Tamper-Resistant Arithmetic for Public-Key Cryptography

by
Gunnar Gaubatz

A Dissertation
Submitted to the Faculty
of the

Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Doctor of Philosophy
in
Computer Engineering

March, 2007

Approved:

Prof. Berk Sunar
ECE Department
Dissertation Advisor

Prof. Mark G. Karpovsky
ECE Department
Boston University

Prof. William J. Martin
Department of Mathematical Sciences

Prof. Brian M. King
ECE Department

Prof. Fred J. Looft
ECE Department Head

For my parents.

Abstract

Cryptographic hardware has found many uses in ubiquitous and pervasive security devices with a small form factor, e.g. SIM cards, smart cards, electronic security tokens, and soon even RFIDs. With applications in banking, telecommunication, healthcare, e-commerce and entertainment, these devices use cryptography to provide security services like authentication, identification and confidentiality to the user.

However, the widespread adoption of these devices into the mass market, and the lack of a physical security perimeter have increased the risk of theft, reverse engineering, and cloning. Despite the use of strong cryptographic algorithms, these devices often succumb to powerful side-channel attacks. These attacks provide a motivated third party with access to the inner workings of the device and therefore the opportunity to circumvent the protection of the cryptographic envelope. Apart from passive side-channel analysis, which has been the subject of intense research for over a decade, active tampering attacks like fault analysis have recently gained increased attention from the academic and industrial research community.

In this dissertation we address the question of how to protect cryptographic devices against these kinds of attacks. More specifically, we focus our attention on public key algorithms like elliptic curve cryptography and their underlying arithmetic structure. In our research we address challenges such as the cost of implementation, the level of protection, and the error model in an adversarial situation. The approaches that we investigate all apply concepts from coding theory, in particular the theory of cyclic codes. This seems intuitive, since both public key cryptography and cyclic codes share finite field arithmetic as a common foundation.

The major contributions of our research are (a) a generalization of cyclic codes that allow embedding of finite fields into redundant rings under a ring homomorphism, (b) a new family of non-linear arithmetic residue codes with very high error detection probability, (c) a set of new low-cost arithmetic primitives for optimal extension field arithmetic based on robust codes, and (d) design techniques for tamper-resilient finite state machines.

Acknowledgements

First and foremost, I would like to thank my advisor and mentor Dr. Berk Sunar for his tremendous support and guidance throughout the years. His patience and insights have resulted in a fruitful collaboration that I am truly thankful for.

I would like to extend my sincere gratitude to my dissertation committee members, Drs. Mark Karpovsky, Brian King and William Martin, whose valuable suggestions and inputs have improved this dissertation measurably. Additionally, the generous support of the National Science Foundation (under grants No. NSF-ANI-0112889 and NSF-ANI-0133297), of Intel Corporation, as well as that from WPI's Axel F. Backlin scholarship fund is gratefully acknowledged.

As important as academic advice and funding is, the support that comes from the people around us shall not be forgotten, either. Many thanks to my dear friends, colleagues and collaborators in the lab: Jens-Peter Kaps, Selçuk Baktır, Erdinç Öztürk, Deniz Karakoyunlu, and Ghaith Hammouri; to my colleagues from the lab upstairs: Dave Holl and Ben Woodacre; to the helpful and indispensable office staff: Cathy Emmerton, Brenda McDonald, and Colleen Sweeney; and to the fearless leader of the ECE department and general good sport: Dr. Fred Looft. Thank you all!

Words cannot express how thankful I am to my parents, Erwin and Inge Gaubatz, for all their love and support throughout the years: my Dad inspired me to become an engineer, showed me how things work, and how to fix them if they don't; my Mom with her wonderful big heart taught me responsibility, was always there for me, and let me go out into the world when I had to. I am indebted to my sister Corinna, who is there for our parents when I cannot. To my parents- and sisters-in-law, Monir, Khosro, Yasi and Kiana, for their love and support: kheili mamnoon!

Most importantly, however, I would like to thank my wife and best friend Nastaran for all her love, care and encouragement. She stuck with me through some hard times, gave me motivation when I was discouraged, and was always close to me—regardless of the North American continent that physically separated us for most of our graduate student life.

San Jose, CA, April 2007

Gunnar Gaubatz

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

In the course of only a few decades the field of cryptography has changed from a black art that only few people in the military and secret services knew about, to an active and openly researched science that is one of the key ingredients of the modern internet economy. Beyond e-commerce it also starts to affect more and more aspects of the public sector, with some examples being the adoption of cryptographic smart cards for health care providers, and digital signature enabled electronic passports.

The pervasive use of cryptography brings with it many new challenges that were not foreseen only a few years back when information security was mainly server-centric. Ubiquitous security devices such as smart cards and security tokens lack the traditional physical security perimeter of a server-based infrastructure. The small form factor of these devices facilitates theft and reverse engineering techniques, and thus requires new protection mechanisms for the sensitive data that is stored on them.

The current generation of cryptographic algorithms and protocols is rarely challenged by the computational resources of an attacker. Key sizes of 128 bits and more for symmetric schemes and matching sizes for public key schemes offer sufficiently large security margins to withstand even huge leaps in cryptanalytical progress. The real, tangible threat stems from side-channel attacks in which an attacker uses certain

implementation-specific physical phenomena of the device to break the cryptosystem, rather than to attack the algorithm directly.

Over the last decade a large body of industrial as well as academic research has been devoted to the study of side channel attacks. In terms of high-level classification we need to distinguish between passive attacks (attacker confined to the role of observer), and active attacks (attacker can manipulate the device). Most of the R&D effort to date has been focused on the development of countermeasures against passive attacks, although early on Boneh, DeMillo and Lipton [BDL97] and also Joye, Lenstra and Quisquater[JLQ99] effectively demonstrated the strong need for the protection of various public-key systems against active attacks. Biham and Shamir [BS97] and later Piret and Quisquater [PQ03] demonstrated successful fault attacks against the data paths of various block ciphers, while only recently attacks on the control logic of block ciphers have been reported by Choukri and Tunstall [CT05].

Fault injection based attacks have potentially devastating consequences on the security of embedded systems. While their uses have been well known among practitioners in the field, i.e. hackers and security evalutation professionals, treatment from the academic community was until recently quite sparse compared to passive attacks such as timing or power analysis. Even then, most papers on the subject matter seem to focus mainly on the invention of new attacks or modification of existing attacks to various cryptosystems. The contributions that deal with countermeasures to these attacks seem to be in the minority.

Designers of security-critical embedded systems are in a perpetual struggle to devise new countermeasures against adversaries, while at the same time keeping a bound on the cost of the implementation. Ever new attack vectors keep appearing which are easy to implement and difficult to anticipate. It seems surprising that passive attacks are generally better studied in the open literature than active tampering attacks, given that the latter have been successfully (ab-)used in real-world scenarios

such as to produce counterfeit smart-cards for accessing pay-TV services [AK96].

For manufacturers of devices prone to any sort of tampering, it seems prudent to invest not only in physical countermeasures such as tamper-proof coating and environmental sensors, but also in error detection mechanisms. After all, the methods of fault-injection are manifold (photo-flash, laser, ion-beam, power-glitch, radiation, etc.), while the outcome is always the perturbation of data. Sensors for detecting tampering activity are certainly a useful tool for increasing the level of physical security, but they can never protect against all possible attack-vectors. By this reasoning, error detection is an essential building block for secure systems under adversarial conditions.

The current research effort to prevent active fault attacks is based on two strategies which complement each other. On one hand there are techniques to reduce side channel leakage in case of a successful fault insertion [JY02]. The other strategy is aimed at increasing the fault-tolerance of cryptographic devices. The work presented in this dissertation falls into the latter category.

1.1 Motivation

The security of several public-key cryptosystems relies on the intractability of the discrete logarithm problem (DLP) in large finite groups. Such groups may be large multiplicative subgroups of a finite field $\mathbb{K}$, or the group of points on an elliptic curve defined over $\mathbb{K}$. The most common choices for $\mathbb{K}$ are prime fields $\mathbb{F}_p$ and binary extension fields $\mathbb{F}_{2^k}$. In the literature these are often referred to as *Galois Fields*, in honor of the french mathematician *Évariste Galois*, and denoted as GF(p) and GF(2^k).

Arithmetic in Galois Fields serves as the mathematical foundation not only of public key cryptography, but also many concepts in coding theory. Indeed cryp-

tography and coding theory are two very closely related fields with a large amount of overlap. Finite fields and rings are mathematical structures that are common to both areas and have found applications in the definition of cyclic codes, public-key and symmetric cryptosystems. While the applications of both areas can be broadly characterized as communication-centric, in practice they have traditionally remained separate. Stated simply, coding theory provides for reliable communication over a noisy channel, while cryptography provides for secure communication over an ideal channel.

By viewing the data path of a public key arithmetic operation as a noisy environment that may be disturbed by an attacker, we aim to apply concepts from coding theory to this "computational channel" in the hope to provide for reliable arithmetic operation. This is not unlike early techniques in fault tolerant computing that aimed to protect the arithmetic unit of a space-flight computer against errors caused by radiation, albeit more focussed on the specific properties of finite field arithmetic.

1.2 Dissertation Outline

This dissertation consists of four major chapters (5 through 8) that represent the major results of the author's research. It is organized as follows: In Chapter 2 we will provide the reader with an overview of related work specific to each of the four major chapters. In Chapter 3 we will give an introduction into common side-channel analysis techniques and their classification. Chapter 4 defines the adversarial error model that forms the basis of our research. Chapter 5 contains the results of our research on homomorphic embedding techniques for finite fields. We show how these techniques may be used to provide redundancy and error detection mechanisms, and how they relate to the theory of cyclic codes. In Chapter 6 we consider a worst-case scenario with respect to attacker capability and introduce robust arithmetic codes

which offer nearly perfect error detection properties. Since the strong error detection properties of these codes come at the price of large overhead, we show in Chapter 7 how one may strike a balance between robustness and low overhead, based on a special class of finite fields. The data path-centric view of the previous chapters is complemented by Chapter 8 in which we present novel techniques for the protection of control structures in cryptographic circuits.

Chapter 2

Related Work

2.1 Homomorphic Embedding

Early work on fault tolerant cryptography has either revolved around the use of simple parity prediction schemes or adapted traditional mechanisms like *triple modular redundancy (TMR)* and *time redundancy* for determining the correct result in the presence of errors. It seems, however, that most of the current effort is concentrated on *concurrent error detection (CED)* schemes for symmetric ciphers. Efforts to provide error detection capabilities to public key schemes based upon finite field arithmetic have so far only seen sporadic treatment. The prevailing approach is augmentation of finite field multipliers over $\mathbb{F}_2^k$ with parity prediction capabilities [RH02, RMH04]. These techniques, however, do not make use of the rich mathematical structures provided by finite rings and fields, which form the arithmetic foundation for many cryptographic schemes. Furthermore, their error detecting capabilities are mostly aimed at simple faults with only a single bit-flip, caused for example by *single event upsets* (SEU) due to background radiation, but not multiple faults induced by an intelligent attacker.

A strategy that has not been explored yet is the use of *error detecting codes*

(EDC) with an arithmetic structure, specifically cyclic binary and arithmetic codes, upon which our approach is based. The main difficulty is that our purpose is not only to encode and decode data for transmission over a noisy channel. In viewing the computation itself as a noisy channel, we aim to compute with encoded operands while preserving arithmetic operations. We thus propose to embed finite field elements into a larger ring via a suitable ring homomorphism, and to utilize the redundancy for error detection purposes. Embedding techniques for finite fields have been used before, e.g. for the implementation of efficient finite field multiplier architectures based on redundant representation in cyclotomic rings [WHBG02]. The authors, however, have not explored the usefulness of this redundancy for fault tolerance. Our work on *scaled embedding* was motivated by earlier work on modulus scaling [Wal92, ÖSS04], and the connection to coding theory. In [ÖSS04] a scaled modulus of special low Hamming-weight form was used to enable low-complexity modular reduction, but the redundancy was not used for error detection. By additionally scaling the operands with a constant factor, the information is spread out across an extended range of bits. This allows the detection of errors by simply dividing out that factor and checking for the remainder to be zero.

