **Reverse Engineering:** By depackaging the chip and then delayering it, an adversary can reverse engineer the internal design of the chip and use that information to exploit a vulnerability or change the behaviour in the target chip of the same design. Reverse engineering in the final product is not easy and requires considerable resources, time, patience and knowledge on the adversary's part.

- **Side Channel Attacks:** Hardware device could leak some information in the form of heat signatures, power consumption, sound or electromagnetic radiation. After enough data are accumulated, the data can be analyzed and associated

with the operation being processed inside the chip. This enables the adversary to gain insight about the confidential data such as encryption/decryption keys from the memory on the chip. This side channel information from modern microprocessors was the concern behind the Spectre and Meltdown vulnerabilities in the modern processors, discovered in 2017. The vulnerabilities allowed an adversarial process to gain the data of another process from the memory, which ideally is a security design flaw.

1.2 Hardware Security

Due to the growing discovery of such potential attacks and the alarmingly rising risk they put on the people, infrastructure, governments, military and trade, designers have to come up with methods to protect the IP of their designs. It has become necessary to develop methods against reverse engineering for hardware devices. Widely used countermeasures to reverse engineering are discussed as follows.

1.2.1 Circuit Camouflaging

IC camouflaging is a technique to design the layout in a manner to thwart the attempts of reverse engineering attempted through electron microscopy. Electron microscopy allows the adversary to visually see the cells on the IC, enabling him to potentially obtain the gate level layout of the chip. To implement camouflaging, some standard gate cells are made identical in shape so that they are difficult to identify in the images obtained through an electron microscope. Additionally, dummy contacts are also created which appear to be connected in the images but are actually disconnected being only a small distance apart. It makes it difficult for the attacker to deduce the exact functionality and netlist when these techniques are used.

1.2.2 Logic Encryption

Adding some additional gates known as the key gates to the design, designers can lock the correct functionality of the chip. The key gates obtain input from the chip's I/O. A correct input to the key gates activates the correct functionality of the chip. This technique is also called logic locking or logic obfuscation.

1.3 Cryptography and Error Correcting Codes

Cryptography is the area of research where methods and algorithms are developed to hide information from unauthorized access and to authenticate the source while maintaining the integrity of the message. Error correcting codes (ECC) are used to detect and correct errors that might infect a message when received via a noisy communication channel, therefore protecting the integrity of the message from sending party to the receiving end. ECC and cryptography are used in different scenarios. However, both perform the encoding of the information to execute their respective purposes.

1.4 Contribution and Thesis Organization

In this dissertation, we focus on the logic obfuscation methods for error correcting codes and crytographic algorithms that are primarily based on finite field algebra. The hardware implementations of coding theory based algorithms usually employ a number of basic arithmetic units such as adders and multipliers. The adders and multipliers are specifically designed for finite field based addition and multiplication operations. The devices implementing cryptography and ECC are one of the most popular applications that work on the concept of finite field coding theory, and use a

numerous finite field adders and multipliers. We focus on such arithmetic circuits in our work.

We propose reconfigurable designs by exploiting the concepts of hardware obfuscation for finite field arithmetic. The design of the adder and the multiplier is dependent on a set of chosen parameters. Thus, to prevent reverse engineering of the hardware, the design and functionality of the arithmetic units should be obfuscated. Our work provides a method to deceive the adversary and making it difficult for him to guess the set of chosen parameters that make the circuit work correctly. Our method also results in very low hardware overheads which is an important aspect for real world application of the new hardware design.

1.5 Thesis Overview

The thesis is organized as follows: In Chapter 1, we discussed the various security challenges that the semiconductor industry faces. Our main focus is to propose a design methodology for improving the security of ICs.

In chapter 2, we present the background of the process of IC design and manufacturing and discuss the potential points of attacks. Additionally, we elaborate on the types of attacks in the field of hardware security and go over the recent research progress in the field of hardware obfuscation. Cryptography and Error Correction Codes and their applications have also been discussed to emphasize the importance and potential contribution of this work in the chip design for such applications.

In chapter 3, we provide a background on finite field arithmetic and its application in coding theory. We also describe the process of Reed Solomon encoding and decoding to provide a foundation for the reader to understand our research methodology and the nature of this research work.

In chapter 4, we go over the recent findings and proposed designs that helped us shape and improve our experimental setup. We also describe the implementation of our version of the Reed-Solomon decoder inspired from recent academic research findings and discuss the methodology used to obtain and analyze the results.

In chapter 5, we present our findings from our experiment. Our experiment involves using our method for obfuscation of finite field based circuits to implement obfuscation in Reed Solomon error correcting codes by the use of re-configurable units of finite field computation. We produce results for different scenarios and analyze them to conclude and strengthen the confidence of our findings.

In chapter 6, we present the conclusion of our research findings and discuss the potential implications of our results. We also briefly talk about potential future research in this field.

Chapter 2

Hardware Obfuscation

Hardware obfuscation transforms an original design into a functional equivalent version that is much harder to reverse engineer. Hardware obfuscation can be realized via various ways [45, 28, 25, 9, 8, 16, 18, 40, 27, 20].

2.1 Importance of Hardware Obfuscation

There have been events concerning IP theft and reverse engineering among big firms as well as small agencies. In 2016, FPGA manufacturer Xilinx accused a popular chip supplier Flextronics of violating its IP contract. Xilinx stated that Flextronics bought chips from Xilinx on the pretext of a different consumer market and then sold the chips at higher rates after branding them as higher grade chips [33]. In the economics of it, Xilinx was exposed to higher liabilities.

Understanding the competitor's product is something not very unexpected in today's era and large firms do spend time and resources reverse engineering competitors' products. The laws of IP protection vary vastly from country to country and it is not easy for a company to protect its IP [44] and prevent another corporation from

using the IP to develop and sell products with no compensation to the IP holder. Because of the variation in laws and degree of law enforcement in different areas, patents and copyrights do not provide complete assurance to the IP holder. Thus, IP protection needs to be enforced actively by the designers if they want to avoid loss of profits and trust.

Obfuscation cannot be converted into such mathematical algorithms. Also, the encryption and decryption of the hardware design requires additional gate logic added to the design. Since encryption algorithms involve multiple copies of similar manipulation blocks, it also increases the area and power overheads considerably.

Additionally, due to the presence of multiple identical blocks in the design, it is easy for an adversary at the fabrication house having access to the layout for fabrication purpose, to deduce the gates responsible for the encryption logic. The adversary can simply remove these gates to produce the unencrypted fully functional chip. Add to this the logistic overhead and inconvenience that would be encountered while sending these chips for testing.

An obfuscated form $P'()$ of an operation $P()$ is analogous to a virtual black box that can be used by a user to obtain an output $P(x)$ for an input x without knowing the exact structure of $P()$. It was stated in [2] that perfect virtual black box obfuscation is unachievable for all operations. Specifically, it was argued that Boolean circuits were inherently unobfuscable. Consequently, an obfuscated form $P'()$ of the program $P()$ can be exploited to extract a key information about $P()$ or the entire program $P()$ itself with a probability greater than zero.

Though perfect obfuscation which leaks no information about the output to input relation of the program, is impossible to achieve, there is a 'best possible obfuscation' with relatively relaxed constraints that can be achieved. Best possible obfuscation relaxes the strict constraint of the perfect black box obfuscation and relies

on the following constraint: An obfuscated program $P'()$ is allowed to leak the least amount of information that can be leaked by any other program which implements the same function $F()$ as being implemented by the program $P()$. Consequently, from the set of all programs that perform the function $F()$, none should reveal any information more than that revealed by any other program in the set about the function $F()$ to the adversary.

Another concept that is related to the above is indistinguishable obfuscation [6]. Indistinguishable obfuscation means that for two programs $P()$ and $Q()$ that implement the same function $F()$, there exists an indistinguishability obfuscator $O()$ such that the respective obfuscated programs $O(P()) = P'()$ and $O(Q()) = Q'()$ are indistinguishable from each other. However, it has been shown that indistinguishability is not achievable in practice [2]. Nonetheless, there have been consistent efforts in the academic community towards enhancing the security strength of partial obfuscation. In the next section, we discuss some of the representative work in the field of hardware obfuscation.

2.2 Recent Work in Hardware Obfuscation

Numerous hardware obfuscation techniques have been proposed by researchers in the literature. Figure 2.1 shows a taxonomy of various techniques of implementing hardware obfuscation. We discuss some of those methods briefly below.

2.2.1 Layout Level Obfuscation

Layout level obfuscation is used as a countermeasure against reverse engineering [34]. The most popular method is camouflaging [13, 29]. Standard cells such as XOR, NAND and NOR gates are designed to appear identical in the layout by

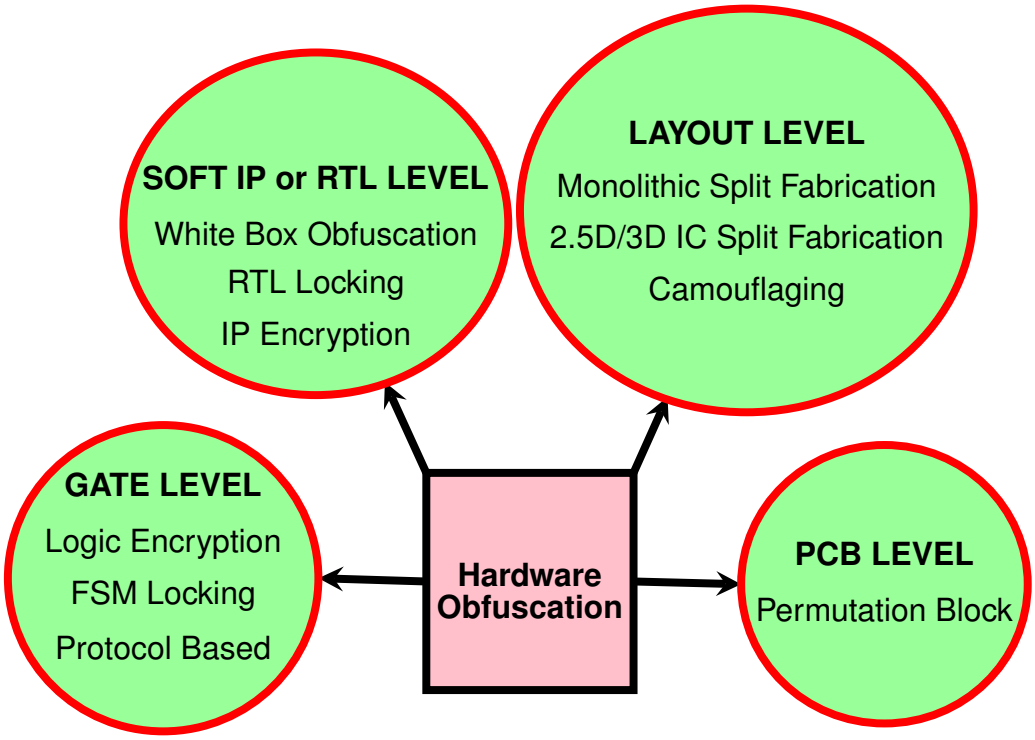


Figure 2.1: Hardware obfuscation designs

having the same shape and size. Additionally, contacts that are not actually forming a closed circuit, known as dummy contacts which appear to connect two cells in the layout images are helpful in confusing the adversary about the actual design of the circuit. This makes it difficult for the adversary to retrieve the exact netlist. However, to have a considerable level of security, a large number of such camouflaged gates might be required, thereby increasing the overhead considerably. Also, recently SAT solvers have been demonstrated to be successful in revealing the correct functionality of a highly camouflaged design, raising doubts about the security of this technique [32, 48, 17, 3].

