

**T.R.**  
**GEBZE INSTITUTE OF TECHNOLOGY**  
**INSTITUTE OF SCIENCES AND ENGINEERING**

**NONLINEAR SYSTEMATIC FAULT  
DETECTION CODES FOR  
CRYPTOGRAPHIC ALGORITHMS AND  
THEIR IMPLEMENTATIONS**

**KAZIM YUMBUL**  
**PH. D. THESIS**  
**ELECTRONICS ENGINEERING DEPARTMENT**

**GEBZE**

**2013**

**T.R.**  
**GEBZE INSTITUTE OF TECHNOLOGY**  
**INSTITUTE OF SCIENCES AND ENGINEERING**

**NONLINEAR SYSTEMATIC FAULT  
DETECTION CODES FOR  
CRYPTOGRAPHIC ALGORITHMS AND  
THEIR IMPLEMENTATIONS**

**KAZIM YUMBUL**  
**PH. D. THESIS**  
**ELECTRONICS ENGINEERING DEPARTMENT**

**ADVISORS**  
**Yrd. Doç. Dr. Serdar Süer Erdem**  
**Doç.Dr.Erkay Savaş**

**GEBZE**

**2013**



## DOKTORA TEZİ JÜRİ ONAY SAYFASI

G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun 04/01/2013 tarih ve 2013/01 sayılı kararıyla oluşturulan jüri tarafından 17/01/2013 tarihinde tez savunma sınavı yapılan Kazım YUMBUL'un tez çalışması Elektronik Mühendisliği Anabilim Dalında DOKTORA tezi olarak kabul edilmiştir.

### JÜRİ

ÜYE

(TEZ DANIŞMANI) : Yrd.Doç.Dr. Serdar Süer ERDEM

*S. S. Eellen*

(TEZ DANIŞMANI) : Doç.Dr. Erkay SAVAŞ

*Erkay Savaş*

ÜYE

: Prof.Dr. İbrahim SOĞUKPINAR

*İbrahim Soğukpınar*

ÜYE

: Doç.Dr. İsmail Hakkı TAYYAR

*İsmail Hakkı Tayyar*

ÜYE

: Yrd.Doç.Dr. Tuğrul YANIK

*Tuğrul Yanık*

ÜYE

: Yrd.DoçDr. Köksal HOCAOĞLU

### ONAY

G.Y.T.E. Mühendislik ve Fen Bilimleri Enstitüsü Yönetim Kurulu'nun ...../...../20... tarih ve ...../..... sayılı kararı.

İMZA/MÜHÜR

## SUMMARY

**TITLE** : NONLINEAR SYSTEMATIC FAULT DETECTION CODES FOR CRYPTOGRAPHIC ALGORITHMS AND THEIR IMPLEMENTATIONS.

**AUTHOR** : Kazım YUMBUL

In this thesis, we design and implement a robust execution unit as an extension to a configurable processor, which detects the data faults while performing the arithmetic operations. Assuming a capable adversary who can inject faults to the cryptographic computation with high precision, a nonlinear error detection code with high error detection capability is used. The designed units are tightly integrated to the datapath of the configurable processor using its tool chain. For different configurations, we report the increase in space and time complexities of the configurable processor. Also, we present performance evaluations of the software implementations using the robust execution units. Implementation results show that it is feasible to implement robust arithmetic units with relatively low overhead in an embedded processor.

In addition to hardware implementation of the nonlinear error detection codes, we re-consider a class of nonlinear error detection codes in more realistic an adversarial model than assumed in similar works in literature, where a powerful attacker can introduce faults with high precision and accuracy. We present two analysis techniques that can lead to successful attacks against quadratic codes if the modulus is not chosen carefully. We provide in-depth theoretical analysis that covers wide range of attacks and present results of practical concerns such as exact number of undetected faults and effective countermeasures. Our analysis is generic in the sense that it can be extended to other similar error-detection codes.

**Keywords:** Security, Cryptographic Algorithms, Fault Attack, Fault Detection, Instruction Set Extension, Computer Architecture

## ÖZET

**BAŞLIK : KRİPTOGRAFİK ALGORİTMALAR İÇİN**  
**DOĞRUSAL OLMAYAN SİSTEMATİK HATA**  
**KORUMALI KODLAR VE UYGULAMALARI**

**YAZAR ADI:** Kazım YUMBUL

Bu tezde, aritmetik işlemler gerçekleştirilirken veri hatalarını belirleyebilen yapılandırılabilir bir işlemciye ek olarak gürbüz bir yürütme birimi tasarlanmış ve gerçekleştirilmiştir. Yetenekli bir saldırganın yüksek hasasiyetle hata oluşturma kabiliyetinin olduğu varsayılırsa, bu hatalardan korunmak için yüksek hata belirleme kabiliyetine sahip olan doğrusal olmayan hata belirleme tekniği kullanılabilir. Tasarlanan birimler yapılandırılabilir işlemcinin veriyoluna kendi geliştirme araçları kullanılarak sıkı bir şekilde entegre edilmiştir. Zaman ve alan bakımından değişik kofigürasyonlar için yapılandırılabilir işlemcinin kompleksliğinin arttığı raporlanmıştır. Ayrıca, gürbüz yürütme birimi kullanılarak yazılım uygulamalarının performans değerlendirilmeleride verilmiştir. Uygulama sonuçları bir gömülü işlemcide nispeten düşük bir masraf ile gürbüz işlemci birimlerinin gerçekleştirilmesinin mümkün olduğu gösterilmiştir.

Doğrusal olmayan hata belirleme kodlarının donanım gerçeklemesine ek olarak bu kodlar güçlü bir saldırganın hataları yüksek çözünürlükte ve doğrulukta uygulayabildiği bir saldırı modelinde tekrar ele alınmıştır. Modüllerin dikkatli bir şekilde seçilmediği durumlarda quadratik kodlara karşı başarılı saldırılar yapılabilecek iki analiz tekniği sunulmuştur. Belirlenemeyen hataların tam sayısı ve etkin karşı önlemler gibi pratik işlerin sonuçları sunulmuş ve geniş bir yelpazede çeşitli ataklar kapsanarak derinlemesine bir teorik analiz yapılmıştır. Yapmış olduğumuz analiz genel bir analizdir ve diğer benzer hata belirleme kodlarına da uygulanabilir.

**Anahtar Kelimeler:** Güvenlik, Kriptografik Algoritmalar, Hata Saldırısı, Hata Belirleme, Komut Seti Uzantisi, Bilgisayar Mimarisi

## ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all those who have helped me in my Ph.D. studies at Gebze Institute of Technology. I am indebted to my thesis advisors, Dr. Serdar Süer Erdem and co-advisor Dr. Erkey Savaş, for the knowledge they have imparted with me. I also would like to express my sincere gratitude for the continued encouragement and invaluable suggestions.

I thank Prof. Dr. İbrahim Soğukpınar, Dr. Tuğrul Yanık, Dr. Köksal Hocaoglu, and Dr. İsmail H. Tayyar, my committee members, for the time and consideration given to this work.

I would like to thank to ASELSAN, MGEO Electronics Design Engineering Departments for the support given throughout this study.

Finally, I am deeply indebted to my mother Nevin Yumbul, my father A.Fikri Yumbul, and my brother M. Gökhan Yumbul for their endless love and support. You make life much more bearable.

# TABLE OF CONTENTS

|                                                        |    |
|--------------------------------------------------------|----|
| SUMMARY                                                | iv |
| ÖZET                                                   | v  |
| ACKNOWLEDGEMENTS                                       | vi |
| TABLE OF CONTENTS                                      |    |
| NOTATIONS AND ABBREVIATIONS                            |    |
| LIST OF FIGURES                                        |    |
| LIST OF TABLES                                         |    |
| 1 INTRODUCTION                                         | 1  |
| 2 CRYPTOGRAPHIC BACKGROUND AND FAULT ATTACKS           | 5  |
| 2.1 Basic Cryptographic Background                     | 5  |
| 2.1.1 Symmetric Key Algorithm                          | 5  |
| 2.1.2 Public Key Algorithm                             | 5  |
| 2.2 Side-Channel and Fault Attacks                     | 6  |
| 2.2.1 Timing Attack                                    | 7  |
| 2.2.2 Power Attack                                     | 8  |
| 2.2.3 Radiation Source Attack                          | 8  |
| 3 ROBUST ARITHMETIC CODES                              | 10 |
| 3.1 Quadratic Residue Codes                            | 10 |
| 3.2 Dual Residue Codes and Their Performance           | 11 |
| 3.2.1 Dual Residue Codes with Identity Function        | 11 |
| 3.2.2 Dual Residue Codes with Hybrid Function          | 12 |
| 3.2.3 Quadratic Dual Residue Codes                     | 14 |
| 4 ROBUST EMBEDDED PROCESSOR IMPLEMENTATION             | 16 |
| 4.1 Error Detection Strategy for An Embedded Processor | 16 |
| 4.2 General Architecture                               | 17 |

|       |                                                                        |    |
|-------|------------------------------------------------------------------------|----|
| 4.2.1 | Robust Adder                                                           | 18 |
| 4.2.2 | Robust Multiplier                                                      | 20 |
| 4.2.3 | Pipeline Integration of The Proposed Robust Functional Units           | 22 |
| 4.3   | Processor Configuration for Secure Software                            | 23 |
| 4.3.1 | Configuration 0                                                        | 24 |
| 4.3.2 | Configuration 1                                                        | 25 |
| 4.3.3 | Configuration 2                                                        | 26 |
| 4.3.4 | Configuration 3                                                        | 27 |
| 4.3.5 | Configuration 4                                                        | 28 |
| 4.3.6 | Configuration 5                                                        | 28 |
| 4.3.7 | Configuration 6                                                        | 28 |
| 4.3.8 | Configuration 7                                                        | 29 |
| 4.3.9 | Configuration 8                                                        | 29 |
| 4.4   | Implementation Results                                                 | 29 |
| 5     | ON SELECTION OF MODULUS OF QUADRATIC CODES                             | 34 |
| 5.1   | Fault and Attacker Model                                               | 34 |
| 5.2   | Attacks on Quadratic Codes                                             | 36 |
| 5.2.1 | Exploiting Sparse Modulus Multiples                                    | 37 |
| 5.2.2 | Exploiting Dense Modulus Multiples                                     | 38 |
| 5.2.3 | Exploiting Modulus Multiples Having Many Consecutive<br>Nonzero Digits | 39 |
| 5.2.4 | Exploiting Modulus Multiples Having Many Consecutive Zero<br>Digits    | 41 |
| 5.3   | Data Word Attacks Using Multiples of Modulus                           | 43 |
| 5.3.1 | Data Words for Which Attack is Undetectable                            | 44 |
| 5.3.2 | Exact Count of Detection Failures                                      | 45 |
| 5.3.3 | Basic Attack Strategy                                                  | 47 |
| 5.3.4 | Maximum Number of Detection Failures                                   | 47 |

|                                                  |                                                       |    |
|--------------------------------------------------|-------------------------------------------------------|----|
| 5.3.5                                            | Experimental Results                                  | 51 |
| 5.4                                              | Data Word Attacks Using Linear Expressions of Modulus | 54 |
| 5.4.1                                            | Data Word for which Attack is Undetectable            | 55 |
| 5.4.2                                            | Exact Count of Detection Failures                     | 56 |
| 5.4.3                                            | Modulus Multiples in Redundant Binary Forms           | 58 |
| 5.4.4                                            | Basic Attack Strategy                                 | 60 |
| 5.4.5                                            | Maximum Number of Detection Failures                  | 61 |
| 5.5                                              | Estimated Count of Detection Failures                 | 67 |
| 5.5.1                                            | For Multiples of Modulus                              | 67 |
| 5.5.2                                            | For Linear Expressions of Modulus                     | 69 |
| 5.6                                              | How to Choose Modulus                                 | 71 |
| 5.6.1                                            | Evaluating Detection Performance of a Modulus         | 71 |
| 5.6.2                                            | The Properties and Example of Good Moduli             | 74 |
| 6                                                | CONCLUSION                                            | 76 |
| APPENDICES                                       |                                                       |    |
| Appendix-A Finding Detection Failure Counts      |                                                       |    |
| Appendix-B Finding $ E $ , $ S_A $ , and $ S_B $ |                                                       |    |
| BIBLIOGRAPHY                                     |                                                       |    |
| VITA                                             |                                                       |    |

## NOTATIONS AND ABBREVIATIONS

|                          |                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------|
| $Q(\varepsilon, \delta)$ | : Error masking probability                                                         |
| $C$                      | : Codeword                                                                          |
| $p$                      | : Prime integer                                                                     |
| $x$                      | : K-bit data word                                                                   |
| $w$                      | : R-bit parity                                                                      |
| $\varepsilon$            | : Fault in data word                                                                |
| $\delta$                 | : Fault in parity                                                                   |
| $\lambda_p$              | : Multiples of prime integer                                                        |
| $\hat{x}$                | : Manipulated part of $x$                                                           |
| $\bar{x}$                | : Untouched part of $x$                                                             |
| $E$                      | : Flipped bit for multiples of modulus                                              |
| $\lambda_d p$            | : Largest number of 1's                                                             |
| $\lambda_s p$            | : Largest number of 0's                                                             |
| $E_{min}$                | : 1's in $\lambda_s p$                                                              |
| $E_{max}$                | : 1's in $\lambda_d p$                                                              |
| $\mathcal{E}$            | : Maximum number of detection failure                                               |
| $F$                      | : Flipped bit for linear expressions of modulus                                     |
| $S_1$                    | : The set contains $x$ whose manipulated part                                       |
| $S_2$                    | : The set contains $x$ whose untouched part                                         |
| $S_3$                    | : The set contains manipulated part of $x$ for linear expressions of modulus        |
| $S_4$                    | : The set contains untouched part of $x$ for linear expressions of modulus          |
| $S_\phi$                 | : All multiples of signed digit representation of $p$ has $\pm 1$ ' in bit position |
| $S_\Theta$               | : All multiples of redundant binary representation of $p$ has 1' in bit position    |
| $\phi_d p$               | : Largest number of 1's (Redundant binary representation)                           |
| $\phi_s p$               | : Largest number of 0's (Signed digit representation)                               |
| $F_{min}$                | : Smallest set of $\phi_s p$                                                        |
| $F_{max}$                | : Largest set of $\phi_d p$                                                         |
| $\mathcal{F}$            | : Maximum number of detection failure for linear expression                         |

## LIST OF FIGURES

| <b><u>Figure</u></b>                                     | <b><u>Page</u></b> |
|----------------------------------------------------------|--------------------|
| 4-1 A Simple Error Detection Architecture                | 16                 |
| 4-2 An Error Detection Architecture with Input Parities  | 17                 |
| 4-3 Detailed Architecture of the Robust Processor        | 18                 |
| 4-4 Robust Addition Unit                                 | 20                 |
| 4-5 Robust Multiplication Unit                           | 21                 |
| 4-6 Pipeline Stages of Robust Addition Instruction       | 22                 |
| 4-7 Pipeline Stages of Robust Multiplication Instruction | 23                 |
| 4-8 Robust Instruction Behavior at CPU Pipeline          | 23                 |
| 4-9 Robust Addition Unit with no Parity Input            | 25                 |
| 4-10 Robust Multiplication Unit with no Parity Input     | 26                 |
| 4-11 Robust Operations for Configuration-3               | 27                 |
| 5-1 Estimated number of detection failures for large $p$ | 68                 |
| 5-2 Estimated number of detection failures for small $p$ | 68                 |

## LIST OF TABLES

| <b><u>Table</u></b>                                                                                                                          | <b><u>Page</u></b> |
|----------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| 4-1 Clock Cycle Comparison for Montgomery Implementation                                                                                     | 31                 |
| 4-2 Speed Area Information for ASIC Implementation                                                                                           | 32                 |
| 4-3 Speed Gate Information for FPGA Implementation with Time<br>Constraint @33.33 MHz                                                        | 32                 |
| 4-4 Speed Gate Information for FPGA Implementation with No Time<br>Constraint                                                                | 33                 |
| 5-1 Multiples of the modulus $p=19$ whose nonzero bit positions $i \in E = \{0, 2, 4, 6, 7, 11\}$ .                                          | 46                 |
| 5-2 Multiples of the modulus $p=19$ whose nonzero bit positions $i \notin E = \{0, 2, 4, 6, 7, 11\}$ .                                       | 47                 |
| 5-3 The values of the detection failure count, $SA$ and $SB$ when $p=131$ ,<br>$k=12$ , and the faults are the multiples of $p$ in $(0, 2k)$ | 52                 |
| 5-4 The values of the detection failure count, $SA$ and $SB$ when $p=331$ ,<br>$k=12$ , and the faults are the multiples of $p$ in $(0, 2k)$ | 53                 |
| 5-5 Linear expressions $\alpha p + \xi = \alpha 47 + 5$ whose nonzero bit positions<br>$i \in F = \{0, 2, 6, 7, 8, 11\}$ .                   | 58                 |
| 5-6 Linear expressions $\beta p - \xi = \beta 47 - 5$ whose nonzero bit positions<br>$i \notin F = \{0, 2, 6, 7, 8, 11\}$ .                  | 58                 |
| 5-7 Suggested prime numbers                                                                                                                  | 75                 |

# 1 INTRODUCTION

Implementation (both software and hardware) of cryptographic algorithms, which are considered to be secure in theory, can be vulnerable to side-channel and fault attacks in practice. Alternatively referred as the Bellcore attacks, the latter category of attacks are simply based on flipping certain bits during the cryptographic computation in order to generate incorrect results. In a multitude of previous works, [Biham and Shamir, 1997], [Boneh et al., 2001], [Aumüller et al., 2002], [Quisquater and Piret, 2003] just to name a few, it was demonstrated that the incorrect results, when produced deliberately, can be utilized to extract secret keys (or parts of them) used in the cryptographic computations. Recent attacks shown in [Berzati et al., 2010] and [Li et al., 2010] hardly overemphasize the difficulty of protecting the cryptographic device against the fault attacks even in the presence of countermeasures considered to be effective by many.

Assuming a direct physical access to the cryptographic device, an adversary can easily inject faults in the device by manipulating its working conditions such as supply voltage, working temperature, clock rate, or exposing it to some radiation source (white light, laser, X-ray, ion beam, etc.) [Skorobogatov and Anderson, 2003], [Hamid et al., 2006]. A recent work [Trichina and Korkikyan, 2010] achieved to show two faults in different locations of an advanced processor with high precision utilizing laser beams. The latter work emphasizes the need for new adversarial models to capture the increasing capabilities of attackers equipped with powerful devices. Similar findings and claims in other recent publications [Bekkers and König, 2011], [Balasch et al., 2011] support the need for the consideration of a powerful attacker in the analysis of cryptographic implementations.

Fault detection is a widely accepted countermeasure, whereby a cryptographic device is able to detect a faulty computation or result and to take an action preventing the eventual compromise of the secret key such as raising an alarm, immediately stopping the computation, resetting the device, erasing the secret key etc. Error detection codes designed to detect random errors due to mother nature can be utilized as suggested by a multitude of works [Bertoni et al., 2003], [Karri et al., 2002], [Karri et al., 2003], [Reyhani-Masoleh and Hasan, 2002], [Reyhani-Masoleh and

Hasan, 2004], [Gaubatz and Sunar, 2006], [Wang et al., 2012], [Bertoni et al., 2003], [Mathew et al., 2010]. However, one should not forget that the type of fault introduced by an intelligent attacker is not of random nature. The attacker, knowing the algorithmic and implementation details of the cryptographic device, can mount devastating attacks via faults undetectable by conventional codes. For instance, since identical error patterns in data and parity words can lead to undetectable faults in some error detection schemes there is no need for a priori knowledge of the data word.

Systematic non-linear error detecting codes proposed by Karpovsky and Taubin [Karpovsky and Taubin, 2004] can protect the circuit against a much more capable adversary since the success rate of introducing an undetectable fault by these codes is extremely low if no a priori knowledge on data is available. Furthermore, it is also possible to minimize the maxima of the fraction of undetectable faults over all codes (cf. the definition of *robust codes* in [Konrad J. Kulikowski, 2005]) when certain parameters in the code are chosen carefully. For cryptographic use of the systematic non-linear error detecting codes one can profitably refer to [Karpovsky et al., 2004] and [Konrad J. Kulikowski, 2005].

Quadratic (residue) codes proposed in [Gaubatz et al., 2006] for the protection of fundamental arithmetic operations (e.g., addition and multiplication), which are heavily used in public key algorithms, are preferable also from implementation point of view since they require only squaring operation modulo a prime integer. The integration of quadratic codes into the data path of a simple embedded processor in [Yumbul et al., 2010] demonstrates that they are effective in detecting faults as well as being lightweight. A recent work [Yumbul et al., 2011] points out a potential vulnerability of quadratic codes against a powerful adversary if the modulus is not chosen carefully.

In this thesis, we design a variety of execution units for a configurable processor to perform integer addition and multiplication operations, which are secured against the fault analysis. While these execution units perform the integer operations, they also calculate parity information from the input operands and the results, using the non-linear error detecting code given in [Gaubatz et al., 2006]. The execution units (i.e., multiplier and adder) are hence referred as robust arithmetic

units. Also, they issue error signals, if the redundancy checks fail. In other words, any fault (i.e., an unintended bit flipping in a register or an execution unit) can be detected with an overwhelming probability.

We choose a reconfigurable embedded processor core, Xtensa LX3 [Halfhill, 2011] by Tensilica, as the underlying platform to integrate the robust execution units. The robust execution units are developed using Tensilica Instruction Extension Language (TIE), which is a Verilog-like language. Then, using the Tensilica tool chain, we obtain the RTL description of the core processor with the robust execution units being added to its data path. The resulting RTL code can be mapped to ASIC (Application-Specific Integrated Circuit) or FPGA (Field Programmable Gate Array) implementations easily.

The instruction set architecture of Xtensa LX3 is extended with two new instructions that utilize robust addition and multiplication units. Therefore, software implementations of many public key cryptography algorithm are protected against fault attacks to a great extent since overwhelming majority of their execution times are usually dominated by these two operations (e.g. RSA and elliptic curve cryptography). Moreover, the processor core is also enhanced by a new register file to store parity values of the operands to protect the computation against fault attacks while the operands are not active in execution units, but waiting in the register file.

Using synthesis tools, we obtain the space and time complexities for the ASIC and the FPGA implementations of the enhanced processor core with error detection and compare them to those of the base LX3 core with no error detection capability. The results demonstrate that it is feasible to add the robust units even to a simple embedded processor if low to moderate increases in time and area are tolerable.

Also, using robust addition and multiplication instructions, a C implementation of the Montgomery modular multiplication algorithm, commonly used in cryptographic applications [Koç et al., 1996], is developed. New instructions improve the performance of a modular multiplication operation in terms of clock cycle count compared to the base Xtensa LX3 processor core. The base processor is a simple core which has no direct access to the carry bits generated in multi-precision integer addition as well as higher part of the result of a 32-bit multiplication operation. Direct access to them is needed for parity calculations in robust

instructions, which also results in considerable improvements in term of clock cycles in multi-precision arithmetic operations used in cryptographic algorithms as seen in Chapter 4.

In addition to implementation of quadratic residue codes, perceiving the need for a deeper study to gain more insight to the cryptographic use of quadratic codes, we provide an extensive and in-depth study to determine the rules for selecting the modulus to minimize the success rate of an attack by a powerful adversary. It turns out that many primes prove to be insecure and that selecting a secure prime is not a straightforward task in the assumed adversarial model.

Our first theoretical analysis demonstrates that the success rate depends on the binary representations of the modulus multiples and its linear expressions of the form  $\lambda p + \mu$  that can fit into a data word. In fact, we are able to provide a formula to determine the exact number of undetected faults which is a function of the number of nonzero bits in the binary representation of the modulus. Our second analysis shows that the distribution of 1's and 0's in the binary representation also plays an important role in the security of the quadratic codes against fault attacks. Several examples of weak moduli allowing an increased chance of introducing undetectable faults are included. Finally, we provide countermeasures and methods of choosing a *good* modulus that will show better resilience against attacks.

This thesis is organized as follows. Chapter 2 introduces cryptographic background and fault injection attack fundamentals. Chapter 3 discusses the nonlinear error detection codes (robust codes). Chapter 4 describes the general architecture for the hardware extensions as well as presents various designs for the robust addition and multiplication execution units and several configurations of the Xtensa LX3 processor which provide robust arithmetic instructions for software implementations. Chapter 5 provides an introduction of two types of attacks which can be successful in the adopted adversarial model if the modulus is sparse or dense in terms of the number of nonzero bits in its various representations. Also in Chapter 5, we provide a detailed analysis of the first type of attack, which is based on adding a multiple of the modulus to the data word and second type of attack, which utilizes a linear expression of the modulus that may result in undetectable faults. Chapter 6 concludes the thesis by summarizing the achievements.