The results of this chapter were originally presented at the Fault Diagnosis and Tolerance in Cryptography Workshop in 2005 and published in [GS05].

2.2 Robust Arithmetic Residue Codes

During the early years of fault-tolerant computing, residue codes were proposed [RG71] as a means for systematically encoding integer operands and checking their arithmetic operations for errors. The check symbol in residue codes is computed as the remainder of the operand (or its complement) with respect to the check modulus, usually a prime. Several variations such as multi-residue codes were also introduced

early on, as well as non-systematic arithmetic codes such as AN codes[1]. Designed for the purpose of detecting only sporadically occurring bit errors, their arithmetic distance is limited to 2 or 3. Mandelbaum [Man67] introduced arithmetic codes with larger distance properties, however, with an unattractively large amount of redundancy.

Unfortunately, due to the linear nature of their encoding scheme, standard arithmetic residue codes do not offer robustness properties, since any error pattern which itself is a codeword can not be detected, irrespective of the actual data. In this work we extend the idea of residue codes to non-linear residue codes and show how their arithmetic properties can be used to provide strong error detection under even the most challenging adversarial conditions. Our construction for robust arithmetic codes is versatile enough to be applied to any fixed width data-path for digit serial general purpose arithmetic. Using this code it is possible to protect any type of integer ring or prime field arithmetic, e.g. RSA, Diffie-Hellman, Elliptic Curves over GF(p), etc., against active adversaries.

We presented our results at the Fault Diagnosis and Tolerance in Cryptography Workshop in 2006 and published them in [GSK06].

2.3 Low Cost Techniques

A common approach to prevent errors in cryptographic hardware involves the use of existing redundancy, for example by decrypting an encrypted result and comparing to the input. In other cases simple concepts from classic linear coding theory (parity prediction, Hamming codes) are applied to standard circuits, for low-cost protection against basic faults.

However, with the advent of more advanced attack techniques and more accurate

[1]Named after their multiplicative encoding procedure $N' = AN$, where A is the code generator value and N is an operand

error and attacker models, these simple techniques may prove to be inadequate. Effective hardening of systems against sophisticated and malicious fault attacks requires strong error detection under worst case assumptions rather than average case. At the same time there is always the concern about overhead of error detection solutions, since more complex circuits are also more likely to fail. The robust arithmetic residue codes mentioned above, unfortunately fall into this category.

In order to strike a balance between robustness and overhead we show in this chapter how Karpovsky and Taubin's [KT04] original construction may be applied to a type of specialized finite extension fields named optimal extension fields (OEF). This new construction preserves the original code's strong error detection properties, yet with almost negligible overhead.

The results of this chapter are currently under preparation for publication.

2.4 Tamper Resilient Control Structures

So far most of the work in fault tolerant cryptography is centered around concurrent error detection (CED) techniques applied to the data path [RMH04, GS05]. Other techniques take advantage of architectural features, such as independent encryption and decryption units to compute the inverse operation for parallel computation and comparison of the result [JWK04]. While this strategy works well with symmetric key algorithms such as the AES [BBK$^+$02], it may not work in a public-key setting, for example if the private key is not available to check the result of an encryption [ABF$^+$02].

We are currently unaware of any strategies to also protect the control logic (state machine, sequencer, execution pipeline, etc.) of cryptographic systems against interference from the outside. A literature search revealed that there are several decades worth of research on the design of fault-tolerant state machines in classic application domains like automation control, avionics, and space-borne systems

[Ren78, CT91, Ber04]. While it is true that many of those findings could also apply to cryptographic systems, our main concern is that the fault model is fundamentally different. Fault-tolerant digital systems are typically designed around the premise that faults only occur one at a time and that there is sufficient time to recover between faults. Thus all that is needed to build a fault secure system is the ability to recover from the set of single faults [Pra86]. Such an assumption seems reasonable as long as the faults are caused by stochastic events like natural background radiation, or the random failure of components over time. In a cryptographic setting we deal with faults of an adversarial nature, caused by an intelligent attacker, who can be assumed to know about the structure and thus certain weaknesses of the system. In this chapter we therefore explore design techniques for tamper resilient finite state machines that employ error detecting codes with large minimum distance.

Our results have been accepted for publication in the IEEE Transactions on Computers [GSS07].

Chapter 3

Side-channel Analysis: A Taxonomy

The standard literature on cryptography defines a cryptographic system abstractly as a black box with an input and an output. The input accepts the plain text, while the output provides the cipher text. Implicitly we also shall assume that the key is loaded into the black box during a safe time via some appropriate mechanism, and stored for later use in the field, but cannot be retrieved out of the box. In the physical world this description is insufficient, as it is nearly impossible to completely isolate a device from its environment. In reality there are many unintentional input and output channels, in the following named *side-channels*, which can reveal the inner workings of the device.

A side-channel attack, or more formally a side-channel analysis, is an attack on a cryptographic device that makes use of one or more of these covert channels to obtain information about the key material, thus breaking the cryptosystem. Over the last two decades, many different classes of side-channel attacks have been studied in the literature and also applied in practice.

In general one may classify side-channel analysis into two major categories, de-

pending on whether a device is simply observed or whether it is being acted upon by the attacker. Observation-only attacks are termed passive, while attacks involving interaction with the device are termed active. Just like the regular interface of a cryptosystem defines input and output channels, side-channels may also be classified in terms of the direction of information flow. Physically observable phenomena that result from activity inside the device consitute a covert output channel. Similarly, any kind of external manipulation that the device is exposed to and which has an effect on the device's inner workings can be considered a covert input channel. Moreover, an external manipulation that has a discernible effect, i.e. that can be observed through any (overt or covert) output channel, constitutes an interaction with the device.

3.1 Passive Side-Channel Analysis

The following physical phenomena have been identified in the literature as effective for passive side-channel analysis:

- Timing information
- Power consumption
- Electromagnetic emissions (including thermal phenomena)
- Accoustic emissions

Timing analysis, first introduced by Kocher [Koc96] in 1996, works by measuring the data-dependent differences in execution timing of naïve implementations to determine the Hamming weight of the secret key. It applies mainly to modular exponentiation based public key cryptosystems, such as RSA or Diffie-Hellman, where the exponentiation is implemented as a simple square-and-multiply algorithm. Common countermeasures include multiply-always strategies aimed at constant time execution.

Power [KJJ99] and electromagnetic analyses [GMO01, QS01] measure the power consumption and the radio frequency spectrum of a cryptographic device over time to deduce information about the data that is being processed, specifically the key-dependent data. It can be successful even if precautions to timing analysis have been taken. Simple power (SPA) and electromagnetic analysis (SEMA) operate on a single power or EM trace acquisition, while differential analysis (DPA and DEMA) acquires several thousands of traces and computed their average signal to filter out random variations. Then, by computing the difference between the average power trace and a single acquisition, even subtle data-dependent power consumption becomes visible in the trace, making this attack very powerful. While power analysis may be mitigated by supply power filtering at the terminals of the device, EM analysis can bypass these countermeasures by sampling the emissions with high accuracy directly above the circuit using micro-coils.

Common countermeasures against power and EM analysis require a tremendous effort on the part of the circuit designers, who need to go to great lengths to balance power consumption or shield the device from emitting EM radiation. Other approaches use adaptive masking techniques for randomization and hence, de-correlation between the power signature and the secret information.

3.2 Active Side-Channel Analysis

Active attacks are much more powerful than passive attacks, since they no longer confine the attacker into the role of an observer. Through the deliberate insertion of faults into the computation an adversary might cause the leakage of secret key information. The consequences of not employing at least an error detection scheme have been demonstrated vividly in [BDL97]. Protecting against this class of attacks requires more than elaborate circuit tricks; it requires a mechanism for detecting mod-

ification of data, faulty behavior of the arithmetic circuit, or both. Just as traditional mission critical applications like avionics and systems working under harsh environmental conditions require fault tolerant design techniques, it becomes increasingly important for embedded security devices operating in a hostile environment.

Active side channel attacks rely on the manifestation of injected faults as erroneous results which can then be observed either at the output of the device, or through some covert side-channel. The error is therefore the difference between the expected output x and the observed output $\tilde{x} = x + e$. We will define the error model and the capabilities of the adversary more precisely in the following chapter.

In 1996 Boneh, DeMillo and Lipton [BDL97] demonstrated painfully how vulnerable straightforward implementations of public-key cryptographic algorithms are to a class of attacks now commonly referred to as "Bellcore attacks". Here we give a short summary of the attack as it applies to the RSA signature scheme.

It is commonly known that the Chinese Remainder Theorem (CRT) may be used to speed up the computation of the RSA signature algorithm, due to availability of the factors p and q of the RSA modulus N. Let $n = \lceil \log_2 N \rceil$ be the length of the modulus in bits, and thus typically p and q are of size $n/2$ bits. The RSA signature algorithm computes the modular exponentiation $y = x^d \bmod N$, where x is the message digest, d is the private key, and y is the resulting signature. Instead of computing a n-bit exponentiation which has a time complexity of $O(n^3)$ assuming a quadratic complexity multiplication algorithm, the CRT allows one to compute two $n/2$-bit exponentiations $y_p = x^{d_p} \bmod p$ and $y_q = x^{d_q} \bmod q$ and combining them into the final result $y = y_q + q((y_p - y_q)q^{-1} \bmod p)$. This reduces the complexity of exponentiation to $O((n/2)^3)$ and results in an overall speed-up of almost factor 4.

The Bellcore attack consists of obtaining one correct signature S and one faulty one S' by inducing a random fault in one of the two $n/2$-bit exponentiations. The only requirement for success is that the fault only affects either y_p or y_q but not both. After

obtaing S and S', the attacker may simply compute the greatest common divisor of the signatures' difference and the RSA modulus N to obtain the factorization of N: $\gcd(S - S', N) = r$, where either $r = p$ or $r = q$ with high probability.

Shortly after their findings were discussed in the cryptographic community, Shamir [Sha99] proposed a simple and "low-cost" countermeasure which consists of scaling the moduli p, q by some small random factor j and a slightly modified derivation of the partial exponents. These modifications of the standard CRT algorithm allow the detection of random errors during the exponentiation. Shamir's trick, however, was shown to be flawed by Aumüller et al. [ABF+02], since it does not protect all steps of the computation. More advanced protection schemes were proposed by Blömer et al. [BOS03] and Yen et al. [YJ00], and there exist claims that some of them can be broken, too [Wag04], although this seems to be disputed. The discoveries of the Bellcore team triggered additional work by other researchers who extended the technique to other cryptosystems [JLQ99], and also spawned a new type of active attack on symmetric key cryptosystems such as block and stream ciphers, called differential fault attack (DFA) [BS97].

Apart from Bellcore style attacks there exists another type of fault attack, which is aimed at common countermeasures to passive attacks. In order to prevent power and electro-magnetic analysis techniques, many VLSI implementations nowadays employ power balanced logic gate libraries, whose power consumption and hence electromagnetic emanations are data-independent. New fault attacks are aimed at introducing glitches into the circuit which cause such gates to 'lose balance', i.e. reveal data through power imbalances. This opens the door to various classical attacks on the circuit, like simple and differential power (SPA,DPA) and electromagnetic (SEMA,DEMA) analysis. All this demonstrates the urgent need for a truly robust error detection scheme.

Chapter 4

An Adversarial Error Model

Based on the discussion of active tampering attacks we will now establish an adversarial error model which allows us to analyze the effectiveness of countermeasures. We will make use of existing definitions borrowed from the testing and fault-tolerance communities.

Generally one can say that errors are the manifestations of faults occurring in a system. Some classes of faults may not cause an error, others may have devastating consequences and cause multiple errors at the same time. Faults can be broadly categorized into permanent and transient faults:

Permanent Faults can be caused by manufacturing defects, long-term slowly destructive effects such as electromigration, or by a sudden disruptive force, for example a static discharge. In an adversarial setting, a permanent fault may be caused by an attacker who has access to a focussed ion beam (FIB) workstation, intent on manipulating the functionality of a device.