2.2.2 Gate Level Obfuscation

The gate level obfuscation focuses on the protection of the netlist. Logic encryption is one of the major techniques as shown in Figure 2.2. A set of bits are

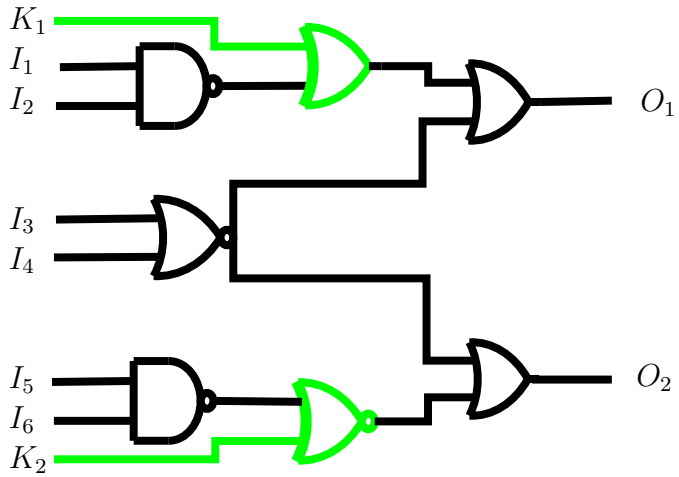


Figure 2.2: Logic Encryption example

used as keys which dominate the function of the netlist. These bits are inputs to some additional logic gates inserted into the netlist. Physically unclonable functions (PUFs) are functions derived from physical properties of tiny semiconductor components that are unique to each component. The key bits for logic encryption can be derived from PUFs, essentially providing a unique key for each chip. Netlist level obfuscation is considered a countermeasure against hardware trojans [10].

However, SAT attacks have been shown to be successful against such logic encryption techniques [39, 47]. For circuits having partial or entire sequential portions, a correct ordered sequence of inputs can be used as a key to unlock the netlist functionality. Since this employs a series of finite states, this is also referred as Finite State Machine (FSM) based locking [4, 31, 43]. Due to the use of multiple finite states, this technique imposes high overheads in terms of time, power and area.

2.2.3 Soft IP Obfuscation

Soft IP obfuscation is implemented at the register transfer level (RTL). Various methods include RTL logic locking and Encryption of IP. In RTL locking methods, non-functional states are added to the functional states of the IP after analyzing the state transition graphs of the Verilog or VHDL code. Until the correct key sequence is applied to the design, the IP remains in the non-functional states producing no or incorrect outputs. When the correct key sequence is applied, the IP becomes functional and operates in one of the functional states depending on the input sequence. Encryption of IP is handled by the trusted design tools i.e. EDA tools that the client uses to use the encrypted IP without knowing the internal structure [37, 26].

Among these, most of the recently proposed obfuscation techniques utilize additional keys to implement logic obfuscation [9, 8], which is also referred to as logic encryption [16]. Recently, a new class of hardware obfuscation techniques that applies high-level transformations to achieve obfuscation has been proposed [28]. Both meaningful and non-meaningful obfuscating modes are introduced in these schemes, which creates another level of ambiguity to protect the device from adversarial attacks.

2.2.4 Printed Circuit Board Level Obfuscation

PCB obfuscation is governed by different constraints than IC obfuscation. These differences include the differences in design dimension, attack challenges and opportunities for design modification. These differences limit the possible choices for PCB level obfuscation. However, gate-level approaches like logic permutation and logic encryption can be applied to PCB level as well.

Since, the connections on the board are easily discoverable as compared to that on an IC, adversary has higher flexibility in identifying, removing or bypassing the

components responsible for obfuscation. In permutation based obfuscation [21, 22], a permutation block permutes a set of selected inter-component connections using a permutation network controlled by a key. Before moving onto obfuscation of finite fields, let's take a look at the most popular application of finite fields in security cryptography and coding theory.

2.3 Cryptography and ECC applications

Encryption hides a message from unauthorized read access. Cryptography also provides authenticity to message as well as the source of the message. Error correcting codes, on the other hand, enables us to evaluate the integrity of the message by detecting and correcting error(s) in a message received from a communication channel.

ECC and cryptography have a few fundamental concepts common to both. There are a few similarities between ECC and cryptography as was also noted by [23]:

- Both perform encoding of the message into another format of information.
- Both have theoretical origins from Shannon's work [42, 41].
- Most algorithms in both domains work on principles of abstract algebra, finite field mathematics and combinatorial theory.

However, one fundamental difference between the two is that ECC embraces the variation in the message, attempting to evaluate the error and retrieve the correct value of the message while, a cryptographic system tries to evaluate any possible variation to identify even the slightest change in the message in order to strongly reject a modified received message. For example, a few bit flips in a message received

by an ECC decoder may still provide the user with the expected output i.e. the message that was intentionally sent. However, a single bit flip in a message received by a decryption algorithm will straight away reject the received message as a bad input and deny the user the access to the correct message.

Owing to the similarities in the mathematical concepts and in terms of the subject of the application, there have been some interesting research in the overlapping fields of ECC and cryptography [23]. Linear error correcting codes have been used in developing cryptographic solutions as in the case of [14]. Linear secret sharing schemes are used in a certain area of cryptography for applications such as secret sharing, multi-party computation as well as two-party primitives. In 2015, Baldi et al invented a method of using public key cryptography using the concepts of error correcting codes [1].

2.4 Hardware Obfuscation for Cryptography and ECC

In [35], a method to protect hardware implementation of Advanced Encryption Scheme (AES) was proposed by shifting down the inversion operation from $GF(2^8)$ to $GF(2^2)$, since linear operations like inversion are easy to mask in smaller fields. They concluded that they could achieve significant masking against first-order side-channel attacks by combining the concepts of multiplicative and additive masking. Obfuscation using high-level transformation was obtained for Digital Signal Processing (DSP) circuits using Finite State Machines (FSM) based obfuscation [28]. They were able to achieve an area overhead of about 17% for an Infinite Impulse Response (IIR) filter using a 128-bit key. [9] proposed RTL obfuscation design using data and control flow

to incorporate FSM based obfuscation. Many have proposed obfuscation techniques using FSMs and other mathematical tools [31, 36, 46, 49]. To the best of my knowledge, there has been no previous work specifically focused on the obfuscation of the hardware implementation of finite field based algorithms, be it cryptography or error correction codes.

2.5 Our Focus

In our work, we focus on hardware obfuscation methodology for finite field arithmetic circuits. This includes many error correcting codes as well as cryptography solutions. The Rijndael AES [15] uses a number of Galois field units. Coding theory also relies heavily on finite field based computation. We demonstrate a useful method for strengthening the security of hardware implementation of such algorithms. We demonstrate our technique and our analysis of its benefits, on one of the most popular coding theory algorithm known as Reed-Solomon error correction code. For proper understanding of our implementation, we provide a background on finite field arithmetic and the conceptual design of the Reed-Solomon decoding process in the next chapter.

Chapter 3

Background

In this chapter, we provide an overview of finite field arithmetic and its application for cryptography and error correcting codes. We will also discuss the use of hardware obfuscation and various methods by which hardware obfuscation can be achieved in today's world.

3.1 Finite Fields

A field is a set of elements on which the operations of addition and multiplication are defined. The operations are commutative, associative, and closed as shown below.

- $a + b = b + a$ (Commutative Property of Addition)
- $ab = ba$ (Commutative Property of Multiplication)
- $a + b = b + a$ (Associative Property of Addition)
- $a(bc) = (ab)c$ (Associative Property of Multiplication)

Closure implies that the sum and product of any two elements in the field are also elements of the field. The distributive law relates multiplication and addition by

- a (b + c) = ab + ac

The additive and multiplicative identities (0 and 1) exist for all elements in the field.

- $a + 0 = a$ (Additive Identity)
- $1 \times a = a$ (Multiplicative Identity)

The additive and multiplicative identities are not the same.

The elements of a field also have additive and multiplicative inverses. The additive inverse b and the multiplicative inverse c of the element a from a field satisfy the following relations:

- $a + b = 0$
- $a \times c = 1$

Since there is an additive inverse, the operation of subtraction is also defined in a field. Similarly, the existence of multiplicative inverse implies the operation of division.

Popularly known and commonly used fields having an infinite number of elements are the one consisting of the real numbers R, the complex numbers C, and the rational numbers Q. because only $+1$ and -1 have multiplicative inverses, the integers under the usual arithmetic (Z), do not constitute a field. Although the real, complex, and rational fields belong to the set of infinite fields. The fields consisting of finite number of elements are called the finite fields. The symbol Z_p refers to the set of integers $0, 1, 2, \ldots, p-1$ using modulo p arithmetic. Z_p is a field if and only if p is a prime number. Regardless of whether or not p is prime, each element x has an

additive inverse with the value $(p - x)$. This follows from the fact that

$$\begin{aligned} x + p - x &= p \\ &= 0 \ mod \ p \end{aligned} \tag{3.1}$$

p being prime, there exists an element y in the field such that $x \times y = 1 \ mod \ p$. y is called the multiplicative inverse of x.

If p is not a prime number, then p can be factored as $p = ab$ where $1 < a$, $b < p$. The product of a and b, $ab = 0 \mod p$. In this case, a and b are the divisors of zero. Fields satisfy a cancellation law: $ac = ad$ implies $c = d$, and the following argument shows that a field cannot have divisors of zero. Z_p for p not prime is not a field. For any such finite field, it will always be the case each row of the addition is a permutation of the values $0, 1, \ldots, p-1$ and each row of the multiplication table except the first row will also be a permutation of the elements of the field. As noted previously, a value of 1 in the multiplication table identifies a pair of multiplicative inverses.

3.1.1 Galois fields

If p is a prime number, then it is also possible to define a field with p^m elements for any m. These fields are named after the great French algebraist Evariste Galois who was killed in a duel at the age of 20. They have many applications in coding theory. The elements of Galois Field $GF(p^m)$ is defined as

$$\begin{aligned} GF(p^m) =& (0, 1, 2, \ldots, p-1) \cup (p, p+1, p+2, \ldots, p+p-1) \\ & \cup (p^2, p^2+1, p^2+2, \ldots, p^2+p-1) \cup \ldots \\ & \cup (p^{m-1}, p^{m-1}+1, p^{m-1}+2, \ldots, p^{m-1}+p-1) \end{aligned} \tag{3.2}$$

where $p \in P$ and $m \in Z^+$. The order of the field is given by p^m while p is called the characteristic of the field. Also note that the degree of polynomial of each element is at most $(m-1)$. The fields, denoted $GF(p^m)$, are comprised of the polynomials of degree $(m-1)$ over the field Z_p. These polynomials are expressed as a $(m-1) \times (m-1) + \cdots + a^1 \times 1 + a^0 \times 0$ where the coefficients a^i take on values in the set $0, 1, \ldots, p-1$. When employed in coding applications p is commonly 2 and thus the coefficients $a^0, \ldots, a^{m-1}$ are taken from the binary digits $0, 1$. Consider the following example where

$$GF(5) = (0, 1, 2, 3, 4) \tag{3.3}$$

which consists of 5 elements where each of them is a polynomial of degree 0 (a constant) while

$$\begin{aligned} GF(2^3) &= (0, 1, 2, 2+1, 2^2, 2^2+1, 2^2+2, 2^2+2+1) \\ &= (0, 1, 2, 3, 4, 5, 6, 7) \end{aligned} \tag{3.4}$$

which consists of $2^3 = 8$ elements where each of them is a polynomial of degree at most 2 evaluated at 2.