## **2 CRYPTOGRAPHIC BACKGROUND AND FAULT ATTACKS**

In this chapter, we provide a brief introduction to cryptographic algorithms and some practical attacks developed against their implementations.

### **2.1 Basic Cryptographic Background**

Nowadays, many electronic devices such as PDAs, personal computers, smart phones, smart cards utilize cryptographic algorithms to protect their secret information. Many of those systems implement the cryptographic applications in software. Crypto systems are usually classified in two groups: symmetric key algorithm and public key algorithm. The brief descriptions of these algorithms are given in the following sections.

#### **2.1.1 Symmetric Key Algorithms**

In this type of cryptographic algorithm, both sender and receiver generally use the same key for encryption and decryption. Therefore, they must agree on a secret key before communicating with each other. If an attacker obtains this secret key, he can encrypt and decrypt all exchanged messages. If the secret key is kept secret, the communication between sender and receiver is secured against attacks provided that there is no weakness in the cryptographic algorithm and its implementation.

Symmetric algorithms can be divided into two categories. In the first category consist of stream ciphers which operate on a plain text a single bit at a time. The other one is block ciphers which operate on a plain text in groups of bits. Examples of block ciphers include DES, 3DES, AES and so on, while RC4 and Trivium are examples of stream ciphers.

#### **2.1.2 Public Key Algorithms**

In this type of algorithm, the key used for encryption (i.e., public key) is different from the key used for decryption (i.e., private key). Additionally, the decryption key cannot be calculated from the encryption key. While the encryption operation is public, only the person with the corresponding private key can decrypt

the message. The most popular public key cryptographic algorithms are RSA, which is based on the difficulty of integer factorizations of large integers and, the Diffie-Hellman key exchange algorithm based on the difficulty of discrete logarithm in large groups. These algorithms are used generally for generating and verifying digital signature, verifying certificates and exchanging keys for symmetric key cryptography.

## 2.2 Side-Channel and Fault Attacks

In side-channel attacks, the adversary tries to get secret information utilizing potential weakness in the physical implementation of a cryptosystems. Side channel attacks could be classified into two groups: Invasive and non-invasive attacks. In invasive attack, the adversary accesses the physical layer and tries to manipulate, observe and understand the system internals. Reverse engineering and optical attacks are the examples for invasive attacks. In non-invasive attacks, the adversary does not need the access the physical layer. The attacker investigates the device just by probing and manipulating. There are many forms of the non-invasive attack such as timing attacks, power attacks, and electromagnetic attacks and so on.

Fault attack is a generic name for a class of active attacks, whereby an adversary extracts or infers the secret key by forcing a cryptographic device to perform faulty computation. Assuming that the adversary has physical access to the cryptographic device, a fault can easily be injected by manipulating its working conditions such as supply voltage, working temperature, clock rate, or exposing it to some radiation source (white light, laser, X-ray, ion beam, etc.) [Skorobogatov and Anderson, 2003], [Hamid et al., 2006]. There is a huge body of literature on fault attacks for both secret and public key systems such as [Biham and Shamir, 1997], [Boneh et al., 2001], [Aumüller et al., 2002], [Quisquater and Piret, 2003], to name a few.

More detailed information for non-invasive attacks (power attack and timing attack) and invasive (optical attack) is given in the following sections.

### 2.2.1 Timing Attack

Timing analysis attack was firstly introduced by Paul Kocher in 1996 [Kocher, 1996]. He demonstrated that one can infer the secret key by analyzing small variations in the execution time of a cryptographic algorithm. The essential idea of this type of attack is to make predictions about the secret key, then using the statistical techniques to test the prediction and observe the device reaction whether the prediction is correct or not.

In this type of attack, the adversary begins with a known input (possibly a known input and considering the cryptographic device as a black box) and measures the execution time of the cryptographic algorithm for this input. During the steps of the cryptographic algorithm one key bit is mixed with the input; and furthermore particular steps of the cryptographic algorithm take different amounts of time depending on the key bit involved. For instance, according to a particular key bit value, two operations are possible for the algorithm step, whose execution times are perceptively different.

In order to execute an attack, the adversary applies a set of input and measures the response time of the device for these input sequences. The adversary also forms two models of cryptographic computation for each value of particular key bit whose value results in a perceptible time difference. Then, the adversary compares the measured times and estimated times based on these models and try to give a decision whether the correct bit is one or zero.

One of the techniques to prevent this type of attack is to enforce all operations to take exactly the same amount of time. But most of the time it is not possible to make all computations take exactly the same amount of time due to implementation difficulties. Another technique is adding random delays [Kocher, 1996]. But the latter is not always an effective solution for preventing this type attack. It just increases the number of measurements required to break the cipher.

For preventing this type of attack, a blinding technique [Chaum, 1982] can also be used, whereby message is blinded to prevent the attacker from obtaining the aforementioned model for cryptographic computation that requires a known input. For example, message blinding can be used with RSA and Diffie-Hellman [Kocher, 1996] to prevent attackers to obtain the key with timing measurements.

### 2.2.2 Power Attack

There are two main categories of power analysis attacks: simple power analysis (SPA) and differential power analysis (DPA).

In SPA attacks, the power consumption of the device is directly measured by the attacker. The power consumption of the device varies depending on the particular operation performed. For example, at the RSA implementation, SPA can infer the key from the multiplication and squaring operations during modular exponentiation, since these operations usually require different amount of power due to the fact that multiplication is more complicated than squaring. For DES implementations, the attacker easily infers the key with the SPA by means of the differences between permutation and shift operation [Kocher et al., 1999].

DPA attack is the more complex form of the SPA attack, whereby we use statistical techniques and the key can be figured out from noisy power consumption measurements. This type of attack is used to break both symmetric (DES) and public key (RSA) algorithms [Kocher et al., 1999]. For example, in RSA algorithm, the attacker can infer the key by means of the comparison between candidate and power consumption measurement values. For detailed information about the DPA, see [Kocher et al., 1999].

### 2.2.3 Optical Attack

Standard CMOS circuitry is extremely vulnerable to attack using optical probing. This vulnerability may pose a big problem for the industry, similar to those resulting from probing attacks in the mid-1990s and power analysis attacks in the late 1990s. Unless suitable countermeasures are taken, optical probing might also be used to induce errors in cryptographic computations or protocols, and to disrupt the processor's control flow.

Skorobogatov et al. [Skorobogatov and Anderson, 2003] developed a new class of attacks on secure microcontrollers and smartcards systems. They illuminated on the target transistor by using a simple flashgun and a laser pointer, which induced a transient fault. As an illustration of the power of this attack, they developed techniques to set or reset any individual bit of SRAM in a microcontroller. They described also how the illumination of a certain area of an SRAM cell can be used to set it to either 0 or 1. Other memory technologies, such as EPROM, EEPROM and

Flash, can also be manipulated in this way. They proposed that this simple and cheap optical technique gives rise to many effects used by an attacker.

There are many induction techniques other than the optical attack. One of them is the voltage attack, which is based on the forced changes in the supply voltage of the cryptographic device, such as causing a spike in the supply voltage, forcing the circuit to work with higher or lower voltage levels than its normal operational value. Second of them is clock attack, where adversary tries to change the clock speed. For instance, slowing down the clock speed may be used to observe internal state of the cryptographic devices. Third of them is temperature attack. If the attacker operates the cryptographic device in extreme temperature conditions, he can induce the fault. But this technique is not used in contemporary attacks. The fourth one is the radiation attack. The attack utilizes microwaves to induce fault instead of light. The last one is the eddy current attack. This type of attack is firstly introduced by Quisquater and Samyde [Quisquater and Samyde, 2002]. The main idea is that cryptographic device working conditions can be perturbed by the electromagnetic field produced by alternating current. With this type of attack we can change the content of the RAM, EPROM, EEPROM or Flash.

Differential fault analysis (DFA) is another popular attack technique, proposed in [Biham and Shamir, 1997] and [Quisquater and Piret, 2003]. This type of attack can be used to break both symmetric and public key cryptosystems. The basic idea is based on two observations made with the same plaintext message. The first observation is made for the plaintext when no fault induced on the cryptographic device. The second observation is made when the operation is repeated with the same plaintext, but this time computation is corrupted with an induced fault. Then, the adversary compares the two results and infers the secret key. DES was shown to be broken with DFA [Biham and Shamir, 1997]. This type of attack was also used to break ECC (Elliptic Curve Cryptosystems) [Biehl et al., 2000] and RSA [Berzati et al., 2012].

In this thesis, we are interested in countermeasures based on error detecting codes to protect cryptographic computations against fault attacks. The particular method we use is robust arithmetic codes which were previously proposed as an affective countermeasure against fault attacks.

### 3 ROBUST ARITHMETIC CODES

#### 3.1 Quadratic Residue Codes

The residue codes use a prime integer  $p$  to compute the parity as follows,

$$C = \{(x, w) | x \in Z_{2^k}, w = x^2 \bmod p \in F_p\} \quad (3-1)$$

Where  $x$  is the  $k$ -bit data part and  $w$  is the  $r$ -bit redundancy part of the code, respectively, where  $r = \lceil \log_2(p) \rceil$ . Let  $e_x$  denote the error in the data part  $x$  and  $e_w$  denote the error in the parity part  $w$ . Then, an error  $(e_x, e_w)$  in the code cannot be detected, if

$$\begin{aligned} (x + e_x \bmod 2^k)^2 \bmod p &= w + e_w \bmod 2^r \\ &= (x)^2 \bmod p + e_w \bmod 2^r \end{aligned} \quad (3-2)$$

Assume that an attacker has no a priori information for the data  $x$  and its computed redundancy  $w$ . If the attacker generates the error  $e = (e_x, e_w)$ , the probability that this error remains undetected (error masking probability) can be given by [Gaubatz et al., 2006].

$$Q(e) = Q(e_x, e_w) = \frac{|\{x | (x + e_x, w + e_w) \in C\}|}{|C|} \quad (3-3)$$

If any given code  $C$  minimizes  $Q(e)$  value for all non-zero values of  $x$ , then this code is called robust code. Overall robustness is achieved for  $\max_{e \neq 0} (Q(e)) = 2^{-r}$ . If an upper limit as  $\max_{e \neq 0} (Q(e)) \leq \varepsilon 2^{-r}$  for a code  $C$  is given, then this code called  $\varepsilon$ -robust.  $\varepsilon$  is a constant number much smaller than  $2^r$ . In [Gaubatz et al., 2006], it is shown that

$$\varepsilon = \max(4, 2^k - p + 1) \quad (3-4)$$

where  $k = r$ . For example, we can select  $k = r = 32$ , assuming the word length in many general-purposes microprocessor is 32. If the closest prime  $2^{32}-5$  is selected for parity calculation, the error masking probability becomes,

$$Q(e) = (2^k - p + 1).2^{-k} = 3.2^{-31}.$$

The basically means that the probability of forming an undetected error through random trials by an attacker is approximately one in a billion if the data word  $x$  is unknown. Selecting a prime in a special form as  $p = 2^{32} - 5$  is inexpensive. On the other hand selecting a random prime, e.g. a  $q=0x\text{FB01CDD9}$ , does not have this advantage as its error masking probability becomes significantly large

$$Q(e) = (2^k - p + 1).2^{-k} = 0.0195.$$

While selecting a prime integer is a special form and close to word boundary has many-fold advantages, this method may become vulnerable to simple attacks under certain assumptions on the capabilities of the adversary as will be shown in Chapter 5 .

## 3.2 Dual Residue Codes and Their Performance

Basic idea about dual residue codes is to use two moduli,  $p$  and  $q$ , to obtain parity  $w = w_p \parallel w_q$ , where  $w_p = f_p(x) \bmod p$  and  $w_q = f_q(x) \bmod q$ . The two functions  $f_p(x)$  and  $f_q(x)$  represent simple arithmetic operations, linear or nonlinear. We investigate residue code with identity function first.

### 3.2.1 Dual Residue Codes with Identity Function

The first option is to use the identity operation for both functions. In other words, the parity for data word  $x$  is computed as

$$w = x \bmod p \parallel x \bmod q$$

where  $p$  and  $q$  are two prime integers such that  $q < p < p \times q \approx 2^k$ . This sounds reasonable since  $w_p$  and  $w_q$  are combined to give a unique data word  $x$  using Chinese Remainder Theorem (CRT) in mod  $(p \times q)$ . Therefore, it seems to provide a good protection for data words in the range  $[q, (p \times q)]$  against randomly introduced

errors. On the other hand, for smaller data words  $x < 2^{13}$ , flipping the bits in the same position of  $x$ ,  $w_p$ , and  $w_q$  leads to undetectable errors.

However, a careful analysis shows that the scheme does not protect larger data words and the attack method is capable of introducing undetectable errors with very high probability. Assume the data word  $(p \times q) > x > p > q$  and consider the case where  $x$  is even, i.e.  $x_0 = 0$ . If we attack on  $x_0$  and flip it, we get a corrupt data word  $\tilde{x} = x + 1$ . Then, for parity words we have  $\widetilde{w_p} \equiv w_p + 1 \pmod{p}$  and  $\widetilde{w_q} \equiv w_q + 1 \pmod{q}$ . If we further assume  $w_{p,0} = 0$  and  $w_{q,0} = 0$ , flipping these least significant bits of parity words will produce parity values for the corrupt data word  $\tilde{x}$  if  $w_p, w_q < p - 1$ . This case happens roughly with probability of  $1/8$ .

Consider now the case where  $x$  is an odd integer; i.e.  $x_0 = 1$ . If we flip  $x_0$ , we obtain corrupted data  $\tilde{x} = x - 1$  and naturally  $\widetilde{w_p} \equiv w_p - 1 \pmod{p}$  and  $\widetilde{w_q} \equiv w_q - 1 \pmod{q}$ . If we further assume  $w_{p,0} = 1$  and  $w_{q,0} = 1$  (i.e., both  $w_p$  and  $w_q$  are odd), flipping these bits will produce parity values  $\widetilde{w_p}$  and  $\widetilde{w_q}$  for  $\tilde{x}$  provided that  $w_p, w_q > 0$ . This occurs again with the probability of  $1/8$ . In summary the attack simply flips the least significant bits of  $x$ ,  $w_p$  and  $w_q$  and this generates undetectable errors roughly with total probability of  $1/4$  or 25% for  $x > 2^{13}$ . Therefore, we need different types of functions in our residue codes.

### 3.2.2 Dual Residue Codes with Hybrid Function

Since both single quadratic residue and dual residue codes with identity functions are weak, a better approach is to use multiple moduli, where one parity is found by a quadratic function while the other is by linear. The code can be formulated as follows

$$w_p \equiv x \pmod{p} \text{ and } w_q \equiv x^2 \pmod{q}$$

where  $w = w_p \parallel w_q$  is the parity for the data word  $x$ . For simplicity we assume  $0 < x < n$ , where  $n = p \times q$ .

We can negate the data word  $x$  and this will not change the parity  $w_q$  since  $w_q \equiv (x^2) \pmod{q} \equiv ((-x)^2) \pmod{q}$ . All we have to do is to negate  $w_p$  to find a matching parity for  $(-x) \pmod{n}$ . Negating  $w_p$  can be performed in a simple way just

flipping certain bits if the modulus  $p$  is chosen in a special form, which is usually the case for efficiency reasons. As an example for 32-bit data words, assume that we pick up  $p = 2^{19} - 1$  and  $q = 2^{13} - 1$ . By noting that  $p$  is an all-one binary representation

$$p = 2^{19} - 1 = 1111111111111111111$$

All we have to do is to flip all the bits of  $w_p$  to obtain  $(-w_p) = -x \bmod p$ . If we are able to flip all the bits of  $w_p$ , this give success probability of 100%. However, it may not be easy to obtain the negative of the data word of  $x$  modulo  $n$  by flipping bits. Note that in binary representation of  $n$

$$n = (2^{19} - 1) \times (2^{13} - 1) = 1111\ 1111\ 1111\ 0111\ 1110\ 0000\ 0000\ 0001.$$

There are 13 zero bits. Attack on a single modulus quadratic residue code will be shown in Chapter 5; we only flip the bits of  $x$  corresponding to non-zero bits of the modulus  $n$  and do not touch the other bits of  $x$ . If the other bits of  $x$  corresponding to zero bits of the modulus  $n$  are also 0, the attack succeeds introducing undetectable errors. Since there are 13 zero bits in  $n$ , the success probability of the attack is  $2^{-13}$ . Considering the difficulty of the flipping a number of bits in different locations, this succeed probability is considerably low, rendering the attack useless for many practical purposes. However, there is a more powerful attack that will weaken this code considerably.

Easiest approach is to add (or subtract)  $q$  to both the data word  $x$  and the first parity  $w_p$ , i.e.  $\tilde{x} = x \pm q$  and  $\tilde{w}_p = w_p \pm q$  since

$$\tilde{w}_q \equiv (x \pm q)^2 \bmod q \equiv x^2 \bmod q \equiv w_q$$

$$\tilde{w}_p \equiv x \pm q \bmod p \equiv w_p \pm q \bmod p$$

Now, the question is whether these operations can be performed easily by bit flipping. A probabilistic method that flips the data bits  $x_{13}$  and  $x_0$  is successful in adding  $2^{13} - 1$  to  $x$  only when  $x_{13} = 0$  and  $x_0 = 1$ . Similarly, flipping the bits  $w_{p,13} = 0$  and  $w_{p,0} = 1$  adds  $2^{13} - 1$  to  $w_p$ . Therefore, success probability of this attack is 1/16. If  $x_{13} = w_{p,13} = 1$  and  $x_0 = w_{p,0} = 0$ , flipping those bits of  $x$  and  $w_p$  subtracts  $2^{13} - 1$  from them. Then no error can be detected with probability 1/16.

Consequently, the overall success of the attack can be put roughly at 1/8, which is quite high for a proper protection; therefore we need a better approach.

### 3.2.3 Quadratic Dual Residue Codes

The quadratic dual residue codes use the squaring operation for both part of the parity as follows

$$w_p \equiv x^2 \pmod{p} \text{ and } w_q \equiv x^2 \pmod{q}.$$

Consider now the square roots of  $w_p$  and  $w_q$ .

$$(\pm s_p)^2 \equiv w_p \pmod{p} \rightarrow x \equiv s_p \pmod{p}$$

$$(\pm s_q)^2 \equiv w_q \pmod{q} \rightarrow x \equiv s_q \pmod{q}$$

The CRT gives the following equations

$$\begin{aligned} x_1 &\equiv (s_p \cdot N_p \cdot M_p + s_q \cdot N_q \cdot M_q) \pmod{n} \\ x_2 &\equiv (-s_p \cdot N_p \cdot M_p + s_q \cdot N_q \cdot M_q) \pmod{n} \\ x_3 &\equiv (s_p \cdot N_p \cdot M_p - s_q \cdot N_q \cdot M_q) \pmod{n} \\ x_4 &\equiv (-s_p \cdot N_p \cdot M_p - s_q \cdot N_q \cdot M_q) \pmod{n} \end{aligned} \tag{3-5}$$

where  $N_p = q$ ,  $N_q = p$ ,  $M_p = q^{-1} \pmod{p}$ , and  $M_q = p^{-1} \pmod{q}$ . The CRT simply implies that there are four data words,  $x_1, x_2, x_3, x_4 < p \times q$  with the same parity. In addition, we have the following relationships

$$x = x_1 \equiv -x_4 \pmod{n} \tag{3-6}$$

$$x = x_1 \equiv x_2 \pmod{q} \tag{3-7}$$

$$x = x_1 \equiv -x_2 \pmod{p}. \tag{3-8}$$

Equation (3-6) simply implies an attack based on negating the data word  $x$ . The success of this attack depends on the choice of the prime. For  $p = 2^{19} - 1$  and  $q = 2^{13} - 1$ , our probabilistic bit flipping attack induces undetectable errors with a probability of  $2^{-13}$  due to the fact that there are 13 zeros in  $n = p \times q$ .

Equation (3-7) implies a much harder attack that attempts to turn the word  $x$  to  $x_2$  by adding/subtracting an unknown multiple of  $q$  to  $x$  since  $x_2 = x \pm t \times q$ . The knowledge of  $x$  is required to find the correct multiple  $t$ . Furthermore, adding  $t \times p$  to  $x$  may not be easy. Therefore, this attack is not feasible in our adversary model. Equation (3-8) implies another way of obtaining  $x_2$  from  $x$  (e.g. subtracting  $x$  from some unknown multiple of the prime  $p$ ), which seems to be as difficult. We have a similar relationship between  $x$  and  $x_3$  suggesting a similar probabilistic bit flipping attacks with the same success rate.

Naturally, if adding any multiple of  $n = p \times q$  to a data word  $x$  results in a number smaller than  $2^k$ , parity does not change and we can induce an undetectable error. However, adding  $n$  or its multiple may not be easy. In our example above,  $n = (2^{19} - 1)(2^{13} - 1)$  has 13 zeros in its binary representation, therefore our probabilistic bit flipping attack succeeds with the probability of  $2^{-13}$ . Choosing 31-bit data words will eliminate this attack.

## 4 ROBUST EMBEDDED PROCESSOR IMPLEMENTATION

### 4.1 Error Detection Strategy for an Embedded Processor

Public key cryptography algorithms, in general, rely mainly on simple arithmetic operations, such as addition and multiplication in different mathematical structures (e.g. ring, finite fields, etc.) In software implementations on general-purpose processors, these arithmetic operations are realized on functional units that are capable of single precision arithmetic; e.g. 32-bit. In, for instance, a typical software implementation of RSA or elliptic curve cryptography (ECC), majority of execution time (+95% for RSA, +85% for ECC), is spent in single-precision multiplication operations. Addition and other operation account for the remaining time. Therefore, we elect to re-design word-based multiplication and addition unit with error detection capability; an approach similar to the proposal [Gaubatz et al., 2006].

General architecture of a functional unit with error detection capability can be depicted as in Figure 4-1. As can be observed from the figure, the parity for the result  $c$  computed twice in a sense; one from the result itself using parity generation circuit and the other from the input variables via parity prediction circuit. This architecture protects the operands against fault attacks during computation.

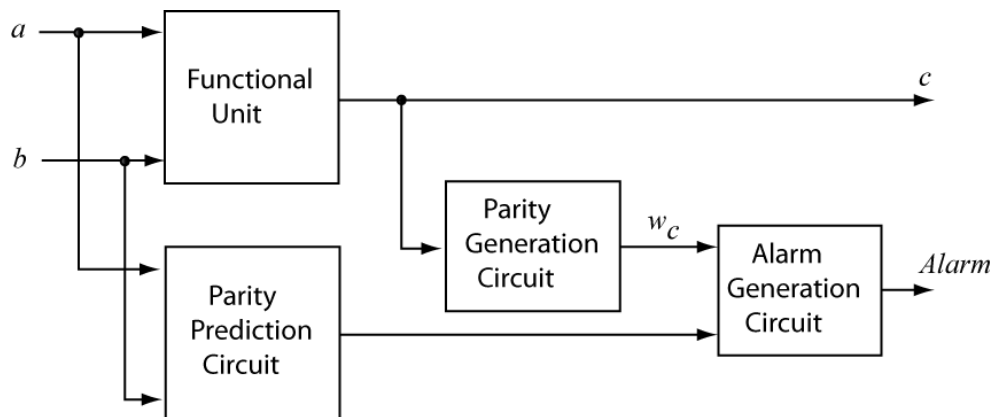


Figure 4-1 A Simple Error Detection Architecture

On the other hand, in software implementation only a small part of the operands are actively processed by functional units during cryptographic computation at a time and the other operands and variables involved may spend considerable amount of time waiting in general-purpose register and/or in memory (cache and/or main). It is imperative to protect variables during this time period as well. To this end, we propose a functional unit which is robust in the sense that is protected against fault attack in Figure 4-2. In the robust functional unit both the operands and their parities are the inputs. The robust unit has the three outputs; the result of the operation, its parity for the subsequent operations and an alarm code indicating detection of an error. Therefore, instructions that use the robust functional units read their input operands and their parities from a register file. The alarm can be used in different ways, such as throwing an exception and stopping the entire computation, depending on the implementation details, which we will not investigate here.