Transient Faults are non-permanent faults, which have an effect only for a limited amount of time, usually only a couple of clock-cycles or fractions thereof. These may be caused by environmental effects that temporarily drive the operating conditions of the device outside of the normal range for which it was

designed, such as fluctuations in supply voltage or temperature. As before, we are more interested in adversarial faults, which may be induced in a variety of ways. An attacker may try to emulate environmental effects and raise the operating temperature of the device or manipulate the power supply. Other specific techniques are described further below.

Successful induction of faults depends on a variety of factors, such as the physical characteristics of the device under attack, the method of fault induction and the existence and type of countermeasures. Starting with the latter, the presence of countermeasures determines which methods of fault injection can be applied in the first place. For example, if the device has a unit for monitoring the quality of the supply voltage and can perform an emergency shutdown if necessary, then a glitching attack is unlikely to succeed.

Every method of fault injection has different characteristics in terms of location and effect of the fault. Some techniques such as power supply glitching may have a more global effect, for example in the case where there is only a single power connector. In other cases such as a general purpose CPU, there may be several ground and supply voltage connectors (GND and V_{CC}), each of which may supply power to a particular functional group of the device. With a finer granularity of power supply lines the attacker also gains better control over the locality of the fault.

The precision of fault injection can be characterized in terms of spatial and temporal resolution of faults, which ultimately determines the precision with which specific error patterns can be caused in the device:

Spatial resolution is a measure of precision with regard to the physical dimensions of the device.

Temporal resolution measures the precision with regard to fault insertion at a particular time.

In addition to the classification of faults based on resolution, we may additionally distinguish between two modes of failure that we define as follows: A *common-mode failure* (CMF) [MM00] is a set of multiple, simultaneously occurring faults resulting from a single cause, e.g. due to a glitch in the power supply, or a large area ionization of an unpackaged integrated circuit with a radiation source. Highly regular circuits made from replicated components are the most prone to common-mode failure, if the cause of the fault is not one of limited locality. In contrast, a *single-mode failure* (SMF) is a single fault that occurs due to a single cause. Oftentimes the effect of the fault is locally limited. Examples of this type of fault are single stuck-at faults caused early on by manufacturing flaws or later through breakdown resulting from electro-migration. In an adversarial setting, a transient single-mode failure may also be induced by means of a 'light-attack'. This refers to the introduction of single bit errors into an unpackaged integrated circuit by means of a highly focused laser beam or other ionization source [SA02]. It requires a high degree of spatial and potentially also temporal precision.

4.1 Faults from an Attacker's Perspective

From an attacker's point of view transient faults are more useful than permanent faults for many reasons. For one, he may only possess a single device to attack and a permanent fault may render the device non-functional. Secondly, several of the known fault attacks require faulty *and* non-faulty output in order to succeed. In the case of permanent faults there is only a single opportunity to obtain useful output on which to mount the attack. The infamous fault attack on the CRT-based RSA signature scheme by Boneh et al. [BDL97], for example, assumes a simple register fault model, in which an almost arbitrary portion of an intermediate value (possibly a single bit) is changed during one of the two modular exponentiations. It does not

matter which of the two exponentiations is affected, as long as it is only one of them and not both.

4.2 Fault Attack Categories

Adversarial faults may be introduced into the cryptosystem in a variety of ways. The spatial resolution with which an attacker is able to carry out the procedure depends largly on the type of fault attack, which may be from one of the following three categories.

Non-invasive Fault Induction In a non-invasive setting the mechanical integrity of the device under attack and its shell is unharmed. This means essentially that no attempt has been made to remove any protective coating of the chip, neither the regular IC packaging, nor an additional layer of tamper-proof coating. Any method of fault induction under this category would be limited to either the device's electrical terminals or electromagnetic radiation.

Invasive Fault Induction When the device's mechanical integrity has been severed, e.g. for modification of the electric circuit or circumvention of anti-tampering countermeasures, we speak of an invasive attack. This method gives the attacker the highest level of control over the fault insertion, but it also carries the highest risk of accidentally erasing key material in the process, for example by triggering a countermeasure such as those mentioned in the beginning of this section.

Semi-invasive Fault Induction The hybrid approach of removing the packaging material of the chip but keeping the circuit intact is characteristic of a semi-invasive attack. With access to the chip's surface, an attacker may use optical ionization sources to trigger transient faults in the device with high spatial resolution.

Advanced attacks have been demonstrated under laboratory conditions, e.g. in [BOS03] where an electric spike generator was used to influence the power supply with precise control. Also, Skorobogatov reported on semi-invasive techniques such as optical fault inductions [SA02], where a high-energy light source, e.g. a camera flash or a laser pointer, were used to induce single bit-flips in SRAM cells of a de-packaged micro-controller. These advanced attacks do not require a great deal of sophistication or expense, yet are very powerful.

4.3 Abstract Error Model

Our aim is to abstract the effect of a fault induction (the error) from its method, in order to obtain an analytical model, which we can then use to derive effective countermeasures. We thus model a successful fault induction as the additive distortion of data or logic values in the device.

Depending on the concrete function of the area of the device which is affected by the fault, there are two principle ways to characterize the error. A logical error is a bitwise distortion of the data, usually modeled as the exclusive OR of data and error, i.e. $\tilde{x} = x \oplus e_x$, while arithmetical errors admit the propagation of carries up to the range limit: $\tilde{x} = x + e \bmod 2^k$, where k is the width of the data path. The former is appropriate for storage dominated devices (register files, RAM, flip-flops, etc.), the arithmetic error model is more useful for arithmetic circuits such as adders and multipliers.

The multiplicity of an error is an important parameter with consequences for the performance of error detection strategies. In information theory the weight of a code word denotes the number of non-zero symbols. Similarly, the distance between two code words denotes the number of symbols in which the two values differ.

Logical Errors can be modeled in exactly the same way that transmission errors in basic coding theory are modeled. The weight of an error term e_x is its Hamming weight (short: $\mathrm{HW}(e_x)$) and the distance between two codewords x, y is the Hamming weight of their difference $\mathrm{HW}(x - y)$.

Arithmetical Errors are modeled slightly different, due to the influence of carry propagation. The binary arithmetic weight (BAW) of an integer N is defined as the number of non-zero coefficients in the minimal non-adjacent representation of $N = \sum_i^k n_i 2^i$ with $n_i \in \{-1, 0, 1\}$.

Due to the arithmetic structure of public-key cryptographic primitives addressed in this dissertation, we model all data-path errors as arithmetical errors and all errors on the control logic as logical errors. As a consequence of the adversarial nature of fault attacks, our error model is not limited to any error patterns of a specific weight, nor does it assume certain errors to occur more frequently than others.

Chapter 5

Cyclic Codes and Homomorphic Embedding

In this chapter we introduce a technique for fault-tolerant finite field arithmetic with applications to public-key cryptosystems defined over finite abelian groups (usually the group of points on an elliptic curve or the multiplicative subgroup of a finite field). Since elliptic curve based systems are also defined over finite fields, fault-tolerant finite field arithmetic is therefore at the heart of tamper-resistant public-key cryptography.

We propose *scaled embedding* as a new approach to fault tolerant finite field arithmetic that is based in principal on two important classes of linear codes, which are arithmetic codes and binary cyclic codes. The usefulness of these codes for our purposes stems from the fact that they possess the same basic arithmetic structure as most public key cryptographic schemes. Public-key arithmetic is commonly based on integer and binary polynomial rings and fields, most notably in *Elliptic Curve Cryptography*. The theories of both classes of codes contain a significant amount of overlap, which suggests a unified treatment. It is possible and useful to view the encoding of operands along with their arithmetic operations as a ring homomorphism.

Under a ring homomorphism elements from the non-redundant ring or field $\mathcal{F}$

can be embeddded into a larger, redundant ring $\mathcal{R}$ by means of multiplication with a constant scaling factor. While the techniques that we introduce in the following sections may not always adhere to the strict definition of cyclic codes, a less stringent definition allows a more flexible choice of scaling factors. When robustness with respect to minimum distance is required, one can always fall back on the special case of cyclic codes, for which a designed distance metric can be calculated.

In our scheme, arithmetic operations executed in $\mathcal{R}$ preserve the operations that otherwise would have to be executed without redundancy in $\mathcal{F}$. This redundancy can be utilized in every step of the computation to detect errors caused either by transient faults due to circuit crosstalk and radiation, or by malicious fault insertion from an adversary. Our method serves two mutually beneficial purposes: on one hand the redundancy of the larger ring can be used for error detection, which is our main objective. On the other hand, the particular low-weight representation of the ring modulus lends itself particularly well to the efficient direct method for modular reduction, based on shifts and additions / subtractions. The negative performance impact of the larger encoded operands is therefore mitigated to some degree.

Our method constitutes a generalization of classic cyclic codes for fault-tolerant arithmetic. This generalization is what enables us to obtain a larger choice of code parameters, thereby allowing hardware implementations with flexible time-space tradeoffs.

5.1 Introduction

Finite field arithmetic operations such as addition and multiplication form the foundations on which most cryptosystems are built. Our goal in this chapter is to define a redundant representation of a finite field along with its associated arithmetic operations, which (a) preserves the original field operation and (b) allows the detection of

errors which may arise due to either random or malicious insertion of faults into the computation.

Our intention is to devise an error detection scheme which utilizes the rich mathematical structure of finite fields and rings. For this we want to use a transformation through which we introduce redundancy in our representation and thereby gain error detection capabilities. We would like to be able to map from a ring or field $\mathcal{F}$ to a redundant ring $\mathcal{G}$, while preserving the arithmetic. Here $\mathcal{F}$ may be, for example, the ring $\mathbb{Z}_m$ of integers modulo m. Formally the elements of $\mathbb{Z}_m$ are cosets of the form $a+m\mathbb{Z}$, but it is usual and customary to simply refer to them as a, where $0 \leq a < m$.

We need a transformation function $\phi : \mathcal{F} \to \mathcal{G}$ which maps between the additive identities and preserves the addition and multiplication operations in the rings (or fields) $\mathcal{F}$ and $\mathcal{G}$, and thus forms a ring homomorphism:

Definition 5.1.1. Let $\mathcal{F}$ and $\mathcal{G}$ be two finite rings. A ring homomorphism is a function $\phi : \mathcal{F} \to \mathcal{G}$, which satisfies the following properties:

1. Addition is preserved: $\phi(a + b) = \phi(a) + \phi(b)$, for all $a, b \in \mathcal{F}$

2. Multiplication is preserved: $\phi(ab) = \phi(a)\phi(b)$, for all $a, b \in \mathcal{F}$

Example 5.1.1. A function $\phi : \mathbb{Z}_{16} \to \mathbb{Z}_{20}$ becomes a ring homomorphism via $a \mapsto 5a$.

There are two potential strategies to achieve fault tolerant arithmetic using ring homomorphisms. They are based on the difference between the two major classes of codes: separable and non-separable. The former ones are also called systematic codes, since each codeword can be split into an information part (the original data in unchanged format) and a redundant part which contains extra digits, the check-symbols. For non-separable codes the roles of information- and check-symbols are not clearly defined, since encoding procedures tend to 'spread' the information over

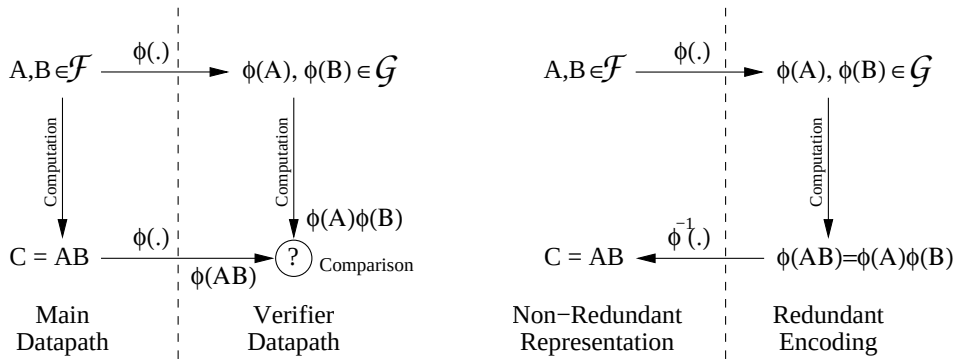
multiple digits in the resulting codeword. We differentiate between the following two principal strategies:

- A homomorphism ϕ which is not one-to-one may be used to create an additional verifier datapath besides the original main data path. It mimics all computations on the main data path using the homomorphic images of the operands obtained through the encoding $\phi(\cdot)$ (cf. Fig. 5.1(a)). At the end of the operation the image $\phi(c)$ of the result from the regular main data path is compared to the output of the verifier data path. This strategy is somewhat similar to that used in parity prediction circuits, e.g. in [RH02], although a bit more general.