In coding applications, for $m <= 32$, it is common to represent an entire polynomial in $GF(2^m)$ as a single integer value in which individual bits of the integer represents the coefficients of the polynomial. The least significant bit of the integer represents the a^0 coefficient.

3.1.2 Finite field polynomial arithmetic in $GF(2^m)$

Addition and multiplication of polynomial coefficients, but not the polynomials themselves in the field $GF(2^m)$ are defined by the rules of Z_2. Addition is defined by the **exclusive OR** operation and multiplication by the **AND** operation.

These operations are used in manipulating the coefficients during multiplication and addition of polynomials, but the basic algorithms used in adding and multiplying polynomials over the integers remain applicable[5].

3.1.2.1 Addition and Additive Inverse

To add two or more polynomials, for each power of x present in the summands, one needs to just add the corresponding coefficients *modulo* 2. If a particular power appears an odd number of times in the summands it will have a coefficient of 1 in the sum. If it appears an even number of times or does not appear at all, it will have a coefficient of 0 in the sum. For example,

$$(x^2 + 1) + (x + 1) + (x^2 + x + 1) = 1. \tag{3.5}$$

Note that the polynomials of degree $(m - 1)$ are closed under polynomial addition. The sum is always a polynomial of degree no more than degree $(m - 1)$.

Consider the following example. Suppose we are working in $GF(2^8)$, then $84 + 247$ is

$$\begin{aligned} 84 + 247 &= (2^6 + 2^4 + 2^2) + (2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 + 2^0) \\ &= 2^7 + 2 \cdot 2^6 + 2^5 + 2 \cdot 2^4 + 2 \cdot 2^2 + 2^1 + 2^0 \\ &= 2^7 + 2^5 + 2^1 + 2^0 \\ &= 163 \end{aligned} \tag{3.6}$$

Alternatively, from binary numeral system perspective,

$$\begin{aligned} 84 + 247 &= 01010100 + 11110111 \\ &= 10100011 \\ &= 163 \end{aligned} \tag{3.7}$$

and the results coincide. Furthermore, because of the XOR method of addition, each polynomial is its own additive inverse.

3.1.2.2 Multiplication and Multiplicative Inverse

Multiplication requires more tedious work. Suppose $a(x)$ and $b(x)$ are polynomials in $GF(p^m)$ and let $p(x)$ be an irreducible polynomial (or a polynomial that cannot be factored) of degree at least m in $GF(p^m)$. We want $p(x)$ to be a polynomial of degree at least n so that the product of two $a(x)$ and $b(x)$ does not exceed $11111111 = 255$ as the product needs to be stored as a byte. If $c(x)$ denotes the resulting product then

$$c(x) = (a(x) \cdot b(x))(mod\ p(x)) \tag{3.8}$$

On the other hand, the multiplicative inverse of $a(x)$ is given by $inv(a(x))$ such that

$$(a(x) \cdot inv(a(x)))(mod\ p(x)) = 1 \tag{3.9}$$

Note that figuring out the product of two polynomials and the multiplicative inverse of a polynomial requires both reducing coefficients *modulo* x and reducing polynomials *modulo* $p(x)$. The reduced polynomial can be calculated easily with long division while the best way to compute the multiplicative inverse is by using Extended Eu-

clidean Algorithm. The details on the calculations in $GF(2^8)$ is best explained in the following example. Suppose we are working in $GF(2^8)$ and we take the irreducible polynomial modulo $p(x)$ to be $x^8 + x^6 + x^5 + x^1 + x^0$. To calculate $13 \cdot 84$, we need to go through several steps. First, we compute the product of the polynomial and reduce the coefficients *modulo* 2.

$$\begin{aligned} 13\dot{8}4 &= ((2^3 + 2^2 + 2^0) \cdot (2^6 + 2^4 + 2^2))(mod\ p(x)) \\ &= (2^9 + 2^8 + 2^7 + 2 \cdot 2^6 + 2^5 + 2 \cdot 2^4 + 2^2)(mod\ p(x)) \\ &= (2^9 + 2^8 + 2^7 + 2^5 + 2^2)(mod\ p(x)) \end{aligned} \tag{3.10}$$

Then we use long division to compute the reduced polynomial as follows

Quotient	*Remainder*
	$2^9 + 2^8 + 2^7 + 2^5 + 2^2$
	$2^8 + 2^6 + 2^5 + 2^1 + 2^0$
$2^1 + 2^0$	2^0

(3.11)

Where the last entry in the first column is the product we seek for. Since the product is 1, it follows that 84 and 13 are multiplicative inverse pairs. If we assume that we do not know the multiplicative inverse of 84. Then to calculate the multiplicative inverse we will use Extended Euclidean Algorithm. Unlike long division, we need to keep track of the auxiliary when we work with Extended Euclidean Algorithm as

follows

Quotient	*Remainder*	*Auxiliary*
	$2^8 + 2^6 + 2^5 + 2^1 + 2^0$	0
	$2^6 + 2^4 + 2^2$	1
2^2	$2^5 + 2^4 + 2^1 + 2^0$	2^2
$2^1 + 2^0$	2^0	$2^3 + 2^2 + 1$

(3.12)

The first two rows in the Remainder column are always the modulo polynomial followed by the polynomial we wish to invert. The first two rows in the Auxiliary are always 0 and 2^0. The remainder and the quotient in row m is then calculated from the division of the remainders in row $(m-1)$ and $(m-2)$ while the auxiliary in row m is given by the sum of the auxiliary in row $(m-2)$ and the product of the quotient and the auxiliary in row $(m-1)$ until the last remainder equals to 2^0. The final entry in Auxiliary happens to be the multiplicative inverse of the product, thus the multiplicative inverse of 84 is $23 + 22 + 20 = 13$ which agrees with the preceding example.

The polynomials of degree $(m-1)$ are not closed under multiplication. For example, $x^{m-1} \times x^{m-1}$ is x^{2m-2}. Thus for all $m > 1$, the degree of the product may exceed $(m-1)$. Our objective is to build a field of 2^m elements in which the operations of addition and multiplication are based upon polynomial addition and multiplication. Thus, we need a mechanism for ensuring that multiplication is closed. To do this we resort again to modular arithmetic.

A generating polynomial for $GF(p^m)$ is a degree m polynomial that is irreducible over Z_p. This simply means that it cannot be factored. For example $(x^3 - 1)$ is not irreducible over Z_2 because it can be factored as $(x^2 + x + 1)(x + 1)$. Note that this factorization works only over Z_2 but not Z. If an irreducible polynomial $p(x)$ can be found, then polynomial multiplication can be defined as standard polynomial mul-

tiplication modulo $p(x)$. That is, to compute the product $a(x)b(x)$, we need to first compute $c(x) = a(x)b(x)$ and then transform $c(x)$ back into the set of polynomials of degree $(m-1)$ by taking the remainder when $c(x)$ is divided by $p(x)$. If $c(x)$ already has degree no larger than $(m-1)$, then the remainder is simply $c(x)$, but if this is not the case, the remainder is guaranteed to have degree no higher than $(m-1)$.

Note that the requirement that $p(x)$ be irreducible is implicit in this definition of multiplication. Suppose $p(x)$ is not irreducible. Then there exist two polynomials $a(x)$ and $b(x)$ such that $p(x) = a(x)b(x)$. However, $p(x) = 0 \ mod \ p(x)$. Hence $a(x)$ and $b(x)$ are divisors of zero, and it has previously been shown that fields may not contain zero divisors.

3.2 Reed-Solomon Codes

Figure 3.1 shows a typical error correction process using the Reed-Solomon (RS) encoder and decoder. The RS encoder takes the message to be transmitted as a block of digital data, and adds redundant bits to the block. A noisy transmission channel introduces errors in the transmitted message block. The decoder processes the received block and attempts to correct the errors to recover the original message data. The redundant bits help the decoder to recover the message from the corrupted received codeword. The error correction capability in terms of the type and the number of errors that the system can correct, is determined by the RS code used.

RS codes defined over $GF(2^m)$ are popular error-correcting codes used in digital communications and data storage. RS codes are a subset of the Bose-Chaudhuri-Hocquenghem (BCH) [7] codes which are a subset of cyclic block codes. Cyclic block codes belong to the family of the block codes for error correction in general. [19].

Given the value m for the Galois field extension $GF(P^m)$, a set of RS codes can

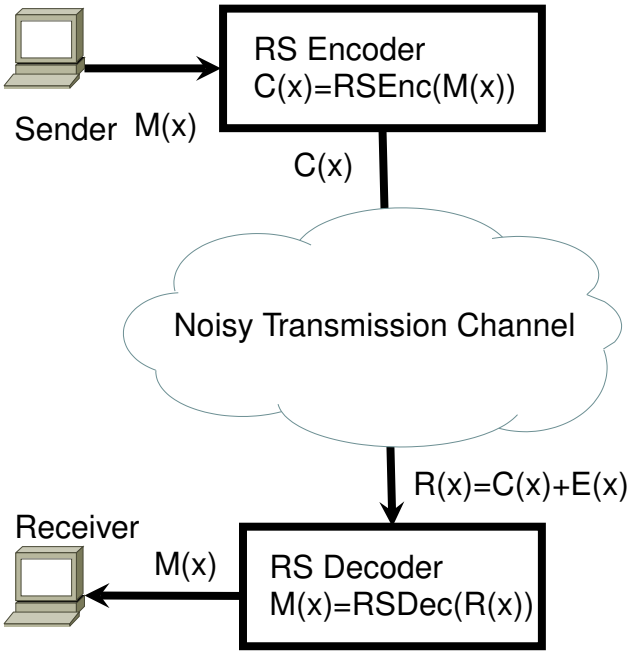


Figure 3.1: Error Correction using RS Codes

be constructed with different block lengths, error correction capabilities and correction rates. The primitive element $a(x)$ and the field generator polynomial $F(x)$ are used to construct each of the P^m unique code symbols. The generator polynomial $p(x)$ having its roots from $GF(P^m)$ field, provides the information for the parity check. The values of $p(x)$, m an n define a n, k RS code. However, when we get into the implementation we need to also know P (2 for ECC in most cases), $F(x)$ (which is the primitive polynomial $p(x)$ in most cases), $a(x)$ ($x = \alpha$ in almost all cases), and α^G (any primitive element of $GF(P^m)$ using $F(x)$ which is almost always set to α^1).