In the chapter 4.2, we will explain how the proposed robust functional units are integrated to a general-purpose embedded processor core.

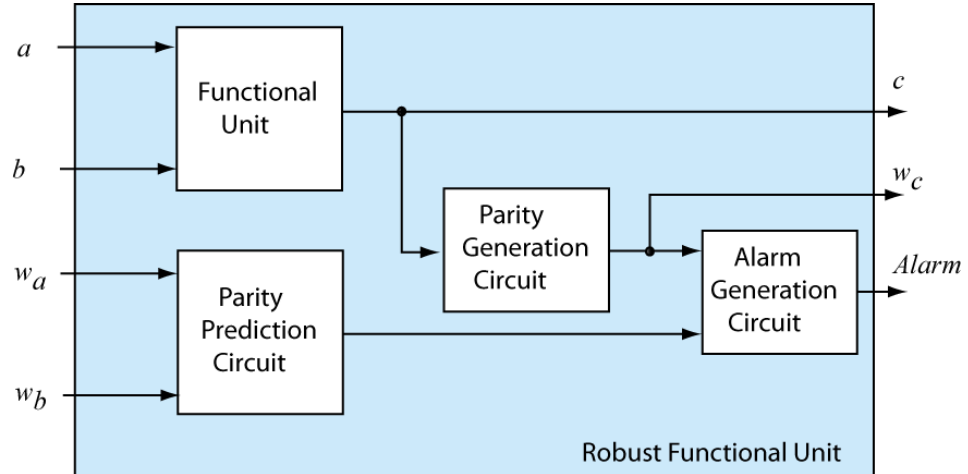


Figure 4-2 An Error Detection Architecture with Input Parities

## 4.2 General Architecture

We select the reconfigurable Xtensa LX3 processor core by Tensilica [Halfhill, 2011] to obtain an embedded general purpose microprocessor where cryptographic operations are protected against fault attacks. Xtensa LX3 core is a simple RISC (Reduced Instruction Set Computing) processor where new functional units can be added to its pipelined data path using the TIE language developed for RTL designs.

We extended the configurable Xtensa LX3 core with two robust functional units; namely robust adder and robust multiplier. We also added a small register file to keep parity values of the operands in cryptographic computations. The detailed data path of the robust processor is shown in Figure 4-3. The data path and the interconnection structure of robust instruction execution are the same as those of the existing integer data path of any RISC processor. Therefore, the new instructions are executed in the same manner as any other native RISC instruction. The only difference is that the two functional units accept input parity values besides input operands. Similarly, they output the result of the operation and its associated parity value. The parity values for the input operands and output are kept in so-called Parity Register File that consists of sixteen 32-bit registers, and is very similar to the Base Register File that is one of the fundamental parts of a general-purpose processor core. The new register file is needed mainly for performance reasons.

In the following subchapters, we provide the microarchitecture level details of robust adder and multiplication units that implement quadratic dual residue codes.

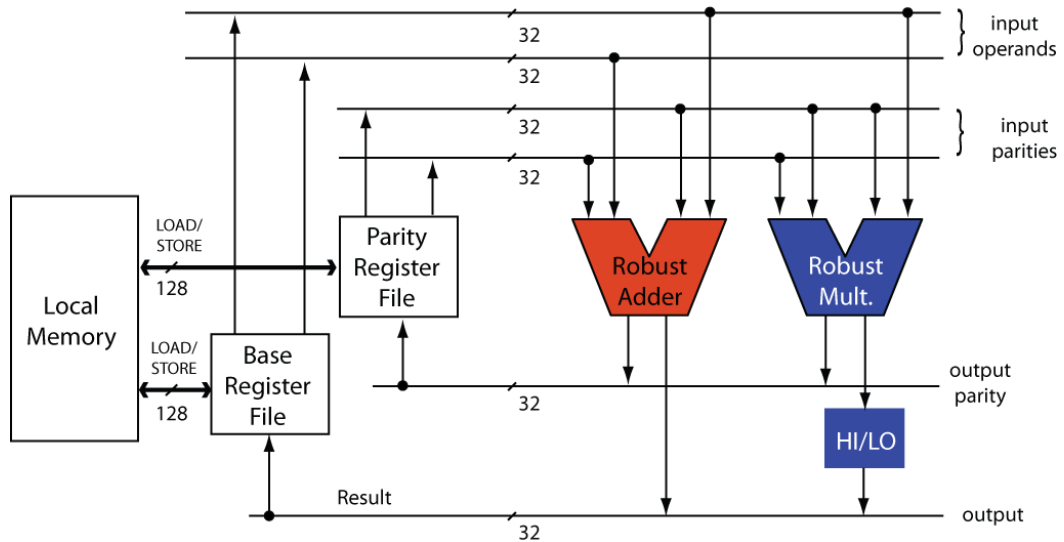


**Figure 4-3 Detailed Architecture of the Robust Processor**

### 4.2.1 Robust Adder

We add the RADDCC (robust addition with carry) instruction to Xtensa LX3 processor to perform each addition operation using robust adder unit. The instruction executes in a similar way to conventional integer addition with carry instruction which is found in many processor cores.

The operands of the RADD instruction are  $a$ ,  $b$ , and  $c_{in}$  and the parity values for input operands  $w_a$  and  $w_b$ . Its outputs are the sum of  $a$  and  $b$ , namely  $c_L$  and the carry  $c_H$ , and the parity of the sum,  $w_C$ . Each parity value has in fact two parts, one for each modulus,  $p$  and  $q$ . Henceforth, we will only deal with the part pertaining to the first modulus  $p$  for the clarity of the discussion. Note, however, everything performed here for the modulus  $p$  needs to be replicated for the other modulus  $q$ . The RADD instruction, as shown in the block diagram of the robust adder unit in Figure 4-4, computes two parity values: the parity of the result,  $w_C = |c^2|_p$  directly from the result and the other from the inputs through parity prediction circuit,  $w_C^* = |c^2|_p^*$ . Note that the new notation  $|x|_p$  denotes residue value of  $x$  with respect to the modulus  $p$ . If there is no error, these two parity values must be equal to each other.  $w_C^*$  can be calculated as follows.

$$w_C^* = |(a + b + c_{in})^2|_p$$

$$w_C^* = ||a^2|_p + |b^2|_p + 2(ab + c_{in}(a + b)) + c_{in}|_p \quad (4-1)$$

And,  $w_C$  can be calculated as follows,

$$w_C = |(c_H \cdot 2^k + c_L)^2|_p$$

$$w_C = |c_H \cdot |2^{2k}|_p + c_H \cdot |c_L|_p \cdot |2^{k+1}|_p + |c_L^2|_p|_p$$

$$w_C = |c_H \cdot (|2^{2k}|_p + c_L \cdot |2^{k+1}|_p) + |c_L^2|_p|_p \quad (4-2)$$

As a result, we check the following equality in the hardware to ensure that the parity values are equal, i.e.  $w_C = w_C^*$

$$|c_H \cdot (|2^{2k}|_p + c_L \cdot |2^{k+1}|_p) + |c_L^2|_p|_p = ||a^2|_p + |b^2|_p + 2(a \cdot b + c_{in}(a + b)) + c_{in}|_p$$

If the above equality holds, it can be said that the obtained result does not contain any error. The block diagram of the robust addition unit in Figure 4-4 is implemented and integrated to the data path of Xtensa LX3 processor. As can we observe from the block diagram, the parity detection and generation circuit share units for efficiency. Note that alarm generation circuit is implemented as a simple

comparison unit. The outputs of the unit are  $c_L$  and its parity  $w_{c_L}$  since the carry out  $c_{out}$  is used separately by subsequent operation.

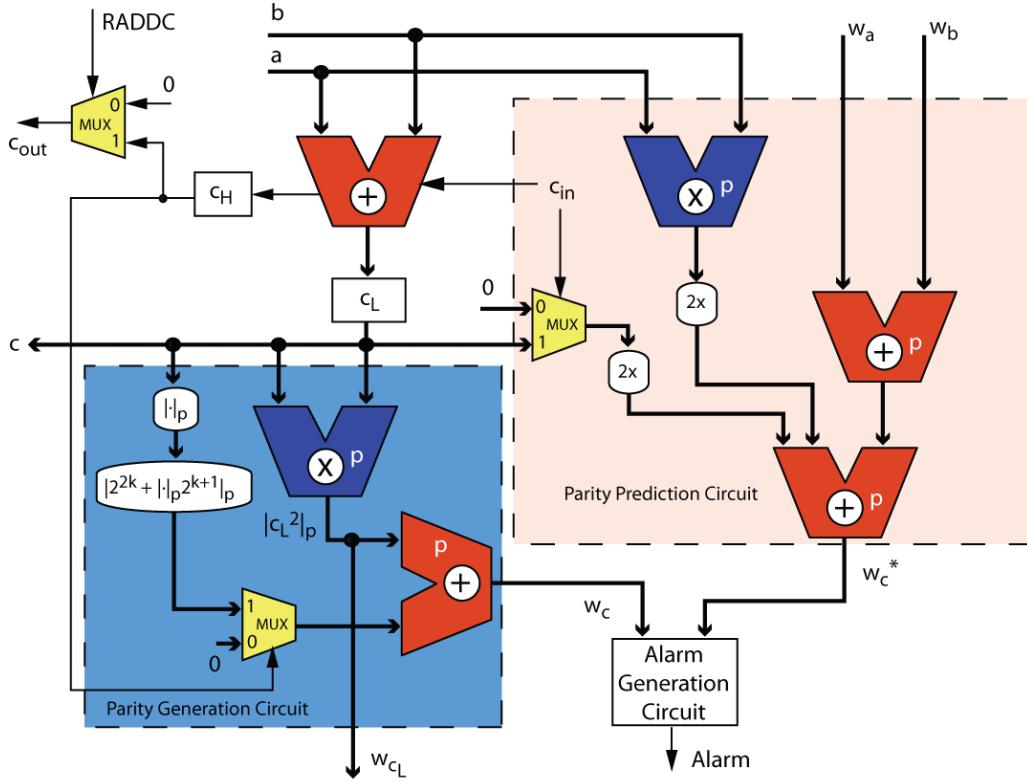


Figure 4-4 Robust Addition Unit

### 4.2.2 Robust Multiplier

The robust multiplication instruction takes two  $k$ -bit inputs,  $a$  and  $b$ , and produces the  $2k$ -bit output  $c = a.b$ . The lower and higher  $k$ -bit parts of the output are denoted by  $c_L$  and  $c_H$ , respectively. Two different parity values  $|c^2|_p$  and  $|c^2|_p^*$  are calculated as in the case of addition. While  $w_c^*$  is calculated from the input operands  $a$  and  $b$  using parity prediction circuit,  $w_c$  is calculated directly from the outputs  $c_L$  and  $c_H$ . If there is no error, these two parity values must be equal to each other. The calculations of  $w_c^*$  and  $w_c$  are performed as follows

$$w_c^* = |(a^2|_p \cdot b^2|_p)|_p$$

$$w_c = |w_a \cdot w_b|_p \quad (4-3)$$

$$w_C = |(c_H \cdot 2^k + c_L)^2|_p$$

$$w_C = |c_H^2 \cdot 2^{2k} + c_L \cdot c_H \cdot 2^{k+1} + c_L^2|_p$$

$$w_C = ||c_H^2|_p \cdot |2^{2k}|_p + |c_L|_p \cdot |c_H|_p \cdot |2^{k+1}|_p + |c_L^2|_p|_p \quad (4-4)$$

As a result, we check the following equality in the hardware to ensure that the parity values are equal, i.e.  $w_C = w_C^*$ . If the equality holds, it can be said that the obtained result does not contain any error. The block diagram of the robust multiplier circuit is shown in Figure 4-5. Note that the parities for the low and high part of the result  $c_L$  and  $c_H$  are separately stored as  $w_{c_L}$  and  $w_{c_H}$ , respectively.

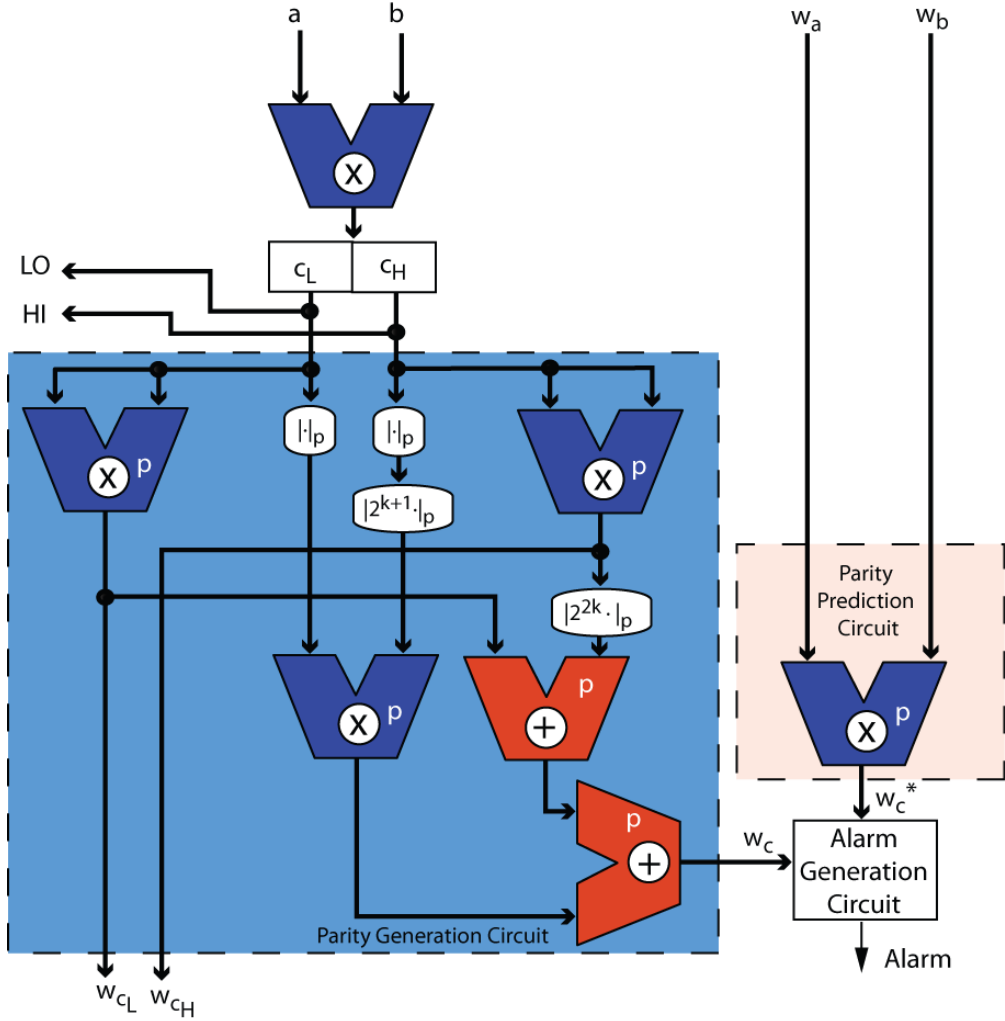


Figure 4-5 Robust Multiplication Unit

### 4.2.3 Pipeline Integration of the Proposed Robust Functional Units

As pointed out earlier, the two new instructions, robust addition and multiplication are executed in the pipeline of the embedded processor. Figure 4-6 illustrates the pipeline stages for robust addition instruction, where there are three execution stages to compute both the result  $c$  and its parity value  $w_c$ . The parity prediction circuit needs its input in the second execution cycle while the parity generation circuit produces the parity for the result at the end of the third execution stage.

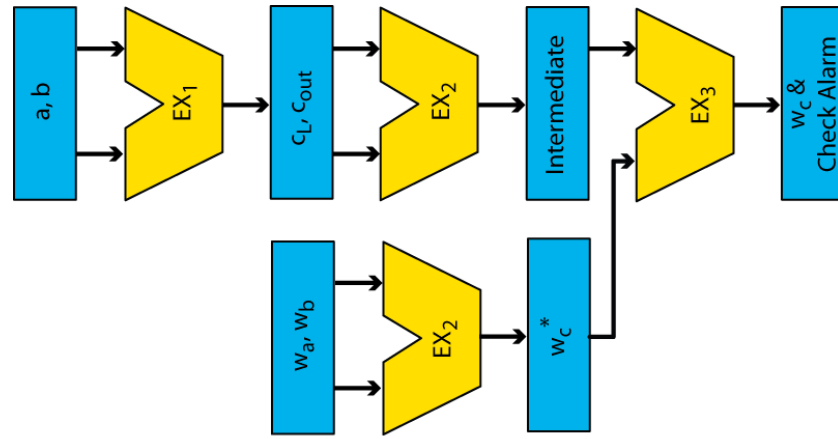


**Figure 4-6 Pipeline Stages of Robust Addition Instruction**

The pipeline stages of the robust multiplication instruction are illustrated in Figure 4-7. Similar to robust addition pipeline, the result of the multiplication is generated at the end of the first execution cycle. Also similar, the parity prediction circuit needs its input in the second execution cycle while the parity generation circuit produces the parity for the result at the end of the third execution stage. Therefore, data dependency between two consecutive robust instructions causes a pipeline stall of one clock cycle.

For instance, a robust addition instruction which depend the result of a robust multiplication operation will stall the pipeline for one clock cycle if it immediately follows the robust multiplication instruction since the dependency cannot be resolved by forwarding. An example scenario is illustrated in Figure 4-8, where one clock cycle is lost due to the aforementioned data dependency. This is an extra cost incurred due to the proposed error detection circuit. However, many dependencies of

this type can be resolved statically by reordering the instructions. Therefore, this does not lead to any significant degradation in overall performance of cryptographic operations. Our implementations in the subsequent chapters confirm that there are no stalls due to this type of data dependencies.

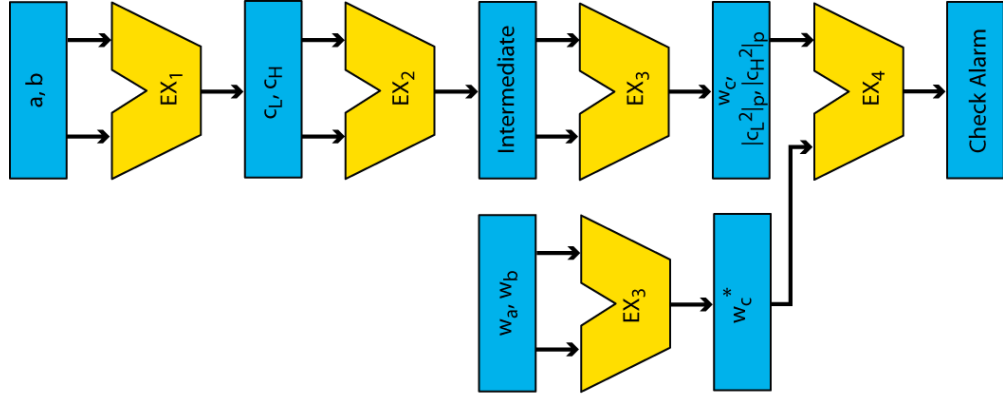


Figure 4-7 Pipeline Stages of Robust Multiplication Instruction

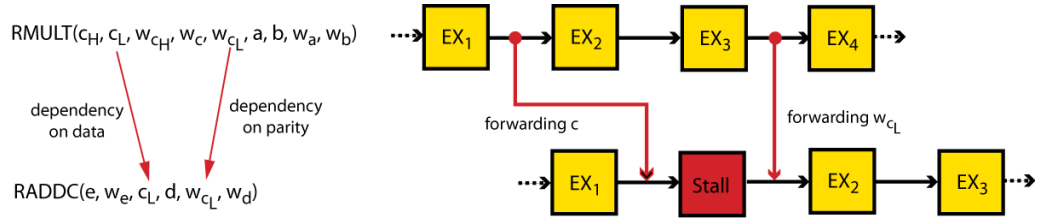


Figure 4-8 Robust Instruction Behavior at CPU Pipeline

Furthermore, it is, in fact, possible to eliminate these pipeline stalls by computing the parity values in an earlier pipeline stage. Indeed, we implemented these versions and observed that with an additional 200 gate equivalent area cost in ASIC implementation, these stalls are eliminated entirely.

### 4.3 Processor Configuration for Secure Software

The Montgomery modular multiplication algorithm [Montgomery, 1985], which is one of the most efficient methods used in many public key cryptography algorithms, is implemented in C language by employing the robust addition and multiplication instructions. In this implementation, we choose the SOS (separate operand scanning) method proposed in the literature for the Montgomery modular multiplication algorithm [Koç et al., 1996]. The C implementation of the robust Montgomery multiplication is simulated on Xtensa LX3 processor, whose instruction set is extended by the robust multiplication and addition instructions. Since these

instructions are integrated to the data path of the processor, the software can use them freely the same way as any native instruction.

We configured the Xtensa LX3 processor in different ways to demonstrate the cost of the proposed embedded error detection mechanism. In the following, we discuss these configurations:

- Configuration 0 : Base Configuration
- Configuration 1 : Residue Calculated in Circuit
- Configuration 2 : Residues Calculated Separately
- Configuration 3 : Software Implementation with Defined Instructions
- Configuration 4 : 31-bit Implementation of Configuration 2
- Configuration 5 : Dual Residue Codes for Hybrid Function
- Configuration 6 : 31-bit Implementation of Configuration 5
- Configuration 7 : Quadratic Dual Residue Code
- Configuration 8 : 31-bit Implementation of Configuration 7

#### **4.3.1 Configuration 0**

This is a simple 32-bit Xtensa LX3 microprocessor, which does not have any protection against fault injection attacks. In this configuration, the embedded processor has the following hardware resources.

- 16-bit MAC with 40 bit Accumulator
- Multiplication instructions MUL16, MUL32, accumulated MUL32, MULUH, and MULSH
- Pipeline length 5
- Widths of Cache and Memory Interface
  - Width of Instruction Fetch Interface 32 bit
  - Width of Data Memory/Cache interface 32 bit
  - Width of PIF interface 32 bit
  - Width of Interface to instruction cache 32 bit
- Instruction Cache size / Line size (Bytes) 4096 / 16
- Data Cache / Line size (Bytes) 4096 / 16
- System RAM size 128K, System ROM size 16K

### 4.3.2 Configuration 1

In this configuration, the robust addition and multiplication instructions are added to Xtensa LX3 microprocessor. The hardware circuits performing these instructions are depicted in Figure 4-9 and Figure 4-10. These circuits are described in TIE language which is developed for Xtensa LX3 microprocessors and integrated to the datapath of the processor core.

The robust addition and multiplication are called as follows.

$$(c_L, c_{out}) = RADD(a, b, c_{in})$$

$$(c_L, c_H) = RMUL(a, b)$$

The Configuration 1 protects the cryptographic computation against faults occurred during the execution of integer multiplication and addition operations. It does not protect temporary values while they are in the registers. Therefore, this configuration provides only limited protection against fault attacks.

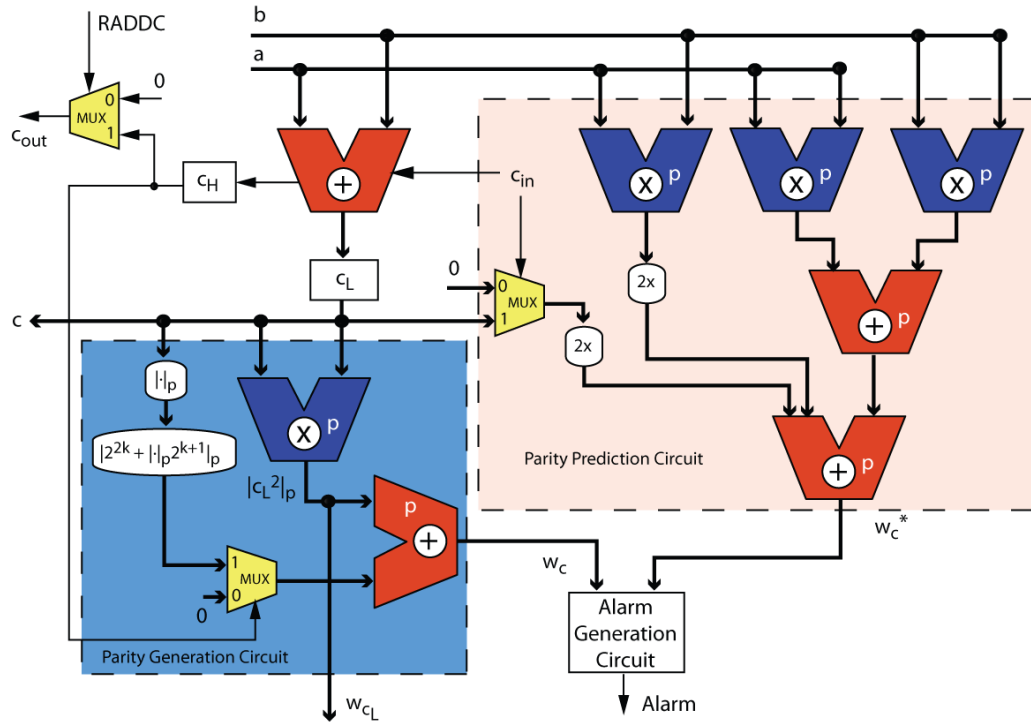


Figure 4-9 Robust Addition Unit with no Parity Input

### 4.3.3 Configuration 2

As mentioned previously, Configuration 1 can detect only the errors that occur during the execution of multiplication and addition operations. It, however, cannot detect the errors which the attackers may insert before and after the execution of these operations. Nevertheless, the inputs and outputs of these operations are vulnerable to fault injection attacks when they are held in the registers or in the memory.