- If ϕ is one-to-one (i.e. $\phi(a) \neq \phi(b)$ whenever $a \neq b$) then we speak of *embedding* $\mathcal{F}$ into $\mathcal{G}$, and the difference in cardinality establishes the amount of redundancy present in the embedding. After mapping all operands from $\mathcal{F}$ to $\mathcal{G}$ via the homomorphism $\phi(\cdot)$, all computation is carried out in $\mathcal{G}$. The integrity of the mapping is verified at intermittent check-points between operations. At the very end of the computation the result is converted back to $\mathcal{F}$ via the decoding function $\phi^{-1}(\cdot)$ (cf. Fig. 5.1(b)), which exists when ϕ is one-to-one.

5.1.1 Separate Codes

The main idea of this approach is to have two individual data paths perform equivalent operations on different, yet related, sets of data. The original data path is unchanged from the non-redundant version, while the redundant data path may be smaller, operating on images of the original data, the check-symbols, which are usually not unique. The smaller bitlength representation of the check-symbols implies that the mapping function $\phi(\cdot)$ is not one-to-one, and $|\mathcal{F}| > |\mathcal{G}|$. If ϕ is a ring homomorphism, it preserves the arithmetic of the original data in the image. We can thus compare



(a) Separate Encoding (b) Homomorphic Embedding

Figure 5.1: Two Strategies for Error Detection Under a Ring Homomorphism

the resulting image $\phi(a) \circ \phi(b)$ to the image of the result of the operation $\phi(a \circ b)$ and check for errors.

The following example illustrates this approach.

Example 5.1.2. Let $\mathcal{F} = \mathrm{GF}(p)[x]/(x^2-1)$ the polynomial quotient ring with $a(x) = a_1x + a_0 \in \mathcal{F}$ an element. Let further $\mathcal{G} = \mathrm{GF}(p)$. The mapping $\phi : \mathcal{F} \to \mathcal{G}$ becomes a ring homomorphism via polynomial evaluation at $x = 1$, i.e. $\phi(a(x)) = a(1) = a_1 + a_0 \bmod p$, since it satisfies the properties in Definition 5.1.1. It preserves arithmetic operations on two elements $a = a_1x + a_0$ and $b = b_1x + b_0$, such as addition

$$\begin{aligned} \phi(a+b) &= \phi((a_1+b_1)x + (a_0+b_0)) \\ &= (a_1+a_0) + (b_1+b_0) \\ &= \phi(a) + \phi(b) \end{aligned}$$

and multiplication

$$\begin{aligned}
\phi(ab) &= \phi(a_1b_1x^2 + (a_0b_1 + a_1b_0)x + a_0b_0) \\
&\equiv \phi((a_0b_1 + a_1b_0)x + (a_1b_1 + a_0b_0)) \pmod{x^2 - 1} \\
&= a_0b_1 + a_1b_0 + a_1b_1 + a_0b_0 \\
&= (a_1 + a_0)(b_1 + b_0) \\
&= \phi(a)\phi(b) \ .
\end{aligned}$$

An implementation of this strategy would require only the addition of a small secondary arithmetic data path to the existing primary ALU. Another example for this strategy is the use of arithmetic residue codes. These are popular in traditional fault-tolerant systems design and detect any single arithmetic error $\pm 2^i$.

Although this approach appears interesting, its merits are limited. While residue codes work nicely for regular integer arithmetic, the finite field and ring arithmetic of most public-key cryptosystems introduces additional constraints which are hard to overcome. For example, the natural mapping from a polynomial quotient ring to its coefficient ground field that works so nicely in Example 5.1.2 is the exception rather than the norm (notice that we evaluated at a root of the modulus $x^2 - 1$). Cryptographic fields are either large prime fields or binary extension fields of prime degree, for which either no such natural subfield mappings exist[1], or their usefulness is severely limited.

5.1.2 Non-separate Codes

This leads us to an alternative approach to defining codes for fault tolerant arithmetic, which is again based on homomorphic mapping. This time, however, the ring $\mathcal{F}$ is mapped into a larger ring $\mathcal{G}$, and arithmetic is only performed in $\mathcal{G}$ instead of in both

[1] With the notable exception of *optimal extension fields (OEF)*.

rings in parallel (cf. Fig. 5.1(b)). We speak of embedding $\mathcal{F}$ into $\mathcal{G}$ and using the additional redundancy for error detection.

One may be tempted to exploit the natural embedding provided by the field / subfield relationship of field extensions. For instance, we may carry out arithmetic in a finite field $\mathcal{F} = \mathrm{GF}(q^k)$ by embedding all operands into an extension $\mathcal{G} = \mathrm{GF}((q^k)^m)$. This method has the advantage of carrying the same field operations, i.e. addition, multiplication, inversion. To construct such an embedding we need a function mapping elements from $\mathcal{F}$ to $\mathcal{G}$, i.e. an injective map $\phi : \mathcal{F} \mapsto \mathcal{G}$. However, having both the domain and the range of the mapping to be fields may prove to be too restrictive, as the smallest extension would already double the size of the representation. As an alternative we may define the mapping from a field $\mathcal{F} = \mathrm{GF}(q^k)$ to a ring $\mathcal{R} = \mathrm{GF}[x]/r(x)$, with $\deg(r(x)) > k$, which offers more flexibility in choosing suitable parameters.

As described in the previous section the mapping function $\phi(\cdot)$ needs to represent a homomorphism and preserve the addition and multiplication operations. Incidentally, this is the same principle behind cyclic codes, which also possess the arithmetic structure of finite rings. Unfortunately, the precise definition of a cyclic code in the literature is relatively restrictive, which makes it hard to find codes that allow the embedding of cryptographically significant prime degree extension fields. By changing the definition slightly we can find a similar class of codes with a much larger set of parameters, in which we can embed these fields. But before we talk about these changes in definition, we would like to start with a brief review of cyclic codes. Also, we will talk about cyclic arithmetic codes which represent the integer analogy to cyclic codes.

5.2 Review of Relevant Coding Theory

In coding theory the alphabet over which codes are defined is typically chosen as a finite field $\mathbb{F}_q = \mathrm{GF}(q)$. The code itself is a subset or, in the case of linear $[n,k]_q$ codes, a k-dimensional subspace of the n-dimensional vector space $\mathbb{F}_q^n$. Codewords are usually represented as ordered n-tuples $c = (c_{n-1}, c_{n-2}, \ldots, c_0)$, with $c_i \in \mathrm{GF}(q)$.

5.2.1 Cyclic Codes

Cyclic codes are a family of linear codes with a rich algebraic structure, which is tremendously helpful in their analysis. It is therefore customary to associate with each codeword $c = (c_{n-1}, c_{n-2}, \ldots, c_0)$ the code polynomial $c(x) = c_{n-1}x^{n-1} + c_{n-2}x^{n-2} + \ldots + c_0$ of degree less than n.

The distinct property of cyclic codes that sets them apart from other linear codes is their invariance under cyclic shifts.

Definition 5.2.1. The *cyclic shift* $c' = c \circlearrowleft 1$ of the codeword c by one position to the left denotes the q-ary n-tuple $(c_{n-2}, c_{n-3}, \ldots, c_0, c_{n-1})$ with its associated polynomial $c'(x) = c_{n-2}x^{n-1} + c_{n-3}x^{n-2} + \ldots + c_0x + c_{n-1}$. A linear code C is a *cyclic code* if for any codeword c in C, $c' = c \circlearrowleft 1$ is also in C.

It is easy to see that the polynomial $c'(x)$ is equivalent to the product $xc(x)$ mod $x^n - 1$, thus establishing the correspondence

$$c \circlearrowleft 1 \longleftrightarrow c(x) \mapsto xc(x) \bmod x^n - 1 \ .$$

The cyclic shift $c'(x) = c(x) \circlearrowleft i$ by i positions is therefore equivalent to computing the remainder modulo $x^n - 1$ of the product $x^i c(x)$: $c'(x) = c_{n-i-1}x^{n-1} + c_{n-i-2}x^{n-2} + \ldots + c_{n-i+1}x + c_{n-i}$. Due to the finite length n of codewords it is straightforward to see that a right cyclic shift can be expressed as a series of $n-1$ left cyclic shifts.

By linearity, any scalar multiple of a code polynomial $ac(x)$, where $a \in \mathbb{F}_q$, is also in C. Accordingly, any linear combination of scalar multiples and cyclic shifts of codewords, again yields a codeword in C, e.g.

$$(a_0 + a_1 x + a_2 x^2)c(x) = a_0 c(x) + a_1 x c(x) + a_2 x(xc(x)) \ .$$

is a codeword (when reduced modulo $x^n - 1$) whenever $c(x)$ is. This provides us with a one-to-one correspondence between cyclic codes of length n over $\mathbb{F}_q$ and the ideals I of the polynomial quotient ring $\mathbb{F}_q[x]/(x^n - 1)$, which is essentially the well-known theorem first recognized by Eugene Prange in 1957 [Pra57].

In order to find cyclic codes, we briefly review an algebraic formulation of the problem. Using this formalism, cyclic codes of length n can be constructed by finding the monic divisors of $x^n - 1$ over GF(q). Specifically we need to construct a generator polynomial $g(x)$ of degree r, which will be used to encode data polynomials of degree less than $k = n - r$ by means of regular polynomial multiplication.

Definition 5.2.2. Let $d_1(x), d_2(x), \ldots, d_\ell(x)$ be the irreducible monic divisors of $x^n - 1$ over GF(q). Any product of these divisors $g(x) = \prod_{i=0}^{\ell} d_i(x)^{c_i}$, $c_i = \{0, 1\}$ can serve as a generator polynomial, since the only requirement for $g(x)$ is that it be a divisor of $x^n - 1$ itself.

For cyclic codes, the encoding process is easy. For a given generator polynomial of degree $r = n - k$ and a data polynomial $a(x)$ of degree less than k we obtain its codeword as $c(x) = a(x)g(x) \bmod x^n - 1$. For example, let $q = 2, n = 7$: the irreducible monic divisors of $x^7 - 1$ over GF(2) are

$$\begin{aligned} d_1 &= x + 1 \\ d_2 &= x^3 + x + 1 \\ d_3 &= x^3 + x^2 + 1 \ , \end{aligned}$$

and thus for $g(x) = d_1 d_3 = x^4 + x^2 + x + 1$, the ideal $C = \{a(x)g(x) : a(x) \in \mathbb{F}_2[x]\}$ is the [7,3]-Simplex code (the dual of the [7,4]-Hamming code) as a set of polynomials in $\mathbb{F}_2[x]/(x^7 - 1)$.

5.2.2 Arithmetic Codes

While linear cyclic codes were developed principally for the transmission of data, arithmetic codes were developed as a means to protect the arithmetic unit (AU) of mission critical computers, most notably the STAR computer at JPL [Avi67].

Non-separate arithmetic codes, often also referred to as AN codes, are conceptually similar to cyclic codes in many aspects, although there are some fundamental differences. Both types of codes form an ideal in a ring, but for arithmetic codes this ring is the integers modulo m, $\mathbb{Z}_m$, as opposed to the previously mentioned polynomial quotient ring for non-arithmetic cyclic codes. Codewords of both types may be represented as n-tuples, but must be interpreted differently due to the difference in arithmetic structure. Specifically, codewords of a cyclic code are n-tuples over the finite alphabet $\mathbb{F}_q$ that may be represented as polynomials, but the codewords of an arithmetic code are elements of the integer ring $\mathbb{Z}_m$, and the components of the n-tuples are digits in base-q representation. Apart from a different notion of distance and weight of codewords (cf. Section 4.3), this has consequences for the propagation of single bit errors in the representation. Despite similar representation an error in a single component of an arithmetic codeword may affect other digits of the result of an addition, while for a non-arithmetic code the error is constrained to the same digit position.