3.2.1 Encoder

The encoders for Reed-Solomon use a systematic encoding architecture using a linear feedback shift register as shown in Figure 3.2. For an $RS(n, k)$ encoder, $n-k$ stages of multiplier-adder pairs are used. The multipliers are constant multipliers with

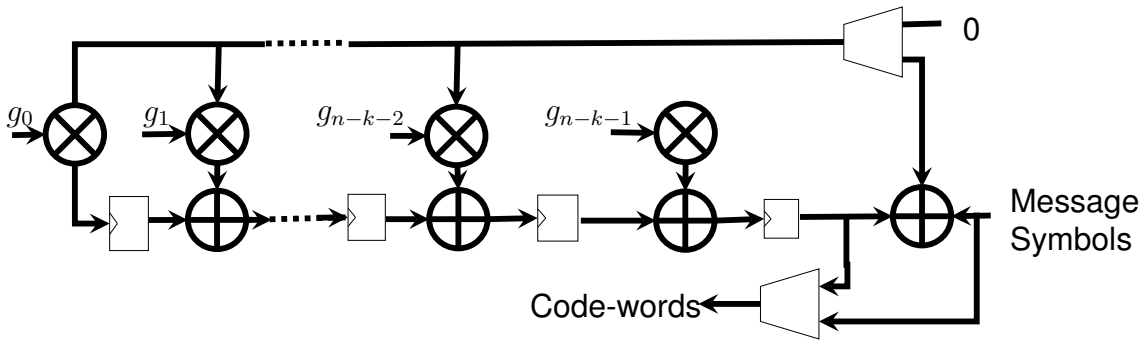


Figure 3.2: Encoding process for Reed-Solomon Code

coefficients of the generator polynomial, $p(x)$ as the constants. This architecture takes n clock cycles to encode an input word. However, parallel architectures have also been invented to perform the encoding at much higher rates.

3.2.2 Decoder

A number of algorithms have been proposed for decoding of RS codes. One category of decoders known as Hard Decision decoders, which have lower error correction capability as well as lower complexity. Soft decision decoders have higher error correction capability and higher complexity. Examples of the hard decision decoding algorithms include the Euclidean algorithm, the Peterson-Gorenstein-Zierler algorithm and Berlekamp-Welch algorithm. In this dissertation, we focus on the Reformulated inversionless Berlekamp-Massey Algorithm (RiBM) for our implementation of the decoder, shown in Figure 3.3.

The decoding process is executed in five-stages involving the following steps:

1. Calculation of the syndromes from the received word.

2. Calculation of the error-locator word from the syndromes.

3. Calculation of the error locations within the received word from the error-locator

numbers.

4. Calculation of the error magnitudes at each error location from the syndromes and the error-locator numbers.

5. Calculation of the decoded codeword from the received word using the calculated error locations and error magnitudes.

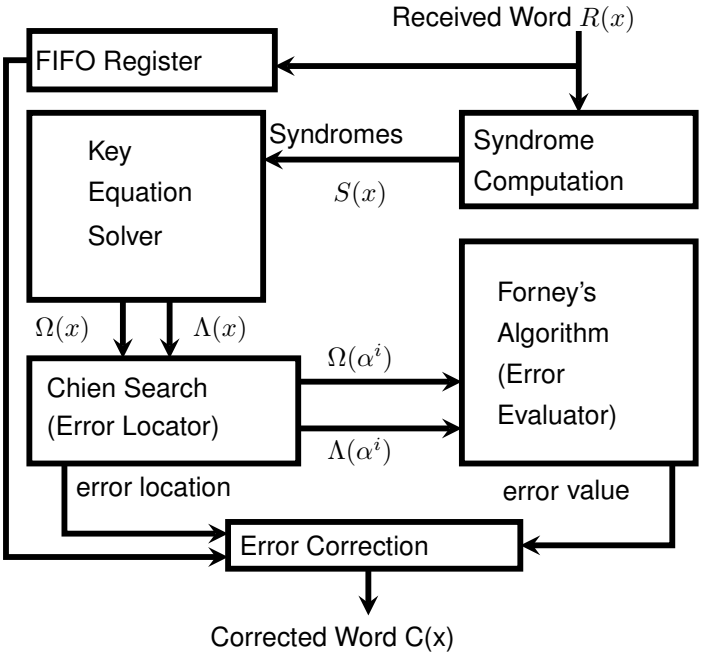


Figure 3.3: Reed-Solomon Decoder Block Diagram

First, we calculate the $2t$ syndrome components or in other words, the syndrome $S(x)$. The syndromes can be determined by either of the two methods:

$$S_i = R(\alpha^I)(= E(\alpha^I)). \tag{3.13}$$

or

$$\begin{aligned} s(x) &= R(x) mod\ p(x) \\ &= REM[R(x)/p(x)]. \end{aligned} \tag{3.14}$$

where $S_i = s(\alpha^I)$ from $I = G_I$ from $p(x)$.

From all the syndromes S_i we calculate the error-locator polynomial $\Omega(x)$. This is done by using one of the two methods: the Berlekamp-Massey's method for error-locator polynomial or the linear recursion method. Next, from the error-locator polynomial $\Omega(x)$, we calculate the error-locator numbers z_i for $i = l, 2, 3, \ldots, T$. Then we calculate the error locations λ_i for $i = l, 2, 3, \ldots, T$. Error locations are the roots of the error magnitude polynomial, $\Lambda(x)$. The error locations can be calculated using one of the two methods, namely, the Explicit Method and the Chien Search Method.

From the error-locator numbers z_i and the syndrome components S_i, we calculate the error values λ_i for $i = l, 2, 3, \ldots, T$. From the error locations, ω_i and the error values λ_i, the estimate of the error polynomial $E(x)$. In the final stage, the determination of the nearest code word $C(x)'$ is done from $R(x)$ and $E(x)$. $R(x)$ and $E(x)$ can be **XOR** added to reproduce the original codeword $C(X)$.

Chapter 4

Research Methodology

We use an implementation of Reed-Solomon Decoder with the Reformulated inversion-less Berlekamp-Massey architecture to illustrate the proposed hardware obfuscation idea. Our idea is to incorporate a reconfigurable design to all the finite field multiplier units to achieve obfuscation without increasing significant overhead.

4.1 RS decoder

As explained in the previous section, the decoder consists of three steps as shown in Figure 3.3: syndrome computation (SC), key equation solver (KES), and Chien search. After this, the obtained Error polynomial, $E(x)$ is bitwise XOR'ed with the received word to obtain the corrected code-word. Our implementation of the RS decoder uses a partial parallel architecture for the syndrome computation block and the RiBM architecture for the KES block.

4.2 Syndrome Computation: J-Parallel Architecture

Recall that α is a primitive element of $GF(2^m)$ and $R(x)$ is the received codeword represented as a binary polynomial. A $RS(n,k)$ code can correct $t = (n-k)/2$ errors in the $R(x)$. For an t error-correcting code, $2t$ syndromes s_j $(1 < j \leq 2t)$ are computed as $R(\alpha^j)$. This requires the use of constant multipliers. Constant multipliers are implemented as binary constant matrix multiplications using **XOR** trees. Using $2t$ syndrome computation cells each with a constant multiplier work in the syndrome computation block to generate $2t$ syndromes for each received word. The syndrome computation can be implemented in a serial manner as illustrated in Figure 4.6a. The value in the register is multiplied with α^j in each clock cycle. The output from the multiplier is added to the input symbol and the result is again stored in the register for calculation of the next syndrome. It takes n clock cycles to calculate j syndromes. However, for faster computation of syndromes, the serial architecture can be upgraded into a parallel architecture by using $2t$ copies of the serial computation unit. Since, syndromes are values evaluated from a polynomial, a design similar to the Chien search can also be used for computation of syndromes.

We use the J-parallel architecture to speed up the process which looks like Figure 4.1 at the RTL Level. Each cell in the block is basically a single serial syndrome computation unit. J coefficients of the received word are grouped together and then the Horner's rule for converting the polynomial in a more computationally efficient is applied to obtain this parallel architecture. A J-parallel architectures need n/J clock cycles to compute each syndrome. If $J = n$, then no feedback loop is required. In such a design, all syndromes can be calculated in a single clock period.

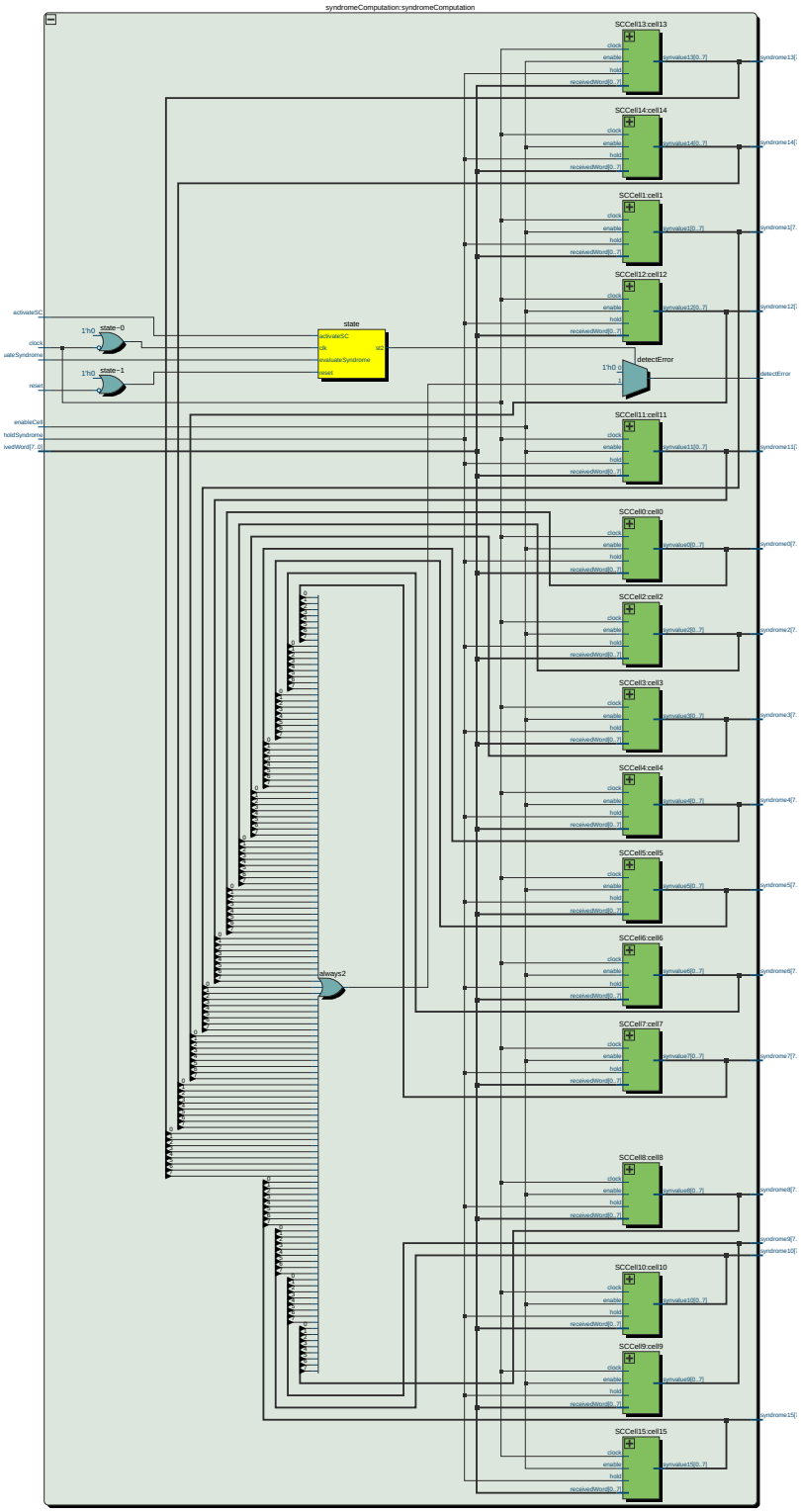


Figure 4.1: Syndrome computation block using J-Parallel architecture showing a stack of syndrome computation cells operating simultaneously to produce $2t$ syndromes per clock cycle