As a remedy to this problem, the robust arithmetic instructions of Configuration 2 take the previous parity values calculated for their input operands as additional inputs and return their outputs together with the updated parity values. The robust addition and multiplication operation are implemented as follows.

$$(c, |c^2|_p, c_{out}) = RADD(a, |a^2|_p, b, |b^2|_p, c_{in})$$

$$(c_L, |c_L^2|_p, c_H, |c_H^2|_p) = RMUL(a, |a^2|_p, b, |b^2|_p)$$

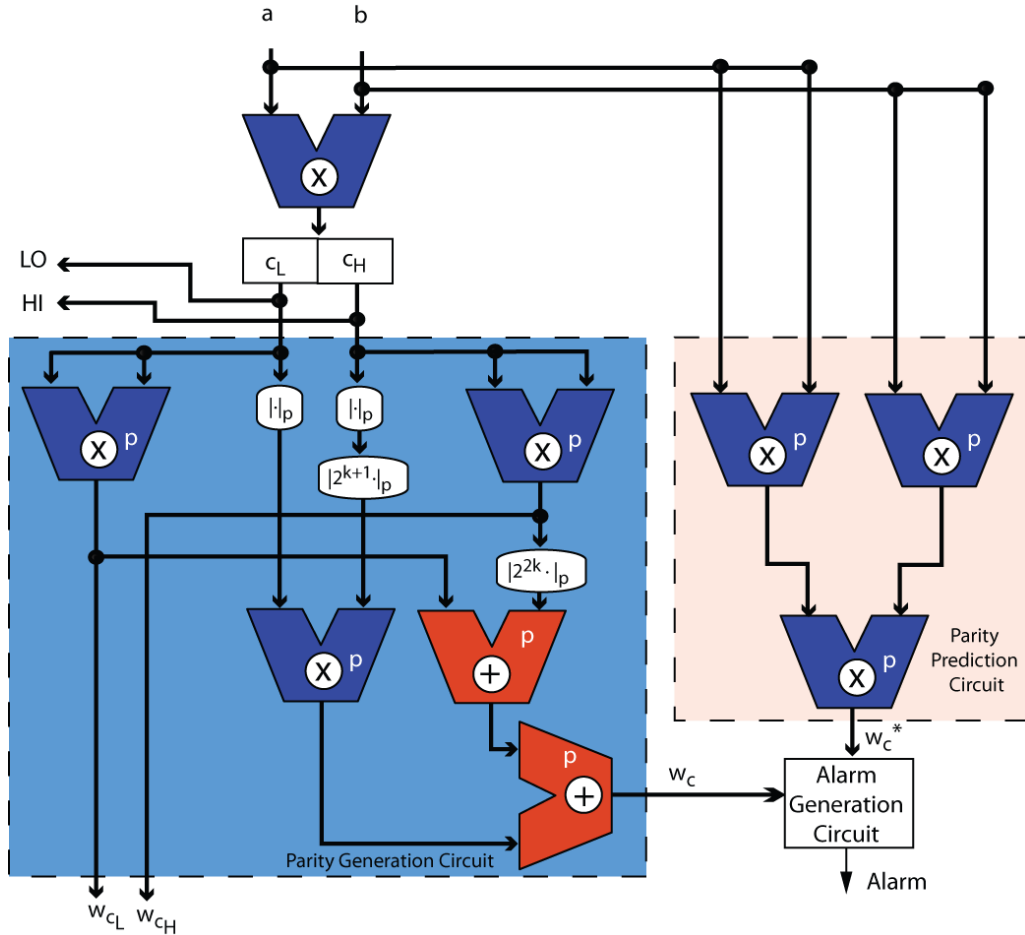


Figure 4-10 Robust Multiplication Unit with no Parity Input

As seen above, these robust instructions not only take  $a$  and  $b$  but also the parity values as input. Similarly, they return the updated parity values together with the operation results. Thus, Configuration 2 can keep track of the parity values throughout the entire execution of a cryptographic algorithm. The hardware circuits performing the robust instructions are depicted in Figure 4-4 and Figure 4-5. These circuits are described in TIE language which is developed to Xtensa LX3 microprocessors and integrated to the data path of the processor core. The Montgomery multiplication operation is implemented using 32-bit words. The error detection circuit utilizes nonlinear quadratic residue code with single modulus, where the modulus is selected as  $p = 2^{32}-5$ . The chosen modulus is the closest 32-bit prime integer to the word size of the processor and proves to render the most efficient implementation in the data path.

Contrary to Configuration 1, Configuration 2 protects the data not only during the arithmetic operations but also when stored in the register file or memory. Therefore, it provides better protection. On the other hand, Configuration 2 has a larger complexity than Configuration 1.

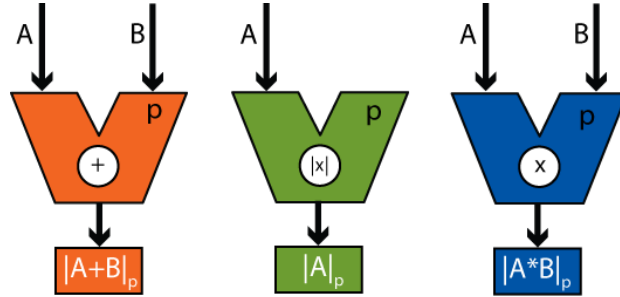


Figure 4-11 Robust Operations for Configuration-3

#### 4.3.4 Configuration 3

In Configuration 3, only instructions whose hardware organization is depicted in Figure 4-11 are used. Robust multiplication and addition operations are implemented in software by using these instructions. The  $a$ ,  $|a^2|_p$ ,  $b$ , and  $|b^2|_p$  variables are used as arguments to the functions implemented in software. The result of multiplication and addition operations and error conditions in performed operations are controlled. In addition to arithmetic results  $|c_H^2|_p$  and parity values are calculated and given as outputs. Multiplication and addition functions are defined as follows:

$$C = RADD C(a, |a^2|_p, b, |b^2|_p, c_{in})$$

$$C = RMUL(a, |a^2|_p, b, |b^2|_p)$$

#### 4.3.5 Configuration 4

This configuration is the same as Configuration 2 with the exception that the modulus are a 31-bit integer, i.e.  $p = 2^{31}-1$ , whereby the integer division needed in reduction is much easier to implement compared to  $p = 2^{32}-5$ . Adding a multiple of the modulus, which is extremely easy with  $p = 2^{31}-1$ , may produce undetectable errors. Note that about half of the 32-bit data words are smaller than  $2^{31}$  and addition of  $p = 2^{31}-1$  to them produces a number which is smaller than the word boundary  $2^{32}$ . This basically means that the success probability of the attack is about one half. Therefore, the software implementation of the modular multiplication algorithm utilizes 31-bit words, which incurs some overhead in clock cycle count of one modular multiplication operation. The robust addition and multiplication operation are implemented as follows.

$$(c, |c^2|_p, c_{out}) = RADD C(a, |a^2|_p, b, |b^2|_p, c_{in})$$

$$(c_L, |c_L^2|_p, c_H, |c_H^2|_p) = RMUL(a, |a^2|_p, b, |b^2|_p)$$

#### 4.3.6 Configuration 5

Configuration 5 implements dual residue codes with hybrid functions in Chapter 3.2.2, where moduli are  $p=2^{19}-1$  and  $q=2^{13}-1$ . The software implementation utilizes 32-bit data words. The robust addition and multiplication operation are implemented as follows.

$$C = RMULL(a, |a|_{\text{mix}}, b, |b|_{\text{mix}})$$

$$C = RADD C(a, |a|_{\text{mix}}, b, |b|_{\text{mix}}, c_{in})$$

#### 4.3.7 Configuration 6

Since the  $n = p \times q$  is a 31-bit integer in Configuration 6 and adding  $n$  to some data words can produce undetectable errors, Configuration 7 utilizes 31-bit data words. Other than this exception, it is the same as the Configuration 6. The robust addition and multiplication operation are implemented as follows.

$$C = RMULL(a, |a|_{\text{mix}}, b, |b|_{\text{mix}})$$

$$C = RADD C(a, |a|_{\text{mix}}, b, |b|_{\text{mix}}, c_{in})$$

#### 4.3.8 Configuration 7

Note that none of previous configuration provides a strong protection as proved in Chapter 3.2.1 and Chapter 3.2.2, and they are included only for comparison purposes. Configuration 7, which implements the quadratic dual residue codes in Chapter 3.2.3 with  $p = (2^{19} - 1)$  and  $q = (2^{13} - 1)$ , provides comparably much better protection. The same architecture in Chapter 4.2 is used for both modules p and q. Some of the functions used for implementation generate results for both moduli concurrently. The robust addition and multiplication operation are implemented as follows.

$$C = RMULL(a, |a|_{\text{mix}}, b, |b|_{\text{mix}})$$

$$C = RADD C(a, |a|_{\text{mix}}, b, |b|_{\text{mix}}, c_{in})$$

#### 4.3.9 Configuration 8

It is the same as Configuration 8, except that it utilizes 31-bit data words to protect the data path. In the chapter 4.4, we provide detailed implementation results for all configurations to evaluate the efficiency of the quadratic dual residue codes. The robust addition and multiplication operation are implemented as follows.

$$C = RMULL(a, |a|_{\text{mix}}, b, |b|_{\text{mix}})$$

$$C = RADD C(a, |a|_{\text{mix}}, b, |b|_{\text{mix}}, c_{in})$$

### 4.4 Implementation Results

This chapter summarizes the results of hardware and software implementations. Table 4-1 enumerates the execution times of the C implementations of the Montgomery multiplication algorithm in terms of number of clock cycles for the nine configurations mentioned previously.

Although it provides no error detection capability, Configuration 0 is the slowest implementation in Table 4-1. This is due to the facts that Configuration 0 is a

simple processor and has no direct access to the carry bit of an addition operation and to the higher part of the result of 32-bit multiplication operation. Several masking and shift operation are needed to have them on general purpose registers. Since we need to consider them in parity computations, the other configurations support the direct access. As can be observed from Table 4-1, Configuration 2 is slower than Configuration 1 since it's more complicated. Configuration 2 robust operations must take the parity data of the previous operations as additional outputs. Configuration 3 is the slowest since it has to perform error detection by the simplest robust operation unit. Configuration 4 is slightly slower than the Configuration 2 since using 31-bit results in more words in the operands (e.g. 67 versus 64 words for operands of 2048-bit). Configurations 7 and Configurations 8 implementing quadratic dual residue codes are comparable to other configurations with protection in terms of clock cycle count while providing superior error detection capability. Configuration 8 is slightly slower than Configuration 7 since using 31-bit words results in more words in the operands.

In summary, the figures in Table 4-1, Table 4-2 , Table 4-3, Table 4-4 basically show that configurations 7 and 8, which implements quadratic dual residue codes in microprocessor data path, performs comparably well with respect to other codes as far as the implementation efficiency is concerned, while it provides superior error detection performance against a powerful adversary.

Table 4-2 summarizes speed and area figures obtained for ASIC implementation of the nine configurations. For the ASIC implementation results, we used 0.13um LV Process geometry for ASIC device and process condition for ASIC device is "Worst Case". This table shows that the error detection capability do not affect the processor speed<sup>1</sup>. However, it causes roughly 50% increase in the circuit area. The configuration with the highest space requirement is Configuration 2. Configuration 1 follows Configuration 2. The other configurations (Configuration 4-8) have roughly the same area consumption. Configuration 7 and 8 are comparable to other configurations with error detection capability in terms of circuit area.

---

<sup>1</sup> The tool chain does not provide a reliable estimate for critical path delay in ASIC realization. For this, FPGA realizations provide more realistic estimates

We also synthesized all configurations into FPGA target device, and generated the bit files to program the Avnet LX200 board that features Xilinx Virtex-4 type FPGA. The implementation figures are obtained after placement-and-routing. Table 4-3 summarizes speed and space information obtained for these FPGA implementations. Timing constraint is chosen as 33.33 MHz (30 ns) as suggested by Tensilica. Table 4-4 enumerates the same results when there is no time constraint.

As can be observed from these tables, error detection circuits incur some overhead both in maximum applicable clock frequency and area consumption. For Configurations 2 and 4, the overhead in area is 20.5% and 34% in terms of total slice and LUT counts, respectively, where the number of LUTs should be considered as the area metric. For Configurations 7 and 8, the overhead in area is 6.23% and 31.3% in terms of total slice and LUT counts, respectively. Configurations 7 and 8 consume slightly less area compared to other configurations with error detection thanks to the fact that it utilizes the hardwired DSP units for multiplication operations in parity generation and prediction circuits. As for the time performance, there is only a slight degradation in clock frequency for the configurations with error detection compared to Configuration 0 when there is a time constraint of 30 ns. The figures in Table 4-4 show a greater penalty in maximum clock frequency.

**Table 4-1 Clock Cycle Comparison for Montgomery Implementation**

| <b>Configurations</b> | <b>2048-bit</b> | <b>1024-bit</b> | <b>512-bit</b> |
|-----------------------|-----------------|-----------------|----------------|
| Configuration 0       | 304,585         | 88,588          | 29,859         |
| Configuration 1       | 129,802         | 36,186          | 12,394         |
| Configuration 2       | 190,791         | 51,845          | 16,543         |
| Configuration 3       | 1,913,977       | 483,143         | 124,457        |
| Configuration 4       | 208,571         | 57,936          | 18,126         |
| Configuration 5       | 190,791         | 51,845          | 16,592         |
| Configuration 6       | 208,564         | 57,894          | 18,119         |
| Configuration 7       | 190,791         | 51,845          | 16,592         |
| Configuration 8       | 208,564         | 57,894          | 18,119         |

In summary, the figures in Table 4-1, Table 4-2, Table 4-3, Table 4-4 basically show that configurations 7 and 8, which implements quadratic dual residue codes in

microprocessor data path, performs comparably well with respect to other codes as far as the implementation efficiency is concerned, while it provides superior error detection performance against a powerful adversary.

**Table 4-2 Speed Area Information for ASIC Implementation**

| <b>Configurations</b> | <b>CPU Speed</b> | <b>Base CPU Area</b> | <b>TIE Area</b> | <b>Total Area</b> |
|-----------------------|------------------|----------------------|-----------------|-------------------|
| Configuration 0       | 320 MHz          | 64,000               | 0               | 64,000            |
| Configuration 1       | 320 MHz          | 64,000               | 33,076          | 97,076            |
| Configuration 2       | 320 MHz          | 64,000               | 33,761          | 97,761            |
| Configuration 3       | 320 MHz          | 64,000               | 25,069          | 89,069            |
| Configuration 4       | 320 MHz          | 64,000               | 32,985          | 96,985            |
| Configuration 5       | 320 MHz          | 64,000               | 32,300          | 96,300            |
| Configuration 6       | 320 MHz          | 64,000               | 32,289          | 96,289            |
| Configuration 7       | 320 MHz          | 64,000               | 34,918          | 96,918            |
| Configuration 8       | 320 MHz          | 64,000               | 34,912          | 96,912            |

**Table 4-3 Speed Gate Information for FPGA Implementation with Time Constraint @33.33 MHz**

| <b>Configurations</b> | <b>Max Clock Frequency (MHz)</b> | <b>Slice Count</b> | <b>LUT Count</b> | <b>RAM16b</b> | <b>DSP48s</b> |
|-----------------------|----------------------------------|--------------------|------------------|---------------|---------------|
| Configuration 0       | 37,281                           | 7,751              | 19,958           | 272           | 1             |
| Configuration 1       | 33,338                           | 9,263              | 26,761           | 272           | 1             |
| Configuration 2       | 30,747                           | 9,172              | 26,734           | 272           | 1             |
| Configuration 3       | 33,672                           | 7,270              | 25,068           | 272           | 0             |
| Configuration 4       | 34,338                           | 9,395              | 26,701           | 272           | 0             |
| Configuration 5       | 36,004                           | 8,917              | 26,082           | 272           | 2             |
| Configuration 6       | 36,004                           | 8,915              | 26,080           | 272           | 2             |
| Configuration 7       | 33,940                           | 8,235              | 26,202           | 272           | 4             |
| Configuration 8       | 33,940                           | 8,234              | 26,198           | 272           | 4             |

**Table 4-4 Speed Gate Information for FPGA Implementation with No Time Constraint**

| <b>Configurations</b> | <b>Max Clock<br/>Frequency (MHz)</b> | <b>Slice<br/>Count</b> | <b>LUT<br/>Count</b> | <b>RAM16b</b> | <b>DSP48s</b> |
|-----------------------|--------------------------------------|------------------------|----------------------|---------------|---------------|
| Configuration 0       | 57,621                               | 7,751                  | 19,958               | 272           | 1             |
| Configuration 1       | 34,338                               | 9,263                  | 26,761               | 272           | 1             |
| Configuration 2       | 31,231                               | 9,172                  | 26,734               | 272           | 1             |
| Configuration 3       | 40,813                               | 7,270                  | 25,068               | 272           | 0             |
| Configuration 4       | 40,538                               | 9,395                  | 26,701               | 272           | 0             |
| Configuration 5       | 45,460                               | 8,917                  | 26,082               | 272           | 2             |
| Configuration 6       | 45,460                               | 8,915                  | 26,080               | 272           | 2             |
| Configuration 7       | 39,001                               | 8,235                  | 26,202               | 272           | 4             |
| Configuration 8       | 39,001                               | 8,234                  | 26,198               | 272           | 4             |

## 5 ON SELECTION OF MODULUS OF QUADRATIC CODES

### 5.1 Fault and Attacker Model

In this work, we assume that the attacker knows everything about the hardware architecture and fault detection mechanism of the cryptographic device. The attacker can individually flip the data bits and induce faults by this way. Nevertheless, the attacker cannot read out the values of the data bits because a tamper proof coating prevents the direct invasive access to chip. In practice, an attacker can easily manipulate the bits of the data using laser beams and electromagnetic fields.

Let a systematic arithmetic code append each data word  $x \in \mathbb{Z}_{2^k}$  with the redundancy  $r = f(x) \in \mathbb{Z}_{2^l}$  where the function  $f: \mathbb{Z}_{2^k} \rightarrow \mathbb{Z}_{2^l}$ . We denote the resulting codewords with the pair  $(x, r)$ . We denote a fault in a codeword  $(x, r)$  by the pair  $(\varepsilon, \delta)$  where  $\varepsilon \in \mathbb{Z}_{2^k}$  and  $\delta \in \mathbb{Z}_{2^l}$  are the faults in  $x$  and  $r$ , respectively.

Any nonzero bit in  $\varepsilon$  and  $\delta$  corresponds a bit flip in our fault model. The bit flip faults can be expressed as the XOR of codeword bits with the fault bits as follow.

$$(x \oplus \varepsilon, r \oplus \delta)$$

In this respect, our work differs from the similar work in [Gaubatz et al., 2006], which uses an additive fault model. Although the additive fault models are easier to analyze, bitwise XOR operations represent the bit flips more realistically. In the proposed attacker model, in which neither data word nor residue is known, the attacker aims to flip a certain subset of data or residue bits in the best possible scenario. On the other hand, additive fault model assumes that the attacker can add an error to either data word or residue (or both) with equal probability of success. However, as shown in this paper, introducing some additive errors is much harder than others.

Moreover, XORing the data with the fault can also be expressed as additions and subtractions of the data bits and the fault bits. In fact, one can perform the mathematical analysis of the strength of the code against fault attacks only after converting the bitwise XOR operation to the addition and subtraction operations.

*Proposition 1:* Let  $\varepsilon = \sum_{i \in E} 2^i$  for some set  $E \subseteq \mathbb{Z}_k$ . Then, the bitwise XOR with  $\varepsilon$  is converted to the addition and subtraction operations as follows.

$$x \oplus \varepsilon = x \oplus \sum_{i \in E} 2^i$$

$$x \oplus \varepsilon = \varepsilon - \sum_{i \in E} (x_i)2^i + \sum_{i \notin E} (x_i)2^i \quad (5-1)$$

where  $x_i$  denotes the  $i$ th binary digit  $x$ .

*Proof:* If the XOR operation is written in bit-level,

$$x \oplus \varepsilon = \sum_{i=0}^{k-1} (x_i \oplus \varepsilon_i) 2^i$$

$$x \oplus \varepsilon = \sum_{i \in E} (x_i \oplus 1) 2^i + \sum_{i \notin E} (x_i \oplus 0) 2^i$$

XOR with 0 and 1 is converted to the addition as follows.

$$x \oplus \varepsilon = \sum_{i \in E} (1 - x_i) 2^i + \sum_{i \notin E} (x_i) 2^i$$

$$x \oplus \varepsilon = \sum_{i \in E} 2^i - \sum_{i \in E} (x_i) 2^i + \sum_{i \notin E} (x_i) 2^i$$

In our fault model, the attacker does not know the codeword  $(x, r)$  but he can induce any fault  $(\varepsilon, \delta)$  on it. A detection failure occurs when a manipulated codeword is a valid codeword. In this case,  $f(x \oplus \varepsilon) = r \oplus \delta = f(x) \oplus \delta$ . The probability of missing a fault is given by

$$Q(\varepsilon, \delta) = \frac{|\{x | f(x \oplus \varepsilon) = f(x) \oplus \delta\}|}{2^k} \quad (5-2)$$

when all the data words  $x \in \mathbb{Z}_{2^k}$  are equally likely.

## 5.2 Attacks on Quadratic Codes

The quadratic codes  $(x, r) = (x, x^2 \bmod p)$  are proposed in [Gaubatz et al., 2006] for fault detection in cryptographic devices where the modulus  $p$  is positive integer. The redundancy  $r = x^2 \bmod p$  is used to detect an induced fault  $(\varepsilon, \delta)$ . However the fault codeword

$$(x \oplus \varepsilon, r \oplus \delta) = (x \oplus \varepsilon, (x^2 \bmod p) \oplus \delta) \quad (5-3)$$

Is a valid codeword, if  $(x \oplus \varepsilon)^2 \bmod p = (x^2 \bmod p) \oplus \delta$  holds. In this case, the fault remains undetected.

Inducing faults on both data and redundancy bits may not be the best strategy to attack a systematic quadratic code. Since the data word  $x$  unknown, the calculated residue  $(x \oplus \varepsilon)^2 \bmod p$  can be thought of as a random function which maps the fault  $\varepsilon$  to the  $(p-1)/2$  squares in  $\mathbb{F}_p$ . The fault induced redundancy  $(x^2 \bmod p) \oplus \delta$  must be equal to this residue. However, choosing the correct  $\delta$  for a given  $\varepsilon$  is pure luck. It is very hard for the attacker to develop a clever scheme exploiting the properties of the modulus to find the correct  $\delta$ . As shown in Chapter 5.3, the probability of introducing undetectable fault can be made better than  $2/(p-1)$  by attacking only the data word.

This work proposes several attacks manipulating only the data part of the quadratic code without touching its redundancy part. These attacks induce some strategically chosen fault  $\varepsilon$  in the data word in order to increase the probability that  $x \oplus \varepsilon = \mp x + wp$  for an integer  $w$ . Then the quadratic code fails since the calculated redundancy remains the same as seen below.

$$(x \oplus \varepsilon)^2 \bmod p = (\mp x + wp)^2 \bmod p = x^2 \bmod p \quad (5-4)$$

### 5.2.1 Exploiting Sparse Modulus Multiples

As shown later in Chapter 5.3, when all the data words  $x$  are equally likely, the probability  $Pr\{x \oplus \lambda_p = x + wp\}$  for an integer  $w$  become higher as  $\lambda_p$  has a smaller number of nonzero bits. Thus, successful attacks can be performed by using such modulus multiples.