We will now give a precise definition for arithmetic AN codes.

Definition 5.2.3. [Gor87] Let $m \in \mathbb{Z}^+$. An *arithmetic AN code* is a subgroup $C = \{AN | 0 \leq N < M\}$ of $\mathbb{Z}_m$, where $m = AM$, for some $A, M \in \mathbb{Z}^+$.

According to this definition the arithmetic AN code C is the ideal in $\mathbb{Z}_m$ generated by A, and M is the number of codewords in C.

Definition 5.2.4. For $m = q^n - 1$, the *left cyclic shift* of a q-ary n-tuple $c = (c_{n-1}, c_{n-2}, \ldots, c_0)$ in $\mathbb{Z}_m$ is the n-tuple $c' = (c_{n-2}, c_{n-3}, \ldots, c_0, c_{n-1})$. As before, we say a code is *cyclic* if the left cyclic shift of any codeword is again a codeword.

It is straightforward to see that this representation is equivalent to the product $qc \bmod m$, thus establishing the correspondence

$$c \circlearrowleft 1 \longleftrightarrow c \mapsto qc \bmod m \ .$$

As usual, a right cyclic shift can be represented as a series of $n-1$ left cyclic shifts. One could also define a left cyclic shift of a codeword by i positions, thus obtaining the general correspondence

$$c \circlearrowleft i \longleftrightarrow q^i c \pmod{q^n - 1}$$

or

$$c \circlearrowleft i \longleftrightarrow (c_{n-i-1}, c_{n-i-2}, \ldots, c_0, c_{n-1}, \ldots, c_{n-i+1}, c_{n-i}) \ .$$

Theorem 5.2.1. *We immediately observe that any arithmetic AN code is cyclic in the sense of Definition 5.2.4.*

Example 5.2.1. We can construct a cyclic arithmetic code based on the factorization of $m = 2^n - 1$. Let $n = 6$, then $m = AM$ factors into $A = 9$ and $M = 7$. This AN code consists of the 7 codewords from the set $C = \{0, 9, 18, 27, 36, 45, 54\}$, represented as the binary 6-tuples

$$\begin{array}{lll} (0,0,0,0,0,0) & (0,0,1,0,0,1) & (0,1,0,0,1,0) \\ (0,1,1,0,1,1) & (1,0,0,1,0,0) & (1,0,1,1,0,1) \\ (1,1,0,1,1,0) & & \end{array}$$

and can be used to embed the prime field $\mathbb{Z}_7$ in $\mathbb{Z}_{63}$.

Remark. Due to the Mersenne-like form of the ring modulus there exist two equivalent binary encodings of the zero codeword, the all-zero and the all-one bit vector. Although the latter is strictly speaking not a codeword, it may arise as a result of the addition of two codewords.

5.2.3 Cyclic Code-Based Homomorphic Embedding

We can now begin to describe a connection between cyclic codes (in both senses) and ring homomorphisms. For example, the arithmetic AN code of Example 5.2.1 can be viewed as the image of the homomorphism $\phi : \mathbb{Z}_7 \to \mathbb{Z}_{63}$ via $a \mapsto 9a$. Analogously, the dual of the Hamming code can be viewed as an embedding of GF(8) inside $\mathbb{F}_2[x]/(x^7 - 1)$ via $\phi : (a_2, a_1, a_0) \mapsto (a_2x^2 + a_1x + a_0)(x^4 + x^2 + x + 1)$. In both above cases, while the additive structure is preserved in the ring, field multiplication is not preserved under the product operation of the target ring, due to the extra multiplicative factor introduced by encoding with the generator, i.e.

$$\phi(a \cdot b) = g \cdot a \cdot b \neq g^2 \cdot a \cdot b = \phi(a) \cdot \phi(b) \ .$$

This can be corrected in several ways, e.g. with an alternative definition of the product operation, as we shall see a bit later on.

Choosing cyclic code-based embedding for error detection has an extremely useful side benefit: by embedding the finite field into a quotient ring, arithmetic operations are no longer subject to modular reduction by the field defining irreducible polynomial or modulus. Any result of an arithmetic operation with a representation that exceeds n digits, can be reduced very efficiently due to the congruences $x^n \equiv 1$ and $q^n \equiv 1$. Low-weight ring and field moduli are well studied in applied cryptography: The finite fields recommended by NIST for the elliptic curve digital signature algorithm (EC-DSA [oSN00]) are defined by low-weight primes and irreducible trinomials or pentanomials. In [ÖSS04], modulus scaling was used to obtain low-weight ring moduli

for efficient scaled modular arithmetic. The authors, however, did not make use of the redundancy for error detection, which is the objective in this dissertation.

5.3 Cryptographically Significant Finite Fields

It is our goal to embed a finite field GF(q) into a larger ring, for q some prime power. In the case where the target is to be a binary cyclic code, i.e. $q = 2^k$, this means that $x^n - 1$ needs as a factor some degree k irreducible polynomial over GF(2). Similarly, in the case of arithmetic codes, we require $2^n - 1$ to contain a large prime factor p of bitlength $|p| \approx k$ bits.

At this point it seems as if all we need to do is to search for some suitable parameters n, k, by way of factorization of the ring modulus. Unfortunately, with respect to specific applications such as Elliptic Curve Cryptography (ECC) not all finite fields are suitable choices for defining secure elliptic curve groups. Binary fields GF(2^k) with k composite are believed to be susceptible to a class of attacks based on *Weil descent* [GHS02]. Without going too much into the details, Weil descent allows one to map the elliptic curve discrete logarithm problem (EC-DLP), upon which the security of elliptic curve cryptosystems is based, to a related problem on a hyper-elliptic curve of larger genus, but much smaller subfield. This gives rise to index-calculus attacks on the EC-DLP for which the time complexity depends largely on the field size. In short, composite degree field extensions may reduce the security of elliptic curve cryptosystems.

As a result we want to find cyclic $[n, k]_2$ codes where $x^n - 1 \pmod 2$ factors into a large irreducible polynomial of *prime* degree k and some other smaller polynomials. For fields with an extension degree in the range that is of interest for elliptic curve cryptography, i.e. $130 \leq k \leq 500$, there are exactly ten suitable choices, and each carries a large amount of redundancy, in fact, they all have $n = 2k + 1$ as shown

in Table 5.1, which means that codewords are more than twice as long as the field elements which we would like to embed. The situation is only slightly better with cyclic arithmetic codes which are defined in a similar way. In this case the field modulus and the amount of redundancy is determined by the integer factorization of $2^n \pm 1$.

The advantage of using cyclic codes for embedding is that we can make certain statements about the worst-case minimum distance (designed distance) of the code, and therefore about its error detection capability, based on the BCH bound [van92]:

Theorem (BCH bound). *Let C be a q-ary $[n, k]$ cyclic code with generator polynomial $g(x)$. Let m be the multiplicative order of q modulo n ($\mathrm{GF}(q^m)$ is thus the smallest extension field of $\mathrm{GF}(q)$ that contains a primitive n-th root of unity). Let α be such a primitive n-th root of unity in $\mathrm{GF}(q^m)$. Select $g(x)$ to be a minimal-degree polynomial in $\mathrm{GF}(q)[x]$ such that $g(\alpha^b) = g(\alpha^{b+1}) = \ldots = g(\alpha^{b+\delta-2}) = 0$ for some integers $b \geq 0$ and $\delta \geq 1$, so $g(x)$ has $(\delta - 1)$ consecutive powers of α as zeros. It follows that the code defined by $g(x)$ has minimum distance at least δ.*

The designed distance δ given by this theorem (and listed in Table 5.1 for the code parameters mentioned earlier) is not necessarily a tight bound. The true minimum distance is often larger, so this theorem only gives a lower bound.

n	263	359	383	479	503	719	839	863	887	983
k	131	179	191	239	251	359	419	431	443	491
δ	8	9	9	13	9	11	11	9	9	11

Table 5.1: Cyclic Codes with Prime Degree Irreducible Divisors

5.4 A Generalization of Cyclic Codes

In order to mitigate this scarcity of useful code parameters, we move from purely cyclic codes to a new and more general family of codes with a slightly different definition. This departure from cyclic codes was inspired by the techniques for scaled modular arithmetic described in the reference mentioned earlier [ÖSS04]. By relaxing the notion that the code has to be strictly cyclic, we can find rings in which we can embed fields of nearly arbitrary cardinality, with flexible trade-offs between field size and amount of redundancy. We thus obtain a generalized interpretation of codes with homomorphic structure, in which arithmetic and cyclic codes constitute special cases. The ring modulus can now take any form, although for performance reasons we prefer a "Mersenne-like" form $q^n \pm u$ ($x^n - u(x)$ in the polynomial case). Since each arithmetic operation in the ring is followed by a modular reduction step, it is desireable to keep the complexity of modular reduction low. Therefore, the best choices for u (and likewise $u(x)$) are typically those that are small (resp., of small degree) and of low weight. [2]

For lack of better terminology we shall name this class of codes 'accumocyclic'. This name was chosen for two reasons: (a) to indicate the close relationship to cyclic codes and (b) to distinguish its behavior with regard to shifted codewords. Upon a single shift of to the left, the ring defining polynomial mandates that the most significant coefficient does not just simply wrap around back to the least significant position, but that it is multiplied with $u(x)$ and added to the remaining shifted coefficients, i.e. $c_{n-1}u(x)$ is accumulated in the lower part of the codeword. We define this concept formally as an accumocyclic shift with respect to $u(x)$.

Definition 5.4.1. Let $u = (u_{n-1}, u_{n-2}, \ldots, u_0)$ denote the q-ary n-tuple associated

[2] Since implementation of q-ary arithmetic in digital circuits is more complex and less common for bases $q \neq 2$, we shall focus on codes with a binary representation throughout the remainder of this chapter.

with the polynomial $u(x)$ over GF(q). The *accumocyclic shift* $c' = c \overset{+}{\circlearrowleft}_u 1$ of the codeword c by one position to the left with respect to u denotes the q-ary n-tuple resulting from the linear combination of $(c_{n-2}, c_{n-3}, \ldots, c_0, 0)$ and the scalar multiple $c_{n-1}u$ over GF(q), i.e. $c' = (c_{n-2} + c_{n-1}u_{n-1}, c_{n-3} + c_{n-1}u_{n-2}, \ldots, c_0 + c_{n-1}u_1, c_{n-1}u_0)$.

Derive an associated polynomial $c'(x) = (c_{n-2}+c_{n-1}u_{n-1})x^{n-1}+(c_{n-3}+c_{n-1}u_{n-2})x^{n-2}+ \ldots + (c_0 + c_{n-1}u_1)x + c_{n-1}u_0$. It is easy to see that this polynomial is equivalent to the product $xc(x) \mod x^n - u(x)$, thus establishing the correspondence

$$c \overset{+}{\circlearrowleft}_u 1 \longleftrightarrow c(x) \mapsto xc(x) \bmod x^n - u(x) \ .$$

Accumocyclic shifts by i positions and those to the right can be defined in a similar manner. We can now define accumocyclic codes as codes invariant under accumocyclic shifts according to Definition 5.4.1:

Definition 5.4.2. Let C a linear block code of length n over GF(q), viewed as a vector subspace of GF(q)[x]. Further, let $u(x)$ denote a polynomial of some degree $m < n$, specific to the code. We say the code C is *accumocyclic* with respect to u if and only if any accumocyclic shift of a codeword $c \in C$, according to Definition 5.4.1 again yields a codeword $c' \in C$.