4.3 KES Block: RiBM Architecture

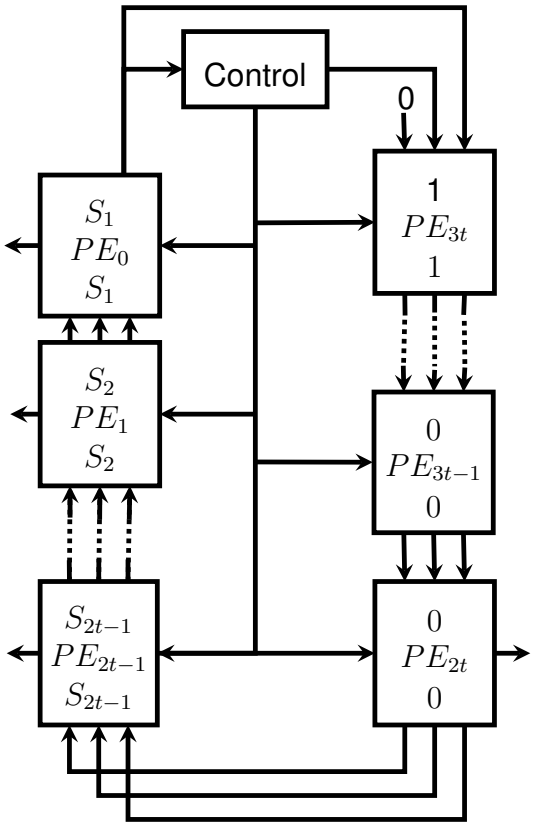


Figure 4.2: Key Equation solver (KES) block based on the RiBM architecture.

The syndromes are used by the KES block to calculate the two polynomials, viz. $\Lambda(x)$ and an error evaluator polynomial $\Omega(x)$ are computed from the syndromes in the KES step. Most practical systems adopt finite field $GF(2^m)$ ($m \in Z^+$). $GF(2^m)$ consists of 2^m elements. Each element is represented by an m-bit binary vector or equivalently as the coefficients of a degree $(m-1)$ binary polynomial. Addition operation is evaluated as a bit-wise XOR. The multiplication between $a(x), b(x) \in GF(2^m)$ is defined as $c(x) = a(x)b(x) \mod p(x)$, where $p(x)$ is a degree-m binary irreducible polynomial. Let

$$a(x) = a_{m-1}x^{m-1} + a_1 x + a_0. \tag{4.1}$$

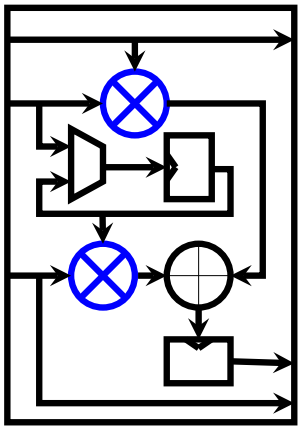


Figure 4.3: RiBM's PE architecture

Then the field multiplication of $a(x)$ and $b(x)$ is calculated as:

$$c(x) = a_0 b(x) + a_1(b(x)x) \ mod \ p(x) + \cdots + a_{m-1}(b(x)x^{m-1} \ mod \ p(x)). \tag{4.2}$$

where

$$b(x)x \ mod \ p(x) = (b_{m-2} \oplus p_{m-1})x^{m-1} + (b_{m-3} \oplus p_{m-2})x^{m-2} + \cdots + (b_0 \oplus p_1)x + p_0 x. \tag{4.3}$$

and $b(x)x^i$ for $i > 1$ are computed in the same manner iteratively.

One of the most efficient architectures for the KES step is the reformulated inversion-less Berlekamp-Massey (RiBM) architecture [38] shown in Figure 4.2. It consists of $(3t + 1)$ copies of the processing element (PE) block shown in Figure 4.3. Of these, each of the $3t$ instances of the PE block contains two instances of a $GF(2^m)$ multiplier. The remaining one PE unit contains one $GF(2^m)$ multiplier. According to equation (4.3), the multiplier is implemented by the architecture in Figure 4.4, and the XTime block implements multiplication with x. The XTime block design is dependent on the primitive polynomial, $p(x)$. In Figure 4.5, an example for the XTime block is shown for $p(x) = 111000011$. The bit outputs corresponding to a set

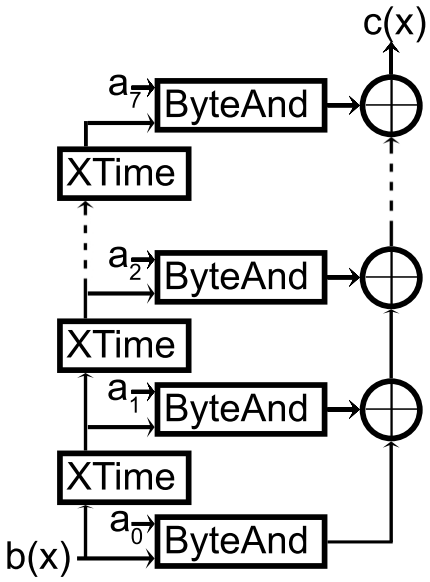


Figure 4.4: GF Multiplier

bit in the primitive polynomial, $p(x)$, are bitwise XORs of the corresponding previous bit and the most significant bit of the input byte. Thus for different $p(x)$, the XTime architecture will be different.

4.4 Chien Search and Error Evaluator Block: Partial Parallel Design

The roots of the error locator polynomial, $\Lambda(x)$ provide the locations of the error in the received codeword. $\Lambda(x)$ has a root of α^{-i} indicates that the i^{th} symbol in the received codeword is corrupted by an error value. Similarly to obtain the value of the error that has infected a symbol, roots of the error evaluator, also known as error magnitude polynomial, $\Omega(x)$ are required to be calculated. Adding the error magnitude to the infected symbol indicated by the roots of $\Lambda(x)$ will give the correct symbol that was originally transmitted. Chien search is carried out on $\Lambda(x)$ and $\Omega(x)$ to find the roots, which are used to generate the error locations and magnitudes.

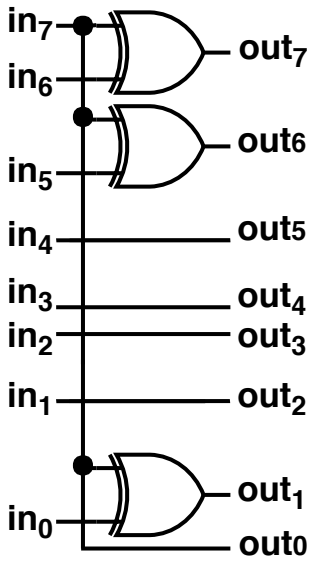


Figure 4.5: XTime for $p(x) = 111000011$

In the original Block Matching Algorithm (BMA) algorithm, calculation of roots of the $\Omega(x)$ is only possible after the roots of $\Lambda'(x)$ have been obtained, where $\Lambda'(x) = \Lambda_1 + \Lambda_3 x^2 + \Lambda_4 x^3 \ldots$. In the case of binary finite fields, consider the two polynomials derived from the even and odd coefficients of $\Lambda(x)$, i.e. for

$$\Lambda(x) = \Lambda_0 + \Lambda_1 x + \Lambda_2 x^2 \ldots. \tag{4.4}$$

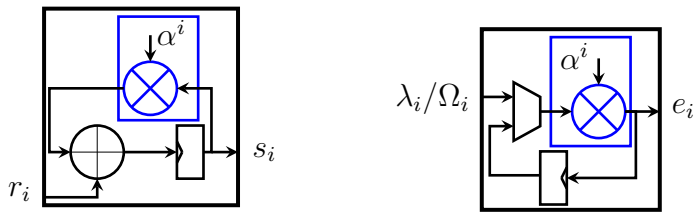
its even and odd polynomials are

$$\Lambda_{odd}(x) = \Lambda_1 + \Lambda_3 x^3 + \Lambda_5 x^5 \ldots; \quad \Lambda_{even}(x) = \Lambda_0 + \Lambda_2 x^2 + \Lambda_4 x^4 \ldots. \tag{4.5}$$

Consequently,

$$\Lambda_{odd}(x) = x\Lambda'(x). \tag{4.6}$$

Therefore, the values evaluated from a separate Chien search block over the odd polynomial can be used for calculation of the error magnitude values [11].



(a) SC Cell (b) CS Cell

Figure 4.6: Individual cells in the parallel architectures of the syndrome computation and Chien search blocks

Thus, the Chien search architecture can also be made to be partially parallel. The design we implemented in our experiment is shown in Figure 4.7. As can be noted from the design of the individual cells in the Chien Search block shown in Figure 4.6b, each Chien search block employs a constant multiplier in which the constant is a power of α, a primitive element in $GF(2^m)$. The Chien search architecture also consists of arrays of constant multipliers.

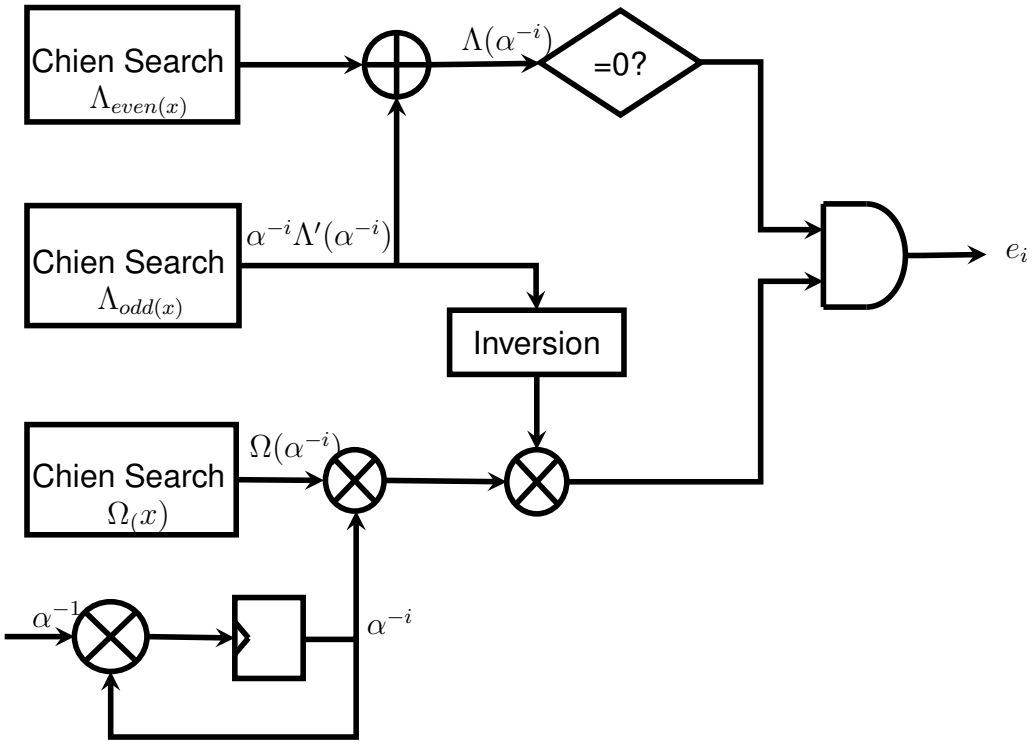
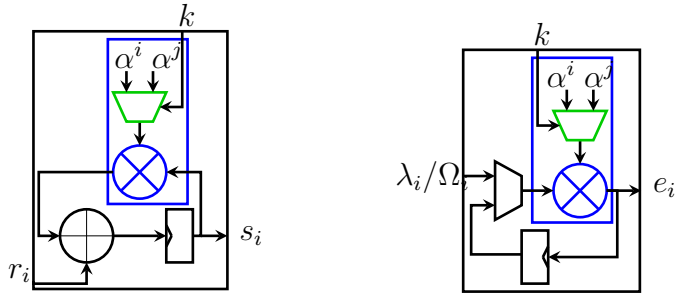