Let us illustrate this with an example. Let the data word size  $k = 12$  bits and the modulus

$$p = 683 = (0\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1)_2$$

We look for the 12-bit multiples of  $p$  including  $p$  itself.

$$p = 2049 = (1\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 0\ 1)_2$$

is very sparse. Thus, we choose the fault  $\lambda_p = 3p$ . Notice that  $x \oplus 3p = x + 3p$  for the case

$$\begin{aligned} x &= (0*****0)_2, \\ 3p &= (100000000001)_2 \\ x \oplus 3p &= (1*****1)_2 \end{aligned}$$

And  $x \oplus 3p = x - 3p$  for the case

$$\begin{aligned} x &= (1*****1)_2, \\ 3p &= (100000000001)_2 \\ x \oplus 3p &= (0*****0)_2 \end{aligned}$$

Where  $*$  denotes an arbitrary bit. Then, the redundancy

$$(x \oplus 3p)^2 = (x \mp 3p)^2 = x^2 \bmod p$$

remains the same. As a result, the quadratic code cannot detect the fault  $3p$  for the data words

$$\begin{aligned} x &= (0*****0)_2 \\ x &= (1*****1)_2 \end{aligned}$$

among all the possible 12-bit data words. Then, the fault  $3p$  cannot be detected for at least half of the data words and detection failure probability is at least  $1/2$ . Naturally, this is an unacceptable detection performance for a code with 12-bit residue. Also, there are additionally the following undetected cases.

$$x = (010101010111)_2 = 1367$$

$$x = (110101010110)_2 = 3414$$

$$x = (000000000001)_2 = 1$$

$$x = (100000000000)_2 = 2048$$

### 5.2.2 Exploiting Dense Modulus Multiples

As shown later in Chapter 5.3, when all the data words  $x$  are equally likely, the probability  $Pr\{x \oplus \lambda_p = wp - x\}$  for an integer  $w$  become higher as  $\lambda_p$  has a larger number of nonzero bits. Thus, successful attacks can be performed by using such modulus multiples.

Let us illustrate this with an example. Let the data word size  $k = 12$  bits and the modulus

$$p = 4093 = (1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1)_2$$

We look for the 12-bit multiples of  $p$  including  $p$  itself. The modulus  $p$  itself is very dense. Therefore, we choose the fault  $\lambda_p = p$ . Notice that  $x \oplus p = p - x$  for the case

$$x = (\text{*****}0*)_2,$$

$$p = (11111111101)_2$$

$$x \oplus p = (\overline{\text{*****}0\overline{*}})_2$$

Where  $*$  denotes an arbitrary bit and  $\overline{*}$  denotes its negation. Then, the redundancy

$$(x \oplus p)^2 = (p - x)^2 = x^2 \bmod p$$

remains the same. As a result, the quadratic code cannot detect the fault  $p$  for the data words

$$x = (\text{*****0*})_2$$

among all possible 12-bit data words. Then, the fault  $p$  cannot be detected for at least half of the data words and the detection failure probability is at least 1/2. Also, there additionally the following undetected cases.

$$x = (000000000010)_2 = 2$$

$$x = (111111111111)_2 = 4095$$

### 5.2.3 Exploiting Modulus Multiples Having Many Consecutive Nonzero Digits

If an integer has a binary representation with long sequences of 1's, it has also a redundant binary representation with large number of 0's. To obtain such a representation, we can use the redundant binary representation with the digit set  $\{-1, 0, 1\}$  (signed digit representation).

Let  $\lambda_p$  be a modulus multiple such that  $\pm 1$ 's in its redundant binary representation with the digit set  $\{-1, 0, 1\}$  are given by the set  $F$ . Then,

$$\lambda_p = \sum_{i \in F} \mp 2^i \quad (5-5)$$

and, if we convert all -1's to +1's, the linear expression

$$\sum_{i \in F} 2^i = \lambda_p + \mu \quad (5-6)$$

is obtained for some integer  $\mu$ . In the attack, the data bits  $x_i$  for  $x_i$  for  $i \in F$  are flipped without touching the redundancy bits. Then, the fault induced data word is

$$x \oplus \sum_{i \in F} 2^i = x \oplus (\lambda_p + \mu) \quad (5-7)$$

As shown later in Chapter 5.4, when all the data words  $x$  are equally likely, the probability  $Pr\{x \oplus (\lambda_p + \mu) = x + wp\}$  for an integer  $w$  becomes higher as  $F$  gets smaller, i.e. as the signed digit representation of  $\lambda_p$  get sparser. Successful attacks can be performed by using this fact.

Let us illustrate this with an example. Let the data word size  $k = 12$  bits and the same modulus multiple

$$\lambda_p = 127 = (0\ 0\ 0\ 0\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 1)_2$$

This number has a long sequence of ones, and thus it has the following signed digit representation which contains a large number of zeros.

$$\lambda_p = 127 = (0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ \bar{1})_2$$

Where  $\bar{1}$  denotes -1. Mapping  $\bar{1} \rightarrow 1$  in this representation, we obtain the fault used in the attack, as follows.

$$\lambda_p + \mu = 129 = (0\ 0\ 0\ 0\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)_2$$

Here,  $\mu=2$ . Note that  $x \oplus (\lambda_p + \mu) = x + \lambda_p$  for the case

$$\begin{aligned} x &= (****0*****1)_2, \\ \lambda_p &= (00001000000\bar{1})_2 \\ \lambda_p + \mu &= (000010000001)_2 \\ x \oplus (\lambda_p + \mu) &= (****1*****0)_2 \end{aligned}$$

$x \oplus (\lambda_p + \mu) = x - \lambda_p$  for the case

$$\begin{aligned} x &= (****1*****0)_2, \\ -\lambda_p &= (0000\bar{1}0000001)_2 \\ \lambda_p + \mu &= (000010000001)_2 \\ x \oplus (\lambda_p + \mu) &= (****0*****1)_2 \end{aligned}$$

Where \* denotes an arbitrary bit. Then the redundancy

$$(x \oplus (\lambda_p + \mu))^2 = (x \pm \lambda_p)^2 = x^2 \bmod p$$

remains the same. As a result, the quadratic code cannot detect the fault  $\lambda_p + \mu$  for the data words

$$\begin{aligned} x &= (****0*****0)_2, \\ x &= (****1*****1)_2 \end{aligned}$$

among all the possible 12-bit data words. Then, the fault  $\lambda_p + \mu$  cannot be detected for at least half of the data words and the detection failure probability is at least 1/2. (Actually, there are 32 more undetected cases.)

#### 5.2.4 Exploiting Modulus Multiples Having Many Consecutive Zero Digits

If an integer has a binary representation with long sequences of 0's, it has also a redundant binary representation with large number of 1's. To obtain such a representation, we can use the redundant binary representation with the digit set  $\{0, 1, 2\}$ .

Let  $\lambda_p$  be a modulus multiple such that 1's and 2's in its redundant binary representation with the digit set  $\{0, 1, 2\}$  are given by the set  $F$  and  $B$ , respectively. Then,

$$\lambda_p = \sum_{i \in F} 2^i + 2 \sum_{i \in B} 2^i \quad (5-8)$$

and, if we convert all 2's to 0's, the linear expression

$$\sum_{i \in F} 2^i = \lambda_p - \mu \quad (5-9)$$

is obtained for some integer  $\mu$ . In the attack, the data bits  $x_i$  for  $x_i$  for  $i \in F$  are flipped without touching the redundancy bits. Then, the fault induced data word is

$$x \oplus \sum_{i \in F} 2^i = x \oplus (\lambda_p - \mu) \quad (5-10)$$

As shown later in Chapter 5.4, when all the data words  $x$  are equally likely, the probability  $Pr\{x \oplus (\lambda_p - \mu) = wp - x\}$  for an integer  $w$  becomes higher as  $F$  gets larger, i.e. as 1's increase in the representation of  $\lambda_p$  in base 2 with digit 0, 1, 2. Successful attacks can be performed by using this fact.

Let us illustrate this with an example. Let the data word size  $k=12$  bits and the same modulus multiple

$$\lambda_p = 3841 = (1\ 1\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 0\ 1)_2$$

This number has the following redundant binary using the digit set  $\{0, 1, 2\}$ .

$$\lambda_p = 3841 = 1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 2\ 1)_2$$

Note that a long sequence of zeros is converted to a large number of ones thanks to the redundant representation. Mapping  $2 \rightarrow 0$  in this representation, we obtain the fault which will be used in the attack, as follows.

$$\lambda_p - \mu = 3837 = (1\ 1\ 1\ 0\ 1\ 1\ 1\ 1\ 1\ 1\ 0\ 1)_2$$

Here,  $\mu=4$ . Note that  $x \oplus (\lambda_p - \mu) = \lambda_p - x$  for the case

$$\begin{aligned} x &= (**0*****1*)_2, \\ \lambda_p &= (11101111121)_2 \\ \lambda_p - \mu &= (11101111101)_2 \\ x \oplus (\lambda_p - \mu) &= (\overline{***}\ 0\ \overline{*****}\ 1\ \overline{*})_2 \end{aligned}$$

Then the redundancy

$$(x \oplus (\lambda_p - \mu))^2 = (\lambda_p - x)^2 = x^2 \bmod p$$

remains the same. As a result, the quadratic code cannot detect the fault  $\lambda_p - \mu$  for the data words

$$x = (**0*****1*)_2,$$

among all the possible 12-bit data words. Then, the fault  $\lambda_p - \mu$  cannot be detected for at least one fourth of the data words and the detection failure probability is at least  $1/4$ . (Actually, there are 12 more undetected cases.)

### 5.3 Data Word Attacks Using Multiples of Modulus

Let  $\lambda$  be some integer such that the modulus multiple  $\lambda p > 0$  fits into a  $k$ -bit data word. That is,  $\lambda p \in \mathbb{Z}_{2^k}$  and

$$\lambda p = \sum_{i \in E} 2^i$$

for some nonempty set  $E \subseteq \mathbb{Z}_k$ . By flipping the bits of  $x$  in the position  $i \in E$  without touching the redundancy bits, the attacker can induce the fault  $(\varepsilon, \delta) = (\lambda p, 0)$ . Then the codeword  $(x, x^2 \bmod p)$  becomes

$$(x \oplus \varepsilon, (x^2 \bmod p) \oplus \delta) = (x \oplus \lambda p, x^2 \bmod p) \quad (5-11)$$

Let us split  $x$  into two integers  $\hat{x}$  and  $\bar{x}$  using the set of the flipped bit positions  $E \subseteq \mathbb{Z}_k$  as follows.

$$\begin{aligned} \hat{x} &= \sum_{i \in E} x_i 2^i, \\ \bar{x} &= \sum_{i \notin E} x_i 2^i = \sum_{i \in (\mathbb{Z}_k \setminus E)} x_i 2^i, \\ \hat{x} + \bar{x} &= x = \sum_{i=0}^{k-1} x_i 2^i. \end{aligned} \quad (5-12)$$

Here,  $\hat{x}$  is the manipulated part and  $\bar{x}$  is the untouched part of  $x$ . Due to (5-1), the fault induced data word is

$$\begin{aligned} x \oplus \lambda p &= \lambda p - \sum_{i \in E} x_i 2^i + \sum_{i \notin E} x_i 2^i \\ x \oplus \lambda p &= \lambda p - \hat{x} + \bar{x} \\ x \oplus \lambda p &= \lambda p + x - 2\hat{x} \\ x \oplus \lambda p &= \lambda p - x + 2\bar{x}. \end{aligned} \quad (5-13)$$

The fault induced codeword in (5-11) cannot be detected, if

$$\begin{aligned} (x \oplus \lambda p)^2 \bmod p &= x^2 \bmod p \\ (\lambda p - \hat{x} + \bar{x})^2 \bmod p &= (\hat{x} + \bar{x})^2 \bmod p. \end{aligned}$$

After simplifying the above equality, we can find that the fault  $(\varepsilon, \delta) = (\lambda p, 0)$  cannot be detected, if

$$4\hat{x}\bar{x} = 0 \bmod p \quad (5-14)$$

### 5.3.1 Data Words for Which Attack is Undetectable

If  $p$  is prime, the data words  $x$  in the sets  $S_1$  and  $S_2$  defined below constitute all the solutions of (5-14), and thus the fault  $\lambda p = \sum_{i \in E} 2^i$  cannot be detected for  $x \in S_1 \cup S_2$ .

$$\begin{aligned} S_1 &= \bigcup_{\alpha p \in S_A} \{x \in \mathbb{Z}_{2^k} \mid \sum_{i \in E} x_i 2^i = \alpha p\} \\ S_2 &= \bigcup_{\beta p \in S_B} \{x \in \mathbb{Z}_{2^k} \mid \sum_{i \notin E} x_i 2^i = \beta p\} \end{aligned} \quad (5-15)$$

where

$$\begin{aligned} S_A &= \{\alpha p \mid \alpha p = \sum_{i \in A} 2^i \text{ for } \exists A \subseteq E\} \\ S_B &= \{\beta p \mid \beta p = \sum_{i \in B} 2^i \text{ for } \exists B \subseteq (\mathbb{Z}_k \setminus E)\} \end{aligned} \quad (5-16)$$

Here,  $S_A$  always contains  $\sum_{i \in \emptyset} 2^i = 0$  and  $\sum_{i \in E} 2^i = \lambda p$ . Similarly,  $S_B$  always contains  $\sum_{i \in \emptyset} 2^i = 0$ . Thus,

$$S_A \supseteq \{0, \lambda p\}, \quad S_B \supseteq \{0\} \quad (5-17)$$

#### 5.3.1.1 Data words Satisfying $x \oplus \lambda p = x + wp$

Since  $\hat{x} = \sum_{i \in E} x_i 2^i = \alpha p$  when  $x \in S_1$ , it follows from (5-13) that the fault induced data word

$$x \oplus \lambda p = \lambda p + x - 2\hat{x}$$

$$x \oplus \lambda p = \lambda p + x - 2\alpha p = x + wp|_{w=(\lambda-2\alpha)}$$

and XOR operation becomes addition with some modulus multiple. In fact, this is why quadratic code fails when  $x \in S_1$ . Also, it follows from (5-15) and (5-16) that

$$|S_1| = |S_A| 2^{k-|E|} \quad (5-18)$$

### 5.3.1.2 Data words Satisfying $x \oplus \lambda p = wp - x$

Since  $\bar{x} = \sum_{i \notin E} x_i 2^i = \beta p$  when  $x \in S_2$ , it follows from (5-13) that the fault induced data word

$$x \oplus \lambda p = \lambda p - x + 2\bar{x}$$

$$x \oplus \lambda p = \lambda p - x + 2\beta p = wp - x|_{w=(\lambda+2\beta)}$$

and XOR operation becomes subtraction from some modulus multiple. In fact, this is why quadratic code fails when  $x \in S_2$ . Also, it follows from (5-15) and (5-16) that

$$|S_2| = |S_B|2^{|E|} \quad (5-19)$$

### 5.3.2 Exact Count of Detection Failures

Let a modulus multiple  $\lambda p = \sum_{i \in E} 2^i$  for a nonempty set  $E \subseteq \mathbb{Z}_k$ . If  $p$  is prime, the fault  $(\varepsilon, \delta) = (\lambda p, 0)$  cannot be detected exactly for

$$|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B| \quad (5-20)$$

Distinct value of the data word  $x$ , where  $S_A$  and  $S_B$  are given by (5-16). This content is equal to  $|S_1 \cup S_2|$ . As said before in this chapter,  $|S_1| = |S_A|2^{k-|E|}$  and  $|S_2| = |S_B|2^{|E|}$ . Also,  $|S_1 \cap S_2| = |S_A||S_B|$  since

$$S_A \cap S_B = \bigcup_{\substack{\alpha p \in S_A \\ \beta p \in S_B}} \left\{ x \mid \sum_{i \in E} x_i 2^i = \alpha p, \sum_{i \notin E} x_i 2^i = \beta p \right\} \quad (5-21)$$

Let us illustrate how to calculate the number of the detection failures using (5-20) with an example. Let  $x \in \mathbb{Z}_{2^k}$  be encoded into the codeword  $(x, x^2 \bmod p)$  for  $k = 12$  and  $p = 19$ . Let the attacker induced the fault

$$(\varepsilon, \delta) = \lambda p, 0 = (119p, 0) = (2261, 0)$$

in the codeword. For this, the attacker flips the data bits in the positions corresponding to nonzero bits of

$$\lambda p = 2261 = (100011010101)_2$$



**Table 5-2 Multiples of the modulus  $p=19$  whose nonzero bit positions  $i \notin E = \{0, 2, 4, 6, 7, 11\}$ .**

| Bit No | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|--------|----|----|---|---|---|---|---|---|---|---|---|---|
| 96p =  | 0  | 1  | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 56p =  | 0  | 1  | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 54p =  | 0  | 1  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 14p =  | 0  | 0  | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| 0p =   | 0  | 0  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

### 5.3.3 Basic Attack Strategy

A simple way of generating large number of detection failures is to choose the fault  $\lambda p = \sum_{i \in E} 2^i$  such that the flipped bit set  $E$  is very small or very large (i.e.,  $\lambda p$  is very sparse or very dense). Then, the exponential terms  $2^{k-|E|}$  and  $2^{|E|}$  blow up the detection failure count (5-20).

As seen from (5-17),  $S_A \geq 2$  and  $S_B \geq 1$ . Thus, the detection failure count

$$|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B| \geq 2^{k+1-|E|} + 2^{|E|} - 2$$

for any fault  $\lambda p = \sum_{i \in E} 2^i$ . Note that this lower bound has a minimum at  $|E| = (k+1)/2$  as seen below.

$$2^{k+1-|E|} + 2^{|E|} - 2 \geq 2^{(k+3)/2} - 2 \quad (5-22)$$

And, it increases exponentially as  $|E|$  gets smaller or larger than  $(k+1)/2$ .

### 5.3.4 Maximum Number of Detection Failures

*Definition 1:* Let the fault  $\lambda p = \sum_{i \in E} 2^i$  be induced in an attack for some flipped bit set  $E \subseteq \mathbb{Z}_k$ .

$$\mathcal{E} = \max_{\lambda p} (|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B|) \quad (5-23)$$

Denotes the maximum number of detection failures.

As said before, the attacker can approach this maximum by choosing  $\lambda p$  very sparse or very dense. Now, we show the relationship of this maximum with the sparsest and densest modulus multiples.

*Definition 2:*  $\lambda_s p$  denotes the modulus multiple in  $(0, 2^k)$  whose binary representation has the largest number of 0's (sparsest).  $E_{min}$  denotes the set of 1's in the binary representation of  $\lambda_s p$ . ( $\lambda_s p$  may not be unique.)

*Definition 3:*  $\lambda_d p$  denotes the modulus multiple in  $(0, 2^k)$  whose binary representation has the largest number of 1's (densest).  $E_{max}$  denotes the set of 1's in the binary representation of  $\lambda_d p$ . ( $\lambda_d p$  may not be unique.)

Note that, according to these definitions,

$$1 \leq |E_{min}| \leq |E| \leq |E_{max}| \leq k$$

for any modulus multiple  $\lambda p = \sum_{i \in E} 2^i$  in  $(0, 2^k)$ .

*Lemma 1:* Let set sizes used in the detection failure count in (5-20) are follows.  $|S_A| = 2$ ,  $|S_B| \geq 1$  when  $|E_{min}| \leq |E| < 2|E_{min}|$  and  $|S_A| \geq 2$ ,  $|S_B| = 1$  when  $|E_{max}| - |E_{min}| < |E| \leq |E_{max}|$ . Also, the maximum detection failure count

$$\mathcal{E} = 2^{k+1-u} + 2^u - 2 \geq 2^{(k+3)/3} - 2 \quad (5-24)$$

where  $u = \max(k + 1 - |E_{min}|, |E_{max}|)$ .

*Theorem 1:* Let the modulus  $p$  is chosen such that  $|E_{max}| < 2|E_{min}|$  holds for its multiples  $\lambda p \in (0, 2^k)$ . Then,  $|S_A| = 2$  and  $|S_B| = 1$ , and the detection failure count (5-20) is

$$2^{k+1-|E|} + 2^{|E|} - 2 \quad (5-25)$$

This count reaches the maximum  $\mathcal{E}$  when the flipped bit set either  $E = |E_{min}|$  or  $E = |E_{max}|$ . Also,

$$\mathcal{E} = 2^{k+1-u} + 2^u - 2 \geq 2^{(k+3)/3} - 2 \quad (5-26)$$

where  $u = \max(k + 1 - |E_{min}|, |E_{max}|)$ .

*Proof:* It can be proven by contradiction that  $S_A = \{0, \lambda p\}$  when  $|E_{min}| \leq |E| \leq 2|E_{min}|$ . Note that

$$S_A = \{\alpha p \mid \alpha p = \sum_{i \in A \subseteq E} 2^i\}$$

as seen from (5-16). If we had  $S_A = \{0, \alpha p, \lambda p\}$  for some nonzero  $\alpha p \neq \lambda p$ , the following three distinct multiples of  $p$  would exist in the interval  $(0, 2^k)$ .

$$\lambda p = \sum_{i \in E} 2^i, \quad \alpha p = \sum_{i \in A \subseteq E} 2^i, \quad (\lambda - \alpha)p = \sum_{i \in E \setminus A} 2^i$$

These multiples cannot be sparser than the sparsest modulus multiple. Thus,

$$|E_{min}| \leq |A \subseteq E|, |E \setminus A|, |E|$$

Then,  $2|E_{min}| \leq |A| + |E \setminus A| = |E|$  and this contradicts with  $|E_{min}| \leq |E| \leq 2|E_{min}|$ .

$S_B = \{0\}$  when  $|E_{max}| - |E_{min}| < |E| \leq |E_{max}|$ . This can be proven by contradiction. Note that

$$S_B = \{\beta p \mid \beta p = \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus E)} 2^i\}$$

as seen from (5-16). If we had  $S_B = \{0, \beta p\}$  for some nonzero  $\beta p$ , the following three distinct multiples of  $p$  would exist in the interval  $(0, 2^k)$ .

$$\lambda p = \sum_{i \in E} 2^i, \quad \beta p = \sum_{i \in B} 2^i, \quad (\lambda + \beta)p = \sum_{i \in (E \cup B)} 2^i$$

These multiples cannot be sparser than the sparsest or denser than densest modulus multiples. Thus,

$$|E_{min}| \leq |E|, |B|, |E \cup B| \leq |E_{max}|$$

Also remember that  $B \subseteq (\mathbb{Z}_k \setminus E)$ . Then,  $|E| = |E \cup B| - |B| \leq |E_{max}| - |E_{min}|$  and this contradicts with  $|E_{max}| - |E_{min}| < |E| \leq |E_{max}|$ .

In summary,  $|S_A| = 2$  when  $|E_{min}| \leq |E| < 2|E_{min}|$  and  $|S_B| = 1$  when  $|E_{max}| - |E_{min}| < |E| \leq |E_{max}|$ , as stated in Lemma 1. Also,  $|E_{max}| < 2|E_{min}|$  implies that

$$|E_{max}| - |E_{min}| < |E_{min}| \leq |E| \leq |E_{max}| < 2|E_{min}|$$

Thus,  $|S_A| = 2$ ,  $|S_B| = 1$ , and the detection failure count

$$|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B| = 2^{k+1-|E|} + 2^{|E|} - 2$$

when  $|E_{max}| < 2|E_{min}|$ , as stated in the Theorem 1.

$|S_A| \geq 2$  and  $|S_B| \geq 1$  due to (5-17). Thus, the detection failure are always more than  $2^{k+1-|E|} + 2^{|E|} - 2$  for any fault  $\lambda p = \sum_{i \in E} 2^i$  including  $\lambda_s p$  and  $\lambda_d p$ . Then, as stated in the Lemma 1, the maximum detection failure count

$$\mathcal{E} \geq 2^{k+1-|E_{min}|} + 2^{|E_{min}|} - 2,$$

$$\mathcal{E} \geq 2^{k+1-|E_{max}|} + 2^{|E_{max}|} - 2,$$

$$\mathcal{E} \geq 2^{k+1-u} + 2^u - 2$$

where  $u = \max(k+1-|E_{min}|, |E_{max}|)$ . However, when  $|E_{max}| < 2|E_{min}|$ , this inequality turns into equality as stated in the Theorem 1. This is because  $|S_A| = 2$  and  $|S_B| = 1$  when  $|E_{max}| < 2|E_{min}|$ . Then,

$$\mathcal{E} = \max_{\lambda p} (2^{k+1-|E|} + 2^{|E|} - 2)$$

as seen from (5-23).  $2^{k+1-|E|} + 2^{|E|} - 2$  increases as  $|E|$  gets smaller or larger than  $(k+1)/2$ . Thus, the maximum occurs for either  $|E| = |E_{min}|$  or  $|E| = |E_{max}|$ . Then,

$$\mathcal{E} = 2^{k+1-u} + 2^u - 2$$

where  $u = \max(k+1-|E_{min}|, |E_{max}|)$ .