Observe that, due to linearity, any addition of two codewords a and b modulo q yields another codeword. Even though this new class of codes is not cyclic according to Definition 5.2.1, its codes possess certain useful properties which are of interest to us, mainly the possibility to embed cryptographically significant fields and efficient modular arithmetic. We can obtain a broad range of suitable field and scaling factor parameters via factorization of the ring moduli. For illustration we have included a selection of parameters in Tables A.1, A.2 and A.3 in the Appendix.

Example 5.4.1. Let $n = 18$, $k = 13$, $q = 2$, and $u(x) = x + 1$. One can easily verify that $g(x) = x^5 + x^2 + 1$ is a polynomial such that $g(x)|x^n + u(x)$. The accumocyclic

code $C = \{g(x)a(x) : a(x) \in \mathrm{GF}(2)[x]\}$ generated by $g(x)$ thus forms an ideal in the quotient ring $\mathrm{GF}(2)[x]/(x^{18} + x + 1)$. It is the set of all codewords $c(x) = g(x)a(x)$ for data polynomials $a(x)$ of degree less than k.

We want to illustrate the difference to classical cyclic codes. Consider the codeword $c(x) = x^{17} + x^{14} + x^9 + x^5 + x^4 + 1$. In contrast to a cyclic code, a cyclically shifted codeword $c'(x) = a(x) \circlearrowleft 1 = x^{15} + x^{10} + x^6 + x^5 + x + 1$ is not in C, since it is not a multiple of $g(x)$. Yet, an *accumocyclic shift* of a codeword with respect to $u(x)$

$$\begin{aligned} c''(x) &= c(x) \overset{+}{\circlearrowleft}_{x+1} 1 \\ &= xa(x) \pmod{x^{18} + x + 1} \\ &= x^{18} + x^{15} + x^{10} + x^6 + x^5 + x \pmod{x^{18} + x + 1} \\ &= x^{15} + x^{10} + x^6 + x^5 + 1 \end{aligned}$$

produces a new codeword, since $c''(x) = x^{15} + x^{10} + x^6 + x^5 + 1 = g(x)(x^{10} + x^7 + x^4 + x^2 + 1)$.

5.5 Homomorphic Embedding in Rings

Consider the case when a field $\mathcal{F}$ is embedded in a larger ring $\mathcal{R}$. Embedding works by mapping any element $a \in \mathcal{F}$ to an element $\phi(a) \in \mathcal{R}$ via a suitable embedding function, i.e. ϕ is an injective ring homomorphism. Therefore we can carry out all arithmetic operations originally defined for $\mathcal{F}$ in $\mathcal{R}$ instead. At the end of the computation it will be required to use the inverse mapping $\phi^{-1} : \mathcal{R} \to \mathcal{F}$ to transform the result back from the ring to the field. In the following we will define explicit mapping functions to make our approach more transparent. We investigate two different methods for embedding: *Basic* and *Idempotent* scaled embedding. Both methods tie into the theory of cyclic codes for certain parameter selections, but can also be seen as a generalization of the concept without making claims about any sort of minimum

distance metric. We already indicated this link to coding theory in Section 5.2.3. We would like to point out that both methods apply equally well to integer *and* polynomial rings. For sake of simplicity, however, we will refrain from making explicit distinctions in notation unless such a distinction is required.

The idea of scaled embedding is closely related to the modulus scaling techniques set forth in [ÖSS04]. There a scaling factor s was applied to the field modulus p to obtain a scaled modulus $m = s \cdot p$ of Mersenne-like form that allows efficient shift-and-subtract modular reduction. The redundancy introduced by scaling the modulus allows us to implement an error detection scheme. A naïve direct mapping $\phi(a) = a$, however, does not provide error detection capabilities, since mapped elements do not form an ideal over the ring and errors would be indistinguishable from data. *Scaled embedding*, on the other hand, multiplies operands by a generator value g which may actually be distinct from the modulus scaling factor s. It effectively partitions the ring $\mathcal{R}$ into cosets, of which only one contains valid codewords. Error detection can therefore be based upon checking for membership in the correct coset. With respect to the choice of a suitable modulus scaling factor we strive to achieve two goals:

1. to select the target ring large enough to have sufficient redundancy for error detection purposes, i.e. an amount proportional to the length of the scaling factor, i.e. $\log_2 s$;

2. to obtain a ring modulus $m = p \cdot s$ for which an efficient reduction technique exists. This helps to offset some of the overhead in complexity that we incur by adding redundancy.

If the field $\mathcal{F}$ and its associated modulus (prime integer or irreducible polynomial) p are determined by the application, then the choices for a suitable scaling factor might be limited. If, however, the value of the modulus is not fixed, then one can choose a suitable pair (s, p) based on the required levels of security (modulus size)

and redundancy. The computationally most efficient moduli of Mersenne-like form have very small Hamming-weight, e.g. less than 5. Therefore, in order to find such a pair we let $m = 2^n \pm u$, with u small and $n = \lceil \log_2 p + \log_2 s \rceil$, be the preferred ring modulus and find a suitable field by way of factorization. Depending on the size of n factorization might take a long time, especially in the case of finding suitable prime fields.

For polynomial moduli the ideal form is a binomial $m(x) = x^n \pm 1$, as in cyclic codes, but other moduli $x^n \pm u(x)$ with small degree $u(x)$ are also conceivable. When $u(x) = 1$ then the reduction of partial products, e.g. during the shifting step of bit-serial multiplication, becomes trivial since the shift with reduction can be simplified to a bit rotation due to the equivalence $x^n \equiv \mp 1 \bmod p(x)$. The factorization of binomials is well studied, and there exist many efficient methods. What we are looking for specifically are large irreducible factors of prime degree. This is important mainly for applications in elliptic curve cryptography, as previously discussed.

5.5.1 Basic Scaled Embedding

Once we have found suitable parameters (p, s) for the modulus and its scaling factor, we can encode the input operands by means of multiplication with the generator value g. For basic scaled embedding this is the same value as the modulus scaling factor s, i.e. $g = s$. The function $\phi_s(a) = g \cdot a$ maps an element a from $\mathcal{F}$ to $\mathcal{R}$. It provides error detection capabilities since all valid elements of $\mathcal{R}$ must be proper multiples of g. Note that while the mapping preserves addition, it does not preserve regular multiplication, i.e.

$$\phi_s(a \cdot b) = g \cdot a \cdot b \neq \phi_s(a) \cdot \phi_s(b) \ .$$

With an alternative definition[3] of the product operation of the ring, however, that implicitly absorbs the extra scaling factor, $\phi_s(\cdot)$ becomes a ring homomorphism with

[3]We use the symbol $\star$ to prevent confusion with regular multiplication.

respect to $(+, \cdot)$ and $(+, \star)$ operations, i.e. $\phi_s(a \cdot b) = \phi_s(a) \star \phi_s(b)$.

Definition 5.5.1 (g-absorbing multiplication)**.** Let $\mathcal{R}$ be a ring under $+$ and $\cdot$, $g \in \mathcal{R}$. Further, let $a, b \in \mathcal{F}$ and $A = \phi_s(a), B = \phi_s(b) \in \mathcal{R}$. Define $\star : \mathcal{R} \times \mathcal{R} \to \mathcal{R}$ via $A \star B = AB/g$, when this belongs to $\mathcal{R}$, otherwise $A \star B$ will be left undefined.

Observe that $\star$ is associative

$$(A \star B) \star C = A \star (B \star C)$$

and distributive

$$A \star (B + C) = A \star B + A \star C$$

when defined.

Therefore, multiplications in $\mathcal{R}$ are implemented using the $\star$ operation instead of regular ring multiplication, while addition in the ring remains the same. Since the value for g is constant for a specific modulus, division may be implemented more efficiently than in the general case. Algorithms for division by constants have been treated, for example, in [Par00].

Error Detection

As mentioned earlier, detection of errors can be based on computing the remainder of a division by g. A value of zero indicates that the operand is likely to be free of errors. We have to use caution here, because quite naturally our scheme cannot detect error patterns that are proper multiples of g. We apply a relatively simple error model to determine the error coverage of this method: We assume that errors only occur as additive terms on input operands and that the operation itself is fault-free. Consequently, the output is the sum of the correct result and another additive error term related to the input errors and the operation. While such an error model may be rather simple, we would like to make a point for its validity in the context of

fault-insertion attacks. From an adversarial point of view, the most accessible targets with high probability of success for introducing an error are storage elements like registers and SRAM memory cells. A glitch attack on such a bi-stable device, e.g. using optical fault induction with a focussed laser beam [SA02], is able to cause an error regardless of the exact point in time during which it is carried out (with respect to the clock interval). A glitch in a combinational part of a circuit will manifest itself as an error only if it reaches the next register in time for the next clock edge and if it does not significantly violate setup and hold time requirements.

We will now determine the conditions under which we can detect errors. Let $A = \phi_s(a)$, $B = \phi_s(b) \in \mathcal{R}$ denote the fault-free input operands of the multiplication operation $A \star B$ and C the fault-free result.

$$C = A \star B = (A \cdot B)/g \pmod{m} = ((g \cdot a \cdot g \cdot b)/g) \pmod{m} = g \cdot a \cdot b \pmod{m}$$

Furthermore let C' denote the result in the presence of additive error terms e_A, $e_B \in \mathcal{R}$ on the inputs:

$$\begin{aligned} C' &= (A + e_A) \star (B + e_B) \\ &= (g^2 \cdot a \cdot b + g(a \cdot e_B + b \cdot e_A) + e_A \cdot e_B)/g \pmod{m} \\ &= C + a \cdot e_B + b \cdot e_A + \frac{e_A \cdot e_B}{g} \pmod{m} \\ &= C + e_C \end{aligned}$$

Here $e_C \in \mathcal{R}$ is the resulting error on the output. Certain errors can be detected immediately during the division step of the $\star$ multiplication procedure, e.g. when the remainder $C' \bmod g \neq 0$, which means that g does not divide $e_A \cdot e_B$. There are two other non-trivial cases of potentially undetectable errors:

1. Certain one-sided errors, e.g. $e_A = 0,\ e_B \neq 0$, such that $e_C = a \cdot e_B \bmod m$, which are not detectable if $g | e_C$.

2. Some two-sided errors $e_A, e_B \neq 0$. Now we have $e_C = a \cdot e_B + b \cdot e_A + \frac{e_A \cdot e_B}{g} \bmod m$. The error is undetectable iff $g | e_C$.

The procedure for error detection is based on modular reduction of the operands with respect to the scaling factor g and checking for a non-zero remainder. Here it can be performed outside of the critical path of the computation. As long as there is no error in any of the previous operations, the result can immediately be used as the input for subsequent operations, while an error check is performed in parallel. The major problem we face with basic scaled embedding is the division step that is intrinsic to the $\star$ multiplication, since it adds to the critical path. Division is notoriously complex in hard- and software implementations unless the divisor is a constant of special form, which is not usually the case. In the next section we present a modification to the basic scaled embedding idea, which completely avoids the division step of $\star$ multiplication.

5.5.2 Idempotent Scaled Embedding

The division step of $\star$ multiplication in the basic scaled embedding scheme is required because both operands contain a multiplicative factor g which results in a square factor g^2 for the product. One way to avoid the extra g is to perform multiplication with only one scaled input, the other unscaled, but this introduces a host of other problems. First we would lose error detection capabilities in the unscaled operand, and secondly the product of two results from previous multiplications would again require division.