Figure 4.7: Chien Search and error evaluation architecture

4.5 Reconfigurable Finite Field Units

As discussed in the previous sections, the implementation of the XTime block is decided by the irreducible polynomial $p(x)$. Since this polynomial is not unique, the general multipliers can be designed to be reconfigurable according to a different $p(x)$ which can be efficiently exploited for hardware obfuscation. A key can be used to control the mode of operation of the XTime block where different modes correspond to different primitive polynomials incorporated in the unit's reconfigurable design. Other legitimate polynomials integrated into the design can serve as meaningful obfuscating modes. In addition, this reconfigurable GF multiplier can be leveraged to operate on the input for the same number of iterations to maintain nearly indistinguishable side-channel leakage profile, while still yielding incorrect outputs for the incorrectly applied keys. From the section 4.3 we also know that the XTime block is instantiated $(m-1)$ times in a $GF(2^m)$ multiplier that operates on the input $(m-1)$ times recursively. This creates a large degree of freedom for the obfuscation combined by all the instances that are extremely suitable for hardware obfuscation.

A similar concept also applies to the syndrome computation block as well as the Chien search block. As discussed in the previous section and shown in Figure 4.6, both have constant multipliers for multiplication with $m \times m$ binary constant matrices. The entries of the constant multipliers are determined according to $p(x)$. Figure 4.8a and Figure 4.8b show the modified designs of cell templates in the syndrome computation and Chien search blocks, respectively. In Figure 4.6, we observed that each root computation cell in the syndrome computation block as well as in each of the Chien search block, contains a constant field multiplier unit.



(a) Reconfigured SC Cell (b) Reconfigured CS Cell

Figure 4.8: Individual cells using a control signal for reconfiguration.

4.6 Final Design

In the final design of our version of RS decoder, we leverage the concepts of reconfigurability discussed above to exploit it to build a hardware design of the decoder which should also provide security against IP theft and reverse engineering.

In the syndrome computation and Chien search blocks, we replace the default constant multipliers with instances of the new reconfigurable constant multiplier designed for a number of primitive polynomials. A control signal, S will switch the mode of operation of the multiplier. The value of the control signal will be dependent on the applied key. For correct key value, the control signal activates the mode of operation of the constant multiplier corresponding to the correct value of $p(x)$. For an incorrect value of the applied key, the activated mode can be any of those associated with the other primitive polynomials. For an adversary, both of these modes appear to be meaningful modes, since in both modes, the computation inside the decoder is similar to a decoder with the corresponding polynomial as the correct polynomial. The decision about the length of the key is left to the designer. It is a widely accepted fact that a longer key generally provides a higher level of security. However, longer keys may incur higher overheads of hardware complexity.

The general GF multiplier in the KES' PE units, are directly configured ac-

cording to the primitive polynomials and the key length. The modifications are similar to that for constant multipliers, the difference being that the design changes are in the XTime block which is instantiated in the GF multipliers. We tested multiple configurations and key lengths, which we describe in the next section.

4.7 Configurations

Numerous different configurations can be employed in our proposal of reconfigurable finite field units. We consider two main configurations.Two primitive polynomials and four polynomials for reconfigurations.

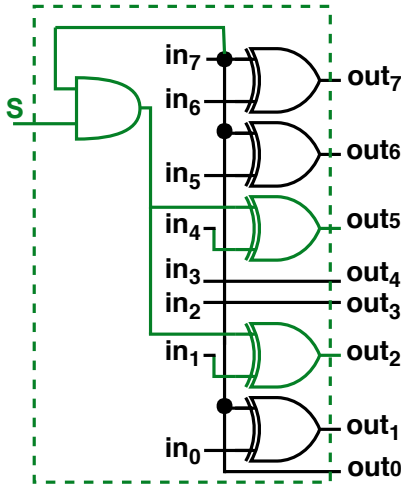


Figure 4.9: Obfuscated XTime block with an 1-bit key: $x = 0$, XTime works for $p(x) = 111000011$; $x = 1$, XTime works for $q(x) = 111100111$.

In the case of using two polynomials, we assume that one correct polynomial is $p(x)$ and the incorrect polynomial is $q(x)$. Since there are only two modes of operation for the XTime block, a one-bit control signal is sufficient. This means that if the applied key to the chip is correct, then the control signal puts the XTime block in the mode corresponding to $p(x)$ which enables the decoder to work as a normal functional decoder for polynomial, $p(x)$. On the other hand, incorrect key causes

the control signal to put the chip into a mode suitable for $q(x)$ but not for $p(x)$. An implementation for the reconfigurable XTime block, with $p(x) = 111000011 = x^8 + x^7 + x^6 + x + 1$ and $q(x) = 111100111 = x^8 + x^7 + x^6 + x^5 + x^2 + x + 1$, is shown in Figure 4.9, where S represents the single-bit control signal.

This means that the decoder will still try to find any errors in $R(x)$ but will possibly fail. It will fail because according to the syndromes produced the number of errors will be beyond its error correction capability. The control signal or the key

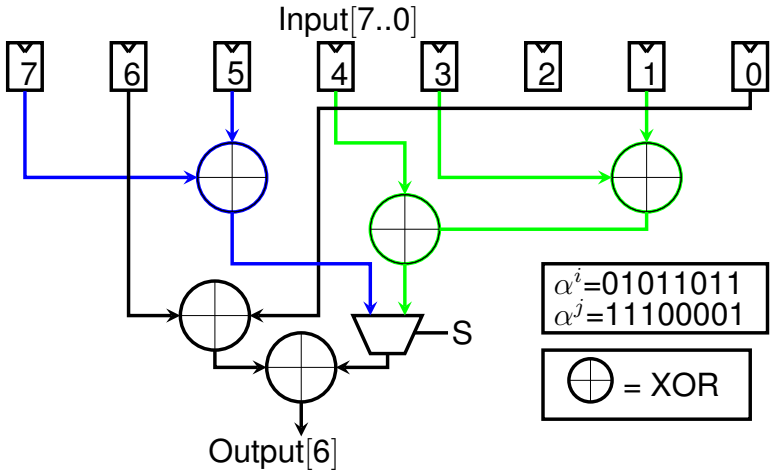


Figure 4.10: An example of the reconfigurable constant multiplier.

itself needs to be propagated from the higher level main external module to each of the XTime blocks inside each of the GF multipliers, as well as inside each of the constant multiplier. The constant multiplier is also redesigned into a reconfigurable constant multiplier design for different α^i value of the multiplier's constant. An example design for $\alpha^i = 11100001$ corresponding to one polynomial, and $\alpha^j = 11100001$ corresponding to another polynomial, with a single bit control signal, S is shown in Figure 4.10. For our experiment, we considered two sub-cases for the setup of one correct primitive polynomial and one incorrect polynomial - a single bit control signal, and a two bit control signal. Additionally, we also experiment with the case of

four polynomials, in which there is one correct polynomial, $p(x)$ and three incorrect polynomials, $q_1(x)$, $q_2(x)$ and $q_3(x)$. Let us discuss each of the three cases in more detail.

4.7.1 One additional primitive polynomial with a single-bit control signal

S	**polynomial**	**mode**	**operation**
1	$p(x)$	meaningful	*correct*
0	$q(x)$	non-meaningful	incorrect

Table 4.1: Modes for a single-bit control signal and two polynomial reconfigurable design

When a single-bit control signal is used in the multiplier design for polynomial selection, a NOT gate is required to invert the signal. This is considered an optimized design where inversion is done only once. Thus an overhead of only one gate cell for each of the XTime blocks in the general GF multipliers, and one gate cell overhead for each constant multiplier block is achieved. Outside the multipliers, only one control signal wire is routed across the parent blocks such as the cells in syndrome computation and Chien search blocks, and the PE units in the KES block.

4.7.2 One additional primitive polynomial with a two-bit control signal

The overhead of inverting the control signal in every constant multiplier and every XTime block in the general multipliers can be avoided by using a two-bit control signal (one bit for each primitive polynomial). This design requires no inverters since each control signal directly maps to one of the modes. Additionally, this also adds

S_1	S_2	**polynomial**	**mode**	**operation**
0	0	–	non-meaningful	incorrect
0	1	$p(x)$	meaningful	*correct*
1	0	–	non-meaningful	incorrect
1	1	$q(x)$	meaningful	incorrect

Table 4.2: Modes for a two-bit control signal and two polynomial reconfigurable design

two incorrect modes or non-meaningful modes, which do not correspond to any of the polynomials considered for the design. Therefore, another layer of ambiguity is introduced in this case without adding any complexity and overhead for two more polynomials.

4.7.3 Three additional polynomials (with four bit control signal)

S_1	S_2	S_3	S_4	**polynomial**	**mode**	**operation**
0	0	0	0	–	non-meaningful	incorrect
0	0	0	1	$p(x)$	meaningful	*correct*
0	0	1	0	$q_1(x)$	meaningful	incorrect
0	1	0	0	$q_2(x)$	meaningful	incorrect
1	0	0	0	$q_3(x)$	meaningful	incorrect
0	0	1	1	–	non-meaningful	incorrect
⋮	⋮	⋮	⋮	⋮	⋮	⋮
1	1	1	1	–	non-meaningful	incorrect

Table 4.3: Modes for a four-bit control signal and four polynomial reconfigurable design

For a higher level of security, the number of operating modes can be increased by having more polynomials incorporated in the GF multipliers and the constant multipliers. A total of four operating modes will need a control signal of a minimum of two bits driven by the values of the applied key. We tested this case with a four-bit control signal to select from the four modes of operation. Similar to two bit

control signal for two polynomial modes, four bit control signal for a four polynomial modes design would discount the overhead of at least two inverters per constant multiplier and two inverters per XTime block. In this configuration, we would have four meaningful modes and twelve non-meaningful modes.

In the next chapter, we present our results and analysis of the results.

Chapter 5

Experimental Results

We implemented the entire $GF(2^8)$ RS(255,239) decoder design in Verilog HDL based on the RiBM architecture with some modifications for higher efficiency. These modifications are adopted from some of the most efficient designs proposed in the literature. We gave preferences to efficient designs that incorporated parallel processing of data which can be explored for observation.