### 5.3.5 Experimental Results

Here, we give some experimental result to illustrate the implications of Lemma 1 and Theorem 1. We take the multiples of two arbitrary primes 131 and 331. We grouped them according to their nonzero bit count  $|E|$  and found the actual number of detection failures generated when they are induced as fault, using the Maple program in Appendix B. This program also finds the size of the sets  $S_A$  and  $S_B$ . Table 5-3 and Table 5-4 give all results for 131 and 331, respectively.

The first column of Table 5-3 contains the multiples of 131, 262, 1048, and 2096 with  $|E| = 3$  nonzero bits. The first column also lists the values of the detection failure count and the pair  $(|S_A|, |S_B|)$  when these multiples are induced as fault. As seen, when the multiples 131, 262, 524, 1048, and 2096 are induced as fault, the detection failure counts are 1072, 1048, 1060, 1054, and 1060, respectively. Also,  $(|S_A|, |S_B|)$  pairs are (2, 8), (2, 4), (2, 6), (2, 5), and (2, 6), respectively. The other columns are organized in the same manner.

Inspection of Table 5-3 leads to the conclusions below.

- The largest numbers of detection failures are obtained when very sparse or very dense multiples are induced as fault.
- The detection failures are not less than the lower bound  $2^{k+1-|E|} + 2^{|E|} - 2$ , given in Chapter 5.3.3.
- $|E_{min}| = 3$ ,  $|E_{max}| = 10$ , and  $\mathcal{E} = 1072$ . (The maximum number of detection failure is 1072 and this maximum is obtain for the fault  $\lambda p = 131$ .)
- As stated in Lemma 1,

$$\mathcal{E} = 1072 \geq 2^{k+1-u} + 2^u - 2 = 1030$$

where  $u = \max(k + 1 - |E_{min}|, |E_{max}|)$ . Additionally,  $|S_A| = 2$  when  $|E_{min}| \leq |E| < 2|E_{min}|$  and  $|S_B| = 1$  when  $|E_{max}| - |E_{min}| < |E| \leq |E_{max}|$ .

- For each  $|S_A|$  and  $|S_B|$  in Table 5-3, the corresponding detection failure in the same table is equal to  $|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B|$ . For example, the first pair in the first column is  $(|S_A|, |S_B|) = (2, 8)$  and the first detection failure in the first column is

$$|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B| \Big|_{\substack{|S_A|=2 \\ |S_B|=8}} = 1072$$

**Table 5-3 The values of the detection failure count,  $S_A$  and  $S_B$  when  $p=131$ ,  $k=12$ , and the faults are the multiples of  $p$  in  $(0, 2^k)$**

| $ E $                                     | 3                                    | 4                          | 5            | 6                                                                  | 7                                            | 8            | 9            | 10   |
|-------------------------------------------|--------------------------------------|----------------------------|--------------|--------------------------------------------------------------------|----------------------------------------------|--------------|--------------|------|
| Multiples of modulus 131 less than $2^k$  | 131<br>262<br>524<br>1048<br>2096    | 393<br>786<br>1572<br>3144 | 1441<br>2882 | 655<br>917<br>1179<br>1310<br>1834<br>2227<br>2358<br>2620<br>3668 | 1703<br>2489<br>3013<br>3275<br>3406<br>3537 | 1965<br>3930 | 2751<br>3799 | 4061 |
| Detection failure count found by software | 1072<br>1048<br>1060<br>1054<br>1060 | 568<br>540<br>554<br>582   | 316<br>316   | 376<br>252<br>316<br>316<br>190<br>376<br>316<br>376<br>314        | 252<br>252<br>316<br>252<br>252<br>316       | 346<br>346   | 568<br>540   | 1054 |
| $2^{k+1- E } + 2^{ E } - 2$               | 1030                                 | 526                        | 286          | 190                                                                | 190                                          | 286          | 526          | 1030 |
| $ S_A $ and $ S_B $ found by software     | 2,8<br>2,4<br>2,6<br>2,5<br>2,6      | 2,4<br>2,2<br>2,3<br>2,5   | 2,2<br>2,2   | 4,2<br>2,2<br>4,1<br>4,1<br>2,1<br>4,2<br>4,1<br>4,2<br>2,3        | 4,1<br>4,1<br>2,2<br>4,1<br>4,1<br>2,2       | 6,1<br>6,1   | 8,1<br>4,1   | 10,1 |
| Mean $ S_A $                              | 2.00                                 | 2.00                       | 2.00         | 3.33                                                               | 3.33                                         | 6.00         | 6.00         | 10.0 |
| Mean $ S_B $                              | 5.80                                 | 3.50                       | 2.00         | 1.67                                                               | 1.33                                         | 1.00         | 1.00         | 1.00 |

As seen, the result produce by the software implementation comply with the theoretical results.

Table 5-4 gives the multiples of the prime 331 and the number for the detection failures generated when they are induced as fault. Inspection of Table 5-4 leads to the conclusions below.

- The detection failures are equal to the lower bound  $2^{k+1-|E|} + 2^{|E|} - 2$ , given in Chapter 5.3.3.
- $|E_{min}| = 5$ ,  $|E_{max}| = 8$ , and  $\mathcal{E} = 286$ .
- $|E_{max}| < 2|E_{min}|$ . Thus, as stated in Theorem 1,

$$\mathcal{E} = 2^{k+1-u} + 2^u - 2 = 286$$

where  $u = \max(k + 1 - |E_{min}|, |E_{max}|)$ . Additionally,  $|S_A| = 2$  and  $|S_B| = 1$  for all modulus multiples.

- As a result, the detection failure count is

$$|S_A|2^{k-|E|} + |S_B|2^{|E|} - |S_A||S_B| = 2^{k+1-|E|} + 2^{|E|} - 2.$$

**Table 5-4 The values of the detection failure count,  $S_A$  and  $S_B$  when  $p=331$ ,  $k=12$ , and the faults are the multiples of  $p$  in  $(0, 2^k)$**

| $ E $                                        | 5                                  | 6                   | 7            | 8            |
|----------------------------------------------|------------------------------------|---------------------|--------------|--------------|
| Multiples of modulus 331<br>less than $2^k$  | 331<br>662<br>1324<br>2317<br>2648 | 993<br>1986<br>3972 | 2979<br>3641 | 1655<br>3310 |
| Detection failure count<br>fount by software | 286<br>286<br>286<br>286<br>286    | 190<br>190<br>190   | 190<br>190   | 286<br>286   |
| $2^{k+1- E } + 2^{ E } - 2$                  | 286                                | 190                 | 190          | 286          |
| $ S_A $ and $ S_B $ found by<br>software     | 2,1<br>2,1<br>2,1<br>2,1<br>2,1    | 2,1<br>2,1<br>2,1   | 2,1<br>2,1   | 2,1<br>2,1   |

## 5.4 Data Word Attacks Using Linear Expressions of Modulus

Let  $\lambda$  and  $\mu$  be two integers such that the linear expression  $\lambda p + \mu > 0$  fits into a  $k$ -bit data word. This means that  $\lambda p + \mu \in \mathbb{Z}_{2^k}$  and

$$\lambda p + \mu = \sum_{i \in F} 2^i$$

for some nonempty set  $F \subseteq \mathbb{Z}_k$ . By flipping the bits of  $x$  in the position  $i \in F$ , the attacker can induce the fault  $(\varepsilon, \delta) = (\lambda p + \mu, 0)$ . Then, the codeword becomes

$$(x \oplus \varepsilon, (x^2 \bmod p) \oplus \delta) = (x \oplus (\lambda p + \mu), x^2 \bmod p) \quad (5-27)$$

Let us split  $x$  into its manipulated part  $\hat{x} = \sum_{i \in F} x_i 2^i$  and its untouched part  $\bar{x} = \sum_{i \notin F} x_i 2^i$  as done in (5-12). Moreover, let us convert the XOR operation into the addition and subtraction as we have done in (5-13). Then, we obtain the fault induced data word as follows.

$$\begin{aligned} x \oplus (\lambda p + \mu) &= \lambda p + \mu - \hat{x} + \bar{x} \\ x \oplus (\lambda p + \mu) &= \lambda p + \mu + x - 2\hat{x} \\ x \oplus (\lambda p + \mu) &= \lambda p + \mu - x + 2\bar{x}. \end{aligned} \quad (5-28)$$

The faulty codeword in (5-27) cannot be detected, if

$$(x \oplus (\lambda p + \mu))^2 \bmod p = x^2 \bmod p$$

$$(\lambda p + \mu - \hat{x} + \bar{x})^2 \bmod p = (\hat{x} + \bar{x})^2 \bmod p$$

After simplifying the above equality, we can find that the fault  $(\varepsilon, \delta) = (\lambda p + \mu, 0)$  cannot be detected, if

$$4(\hat{x} - \xi)(\bar{x} + \xi) = 0 \bmod p \quad (5-29)$$

Where  $\hat{x} = \sum_{i \in F} x_i 2^i$ ,  $\bar{x} = \sum_{i \notin F} x_i 2^i$ , and

$$\xi = 2^{-1} \mu \bmod p \quad (5-30)$$

### 5.4.1 Data Word for which Attack is Undetectable

If  $p$  is prime, the data words  $x$  in the sets  $S_3$  and  $S_4$  defined below constitute all the solution of (5-29), and thus the fault  $\lambda p + \mu = \sum_{i \in F} 2^i$  cannot be detected for  $x \in S_3 \cup S_4$ .

$$\begin{aligned} S_3 &= \bigcup_{(\alpha p + \xi) \in S_C} \{x \in \mathbb{Z}_{2^k} \mid \sum_{i \in F} x_i 2^i = \alpha p + \xi\} \\ S_4 &= \bigcup_{(\beta p - \xi) \in S_D} \{x \in \mathbb{Z}_{2^k} \mid \sum_{i \notin F} x_i 2^i = \beta p - \xi\} \end{aligned} \quad (5-31)$$

where

$$\begin{aligned} S_C &= \{\alpha p + \xi \mid \alpha p + \xi = \sum_{i \in A} 2^i \text{ for } \exists A \subseteq F\} \\ S_D &= \{\beta p - \xi \mid \beta p - \xi = \sum_{i \in B} 2^i \text{ for } \exists B \subseteq (\mathbb{Z}_k \setminus F)\} \end{aligned} \quad (5-32)$$

#### 5.4.1.1 Data words Satisfying $x \oplus (\lambda p + \mu) = x + wp$

Since  $\hat{x} = \sum_{i \in F} x_i 2^i = \alpha p + \xi$  when  $x \in S_3$ , it follows from (5-28) that the fault induced data word

$$\begin{aligned} x \oplus (\lambda p + \mu) &= \lambda p + \mu + x - 2\hat{x} \\ x \oplus (\lambda p + \mu) &= \lambda p + \mu + x - 2(\alpha p + \xi) \\ x \oplus (\lambda p + \mu) &= x + (\lambda p - 2\alpha)p + \mu - 2\xi \end{aligned}$$

and it follows from (5-30) that  $\mu - 2\xi = 0 \pmod p$ . Thus,

$$x \oplus (\lambda p + \mu) = x + wp$$

for some  $w$  and XOR operation becomes addition with a modulus multiple. In fact, this is why quadratic code fails when  $x \in S_3$ . Also, due to (5-31) and (5-32),

$$|S_3| = |S_C| 2^{k-|F|} \quad (5-33)$$

#### 5.4.1.2 Data words Satisfying $x \oplus (\lambda p + \mu) = wp - x$

Since  $\bar{x} = \sum_{i \notin F} x_i 2^i = \beta p - \xi$  when  $x \in S_4$ , it follows from (5-28) that the fault induced data word

$$x \oplus (\lambda p + \mu) = \lambda p + \mu - x - 2\bar{x}$$

$$x \oplus (\lambda p + \mu) = \lambda p + \mu + x - 2(\beta p - \xi)$$

$$x \oplus (\lambda p + \mu) = x + (\lambda p + 2\beta)p + \mu - 2\xi$$

and it follows from (5-30) that  $\mu - 2\xi = 0 \pmod p$ . Thus,

$$x \oplus (\lambda p + \mu) = wp - x$$

for some  $w$  and XOR operation becomes addition with a modulus multiple. In fact, this is why quadratic code fails when  $x \in S_4$ . Also, due to (5-31) and (5-32),

$$|S_4| = |S_D|2^{|F|} \quad (5-34)$$

#### 5.4.2 Exact Count of Detection Failures

Let the linear expression  $\lambda p + \mu = \sum_{i \in F} 2^i$  for nonempty set  $F \subseteq \mathbb{Z}_k$ . If  $p$  is prime, the fault  $(\varepsilon, \delta) = (\lambda p + \mu, 0)$  cannot be detected exactly for

$$|S_C|2^{k-|F|} + |S_D|2^{|F|} - |S_C||S_D| \quad (5-35)$$

Distinct values of the data word  $x$  where  $S_C$  and  $S_D$  are given by (5-32). This count is equal to  $|S_3 \cup S_4|$ .  $|S_3| = |S_C|2^{k-|F|}$  and  $|S_4| = |S_D|2^{|F|}$  as seen from (5-33) and (5-34), respectively. Also,  $|S_3 \cap S_4| = |S_C||S_D|$  since  $S_3 \cap S_4$  is

$$\bigcup_{\substack{(\alpha p + \xi) \in S_C \\ (\beta p - \xi) \in S_D}} \left\{ x \mid \sum_{i \in F} x_i 2^i = \alpha p + \xi, \sum_{i \notin F} x_i 2^i = \beta p - \xi \right\} \quad (5-36)$$

Let us illustrate how to calculate the number of the detection failures using (5-35) with an example. Let the data word  $x \in \mathbb{Z}_{2^k}$  be encoded into the codeword  $(x, x^2 \pmod p)$  for  $k = 12$  and  $p = 47$ . Let the attacker induced the fault

$$(\varepsilon, \delta) = (\lambda p + \mu, 0) = (53p + 10, 0) = (2501, 0)$$

in the codeword. For this, the attacker flips the data bits in the positions corresponding to the nonzero bits of

$$(\lambda p + \mu) = 2501 = (1\ 0\ 0\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 0\ 1)_2$$

without touching the redundancy bits. As seen above,  $\lambda p + \mu = \sum_{i \in F} 2^i$  where  $F = \{0, 2, 6, 7, 8, 11\} \subseteq \mathbb{Z}_k$ . According to (5-30),

$$\xi = 2^{-1}\mu \bmod p = (2^{-1})10 \bmod 47 = 5$$

Table 5-5 and Table 5-6 give the linear expression  $\alpha p + \xi$  and  $\beta p - \xi$ , the position of whose nonzero bits are contained in the set  $F$  and the set  $(\mathbb{Z}_k \setminus F)$ , respectively. From these tables, we find  $S_C$  and  $S_D$  defined in (5-32) as follows.

$$S_C = \{0p + 5, 4p + 5, 49p + 5, 53p + 5\}$$

$$S_D = \{p - 5, 11p - 5, 33p - 5\}$$

Then,  $|S_C| = 4, |S_D| = 3, |F| = 6$ , and Equation (5-35) yields the result  $4 \cdot 2^{12-6} + 3 \cdot 2^6 - 4 \cdot 3 = 436$ . Also the probability of not detecting the fault is calculated as

$$Q(\varepsilon, \delta) = Q(\lambda p, 0) = 436/2^{12} = 0.106$$

When all the values of  $x \in \mathbb{Z}_{2^{12}}$  are equally likely.

We check the result of this example with a Maple program given in Appendix A, as well. The number of detection failures turns out to be indeed 436 when the faults  $(\varepsilon, \delta) = (53p + 10, 0) = (2501, 0)$ . Note that we find 436 for the case of  $\delta = 0$  (no fault is induced in the redundancy). Our test program also investigates the case of  $\delta \neq 0$ . The number of the detection failures for the nonzero  $\delta$  values can be found in Appendix A. These numbers are significantly less than 436.

**Table 5-5 Linear expressions  $(\alpha p + \xi) = (\alpha 47 + 5)$  whose nonzero bit positions  $i \in F = \{0, 2, 6, 7, 8, 11\}$ .**

| Bit No      | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|----|----|---|---|---|---|---|---|---|---|---|---|
| $53p + 5 =$ | 1  | 0  | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| $49p + 5 =$ | 1  | 0  | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $4p + 5 =$  | 0  | 0  | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| $0p + 5 =$  | 0  | 0  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

**Table 5-6 Linear expressions  $(\beta p - \xi) = (\beta 47 - 5)$  whose nonzero bit positions  $i \notin F = \{0, 2, 6, 7, 8, 11\}$ .**

| Bit No      | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|-------------|----|----|---|---|---|---|---|---|---|---|---|---|
| $33p - 5 =$ | 0  | 1  | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $11p - 5 =$ | 0  | 0  | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| $p - 5 =$   | 0  | 0  | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

### 5.4.3 Modulus Multiples in Redundant Binary Forms

*Definition 4:* For a given set of bits  $F \subseteq \mathbb{Z}_k$ ,

$$S_\Phi = \{\phi p \mid \phi p = \sum_{i \in F \setminus A} 2^i - \sum_{i \in A} 2^i = \sum_{i \in F} \pm 2^i, \text{ for } \exists A \subseteq F\} \quad (5-37)$$

The set  $S_\Phi$  contains all the multiples of  $p$  whose signed digit representation has  $\pm 1$ 's in the bit positions  $i \in F$ .  $A \subseteq F$  contains -1's and  $F \setminus A$  contains +1's.

*Definition 5:* For a given set of bits  $F \subseteq \mathbb{Z}_k$ ,

$$S_\Theta = \{\theta p \mid \theta p = \sum_{i \in F} 2^i + 2 \sum_{i \in B} 2^i \text{ for } \exists B \subseteq (\mathbb{Z}_k \setminus F)\} \quad (5-38)$$

The set  $S_\Theta$  contains all the multiples of  $p$  whose redundant binary representation with the digit set  $\{0, 1, 2\}$  has 1's in the bit positions  $i \in F$  and 2's in the bit positions  $i \in B \subseteq (\mathbb{Z}_k \setminus F)$ .

*Theorem 2:* The set  $S_C$  and  $S_D$  given by (5-32) are respectively in one to one correspondence with the set  $S_\Phi$  in (5-37) and the set  $S_\Theta$  in (5-38). Thus,

$$|S_C| = |S_\Phi|, \quad |S_D| = |S_\Theta| \quad (5-39)$$

*Proof:* Notice that  $\mu = 2^{-1}\xi \bmod p$  due to (5-30). Thus,

$$\lambda p + \mu - 2(\alpha p + \xi) \equiv 0 \bmod p,$$

$$\lambda p + \mu - 2(\beta p - \xi) \equiv 0 \bmod p.$$

Then, each  $\alpha p + \xi$  in the set  $S_C$  in (5-32) maps into a unique

$$\phi p = \lambda p + \mu - 2(\alpha p + \xi)$$

$$\phi p = \sum_{i \in F} 2^i - 2 \sum_{i \in A \subseteq F} 2^i$$

$$\phi p = \sum_{i \in F \setminus A} 2^i - \sum_{i \in A \subseteq F} 2^i$$

in the set  $S_\Phi$ . Similarly, each  $\beta p - \xi$  in the set  $S_D$  in (5-32) maps into a unique modulus multiple

$$\theta p = \lambda p + \mu + 2(\beta p - \xi)$$

$$\theta p = \sum_{i \in F} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$$

in the set  $S_\Theta$ .

*Example 1:* Let us find  $S_\Phi$  for the example in Table 5-5.

$$\sum_{i \in F} 2^i = 53p + 10$$

$$\sum_{A \subseteq F} 2^i \in \{0p + 5, 4p + 5, 49p + 5, 53p + 5\} = S_C,$$

$$\sum_{F \setminus A} 2^i \in \{53p + 5, 49p + 5, 4p + 5, 0p + 5\} = S_C,$$

$$\phi p \in \{+53p, +45p, -45p, +53p\} = S_\Phi$$

*Example 2:* Let us find  $S_\theta$  for the example in Table 5-6.

$$\sum_{i \in F} 2^i = 53p + 10$$

$$\sum_{B \subseteq (\mathbb{Z}_k \setminus F)} 2^i \in \{p - 5, 11p - 5, 33p - 5\} = S_D$$

$$\theta p \in \{55p, 75p, 119p\} = S_\theta$$

#### 5.4.4 Basic Attack Strategy

A simple way of generating large number of detection failures is to choose the fault  $\lambda p + \mu = \sum_{i \in F} 2^i$  such that the flipped bit set  $F$  is very small or very large ( $\lambda p + \mu$  is very sparse or very dense). Then, the exponential terms  $2^{k-|F|}$  and  $2^{|F|}$  blow up the detection failure count in (5-35). However, the attacker has to guarantee that either the set  $S_\Phi \neq \emptyset$  or the set  $S_\theta \neq \emptyset$  while choosing the set  $F$ . This is because the detection failure count in (5-35)

$$|S_C|2^{k-|F|} + |S_D|2^{|F|} - |S_C||S_D| = |S_\Phi|2^{k-|F|} + |S_\theta|2^{|F|} - |S_\Phi||S_\theta| \quad (5-40)$$

as seen from (5-39). Thus the attacker must adopt one of the following two methods.

- 1) He finds a modulus multiple whose representation in base 2 with digits -1, 0, 1 has a large number of 0's. That is, he finds a modulus multiple

$$\phi p = \sum_{i \in F \setminus A} 2^i - \sum_{i \in A \subseteq F} 2^i = \sum_{i \in F} \pm 2^i$$

for a very small  $F$ . Then he flips bits in  $F$ .

- 2) He finds a modulus multiple whose representation in base 2 with digits 0, 1, 2 has a large number of 1's. That is, he finds a modulus multiple

$$\theta p = \sum_{i \in F} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$$

for a very small  $F$ . Then he flips bits in  $F$ .

In the first method, the attacker guarantees that

$$S_\Phi \ni \{\pm \phi p = \pm (\sum_{i \in F \setminus A} 2^i) \mp (\sum_{i \in A \subseteq F} 2^i)\}$$

as can be understood from the definition of  $S_\Phi$  in (5-37). Then,  $|S_\Phi| \geq 2$ ,  $|S_\theta| \geq 0$ , and thus the detection failure

$$|S_\Phi|2^{k-|F|} + |S_\Theta|2^{|F|} - |S_\Phi||S_\Theta| \geq |S_\Phi|2^{k-|F|} = 2^{k+1-|F|} \quad (5-41)$$

Here,  $2^{k+1-|F|}$  is very large since  $F$  chosen very small.

In the second method, the attacker guarantees that

$$|S_\Theta| \supseteq \{\theta p = \sum_{i \in F} 2^i + 2 \sum_{i \in B} 2^i\}$$

as can be understood from the definition of  $S_\Theta$  in (5-38). Then,  $|S_\Phi| \geq 0$ ,  $|S_\Theta| \geq 1$ , and thus the detection failure

$$|S_\Phi|2^{k-|F|} + |S_\Theta|2^{|F|} - |S_\Phi||S_\Theta| \geq |S_\Theta|2^{|F|} \geq 2^{|F|}. \quad (5-42)$$

Here,  $2^{|F|}$  is very large since  $F$  chosen very large.

#### 5.4.5 Maximum Number of Detection Failures

*Definition 6:* Let the fault  $\lambda p + \mu = \sum_{i \in F} 2^i$  be induced in an attack for some flipped bit set  $F \subseteq \mathbb{Z}_k$ .

$$\mathcal{F} = \max_{\lambda p + \mu} (|S_C|2^{k-|F|} + |S_D|2^{|F|} - |S_C||S_D|) \quad (5-43)$$

denotes the maximum number of detection failures.

As can be understood from the chapter 5.4.3, the maximum number of detection failures  $\mathcal{F}$  has a relationship with the redundant binary representations of the modulus multiples with large number of 0's and 1's. Now, we show this relationship.