One elegant solution is to find a scaling factor that is idempotent with respect to the scaled modulus, i.e. $g \equiv g^2 \bmod m$. A class of non-separate arithmetic codes known as AN codes use the same encoding principle as scaled embedding and suffer from the same problem of an extra residue of the generator value. In [Pro89] Proudler introduced a class of idempotent AN codes that preserve addition *and* multiplication in the ring. These codes can therefore be used to form a ring homomorphism $\phi_i(\cdot)$

that avoids division altogether. A critical flaw of idempotent AN codes is, however, that a one-sided error, i.e. one that only appears in one of two input operands, will be masked in a multiplication with the other error-free operand due to the distributive law:

$$\begin{aligned} A' &= \phi_i(a) + e_A \\ B &= \phi_i(b) \\ A' \cdot B &= (g \cdot a + e_A)(g \cdot b) \pmod{m} \\ &= g^2 \cdot a \cdot b + g \cdot b \cdot e_A \pmod{m} \\ &= \phi_i((a + e_A)b) \end{aligned}$$

This flaw can be compensated for by extending from idempotent AN to idempotent AN+B codes. These were also introduced in [Pro89] and like AN codes derive their names from the encoding procedure. In addition to being scaled by the generator value g, a constant term c is added to the operands during encoding. AN+B codes exist whenever the ring modulus $m = p \cdot s$ and $\gcd(p, s) = 1$. Then we can construct the values g and c as orthogonal idempotents with respect to the modulus m as follows:

$$g = (s^{-1} \bmod p)\ s \tag{5.1}$$

$$c = (p^{-1} \bmod s)\ p \tag{5.2}$$

where

$$g^2 \equiv g \pmod{m}\ , \tag{5.3}$$

$$c^2 \equiv c \pmod{m} \text{ and} \tag{5.4}$$

$$g \cdot c \equiv 0 \pmod{m}\ . \tag{5.5}$$

Unlike AN codes, AN+B codes are no longer addition preserving. In the presence of a heterogeneous mix of addition and multiplication operations it is therefore necessary

to convert[4] operands back and forth between codes. Luckily this is a rather trivial exercise since both codes share the same generator g. We can re-define multiplication to implicitly handle the conversion steps by adding the constant term c to each operand before multiplication and subsequently subtracting it from the result. The difference to ordinary multiplication in the ring is indicated through the use of the $\star$ symbol:

Definition 5.5.2. Let $A = \phi_i(a)$ and $B = \phi_i(b)$ denote operands embedded in $\mathcal{R}$, where $\phi_i(x) = g \cdot x \bmod m$. Then addition in the ring is defined as usual and multiplication is re-defined as

$$\begin{aligned} A \star B &= (g \cdot a + c) \cdot (g \cdot b + c) - c \pmod m \\ &= g^2 \cdot a \cdot b + c \cdot g(a + b) + c^2 - c \pmod m \\ &= \phi_i(a \cdot b) = g(a \cdot b) \pmod m \end{aligned}$$

due to the equivalences defined in (5.3), (5.4) and (5.5).

We can thus define an idempotent ring homomorphism with respect to $(+, \star)$ and $(+, \cdot)$ operations as

$$\begin{aligned} \phi_i(0) &= 0, \\ \phi_i(a) + \phi_i(b) &= \phi_i(a + b) \text{ and} \\ \phi_i(a) \star \phi_i(b) &= \phi_i(a \cdot b)\ . \end{aligned}$$

Now a one-sided error e_A will not be masked anymore, provided that $s \nmid e_A$:

$$\begin{aligned} A' \star B &= (g \cdot a + c + e_A)(g \cdot b + c) - c \pmod m \\ &= g(a \cdot b) + e_A(g \cdot b + c) \pmod m \\ &\equiv e_A \bmod s \end{aligned}$$

[4]Note that conversion is only necessary at the boundary between heterogeneous operations like addition and multiplication. It can be omitted for homogeneous operations like modular exponentiation, which are based exclusively upon multiplication.

Once all computations have been performed the non-redundant result needs to be converted back from the ring to the field via the inverse homomorphism $\phi_i^{-1}(\cdot)$. This is achieved through modular reduction of the result with respect to the field modulus p.

We now have an efficient method for embedding a field into a larger ring with meaningful redundancy that we want to use for error detection purposes. Since valid codewords need to be proper multiples of the generator value $g \equiv 0 \bmod s$, an error check can be performed by computing the remainder of a division by s. An additive error e_R on the result will be detected as $e'_R = e_R \bmod s$, if it is not evenly divisible by s. Hence, if $e'_R = 0$, the result can be assumed free of errors with high probability. There may be cases, however, in which an error e_A remains undetectable. In the following we establish the probability of this happening.

Error Detection

We apply the same error model as before, which assumes that errors only occur at the input operands. In the presence of additive error terms we can model system behavior for addition as

$$\begin{aligned} A' &= A + e_A\ , \quad B' = B + e_B \\ A' + B' &= g(a \cdot b) + (e_A + e_B) \bmod m \\ e_{R+} &= e_A + e_B \end{aligned} \tag{5.6}$$

and for multiplication as

$$\begin{aligned} A' \star B' &= (g \cdot a + e_A + c) \cdot (g \cdot b + e_B + c) - c \bmod m \\ &= g(a \cdot b) + e_A(g \cdot b + c) + e_B(g \cdot a + c) + e_A \cdot e_B \bmod m \\ e_{R\star} &= e_A(g \cdot b + c) + e_B(g \cdot a + c) + e_A \cdot e_B \bmod m\ . \end{aligned} \tag{5.7}$$

From the reduction modulo s we obtain the detectable portion of the error term. In the case of addition (5.6) this is $e'_{R+} = e_A + e_B \bmod s$. A faulty result is undetectable if

$e_A \equiv -e_B \bmod s$. For simplicity we assume that the errors e_A and e_B are independent and identically distributed random variables from uniform. Thus the probability of an undetectable error is $1/s$.

An error occurring during multiplication will produce the term $e'_{R\star} = e_A + e_B + e_A \cdot e_B \bmod s$ which we obtained through application of the equivalences $g \equiv 0 \bmod s$ and $c \equiv 1 \bmod s$ to (5.7). We can find the probability of an undetectable error during multiplication using the following lemma:

Lemma. *Let X, Y be two independent and identically distributed random variables uniform over $[0, s-1]$ and let the event $A = \{(X = x, Y = y) : x+y+x\cdot y \equiv 0 \bmod s\}$. Then the probability of A occurring is $\Pr[A] = \Phi(s)/s^2$, where $\Phi()$ denotes the Euler totient function.*

Proof. We can rewrite the event A as follows: $A = \{(X = x, Y = y) : y = f(x)\}$, where $f(x) = -x \cdot (x+1)^{-1} \bmod s$. The function $f(x)$ will only be defined if the inverse of $x+1$ exists. For any given modulus s this is the case only for $\Phi(s)$ choices in the range $0 \leq x < s$. Hence, $f(x)$ is defined and has a value with probability $p_R = \Phi(s)/s$. Y takes on a specific value y with probability $p_L = 1/s$. The joint probability of the event A occurring is therefore $p = p_L \cdot p_R = \Phi(s)/s^2$, due to the independence of X (and hence $f(X)$) and Y. □

Here the event A stands for the occurrence of an undetectable error at the output of the multiplier. It is easy to see that the best error coverage can be obtained when the modulus scaling factor s is composite and large. We would like to re-iterate that the error detection mechanism requires a full modular reduction by s, which in the general case does not have a suitable special form as the scaled modulus m. While this might be viewed as a drawback, it should be noted that checking for errors can be done outside of the critical path (in hardware) or at regular intervals (software realization), while the main computation continues operation. As a matter of fact,

the regular field modulus p does not in general have a suitable special low Hamming-weight form either, such that the overhead due to error detection is easily offset by the efficient reduction modulo $m = s \cdot p$.

5.5.3 Error Correction Using Algorithm-Based Fault Tolerance

Quite naturally one would like to build an arithmetic architecture with the ability to also correct errors that occur during computation. For cyclic codes, syndrome decoding allows the correction of the most likely error pattern (assuming blind fault induction). It does not, however, give good results in the presence of burst errors, which would likely occur if an adversary tried to attack the computation. The reason is that in an adversarial setting one has to make worst case assumptions and cannot expect that decoding to the nearest codeword will successfully correct the error.

A different approach is the use of algorithm-based fault tolerance. The principal idea here is to keep valid input operands around until after the computation has finished and an error check determines a valid result. If the error check fails, the computation can be repeated (replayed) until a valid result is available. Alternatively, if the computation fails repeatedly, an alarm can be signaled and the operation canceled. The advantage of this method is clearly its robustness in the presence of transient burst errors. It does not matter which of all possible errors covered under a specific syndrome triggered the detection, when the computation can simply be repeated. Another advantage is the relatively low overhead that is required, which is mostly caused by the storage elements necessary to keep backup copies of operands. A potential disadvantage is that the method does not degrade gracefully, meaning that permanent faults due to stuck-at-0/1 errors can not be compensated for. Circuit defects thus render this method completely useless and do not help, for example, to increase the yield of circuit production.

5.6 Analysis and Discussion

We have presented a novel scheme for fault-tolerant finite field computation with applications in public-key cryptography. Homomorphic scaled embedding is practical and allows designers of cryptographic systems to add fault tolerance with moderate resource overhead. It provides adequate protection against transient faults of either random or adversarial nature under the assumption of reasonable limitations to the attacker's capabilities. This kind of protection is particularly important, due to the continuing success of fault-insertion attacks on cryptographic embedded systems.

Chapter 6

Robust Codes: Tamper Resistance Under a Stronger Error Model

In chapter 4 we introduced a general error model for cryptographic devices in adversarial situations. We left open the question of adversarial capability, i.e. the capacity of an attacker to successfully introduce faults into the system. In general this is a difficult property to assess, as much of it depends on the adversarial's motivation and resources, as well as the system's physical countermeasures.

In the previous chapter we implicitly assumed that the necessary temporal and spatial precision to cause errors of a particular bit-pattern were unavailable to the adversary. If they were, the proposed methods for arithmetic with encoded operands would fail, due to the linearity of the scheme; any linear combination of two codewords will be a codeword again, and therefore the attacker may simply choose to cause an error with the same pattern as a valid codeword. It would be relatively easy to generate a valid codeword, since (paraphrasing Kerckhoff's famous principle [Ker83]) one should always operate under the assumption that basic system parameters are known to the adversary, with the notable exception of key material.

Although linear codes may seem sufficient for the protection of public-key arith-

metic primitives today, advances in fault insertion methodologies may suddenly become available which provide the necessary precision for feasible attacks. In this chapter we address a scenario in which an attacker may introduce arbitrary error vectors reliably, with nearly arbitrary precision.

In light of such worst-case assumptions, a linear code based error detection scheme is bound to fail. We therefore turn our attention to non-linear schemes and develop a family of robust arithmetic codes with a very high error detection probability. It is capable of detecting errors of arbitrary weight, and the small portion of undetected errors is highly data dependent, which makes it nearly impossible to find successful error patterns without a priori knowledge of the data.

6.1 Introduction to Robust Codes

A family of systematic non-linear error detecting codes, termed 'robust codes', was derived from systematic linear block codes in [KT04]. Their use in symmetric ciphers like the AES has been proposed in [KKT04] and later refined in [KKT05]. The robustness of these codes is due to the much more uniform error detection capabilities these codes offer. Unlike linear codes, where the detection capability depends only on the weight of the error vector, for robust codes the probability $Q(e)$ of an undetected error e depends on the combination of error *and* data patterns.

Thus, in the case of a cryptographic key which is not known to the attacker a priori, a fault-injection attack is much more difficult to carry out than with a linear encoding scheme.

Robust codes can achieve optimality according to the minimax criterion, that is, minimizing over all (n, k) codes the maxima of the fraction of undetectable errors $Q(e)$ for $e \neq 0$. The following definition from [KT04] rigorously defines a particular class of non-binary codes.

Definition 6.1.1. Let V be a linear p-ary (n, k) code ($p \geq 3$ is a prime) with $n \leq 2k$ and check matrix $H = [P|I]$ with $\text{rank}(P) = n - k$. Then $C_V = \{(x, w)|x \in GF(q^k), w = (Px)^2 \in GF(q^r)\}$

For binary robust codes ($q = 2$) the definition is slightly different; instead of squaring the check-symbol w is obtained through cubing. We will give a more detailed introduction to robust non-linear block codes in Chapter 7.

While robust codes work well with symmetric ciphers that employ only little more than table look-ups, XORs and byte-wise rotations, they are virtually unusable within the finite field arithmetic structure that forms the basis of most public-key algorithms.[1]

In the following sections we will introduce a construction for robust arithmetic residue codes, inspired by the construction in [KT04]. Based on these codes we present a scheme for robust multi-precision arithmetic over the positive integers. Our scheme lends itself well for straightforward implementation of standard modular multiplication techniques, i.e. Montgomery or Barrett Multiplication, secure against active fault injection attacks.