The Chien search and error evaluator block is implemented using the J-parallel architecture [12] so that the computation of the roots of the error locator polynomial can be partially executed while the computation of the roots of the error evaluator polynomial is in process. There are 350 XTime units in total in the KES module. In our experiment, we first considered construction with $p(x) = 111000011$ as the correct configuration. However, we further developed designs for all the combinations of the primitive irreducible polynomials and observed the area, timing and power overheads in each of the designs. We implemented several obfuscated designs with different legitimate polynomials under $GF(2^8)$ through reconfigurable finite field arithmetic units. Note that in the case of $GF(2^8)$, there are 30 irreducible polynomials, out of which 16 are primitive irreducible polynomials that can be used for construction.

The original architecture and the obfuscated designs are synthesized using Synopsys Design Compiler with the $32/28nm$ generic library.

5.1 Impact of Using Combinations of Polynomials

5.1.1 Case 1: Two legitimate polynomials with a 1-bit control signal

One parameter to select the pair of polynomials is to have the minimum number of additional cells/gates to build the obfuscated design. In this experiment, we select the construction corresponding to $q(x) = 111100111$ as one meaningful obfuscating mode, which only has a Hamming distance of 2 to $p(x)$. In the XTime block for these polynomials, there is an **XOR** gate for each of the input bits except for bits 3 and 4. The XOR gates for bit 2 and 3 do not contribute to the output when the correct key is applied, while the **XOR** gates for bits 2 and 3 become operational when the key is configured into the mode corresponding to $q(x)$. Therefore, only 2 additional gates were required and only one-bit control signal is required for each reconfigurable XTime unit. The reconfigurable XTime block design between these two polynomials is shown in Figure 4.9.

5.1.2 Case 2: Two legitimate polynomials with a 2-bit control signal

We also consider the case where the key is mapped to 2-bit control signals for each reconfigurable finite field arithmetic unit with $q(x) = 111100111$. In this case, each of the additional gate in the reconfigurable XTime block is controlled separately.

5.1.3 Case 3: Four legitimate polynomials with a 4-bit control signal

We implemented the obfuscated design with $q_1(x) = 101011111, q_2(x) = 111001111, q_3(x) = 111110101$ as other legitimate primitive polynomials. A 4-bit control signal is mapped to control each reconfigurable finite field arithmetic unit.

Table 5.1: Overhead for four-polynomial configuration

Area	Power	Time
10.34%	2.78%	1.31%
11.02%	2.81%	1.31%

5.2 Overheads

The overhead results for the above three cases are summarized in Table 5.2, which are calculated based on the entire decoder architecture.

Table 5.2: Overhead for each case configuration

Case	Area	Power	Time
1	6.82%	1.23%	7.81%
2	6.17%	1.04%	2.38%
3	10.34%	2.78%	1.31%

It can be seen that the overhead of the proposed hardware obfuscation methodology is small, compared to prior works [20, 28]. It can also be concluded that given a correct construction, different polynomial combinations that are included in the obfuscating modes will yield different hardware design costs. Higher number of legitimate polynomials would certainly increase the design complexity while achieving a higher level of obfuscation. In addition, our experimental results indicate that introducing

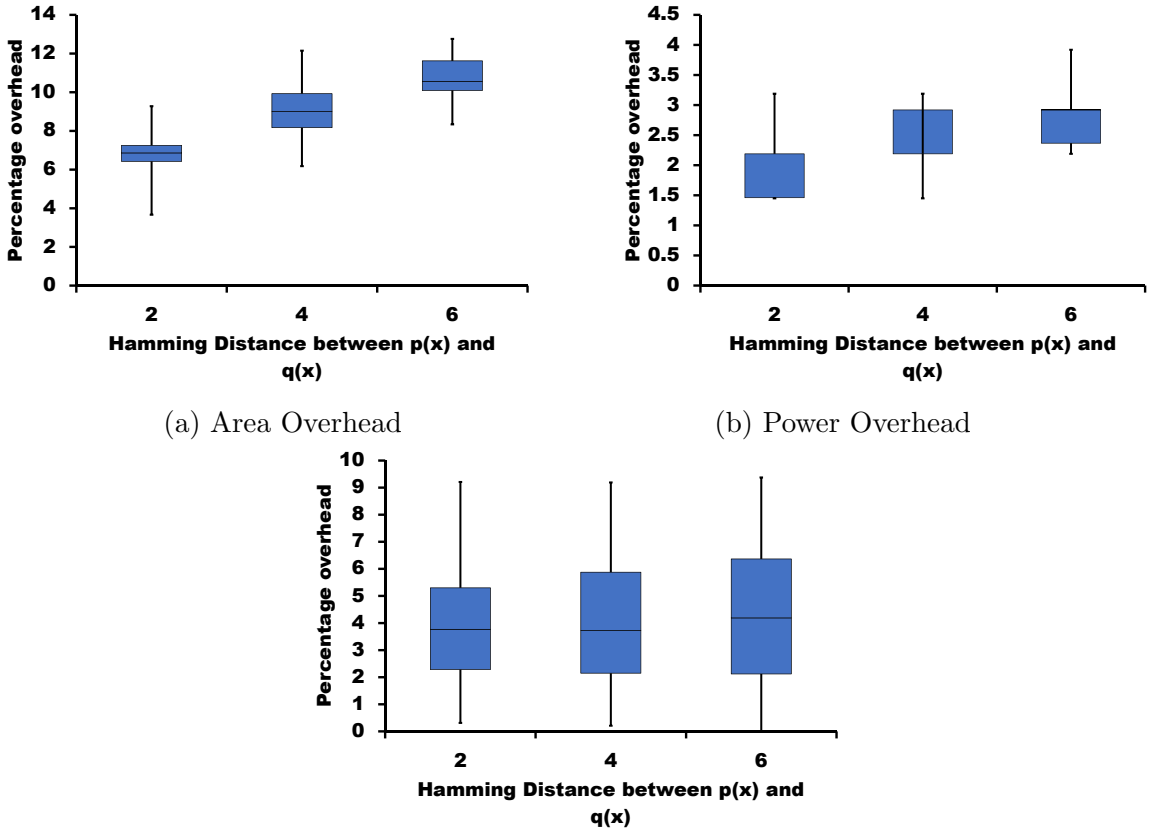
more control signal could reduce the design complexity within the reconfigurable finite field arithmetic units. However, using more bits in the meaningful modes would decrease the degree of freedom for incorporating non-meaningful modes. As a result, the overall obfuscation level might be reduced. Therefore, the detailed obfuscation circuit should be carefully designed according to the performance requirement of a certain application.

5.3 Selecting the Best Polynomial Combination for Secure Reconfigurable Design

Selecting the number of polynomials and the set of those polynomials to implement a reconfigurable multiplier design is crucial to obtaining a low-overhead and secure chip. For a four-polynomial reconfigurable module, one of the $P(16, 4)$ (permutation function, $P(n, r)$) (assuming all four selections are unique) permutations can be used. In order to select the combination wisely the area, power and timing overheads of each combination needs to be compared.

For a two-polynomial reconfigurable module, one of the ${}^{16}P_2$ combinations can be used. We developed RS decoder designs for each of the 16 irreducible primitive polynomials in $GF(2^8)$ as the generator polynomial. We then created the set of reconfigurable decoder designs for each configuration, making them reconfigurable by selecting another irreducible primitive polynomial as the second polynomial (corresponding to the incorrect meaningful mode of the operation).

The variations in the overheads, shown in Figure 5.1 for different values of the Hamming distance is attributed to the reconfigurable constant multiplier designs. Since, there is not much flexibility in modifying the $'i'$ and the $'\alpha'$ values in a RS



(a) Area Overhead

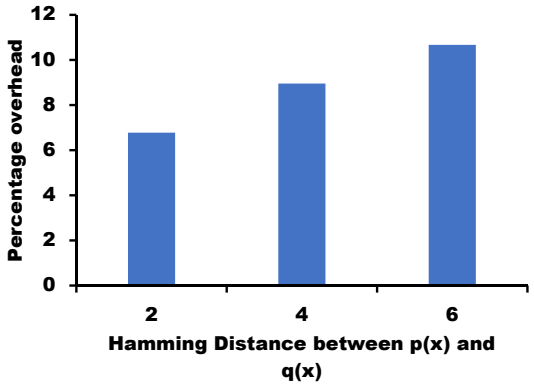
(b) Power Overhead

(c) Timing Overhead

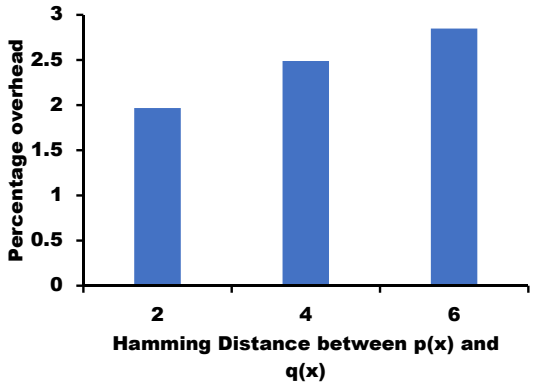
Figure 5.1: Percentage overheads with respect to Hamming Distance between $p(x)$ and $q(x)$

decoder for a specific $p(x)$, the variations in the observed overheads are expected to remain similar for any design of the RS decoder.

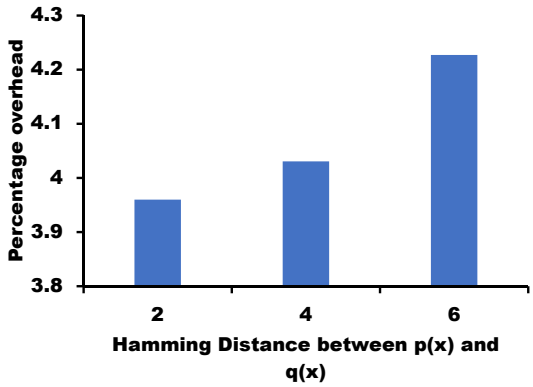
The results are presented graphically in Figure 5.2, which validates our hypothesis that a lower hamming distance between the two polynomials yields lower overall area and power overheads. This is understandable since lower Hamming distance corresponds to less number of gates that are required to incorporate the design associated with the second polynomial.



(a) Average Area Overhead



(b) Average Power Overhead



(c) Average Timing Overhead

Figure 5.2: Percentage average overheads with respect to Hamming Distance between $p(x)$ and $q(x)$

5.4 The Best Pair of $p(x)$ and $q(x)$

Our experiment results also revealed the best combinations that can be used to implement the multipliers with the least overheads in area and power. We present 3 such combinations in Table 5.3.

$p(x)$	$q(x)$	% Area Overhead	% Power Overhead
111001111	110000111	3.67	1.45
111001111	101001101	4.06	1.45
111001111	111000011	4.27	1.45

Table 5.3: The least overhead $p(x)$-$q(x)$ pair

5.5 Security Analysis

The obfuscation level of the design is dependent on the possible configurations in each block. Theoretically, if the control signal of each block is mapped from the key separately, there are maximum 2^k configurations in each block, where k is the length of the key. However, this would require one-to-one mapping for all the control signals, which incurs significant overhead. In general, we assume the possible configurations for each syndrome computation block, GF multiplier in the KES architecture, and constant multiplier in Chien search and Forneys algorithm as 2^{S_1}, 2^{S_2}, and 2^{S_3}, respectively. Since the syndrome computation block produces $2t$ syndromes for a received word, there are $2t \times 2^{S_1}$ possible combinations in the obfuscated design.