*Definition 7:*  $\phi_s p$  denotes the modulus multiples whose k-digit representation in base 2 with digits -1, 0, 1 has the largest number of 0's.  $F_{min} \subseteq \mathbb{Z}_k$  denotes the set of  $\pm 1$ 's in the representation of  $\phi_s p$ . That is,

$$1 \leq |F_{min}| \leq |F| \leq k$$

for any  $\phi p = \sum_{i \in F \subseteq \mathbb{Z}_k} \pm 2^i$ . ( $\phi_s p$  may not be unique.)

*Definition 8:*  $\theta_d p$  denotes the modulus multiples whose  $k$ -digit representation in base 2 with digits 0, 1, 2 has the largest number of 1's.  $F_{max} \subseteq \mathbb{Z}_k$  denotes the set of 1's in the representation of  $\theta_d p$ . That is,

$$1 \leq |F| \leq |F_{max}| \leq k$$

for any  $\theta p = \sum_{i \in F \subseteq \mathbb{Z}_k} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$ . ( $\theta_d p$  may not be unique.)

*Lemma 2:* The set sizes used in the detection failure count in (5-20) are as follows.  $|S_C| = 2$ ,  $|S_D| \geq 1$  when  $|F_{min}| \leq |F| < 2|F_{min}|$  and  $|S_C| \geq 2$ ,  $|S_D| = 1$  when  $|F_{max}| - |F_{min}| < |F| \leq |F_{max}|$ . However, the following exceptions occur depending on the modulus  $p$  and the set of flipped bits  $F \subseteq \mathbb{Z}_k$ .

- $|S_C| = 0$ , if some  $\phi p = \sum_{i \in F \subseteq \mathbb{Z}_k} \pm 2^i$  does not exist.
- $|S_D| = 0$ , if some  $\theta p = \sum_{i \in F \subseteq \mathbb{Z}_k} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$  does not exist.

Also, the maximum detection failure count

$$\mathcal{F} \geq \psi 2^{k+1-v} + 2^v - 2\psi \quad (5-44)$$

where  $\psi \in \{0, 1\}$  and  $v = \max(k+1 - |F_{min}|, |F_{max}|)$ .

*Theorem 3:* Let the modulus  $p$  be chosen such that  $|F_{max}| < 2|F_{min}|$  hold for the multiples of  $p$ . Let the following two cases be defined for the given modulus  $p$  and the set of flipped bits  $F \subseteq \mathbb{Z}_k$ .

Case 1: some  $\phi p = \sum_{i \in F \subseteq \mathbb{Z}_k} \pm 2^i$  exist.

Case 2: Some  $\theta p = \sum_{i \in F \subseteq \mathbb{Z}_k} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$  exist.

$|S_C|$ ,  $|S_D|$ , and the detection failure counts as follows.

- If Case 1 holds,  $|S_C| = 2$ ,  $|S_D| = \psi$ , and the detection failure count is

$$2^{k+1-|F|} + \psi 2^{|F|} - 2\psi \quad (5-45)$$

where  $\psi \in \{0, 1\}$  and  $\psi \neq 0$  only if Case 2 also holds.

- If Case 2 holds,  $|S_C| = 2\psi$ ,  $|S_D| = 1$ , and the detection failure count is

$$\psi 2^{k+1-|F|} + 2^{|F|} - 2\psi \quad (5-46)$$

where  $\psi \in \{0, 1\}$  and  $\psi \neq 0$  only if Case 1 also holds.

- Otherwise,  $|S_C|$ ,  $|S_D|$ , and the detection failure count are all zero.

The detection failure count reaches the maximum  $\mathcal{F}$  when the flipped bit set either  $F = F_{min}$  or  $F = F_{max}$ . Moreover,

$$\mathcal{F} = \psi 2^{k+1-v} + 2^v - 2\psi \quad (5-47)$$

where  $\psi \in \{0, 1\}$  and  $v = \max(k + 1 - |F_{min}|, |F_{max}|)$ .

*Proof:* There the following two important case for the given modulus  $p$  and set of flipped bit  $F \subseteq \mathbb{Z}_k$ .

Case 1: some  $\phi p = \sum_{i \in F \subseteq \mathbb{Z}_k} \pm 2^i$  exist.

Case 2: Some  $\theta p = \sum_{i \in F \subseteq \mathbb{Z}_k} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$  exist.

If Case 1 holds,

$$\begin{aligned} S_\Phi &= \{\pm \phi_j p = \pm (\sum_{i \in F \setminus A_j} 2^i) \mp (\sum_{i \in A_j \subseteq F} 2^i)\} \\ S_\Phi &= \{+\phi_j p = \sum_{i \in F} 2^i, -\phi_j p = \sum_{i \in F} \pm 2^i\} \end{aligned} \quad (5-48)$$

for  $j = 0, 1, 2, \dots$  as seen from (5-37). If Case 2 holds,

$$S_\Theta = \{\theta_j = \sum_{i \in F} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i\} \quad (5-49)$$

for  $j = 0, 1, 2, \dots$  as seen from (5-38). Note that, the set sizes  $|S_C|$  and  $|S_D|$  mentioned in the Lemma 2 and Theorem 3 always satisfy  $|S_C| = |S_\Phi|$  and  $|S_D| = |S_\Theta|$  due to (5-39).

Let Case 1 hold. It can be shown that  $|S_\Phi| = 2$  when  $|F_{min}| \leq |F| < 2|F_{min}|$ . That is  $S_\Phi = \{\pm \phi p\}$  for some  $\phi p$ . If  $|S_\Phi| \geq 2$ , we would have  $S_\Phi \supseteq \{\pm \phi_0 p, \pm \phi_1 p\}$ . Then,

$$\begin{aligned} \phi_0 p + \phi_1 p &= \sum_{i \in F \setminus A_0} 2^i - \sum_{i \in A_0} 2^i + \sum_{i \in F \setminus A_1} 2^i - \sum_{i \in A_1} 2^i \\ &= 2 \sum_{i \in (F \setminus A_0) \cap (F \setminus A_1)} 2^i - \sum_{i \in (A_0 \cap A_1)} 2^i \end{aligned}$$

$$\begin{aligned}
\phi_0 p - \phi_1 p &= \sum_{i \in F \setminus A_0} 2^i - \sum_{i \in A_0} 2^i - \sum_{i \in F \setminus A_1} 2^i + \sum_{i \in A_1} 2^i \\
&= 2 \sum_{i \in (F \setminus A_0) \cap A_1} 2^i - \sum_{i \in A_0 \cap (F \setminus A_1)} 2^i.
\end{aligned}$$

Note that  $(\phi_0 + \phi_1)p/2 = \sum_{i \in \hat{F}} \pm 2^i$  where

$$\hat{F} = ((F \setminus A_0) \cap (F \setminus A_1)) \cup (A_0 \cap A_1)$$

and  $(\phi_0 - \phi_1)p/2 = \sum_{i \in \tilde{F}} \pm 2^i$  where

$$\tilde{F} = ((F \setminus A_0) \cap A_1) \cup (A_0 \cap (F \setminus A_1)).$$

As seen, the sets  $F, \hat{F}$ , and  $\tilde{F}$  give the nonzero digits of modulus multiple  $\phi_0 p, (\phi_0 + \phi_1)p/2$ , and  $(\phi_0 - \phi_1)p/2$  respectively. These sets cannot be smaller than  $F_{min}$ , which gives the nonzero digits of the sparsest modulus multiple  $\phi_s p$ . Also,  $\hat{F} \cup \tilde{F} = F$  and  $\hat{F} \cap \tilde{F} = \emptyset$ . Then,

$$2|F_{min}| \leq |\hat{F}| + |\tilde{F}| = |\hat{F} \cup \tilde{F}| = |F|.$$

Our initial assumption  $|F_{min}| \leq |F| < 2|F_{min}|$  contradicts with this result.

Let Case 2 hold. It can be shown that  $|S_\Theta| = 1$  when  $|F_{max}| - |F_{min}| < |F| \leq |F_{max}|$ . That is  $S_\Theta = \{\theta p\}$  for some  $\theta p$ .  $|S_\Theta| \geq 1$ , we would have  $S_\Theta \supseteq \{\theta_0 p, \theta_1 p\}$  for some  $\theta_0 p$  and  $\theta_1 p$ . Then,

$$\begin{aligned}
\theta_0 p + \theta_1 p &= 2 \sum_{i \in F} 2^i + 2 \sum_{i \in B_0} 2^i + 2 \sum_{i \in B_1} 2^i \\
&= 2 \sum_{i \in F \cup (B_0 \setminus B_1) \cup (B_1 \setminus B_0)} 2^i + 4 \sum_{i \in (B_0 \cap B_1)} 2^i
\end{aligned}$$

$$\begin{aligned}
\theta_0 p - \theta_1 p &= 2 \sum_{i \in B_0} 2^i - 2 \sum_{i \in B_1} 2^i, \\
&= 2 \sum_{i \in (B_0 \setminus B_1)} 2^i - 2 \sum_{i \in (B_1 \setminus B_0)} 2^i, \\
&= 2 \sum_{i \in (B_0 \setminus B_1) \cup (B_1 \setminus B_0)} \pm 2^i.
\end{aligned}$$

where  $B_0 \subseteq (\mathbb{Z}_k \setminus F)$  and  $B_1 \subseteq (\mathbb{Z}_k \setminus F)$ . Note that

$$\begin{aligned}
(\theta_0 + \theta_1)p/2 &= 2 \sum_{i \in F \cup \hat{F}} 2^i - 2 \sum_{i \in B_0 \cap B_1} 2^i, \\
(\theta_0 - \theta_1)p/2 &= \sum_{i \in \hat{F}} \pm 2^i.
\end{aligned}$$

where  $\hat{F} = (B_0 \setminus B_1) \cup (B_1 \setminus B_0)$ . Thus,  $(\theta_0 + \theta_1)p/2$  is in base 2 with digits 0, 1, 2 and the set  $F \cup \hat{F}$  gives 1's in this representation while the set  $B_0 \cap B_1$  gives 2's. Then,  $|F \cup \hat{F}| \leq |F_{max}|$ . Again, note that  $(\theta_0 - \theta_1)p/2$  is base 2 with digits -1, 0, 1 and the set  $\hat{F}$  gives  $\pm 1$ 's in this representation. Then  $|F_{min}| \leq |\hat{F}|$ . Also, since  $B_0, B_1 \subseteq (\mathbb{Z}_k \setminus F)$ , we have always  $F \cup \hat{F} = \emptyset$ . Thus,

$$|F| = |F \cup \hat{F}| - |\hat{F}| \leq |F_{max}| - |F_{min}|$$

Our initial assumption  $|F_{max}| - |F_{min}| < |F| \leq |F_{max}|$  contradicts with this result.

In summary, as stated in the Lemma 2,  $|S_C| = |S_\Phi| = 2$  when  $|F_{min}| \leq |F| < 2|F_{min}|$  and  $|S_D| = |S_\Theta| = 1$  when  $|F_{max}| - |F_{min}| < |F| \leq |F_{max}|$ . However, these are true when Case 1 and Case 2 hold. If Case 1 and Case 2 do not hold,  $S_\Phi = \emptyset$  and  $S_\Theta = \emptyset$ . Then,  $|S_C| = |S_\Phi| = 0$  and  $|S_D| = |S_\Theta| = 0$ .

Let  $\psi_1, \psi_2 \in \{0, 1\}$  be defined for the given set of flipped bits F as follows.

$$\psi_1 = \begin{cases} 1, & \text{if Case 1 Holds,} \\ 0, & \text{otherwise.} \end{cases} \quad \psi_2 = \begin{cases} 1, & \text{if Case 2 Holds,} \\ 0, & \text{otherwise.} \end{cases}$$

$|S_C| = |S_\Phi| \geq 2\psi_1$  due to (5-48) and  $|S_D| = |S_\Theta| \geq \psi_2$  due to (5-49). Thus, the detection failure count

$$|S_C|2^{k-|F|} + |S_D|2^{|F|} - |S_C||S_D| \geq \psi_1 2^{k+1-|F|} + \psi_2 2^{|F|} - 2\psi_1\psi_2.$$

When  $F = F_{min}$ ,  $S_\Phi \supseteq \{\pm\phi_s p\}$ , and thus  $|S_C| = |S_\Phi| \geq 2$ . When  $F = F_{max}$ ,  $S_\Theta \supseteq \{\pm\theta_d p\}$ , and thus  $|S_D| = |S_\Theta| \geq 1$ . Then, as stated in the Lemma 2, the maximum detection failure count

$$\begin{aligned}\mathcal{F} &\geq 2^{k+1-|F_{min}|} + \psi_s 2^{|F_{min}|} - 2\psi_s, \\ \mathcal{F} &\geq \psi_d 2^{k+1-|F_{max}|} + 2^{|F_{max}|} - 2\psi_d, \\ \mathcal{F} &\geq \psi 2^{k+1-v} + 2^v - 2\psi\end{aligned}$$

where  $v = \max(k+1 - |F_{min}|, |F_{max}|)$ , also  $\psi_s, \psi_d$ , and  $\psi$  are some integers in  $\{0, 1\}$ .

Note that  $|F_{max}| < 2|F_{min}|$  implies that

$$|F_{max}| - |F_{min}| < |F_{min}| \leq |F| \leq |F_{max}| < 2|F_{min}|$$

Thus,  $|S_C| = |S_\Phi| \geq 2\psi_1$ ,  $|S_D| = |S_\Theta| \geq \psi_2$ , and the detection failure count is

$$\psi_1 2^{k+1-|F|} + \psi_2 2^{|F|} - 2\psi_1 \psi_2$$

when  $|F_{max}| < 2|F_{min}|$ . This result is in accordance with the theorem. Note that Equation (5-43) now becomes

$$\mathcal{F} = \max_{\lambda p + \mu} (\psi_1 2^{k+1-|F|} + \psi_2 2^{|F|} - \psi_1 \psi_2)$$

where  $v = \max(k+1 - |F_{min}|, |F_{max}|)$ .  $2^{k+1-|F|} + 2^{|F|} - 2$  increases as  $|F|$  gets smaller or larger than  $(k+1)/2$ . Thus, the maximum  $\mathcal{F}$  occurs for either  $|F| = |F_{min}|$  or  $|F| = |F_{max}|$ . That is the maximum detection failure count is on of the following values.

$$\begin{aligned}2^{k+1-|F_{min}|} + \psi_s 2^{|F_{min}|} - 2\psi_s, \\ \psi_d 2^{k+1-|F_{max}|} + 2^{|F_{max}|} - 2\psi_d.\end{aligned}$$

Then, as stated in the theorem,

$$\mathcal{F} = \psi 2^{k+1-v} + 2^v - 2\psi$$

where  $\psi \in \{0, 1\}$  and  $v = \max(k+1 - |F_{min}|, |F_{max}|)$ .

## 5.5 Estimated Count of Detection Failures

### 5.5.1 For Multiples of Modulus

$|S_A|$  and  $|S_B|$  can be estimated in terms of the nonzero bit count  $|E|$  of the fault  $\lambda_p = \sum_{i \in E} 2^i \in \mathbb{Z}_{2^k}$  as follows.

$$\begin{aligned} |S_A| &\approx \max\left(\frac{2^{|E|}}{p}, 2\right) = \frac{\max(2^{|E|}, 2p)}{p} \\ |S_B| &\approx \max\left(\frac{2^{k-|E|}}{p}, 1\right) = \frac{\max(2^{|E|}, 2^k/p)}{2^{|E|}} \end{aligned} \quad (5-50)$$

We can explain this as follows.  $S_A$  and  $S_B$  consist of integers whose nonzero bits are in the bit position  $E$  and  $\mathbb{Z}_k \setminus E$ , respectively as can be understood from (5-16). Thus,  $|S_A| < 2^{|E|}$  and  $|S_B| < 2^{k-|E|}$ . Additionally, the integers in these sets must be the multiples of the modulus  $p$ . If we assume that the numbers in the sets are divisible by  $p$  with probability  $1/p$ ,  $|S_A| \approx 2^{|E|}/p$  and  $|S_B| \approx 2^{k-|E|}/p$ . Moreover, we must have  $|S_A| \geq 2$  and  $|S_B| \geq 1$  due to (5-17).

After using the estimation for  $|S_A|$  and  $|S_B|$ , detection failure count in (5-20) can be estimated as

$$\frac{\max(2^{|E|}, 2p)}{p} 2^{k-|E|} + \frac{\max(2^{|E|}, 2^k/p)}{2^{|E|}} - \frac{\max(2^{|E|}, 2p) \cdot \max(2^{|E|}, 2^k/p)}{p \cdot 2^{|E|}}$$

The estimation count becomes

$$\begin{cases} \frac{p-1}{p} 2^{k+1-|E|} + \frac{2^k}{p} & , \quad 2 \leq 2^{|E|} < 2^k/p \\ 2^{k+1-|E|} + 2^{|E|-2} & , \quad 2^k/p < 2^{|E|} < 2p \\ \frac{p-1}{p} 2^{|E|} + \frac{2^k}{p} & , \quad 2p < 2^{|E|} \leq 2^k \end{cases} \quad (5-51)$$

When  $2p > 2^k/p$  (large  $p > 2^{\frac{k-1}{2}}$ ) and

$$\begin{cases} \frac{p-1}{p} 2^{k+1-|E|} + \frac{2^k}{p} & , \quad 2 \leq 2^{|E|} < 2p \\ \frac{(2p-1)2^k}{p^2} & , \quad 2p < 2^{|E|} < 2^k/p \\ \frac{p-1}{p} 2^{|E|} + \frac{2^k}{p} & , \quad 2^k/p < 2^{|E|} \leq 2^k \end{cases} \quad (5-52)$$

When  $2p < 2^k/p$  (small  $p < 2^{\frac{k-1}{2}}$ ). The estimation for detection failure

count in (5-20) drawn as a function of  $|E|$  in Figure 5-1 and Figure 5-2 for large and small  $p$  respectively.

The following observations can be made for the piecewise functions giving the estimated count.

- The left wing of the piecewise functions decrease as  $|E|$  increases because of term  $2^{k+1-|E|}$ .
- The right wings of the piecewise functions increase as  $|E|$  increases because of term  $2^{|E|}$ .
- When  $p$  is large, the middle part of the piecewise function makes a minimum at  $|E| = (k+1)/2$  because of the term  $2^{k+1-|E|} + 2^{|E|}$ . However, when  $p$  is small, the middle part saturates for  $2^{|E|} \in [2p, 2^k/p]$ .

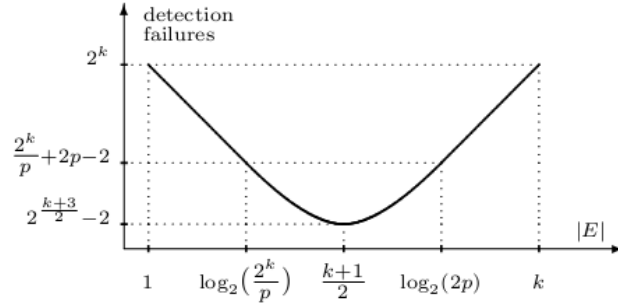


Figure 5-1 Estimated number of detection failures for large  $p$

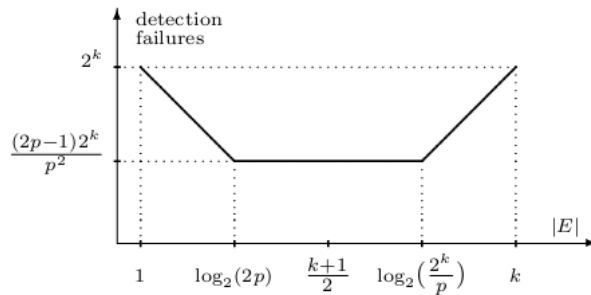


Figure 5-2 Estimated number of detection failures for small  $p$

### 5.5.2 For Linear Expressions of Modulus

The size of the sets in (5-37) and (5-38) can be estimated in terms of the nonzero bit count  $|F|$  of the fault  $\lambda p + \mu = \sum_{i \in E} 2^i \in \mathbb{Z}_{2^k}$  as follows.

$$|S_\Phi| \approx 2^{|F|}/p, \quad |S_\Theta| \approx 2^{k-|F|}/p \quad (5-53)$$

Let us explain this.  $|S_\Phi| \leq 2^{|F|}$  because  $S_\Phi$  is the set of integers whose signed digit representations have  $\pm 1$ 's in the bit position  $i \in F$  and 0's in the bit position  $i \notin F$ . Also,  $|S_\Theta| \leq 2^{k-|F|}$  because  $S_\Theta$  is the set of integers whose redundant binary representation with the digit set  $\{0, 1, 2\}$  has 0's and 2's in the bit positions  $i \notin F$ , and 1's in the bit positions  $i \in F$ . Additionally, the integers in these sets must be the multiple of the modulus  $p$ . If we assume that the numbers in the sets are divisible by  $p$  with probability  $1/p$ , we obtain  $|S_\Phi| \approx 2^{|F|}/p$  and  $|S_\Theta| \approx 2^{k-|F|}/p$ .

#### 5.5.2.1 Attack Ensuring Nonempty $S_\Phi$

Let the attacker induce the fault  $\lambda p + \mu = \sum_{i \in F} 2^i$ , only if there exists a modulus multiple such that

$$\phi p = \sum_{i \in F \subseteq \mathbb{Z}_k} \pm 2^i = \sum_{i \in F \setminus A} 2^i - \sum_{i \in A \subseteq F} 2^i$$

Then the attacker ensures that  $S_\Phi \ni \{\pm \phi p\}$  and  $|S_\Phi| \geq 2$  for  $S_\Phi \ni \{+\phi p, -\phi p\}$ ,  $S_\Theta \ni \{\emptyset\}$ . Consequently, (5-53) becomes

$$|S_\Phi| \approx (2^{|F|}/p, 2), \quad |S_\Theta| \approx 2^{k-|F|}/p$$

and the number of the detection failures in (5-35) (with the equality (5-39)) becomes

$$|S_\Phi| 2^{k-|F|} + |S_\Theta| 2^{|F|} - |S_\Phi| |S_\Theta| \approx \frac{p-1}{p} \max\left(\frac{2^k}{p}, 2^{k+1-|F|}\right) + \frac{2^k}{p}.$$

As seen above, the detection failure count is maximum when  $|F|$  is minimum. Thus, the attacker must keep the manipulated bit set  $F \subseteq \mathbb{Z}_k$  as small as possible.

Then, the best way to maximize the detection failures is to induce the fault  $\sum_{i \in F} 2^i$  where  $F_{min}$  is the smallest set such that

$$\phi_s p = \sum_{i \in F_{min}} \pm 2^i = \sum_{i \in F_{min} \setminus A} 2^i - \sum_{i \in A \subseteq F_{min}} 2^i.$$

Notice that  $\phi_s p$  is the modulus multiples whose signed digit representation has the largest number of zeros (sparsest).

Analyzing this special case, we present a lower bound and an estimate for the maximum of the detection failure count in (5-35) (with the equality (5-39)) under the constraint  $S_\Phi \neq \emptyset$ . Let this maximum be denoted by

$$\mathcal{F}_\phi = \max_{S_\Phi \neq \emptyset} (|S_\Phi|2^{k-|F|} + |S_\Theta|2^{|F|} - |S_\Phi||S_\Theta|).$$

The lower bound and the estimate are follows.

$$\mathcal{F}_\phi \geq 2^{k+1-|F_{min}|}, \quad \mathcal{F}_\phi \approx \frac{p-1}{p} 2^{k+1-|F_{min}|} + \frac{2^k}{p} \quad (5-54)$$

### 5.5.2.2 Attack Ensuring Nonempty $S_\Theta$

Let the attacker induce that the fault  $\lambda p + \mu = \sum_{i \in F} 2^i$  only if there exists a modulus multiple such that

$$\theta p = \sum_{i \in F} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F)} 2^i$$

Then the attacker ensures that  $S_\Theta \supseteq \{\theta p\}$  and  $|S_\Theta| \geq 1$  or  $S_\Theta \supseteq \{\theta p\}$ ,  $S_\Theta \supseteq \{\emptyset\}$ . Consequently, (5-53) becomes

$$|S_\Phi| \approx 2^{|F|}/p, \quad |S_\Theta| \approx \max(2^{k-|F|}/p, 1)$$

and the number of the detection failures in (5-35) (with the equality (5-39)) becomes

$$|S_\Phi|2^{k-|F|} + |S_\Theta|2^{|F|} - |S_\Phi||S_\Theta| \approx \frac{p-1}{p} \max\left(\frac{2^k}{p}, 2^{|F|}\right) + \frac{2^k}{p}.$$

As seen above, the detection failure count is maximum when  $|F|$  is maximum. Thus, the attacker must keep the manipulated bit set  $F \subseteq \mathbb{Z}_k$  as large as possible.

Then, the best way to maximize the detection failures is to induce the fault  $\sum_{i \in F} 2^i$  where  $F_{max}$  is the largest set such that

$$\theta_a p = \sum_{i \in F_{max}} 2^i + 2 \sum_{i \in B \subseteq (\mathbb{Z}_k \setminus F_{max})} 2^i.$$

Notice that  $\theta_d p$  is the modulus multiples whose redundant binary representation with the digit set  $\{0, 1, 2\}$  has the largest number of ones (densest in terms of 1's).

Analyzing this special case, we present a lower bound and an estimate for the maximum of the detection failure count in (5-35) (with the equality (5-39)) under the constraint  $S_\theta \neq \emptyset$ . Let this maximum be denoted by

$$\mathcal{F}_\theta = \max_{S_\theta \neq \emptyset} (|S_\Phi| 2^{k-|F|} + |S_\theta| 2^{|F|} - |S_\Phi| |S_\theta|).$$

The lower bound and the estimate are follows.

$$\mathcal{F}_\theta \geq 2^{|F_{max}|}, \quad \mathcal{F}_\theta \approx \frac{p-1}{p} 2^{|F_{max}|} + \frac{2^k}{p} \quad (5-55)$$

## 5.6 How to Choose Modulus

This chapter first shows a detailed example of evaluating the detection performance of a prime as quadratic code modulus. Then, it gives the basic properties and the examples of good moduli.

### 5.6.1 Evaluating Detection Performance of a Modulus

Let the modulus  $p = 1285677593$  and the data word size  $k = 32$ . Note that  $p$  has the following 32-bit multiples.

| Binary Representation                                | #1's |
|------------------------------------------------------|------|
| $p = (01001100 \ 10100001 \ 11100010 \ 00011001)_2$  | 13   |
| $2p = (10011001 \ 01000011 \ 11000100 \ 00110010)_2$ | 13   |
| $3p = (11100101 \ 11100101 \ 10100110 \ 01001011)_2$ | 18   |

$\lambda_s p = p$  and  $\lambda_d p = 3p$  are the modulus multiples either the largest number of 0's and 1's, respectively.  $E_{min}$  and  $E_{max}$  are the sets of 1's in  $\lambda_s p = p$  and  $\lambda_d p = 3p$ , respectively. Then,

$$E_{min} = \{0, 3, 4, 9, 13, 14, 15, 16, 21, 23, 26, 27, 30\},$$

$$E_{max} = \{0, 1, 3, 6, 9, 10, 13, 15, 16, 18, 21, 22, 23, 24, 26, 29, 30, 31\}.$$

The attacker must flip the data word bits determined by either  $E_{min}$  or  $E_{max}$  to maximize the detection failures. Note that  $2|E_{min}| = 26 > |E_{max}| = 18$ . Then, it follows from Theorem 1 that the detection failure count is

$$2^{k+1-|E_{min}|} + 2^{|E_{min}|} - 2 = 2^{33-13} + 2^{13} - 2 \approx 2^{20}$$

when the flipped bit set is  $E_{min}$  and it is

$$2^{k+1-|E_{max}|} + 2^{|E_{max}|} - 2 = 2^{33-18} + 2^{18} - 2 \approx 2^{18}$$

when the flipped bit set is  $E_{max}$ . Then, the maximum detection failure count  $\mathcal{E} \approx 2^{20}$ . As stated in Theorem 1, let  $u = \max(k + 1 - |E_{min}|, |E_{max}|) = 20$ ,

$$\mathcal{E} = 2^{k+1-u} + 2^u - 2 = 2^{33-20} + 2^{20} - 2 \approx 2^{20}.$$

Unfortunately, the attacker can do better. He can increase the number of 0's or 1's using the redundant binary representations.

- if a bit sequence starts with 011 and ends with 11,
- also if it does not contain any consecutive zeros (00),

0's in this sequence can be increase as follows.

$$(01111111)_2 = (1000000\bar{1})_2,$$

$$(01101011)_2 = (100\bar{1}0\bar{1}0\bar{1})_2.$$

- if a bit sequence starts with 100 and ends with 00,
- also if it does not contain any consecutive zeros (11),

1's in this sequence can be increase as follows.

$$(10000000)_2 = (01111112)_2,$$

$$(10010100)_2 = (01121212)_2.$$

Then, the multiples of  $p$  can be represented as follows.

| Redundant binary with digits -1,0,+1                       | #1's |
|------------------------------------------------------------|------|
| $p = (01001100 \ 10100001 \ 00\bar{1}00010 \ 00011001)_2$  | 11   |
| $2p = (10011001 \ 01000011 \ 0\bar{1}000100 \ 00110010)_2$ | 11   |
| $3p = (11100101 \ 00\bar{1}00101 \ 10100110 \ 01001011)_2$ | 16   |

| Redundant binary with digits 0,1,2                   | #1's |
|------------------------------------------------------|------|
| $p = (01001100 \ 21211121 \ 11011121 \ 11211001)_2$  | 20   |
| $2p = (10010112 \ 12111211 \ 10111211 \ 12110010)_2$ | 20   |
| $3p = (11100101 \ 11100101 \ 10100101 \ 12121011)_2$ | 19   |
| $4p = (12101121 \ 21112111 \ 01112111 \ 21011212)_2$ | 21   |
| $5p = (21111110 \ 11212121 \ 01101010 \ 01111101)_2$ | 21   |
| $6p = (22201011 \ 10112122 \ 12121011 \ 21210110)_2$ | 16   |

There are 32-digit representations of 32-bit or 33-bit multiples of  $p$ .  $\phi_s p = p$  and  $\theta_d p = 4p$  are the modulus multiples with the largest number of 0's and 1's, respectively.  $F_{min}$  and  $F_{max}$  are the sets of  $\pm 1$ 's in  $\phi_s p = p$  and 1's in  $\theta_d p = 4p$ , respectively. Then,

$$F_{min} = \{0,3,4,9,13,17,21,23,26,27,30\},$$

$$F_{max} = \{1,3,4,6,8,9,10,12,13,14,16,17,18,20,21,22,24,26,27,29,31\}.$$

The attacker must flip the data word bits given by either  $F_{min}$  or  $F_{max}$  to maximize the detection failures. Note that  $2|F_{min}| = 22 > |F_{max}| = 21$ . Then, we can reach the following results for some  $\psi \in \{0,1\}$ , using Theorem 3. The detection failure count is

$$2^{k+1-|F_{min}|} + \psi 2^{|F_{min}|} - 2\psi = 2^{33-11} + \psi 2^{11} - 2\psi \approx 2^{22}$$

when the flipped bit set is  $F_{min}$  and it is

$$\psi 2^{k+1-|F_{max}|} + 2^{|F_{max}|} - 2\psi = \psi 2^{33-21} + 2^{21} - 2\psi \approx 2^{21}$$

when the flipped bit set is  $F_{max}$ . Then, the maximum detection failure count  $\mathcal{F} \approx 2^{22}$ . As stated in Theorem 3, let  $v = \max(k+1 - |F_{min}|, |F_{max}|) = 22$ ,

$$\mathcal{F} = \psi 2^{k+1-v} + 2^v - 2\psi = \psi 2^{33-22} + 2^{22} - 2\psi \approx 2^{22}.$$

Then, the worst case probability of a detection failure is  $2^{22}/2^{32} = 1/1024$  when all the 32-bit data words are equally likely.

The quadratic code in this example generates 31 bit residue for 32-bit data words using a 31-bit modulus resulting in a code rate of 32/63. We can increase the rate, choosing a smaller modulus  $p$ , but then  $p$  will have more multiples in  $[0, 2^{32})$ . Thus, the attacker can find sparser and denser multiples more easily. As a result, the maximum detection failure count increases rapidly. In conclusion, the data rate of about 1/2 seems to be the best choice for the quadratic code; a fact that is also pointed out in [Gaubatz et al., 2006].

### 5.6.2 The Properties and Example of Good Moduli

The discussion so far suggests that the detection power of a quadratic code encoding  $k$ -bit data words depends on the multiples of the chosen prime modulus which can be represented in base 2 by  $k$  digits. In the binary representations of these multiples,

- 0's and 1's must be equal in number and
- 0's and 1's must be evenly scattered

in order to keep the maximum detection failure count low. Note that the  $k$  digit multiples of a modulus satisfy the above properties, only if

$$|E_{min}| \approx |E_{max}| \approx |F_{min}| \approx |F_{max}| \approx (k + 1)/2$$

for this modulus.

Table 5-7 gives 32-bit prime modulus example ( $k=32$ ) whose 32-digit multiples satisfy

$$|E_{min}|, |E_{max}|, |F_{min}|, |F_{max}| \in \{\frac{k}{2}, \frac{k}{2} + 1\}$$

Thus, the maximum detection failure counts for them

$$2^{k+1-16} + 2^{16} - 2 = 2^{17} + 2^{16} - 2 = 196\,606,$$

$$2^{k+1-17} + 2^{17} - 2 = 2^{16} + 2^{17} - 2 = 196\,606.$$

by Theorem 1. Consequently, their worst case detection failure probability is

$$(2^{17} + 2^{16} - 2)/2^{32} \approx 3/2^{16}$$

when all the 32-bit data words are equally likely.

**Table 5-7 Suggested prime numbers**

| 32 bit prime | Binary Representation               |
|--------------|-------------------------------------|
| 2362232011   | 10001100 11001100 11001100 11001011 |
| 2577190093   | 10011001 10011100 11001100 11001101 |
| 2738041651   | 10100011 00110011 00110011 00110011 |
| 2863311053   | 10101010 10101010 10101000 11001101 |
| 2908050217   | 10101101 01010101 01010011 00101001 |
| 2908050581   | 10101101 01010101 01010100 10010101 |
| 2908050773   | 10101101 01010101 01010101 01010101 |

## 6 CONCLUSION

We designed and implemented a robust execution unit in an embedded processor that allows efficient and secure execution of cryptographic algorithms. We utilized non-linear arithmetic codes for robustness in detecting faults induced by adversary. We estimated the area overhead of the robust zone for the ASIC implementation. We also provided area usage on an FPGA device after placement-and-routing for the full design including an embedded base processor and robust execution unit. Since the number and organization of the subsystems in the robust execution unit are carefully designed, we observed only a moderate area overhead while no deterioration in maximum applicable clock frequency is reported in ASIC. Although FPGA realizations incur some penalty in maximum allowable clock frequency, overall execution time is not affected in some robust configurations. Robust execution units also provide faster execution of modular multiplication which is the most time-consuming operations in many cryptographic applications. Our results show that it is possible to incorporate a powerful robust error detection circuitry into an embedded processor core transparently with an acceptable cost in area. The proposed error detection capability benefits many cryptographic applications that use basic arithmetic operations.

We also introduce new classes of fault attacks against quadratic codes, which are a class of non-linear arithmetic codes that are proposed for use in the protection of cryptographic devices. The proposed attacks are based on two classes of analysis techniques which prove to be effective in determining the resiliency of a class of error detection codes against a powerful adversary, whose precision and accuracy in introducing multiple bit errors in cryptographic devices are shown to be ever increasing. Our theoretical analysis allows us to express the exact number of undetected faults for a given quadratic code as well as effective countermeasures. Two important findings have practical uses as to how the moduli are chosen in quadratic codes: i) the number of nonzero digits of the modulus should be balanced in various representations (i.e., binary, signed-digit and redundant) and ii) certain distributions of zero and non-zero digits in these representations should be avoided. We provide examples of primes which are resilient against the proposed analysis techniques. Extending the analysis technique presented in the thesis to cubic residue

codes is straightforward, but to other codes calls for further study. Another important result is that the relatively *secure* moduli tend to be costly as far as the implementation is concerned since the modular reduction cannot take advantage of special form moduli. Finding secure moduli that also render cost-effective implementations is left as a future work.

## APPENDICES

### Appendix-A Finding Detection Failure Counts

In the following Maple code,

- The data words in  $\mathbb{Z}_{2^{12}}$  are generated one by one.
- Each data word is XORed by the given error  $\varepsilon$  to generate a faulty data word.
- The quadratic code redundancies for the actual and faulty data words are computed.
- $\delta$  is computed as XOR of the two redundancies.
- deltaList is a list of pairs. Each pair consists of a possible  $\delta$  value and its occurrence. For each computed  $\delta$  value, its occurrence stored in deltaList is incremented by one.

```
p:=19: epsilon:=119*p k:=12:
Settings(defaultbits=k):
deltaList:=[]:
for i from 1 to 2^k do
  actualWord:= i-1:
  faultyWord:= Xor(actualWord, epsilon):
  actualResidue:= actualWord^2 mod p:
  faultyResidue:= faultyWord^2 mod p:
  delta:= Xor(actualResidue, faultyResidue):
  deltaList[delta+1, 2]:= deltaList[delta+1, 2]+1:
end do: deltaList;
```

For  $p = 19$  and  $\varepsilon = 119p$ , Maple result is as follows.

```
[ 0,674] [ 1,276] [ 2,218] [ 3,168] [ 4,128]
[ 5,114] [ 6, 86] [ 7,140] [ 8,112] [ 9, 72]
[10, 92] [11, 10] [12,122] [13,140] [14,228]
[15,180] [16,100] [17,168] [20,230] [21,200]
[22,174] [23,144] [24, 86] [25, 38] [26, 84]
[27,122]
```

As seen,  $\delta = 0$  occurs 674 times while the other  $\delta$  values occur significantly less than 674.

For  $p = 47$  and  $\varepsilon = 53p+10$ , Maple result is as follows.

|           |          |           |           |          |           |
|-----------|----------|-----------|-----------|----------|-----------|
| [ 0, 436] | [ 1, 64] | [ 2, 106] | [ 3, 66]  | [ 4, 70] | [ 5, 102] |
| [ 6, 60]  | [ 7, 66] | [ 8, 68]  | [ 9, 100] | [10, 92] | [11, 110] |
| [12, 48]  | [13, 86] | [14, 74]  | [15, 48]  | [16, 82] | [17, 44]  |
| [18, 68]  | [19, 92] | [20, 74]  | [21, 74]  | [22, 66] | [23, 64]  |
| [24, 82]  | [25, 80] | [26, 60]  | [27, 60]  | [28, 74] | [29, 76]  |
| [30, 56]  | [31, 54] | [32, 58]  | [33, 42]  | [34, 70] | [35, 102] |
| [36, 78]  | [37, 62] | [38, 70]  | [39, 44]  | [40, 28] | [41, 22]  |
| [42, 38]  | [43, 42] | [44, 102] | [45, 18]  | [46, 48] | [48, 28]  |
| [49, 28]  | [50, 42] | [51, 14]  | [52, 28]  | [53, 40] | [54, 50]  |
| [55, 26]  | [56, 44] | [57, 50]  | [58, 14]  | [59, 56] | [60, 68]  |
| [61, 24]  | [62, 24] | [63, 34]  |           |          |           |

## Appendix-B Finding $|E|$ , $|S_A|$ , and $|S_B|$

The following Maple code produces a five-tuple for each modulus multiple  $\lambda_p$ . Each five-tuple contains

- the nonzero bit count  $|E|$  of  $\lambda_p$ ,
- $\lambda_p$  itself,
- the size of the set  $S_A$  for the fault  $\varepsilon = \lambda_p$ ,
- the size of the set  $S_B$  for the fault  $\varepsilon = \lambda_p$ ,
- the detection failure count for the fault  $\varepsilon = \lambda_p$ ,

```
p:=131: k:=12: Settings(defaultbits=k):
tuples:=seq([NULL], i=1..k):
for epsilon from p by p while p<2 ^k do
  count:=0: SA:=0: SB:=0:
  for multiple from 0 by p while p<2 ^k do
    if Or(multiple,epsilon)=epsilon then SA:=SA+1: end if:
    if And(multiple,epsilon)=0 then SB:=SB+1: end if:
  end do:
  for i from 1 to 2 ^k do
    actualWord:=i-1:
    faultyWord:=Xor(actualWord,epsilon):
    actualResidue:=actualWord ^2 mod p:
    faultyResidue:=faultyWord ^2 mod p:
    delta:=Xor(actualResidue , faultyResidue):
    if(delta=0) then count:=count+1: end if:
  end do:
  E:=add(Split(epsilon)[i], i=1..k):
  tuples[E]:=[op(tuples[E]), [E,epsilon,SA,SB,count]]:
end do
```

## BIBLIOGRAPHY

Aumüller, C., Bier, P., Fischer, W., Hofreiter, P. and Seifert, J.-P. (2002) 'Fault Attacks on RSA with CRT: Concrete Results and Practical Countermeasure', CHES, San Francisco, 260-275.

Balasch, J., Gierlichs, B. and Verbauwhede, I. (2011) 'An In-depth and Black-box Characterization of the Effects of Clock Glitches on 8-bit MCUs', FDTC, Nara, 105-114.

Bekkers, R. and König, H. (2011) 'Fault Injection, A Fast Moving Target in Evaluations', FDTC, Nara, 65.

Bertoni, G., Breveglieri, L., Koren, I., Maistri, P. and Piuri, V. (2003) 'Concurrent Fault Detection in a Hardware Implementation of the RC5 Encryption Algorithm', ASAP, The Hague, 423-432.

Bertoni, G., Breveglieri, L., Koren, I., Maistri, P. and Piuri, V. (2003) 'Error Analysis and Detection Procedures for a Hardware Implementation of the Advanced Encryption Standard', *IEEE Transactions on Computers*, vol. 52, no. 4, pp. 492-505.

Berzati, A., Canovas-Dumas, C. and Goubin, L. (2010) 'Public Key Perturbation of Randomized RSA Implementations', CHES, Santa Barbara, 306-319.

Berzati, A., Canovas-Dumas, C., Goubin, L., Joye, M.& and Tunstall, M. (2012) 'A Survey of Differential Fault Analysis Against Classical RSA Implementations', *Fault Analysis in Cryptography*, pp. 111-124.

Biehl, I., Meyer, B. and Müller, V. (2000) 'Differential Fault Attacks on Elliptic Curve Cryptosystems', CRYPTO, Santa Barbara, 131-146.

Biham, E. and Shamir, A. (1997) 'Differential fault analysis of secret key cryptosystems', CRYPTO 97, Santa Barbara, 513-525.

Boneh, D., DeMillo, R.A. and Lipton, R.J. (2001) 'On the Importance of Eliminating Errors in Cryptographic Computations', *Journal of Cryptology*, vol. 14, no. 2, pp. 101-119.

Chaum, D. (1982) 'Blind Signatures for Untraceable Payments', CRYPTO, New York, 199-203.

Gaubatz, G. and Sunar, B. (2006) 'Robust Finite Field Arithmetic for Fault-Tolerant Public-Key Cryptography', FDTC, Yokohama, 196-210.

Gaubatz, G., Sunar, B. and Karpovsky, M.G. (2006) 'Non-linear Residue Codes for Robust Public-Key Arithmetic', FDTC, Yokohama, 173-184.

Halfhill, T.R. (2011) *Tensilica Tweaks Xtensa: Xtensa LX3 and Xtensa 8 Cores Boost Performance, Cut Power*, 20 April, [Online], Available: [http://www.tensilica.com/uploads/pdf/Tensilica-LX3\\_Reprint.pdf](http://www.tensilica.com/uploads/pdf/Tensilica-LX3_Reprint.pdf) [20 April 2011].

Hamid, H.B.-E., Choukri, H., Tunstall, D.N.M. and Whelan, C. (2006) 'The Sorcerer's Apprentice Guide to Fault Attacks', *Proceedings of the IEEE*, vol. 94, no. 2, pp. 370-382.

Karpovsky, M.G., Kulikowski, K.J. and Taubin, A. (2004) 'Robust Protection against Fault-Injection Attacks on Smart Cards Implementing the Advanced Encryption Standard', DSN'04, Florence, 93-101.

Karpovsky, M.G. and Taubin, A. (2004) 'New Class of Nonlinear Systematic Error Detecting Codes', *IEEE Transactions on Information Theory*, vol. 50, no. 8, pp. 1818-1820.

Karri, R., Kuznetsov, G. and Goessel, M. (2003) 'Parity-Based Concurrent Error Detection of Substitution Permutation Network Block Ciphers', CHES 2003, Cologne, 113-124.

Karri, R., Wu, K., Mishra, P. and Kim, Y. (2002) 'Concurrent Error Detection of Fault Based Side Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers', *IEEE Trans. on Computer Aided Design of Integrated Circuits and Systems*, vol. 21, no. 12, pp. 1509-1517.

Koç, Ç.K., Acar, T. and Jr., B.S.K. (1996) 'Analyzing and Comparing Montgomery Multiplication Algorithms', *IEEE MICRO*, vol. 16, no. 3, pp. 26-33.

Kocher, P.C. (1996) 'Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems', CRYPTO, Santa Barbara, 104-113.

Kocher, P.C., Jaffe, J. and Jun, B. (1999) 'Differential Power Analysis', CRYPTO, Santa Barbara, 388-397.

Konrad J. Kulikowski, M.G.K.A.T. (2005) 'Robust Codes for Fault Attack Resistant Cryptographic Hardware', FDTC, Edinburgh, 1-12.

Li, Y., Sakiyama, K., Gomisawa, S., Fukunaga, T., Takahashi, J., Ohta and Kazuo (2010) 'Fault Sensitivity Analysis', CHES, Santa Barbara, 320-334.

Mathew, J., Rahaman, H., Jabir, A.M., Mohanty, S.P. and Pradhan, D.K. (2010) 'On the design of different concurrent EDC schemes for S-Box and GF(p)', ISQED, Santa Clara, 211-218.

Montgomery, P.L. (1985) 'Modular Multiplication Without Trial Division', *Mathematics of Computation*, vol. 44, no. 170, pp. pp. 519-521, Available: <http://www.jstor.org/stable/2007970>.

Quisquater, J.-J. and Piret, G. (2003) 'A Differential Fault Attack Technique Against SPN Structures, with application to the AES and KHAZAD', CHES, Cologne, German, 77-88.

Quisquater, J.J. and Samyde, D. (2002) 'Eddy current for magnetic analysis with active sensor', *Proc. of Esmart*.

Reyhani-Masoleh, A. and Hasan, M.A. (2002) 'Error Detection in Polynomial Basis Multipliers over Binary Extension Fields.', CHES 2002, San Francisco, 515-528.

Reyhani-Masoleh, A. and Hasan, M.A. (2004) 'Towards Fault-Tolerant Cryptographic Computations over Finite Fields', *ACM Transactions on Embedded Computing Systems*, vol. 3, no. 3, pp. 593-613.

Skorobogatov, S.P. and Anderson, R.J. (2003) 'Optical Fault Induction Attacks', CHES 2002, San Francisco, 2-12.

Trichina, E. and Korkikyan, R. (2010) 'Multi Fault Laser Attacks on Protected CRT-RSA', FDTC, Santa Barbara, 75-86.

Wang, Z., Karpovsky, M.G. and Joshi, A. (2012) 'Secure Multipliers Resilient to Strong Fault-Injection Attacks Using Multilinear Arithmetic Codes', *IEEE Trans. VLSI Syst.*, vol. 20, no. 6, pp. 1036-1048.

Yumbul, K., Erdem, S.S. and Savaş, E. (2010) 'Design and implementation of robust embedded processor for cryptographic applications', *SIN, Taganrog*, 178-185.

Yumbul, K., Erdem, S.S. and Savaş, E. (2011) 'On Protecting Cryptographic Applications Against Fault Attacks Using Residue Codes', *FDTC, Nara*, 69-79.

## VITA

Kazım YUMBUL was born in 1977 in Merzifon/Amasya, TURKEY. He received the BS (1998) and MS (2001) degrees in electrical and electronics engineering from the Electrical and Electronics Engineering Department at Ege University. Between 1998 and 2002, he has worked as a research assistant in Electrical & Electronics Engineering department at Ege University, İzmir, Turkey. Between 2002 and 2007, he has worked as a researcher in Information Technology Institute at TUBITAK, Kocaeli, Turkey. He is currently working toward the Ph.D. degree in Department of Electronics Engineering at Gebze Institute of Technology. He is also working as a senior design engineer at ASELSAN Inc. His research interests include embedded system, security, and digital system design.