In addition, the KES block computes the polynomials, $\Lambda(x)$ and $\Omega(x)$ using $(6t+2)(m-1)$ XTime blocks that can take 2^{S_2} arithmetic forms. As each coefficient in KES is computed using the parameters obtained for the previous coefficients, these units cumulatively contribute in final coefficient value. Furthermore, Chien Search and Forneys algorithm also consist of $(4t-2)$ constant multipliers that can be mapped into 2^{S_3} configurations each depending upon the value of control signal. Therefore, the overall obfuscation level of the proposed design is about $(48t^4-56t^3+8t) \times 2^{S_1+S_2+S_3}$.

However, the RS decoder may not change the output if it concludes that the errors are more than it's designed to correct. Adding an FSM based obfuscation layer between these blocks can change this behaviour causing the decoder to produce a different output even if the number of errors is beyond its correction capability.

Chapter 6

Conclusions and Discussion

In Chapter 1, we discussed the various security challenges that today's semiconductor industry and the electronic industry faces, which includes basically all the computing-based industries viz. embedded systems, data centres, cloud servers, web servers, governments, military equipment, wearable technology as well as individuals themselves. The main focus of our work is to propose a design methodology which can be widely employed in different kind of semiconductor chips. These techniques can be useful to semiconductor IP and IC design companies as well as their clients in building trust with their respective clients and also to keep their business essentials secure.

In chapter 2, we presented a brief background of the IC design supply chain, finite field arithmetic and its applications in coding theory, that will enable the reader to understand the applicability and nature of this research work.

In chapter 3, we went over the recent research progress in the field of hardware obfuscation. We particularly discussed obfuscation with regard to logic encryption and finite field circuits. We acknowledged some of the work that has contributed to our study and analysis, the open challenges that we could take up in order to produce

this work.

In chapter 4, we discussed the details of our research methodology and our approach in producing better designs of the hardware units for a RS decoder. We explained the designs we chose as the basis of our initial implementation and the factors that make them a better choice for our work. Further, we detailed the changes we made to the initial designs in order to improve security of the circuits without compromising on the efficiency and resource consumption of the decoder.

In chapter 5, we presented our findings from our experiment. Our experiment involved using our method for obfuscation of finite field based circuits to implement obfuscation in Reed Solomon error correcting codes by the use of reconfigurable units of finite field computation. We provided results for different scenarios and analyze them to conclude and strengthen the confidence of our findings. We showed that using reconfigurable computation units for finite field arithmetic, we can considerably improve the obscurity of the IC design without having a high overhead.

We produced results for various polynomials and various configurations of reconfigurability and report our observation. Our observation is that the reconfigurability only imposes a low overhead over the original design if the key length is small and it increases with the increase of the key length. Our initial hypothesis that the overheads, on average, increase with increasing hamming distance between the polynomials in the set selected for different modes in the reconfigurable arithmetic units.

For a minimal modification to the circuit, we believe that our design does not give out any significant information in terms of side channel leakage to indicate that the circuit was not functioning correctly when the applied key was incorrect. However, with higher key length and no other method of obfuscation employed, the obscurity of the design is slightly compromised. To prevent this, FSM based scramblers can be

used between the major blocks to introduce additional level of obscurity, making the design more robust to reverse engineering.

In our work, we have presented a novel hardware obfuscation methodology through re-configurable finite field arithmetic units. The obfuscated design can achieve a high level of security by incorporating both meaningful and non-meaningful modes. Techniques for developing efficient obfuscated designs have also been discussed with the use of a Reed-Solomon decoder as an example.

Future work will be directed towards developing obfuscated architectures for other applications on the finite field. There are many directions that can be taken from here to improve upon the work. First, what configuration is the most secure or the least taxing configuration when putting on the final product. Another question that immediately follows is how do we verify and prove that this the best possible configuration without doing an exhaustive search and synthesizing each configuration. Third, which characteristics of the considered polynomials, the field and the error correction capability exactly determine if a configuration is supposed to be a secure one or not and taxing in terms of hardware or not. Furthermore, we only covered the constant and general GF multipliers which although are the foundation units of operations in a finite field, may not be the only units that can be augmented or modified to achieve higher security with lower overheads.

Bibliography

[1] Marco Baldi, Marco Bianchi, Franco Chiaraluce, Joachim Jakob Rosenthal, Davide Mose, et al. Method and apparatus for public-key cryptography based on error correcting codes, November 17 2015. US Patent 9,191,199.

[2] Boaz Barak, Oded Goldreich, Rusell Impagliazzo, Steven Rudich, Amit Sahai, Salil Vadhan, and Ke Yang. On the (im) possibility of obfuscating programs. In *Annual International Cryptology Conference*, pages 1–18. Springer, 2001.

[3] James P Baukus, Lap Wai Chow, and William M Clark Jr. Digital circuit with transistor geometry and channel stops providing camouflage against reverse engineering, July 21 1998. US Patent 5,783,846.

[4] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing ic piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 27(1), 2010.

[5] Christoforus Juan Benvenuto. Galois field in cryptography. *University of Washington*, 2012.

[6] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM (JACM)*, 65(6):39, 2018.

[7] R.C. Bose and D.K. Ray-Chaudhuri. On a class of error correcting binary group codes. *Information and Control*, 3(1):68 – 79, 1960.

[8] Rajat Subhra Chakraborty and Swarup Bhunia. Hardware protection and authentication through netlist level obfuscation. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 674–677. IEEE Press, 2008.

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

[10] Rajat Subhra Chakraborty and Swarup Bhunia. Security against hardware trojan attacks using key-based design obfuscation. *Journal of Electronic Testing*, 27(6):767–785, 2011.

[11] Bainan Chen, Xinmiao Zhang, and Zhongfeng Wang. Error correction for multi-level nand flash memory using reed-solomon codes. In *Signal Processing Systems, 2008. SiPS 2008. IEEE Workshop on*, pages 94–99. IEEE, 2008.

[12] Yanni Chen and Keshab K Parhi. Small area parallel chien search architectures for long bch codes. *Ieee Transactions on Very Large Scale Integration (VLSI) Systems*, 12(5):545–549, 2004.

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

[14] Ronald Cramer, Vanesa Daza, Ignacio Gracia, Jorge Jiménez Urroz, Gregor Leander, Jaume Martí-Farré, and Carles Padró. On codes, matroids and secure multi-party computation from linear secret sharing schemes. In *Annual International Cryptology Conference*, pages 327–343. Springer, 2005.

[15] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES-the advanced encryption standard*. Springer Science & Business Media, 2013.

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

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

[18] Domenic Forte, Swarup Bhunia, and Mark M Tehranipoor. *Hardware protection through obfuscation*. Springer, 2017.

[19] William A Geisel. Tutorial on reed-solomon error correction coding. 1990.

[20] W Paul Griffin, Anand Raghunathan, and Kaushik Roy. Clip: Circuit level ic protection through direct injection of process variations. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(5):791–803, 2012.

[21] Z. Guo, M. Tehranipoor, D. Forte, and J. Di. Investigation of obfuscation-based anti-reverse engineering for printed circuit boards. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[22] Zimu Guo, Mark M. Tehranipoor, and Domenic Forte. *Permutation-Based Obfuscation*, pages 103–133. Springer International Publishing, Cham, 2017.

[23] Hideki Imai and Manabu Hagiwara. Error-correcting codes and cryptography. *Applicable Algebra in Engineering, Communication and Computing*, 19(3):213–228, 2008.

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

[25] Farinaz Koushanfar and Yousra Alkabani. Provably secure obfuscation of diverse watermarks for sequential circuits. In *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*, pages 42–47. IEEE, 2010.

[26] Farinaz Koushanfar, Saverio Fazzari, Carl McCants, William Bryson, Peilin Song, Matthew Sale, and Miodrag Potkonjak. Can eda combat the rise of electronic counterfeiting? In *Design Automation Conference (DAC), 2012 49th ACM/EDAC/IEEE*, pages 133–138. IEEE, 2012.

[27] Yingjie Lao and Keshab K Parhi. Protecting dsp circuits through obfuscation. In *Circuits and Systems (ISCAS), 2014 IEEE International Symposium on*, pages 798–801. IEEE, 2014.

[28] Yingjie Lao and Keshab K Parhi. Obfuscating dsp circuits via high-level transformations. *IEEE transactions on very large scale integration (vlsi) systems*, 23(5):819–830, 2015.

[29] Meng Li, Kaveh Shamsi, Travis Meade, Zheng Zhao, Bei Yu, Yier Jin, and David Z Pan. Provably secure camouflaging strategy for ic protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2017.

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

[31] Bao Liu and Brandon Wang. Embedded reconfigurable logic for asic design obfuscation against supply chain attacks. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[32] Duo Liu, Cunxi Yu, Xiangyu Zhang, and Daniel Holcomb. Oracle-guided incremental sat solving to reverse engineer camouflaged logic circuits. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2016*, pages 433–438. IEEE, 2016.

[33] Inez Miyamoto, Thomas H Holzer, and Shahryar Sarkani. Why a counterfeit risk avoidance strategy fails. *Computers & Security*, 66:81–96, 2017.

[34] Gleb Naumovich and Nasir Memon. Preventing piracy, reverse engineering, and tampering. *Computer*, 36(7):64–71, 2003.

[35] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In Henri Gilbert and Helena Handschuh, editors, *Fast Software Encryption*, pages 413–423, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[36] Ahmad Patooghy, Ehsan Aerabi, Hamidreza Rezaei, Miguel Mark, Mahdi Fazeli, and Michel A Kinsy. Mystic: Mystifying ip cores using an always-on fsm obfuscation method. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 626–631. IEEE, 2018.

[37] Jarrod A Roy, Farinaz Koushanfar, and Igor L Markov. Circuit cad tools as a security threat. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 65–66. IEEE, 2008.

[38] Dilip V Sarwate and Naresh R Shanbhag. High-speed architectures for reed-solomon decoders. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 9(5):641–655, 2001.

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

[40] Goutham NC Shanmugam, Yingjie Lao, and Keshab K Parhi. An obfuscated radix-2 real fft architecture. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 1056–1060. IEEE, 2015.

[41] Claude E Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[42] Claude Elwood Shannon. A mathematical theory of communication. *Bell system technical journal*, 27(3):379–423, 1948.

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

[44] USTR. Ustr releases 2018 special 301 report on intellectual property rights, Apr 2018.

[45] James B Wendt and Miodrag Potkonjak. Hardware obfuscation using puf-based logic. In *Computer-Aided Design (ICCAD), 2014 IEEE/ACM International Conference on*, pages 270–271. IEEE, 2014.

[46] Mingfu Xue, Jian Wang, Youdong Wang, and Aiqun Hu. Security against hardware trojan attacks through a novel chaos fsm and delay chains array puf based design obfuscation scheme. In *International Conference on Cloud Computing and Security*, pages 14–24. Springer, 2015.

[47] Muhammad Yasin, Bodhisatwa Mazumdar, Ozgur Sinanoglu, and Jeyavijayan Rajendran. Security analysis of anti-sat. In *Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific*, pages 342–347. IEEE, 2017.

[48] Cunxi Yu, Xiangyu Zhang, Duo Liu, Maciej Ciesielski, and Daniel Holcomb. Incremental sat-based reverse engineering of camouflaged logic circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(10):1647–1659, 2017.

[49] Jiliang Zhang and Lu Wan. Cmos: Dynamic multi-key obfuscation structure for strong pufs. *arXiv preprint arXiv:1806.02011*, 2018.