6.2 Robust Arithmetic Residue Codes

As mentioned before, a class of non-linear systematic error detecting codes, so-called "robust codes", were proposed by Karpovsky and Taubin [KT04]. They achieve optimality according to the minimax criterion, that is, they minimize over all (n, k) codes the maxima of the fraction of undetectable errors $Q(e)$ for $e \neq 0$. While they are suitable for data transmission in channels with unknown characteristics, and also for robust implementation of symmetric-key cryptosystems with little arithmetic structure, they do not preserve arithmetic. We thus propose a new type of non-

[1] Optimal extension fields are an exception, see Chapter 7

linear arithmetic code, based on the concept of arithmetic residue codes. We define robustness as follows:

Definition 6.2.1. Let $C = \{(x, w)|x \in \mathbb{Z}_{2^k}, w = f(x) \in \mathbb{F}_p\}$ be an arithmetic single-residue code with a function $f : \mathbb{Z}_{2^k} \mapsto \mathbb{F}_p$ to compute the check symbol w with respect to the prime check modulus p of length $r = \lceil \log_2 p \rceil$ bits. A non-zero error $e \in \{(e_x, e_w)|e_x \in \mathbb{Z}_{2^k}, e_w \in \mathbb{Z}_{2^r}\}$ is masked for a message x, when $(x+e_x, w+e_w) \in C$, i.e. iff

$$f\left((x + e_x \bmod 2^k)\right) = f(x) + e_w \bmod 2^r \ . \tag{6.1}$$

The error masking probability for a given non-zero error is thus

$$Q(e) = \frac{|\{x|(x + e_x, w + e_w) \in C\}|}{|C|} \ . \tag{6.2}$$

We call the code C *robust*, if it minimizes maxima of $Q(e)$ over all non-zero errors. Total robustness is achieved for $\max_{e\neq 0}(Q(e)) = 2^{-r}$. We also call C ϵ-robust if it achieves an upper bound $\max_{e\neq 0}(Q(e)) \leq \epsilon \cdot 2^{-r}$, where ϵ is a constant much smaller than 2^r.

In the following we propose a class of non-linear single-residue arithmetic codes C_p based on a quadratic residue check symbol, which achieves ϵ-robustness. Since in practice total robustness is hard to achieve, we will from now on refer to ϵ-robustness simply as robustness.

Theorem 6.2.1 (Robust Quadratic Codes)**.** *Let C_p according to Definition 6.2.1, with $f(x) := x^2 \bmod p$. C_p is robust iff $r = k$ and $2^k - p < \epsilon$, and has the error masking equation*

$$(x + e_x \bmod 2^k)^2 \bmod p = w + e_w \bmod 2^k \tag{6.3}$$

Proof. To prove robustness we proceed by proving an upper bound ϵ on the number of solutions of the error masking equation (6.3), as that directly translates into a

bound on $Q(e)$. The modulo 2^k operator from the LHS of (6.3) stems from the limitation of the data path to k-bits. This limits the ranges of both the message and the message error to $0 \leq x, e_x < 2^k$. We can therefore remove the modulo 2^k operator by distinguishing between the two cases $x + e_x < 2^k$ and $x + e_x \geq 2^k$. Similarly, an error is masked only if the faulty check symbol $w < p$, so for $k = r$ we can distinguish between the three cases $w + e_w < p$, $p \leq w + e_w < 2^k$ and $2^k \leq w + e_w < 2^k + p$. This allows us to simplify the RHS of (6.3).

1. Solutions $x < 2^k - e_x$: An error (e_x, e_w) is masked iff

$$(x + e_x)^2 \bmod p = w + e_w \bmod 2^k$$

Simplifying the RHS we have the following three cases:

(a) If $w < p - e_w$, the error is masked iff

$$(x + e_x)^2 \bmod p = w + e_w \tag{6.4}$$

If $e = (p, 0)$ eq. (6.4) has exactly $2^k - p$ solutions. For $e_x \neq p$ and $e_x \geq 2^k - p$ there exists at most a single solution; at most two solutions exist in the case of $e_x < 2^k - p$.

(b) If $p - e_w \leq w < 2^k$, the error will never be masked, since a check symbol $w \geq p$ will always be detected.

(c) For $w \geq 2^k - e_w$ the error will be masked iff

$$(x + e_x)^2 \bmod p = w + e_w - 2^k \ . \tag{6.5}$$

Eq. (6.5) has at most two solutions.

2. Solutions $x \geq 2^k - e_x$: An error (e_x, e_w) is masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w \bmod 2^k$$

For the RHS we distinguish the following three cases:

(a) If $w < p - e_w$, the error is masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w \tag{6.6}$$

Eq. (6.6) has at most two solutions, unless we have an error $e = (2^k - p, 0)$, in which case there are $2^k - p$ solutions.

(b) If $p - e_w \leq w < 2^k$, the error will never be masked, since a check symbol $w \geq p$ will always be detected.

(c) For $w \geq 2^k - e_w$ the error will be masked iff

$$(x + e_x - 2^k)^2 \bmod p = w + e_w - 2^k \ . \tag{6.7}$$

Eq. (6.7) has at most two solutions.

$Q(e)$ is determined by the number of solutions to the error masking equation (6.3). This gives us the following bound:

There are at most $2^k - p + 2$ solutions to (6.3) for errors of the form $(p, 0)$ or $(2^k - p, 0)$, and at most 8 solutions for all other errors.

□

We would like to point out that the transition from linear arithmetic to robust quadratic codes with the same parameters k, r and p results in a much more uniform distribution of the error detecting capability of the code. For example, for linear codes with $k = r$ there are double errors with $e_x = e_w$ such that $Q(e)$ is very close to 1, i.e. the errors cannot be detected. For robust quadratic codes with the same parameters, $Q(e)$ is close to zero for all e.

We now give an intuitive argument to show the existence of practical robust codes for cryptographic purposes with the help of the prime number theorem. The idea here is that for fault-tolerance in an adversarial situation, the probability of not detecting

an error should be insignificantly small. As we saw from the proof, in the worst case we have a probability of at most $Q(e) = 8 \cdot 2^{-k} = 2^{-k+3}$ of not detecting an error (assuming a uniform distribution of messages, and $2^k - p < 8$). Therefore, a $Q(e)$ that makes insertion of an error infeasible for an attacker, requires a sufficiently large digit size k and a prime p close enough to 2^k so that the difference does not increase $Q(e)$ too much. For example, for $k = r = 32$ the distance from 2^k to the closest k-bit prime from below is less than 8, i.e. $p = 2^{32} - 5$, thus bounding $Q(e)$ by 2^{-29}. According to the prime number theorem the number of primes smaller than or equal to x is approximately $x/\ln x$, i.e. for our case $2^k/(k \ln 2)$. Intuitively it thus seems reasonable to expect to find a prime within the interval $[2^k - (k \ln 2), 2^k)$. In Table 6.1 we give the distance of the primes closest to 2^k from below for practical values of k.

Table 6.1: Closest Prime Number Distances from 2^k for Practical Values of k

k	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
$2^k - p$	1	5	1	3	9	3	15	3	39	5	39	57	3	35	1	5
k	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
$2^k - p$	9	41	31	5	25	45	7	87	21	11	57	17	55	21	115	59
k	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
$2^k - p$	81	27	129	47	111	33	55	5	13	27	55	93	1	57	25	59

6.3 Robust Arithmetic Operations

In the previous section we proved the robustness of quadratic codes for digits of size k bits. We now wish to apply them in a generalized framework for multi-precision arithmetic over the positive integers.

Due to the range limitation of the information bits to $0 \leq x < 2^k$, we need to handle any overflow resulting from arithmetic operations. This may be a carry bit

generated by the addition of two k-bit operands, or the $2k$-bit result of a multiplication. The new digits that are created in this manner will need their own check symbols, which cannot be derived from the input operands' check symbols alone. Thus they need to be derived purely from the information bits of the new digits, creating a potential loophole for the insertion of an error. This can be avoided by re-computing the joint check symbol from the newly generated individual check symbols and comparing it to the output of the predictor. This re-computation represents an integrity check which allows us bridge discontinuities introduced by interleaving mixed modulus operations, here the check modulus p and the implicit range limiting modulus 2^k. Once the integrity check is in place we can perform standard arithmetic operations, and implementing an algorithm like Montgomery's for modular arithmetic becomes straightforward.

In the following we show how this check may be implemented for various arithmetic primitives. Let $(a, |a^2|_p)$ and $(b, |b^2|_p)$ denote encoded input operands a and b, where $|x^2|_p$ is short-hand notation for $x^2 \bmod p$. We also introduce mnemonics for these primitives, in order to tie them into a robust variant of the digit serial Montgomery multiplication algorithm in the next section.

Addition (RADD and RADDC): RADD (Robust ADDition) and RADDC (Robust ADDition with Carry) compute the sum of the two input operands. This is depicted in Figure 6.1. For reference, the operators $\oplus_p$ and $\otimes_p$ stand for addition and multiplication modulo p, respectively. The sum $c = a + b$ $(+c_{\text{in}})$ may be larger than 2^k by at most a single bit. Let c_h denote this new carry, and c_l the k-bit sum. The predictor computes the joint check symbol $|c^2|_p$ as the sum of the check symbols and additional terms involving the operands: $|c^2|_p = |(a + b + c_{\text{in}})^2|_p = ||a^2|_p + |b^2|_p + 2(ab + c_{\text{in}}(a + b)) + c_{\text{in}}|_p$. For error detection we first create the check symbol for the k-bit sum $|c_l^2|_p$ (the check symbol for the carry bit is the carry bit

itself). Then we re-compute the joint check symbol as

$$\begin{aligned} |c^2|_p^* &= |(c_h 2^k + c_l)^2|_p \\ &= \left|c_h \cdot |2^{2k}|_p + c_h \cdot |c_l|_p \cdot |2^{k+1}|_p + |c_l^2|_p\right|_p \\ &= \left|c_h \cdot |2^{2k} + c_l \cdot 2^{k+1}|_p + |c_l^2|_p\right|_p \end{aligned} \tag{6.8}$$

If the check $|c^2|_p^* = |c^2|_p$ holds, then the result is deemed to be free from errors. The resulting carry from both RADD and RADDC is held in a register local to the addition circuit. If the following addition operation is RADDC, then that carry is used for computation of the new sum. If it is RADD, then a zero carry is used.

Multiplication (RMUL): The product of a and b and its joint check symbol is $(c, |c^2|_p) = (a \cdot b, ||a^2|_p \cdot |b^2|_p|_p)$. However, the previous tuple is not a code word, since c may exceed 2^k. We therefore split c into two halves c_h and c_l (cf. Figure 6.2), both of which are within the desired range:

$$c = c_h \cdot 2^k + c_l \quad 0 \leq c_h, c_l < 2^k \ .$$

We then compute the check symbols $|c_h^2|_p$ and $|c_l^2|_p$ separately, and establish their integrity with the composite check symbol $|c^2|_p$:

$$\begin{aligned} |c^2|_p^* &= \left|(c_h \cdot 2^k + c_l)^2\right|_p \\ &= \left|c_h^2 \cdot 2^{2k} + c_h \cdot c_l \cdot 2^{k+1} + c_l^2\right|_p \\ &= \left||c_h^2|_p \cdot |2^{2k}|_p + |c_h|_p \cdot |c_l|_p \cdot |2^{k+1}|_p + |c_l^2|_p\right|_p \end{aligned} \tag{6.9}$$

Observe that the values $|2^{2k}|_p$ and $|2^{k+1}|_p$ are constant for a given implementation and that $|c_h|_p$ and $|c_l|_p$ are intermediate results from the computation of the separate halves' check symbols. Hence we have all the necessary ingredients to re-compute the joint check symbol as $|c^2|_p^*$ and compare it to the value obtained from the predictor. If the comparison passes we assume there were no errors.

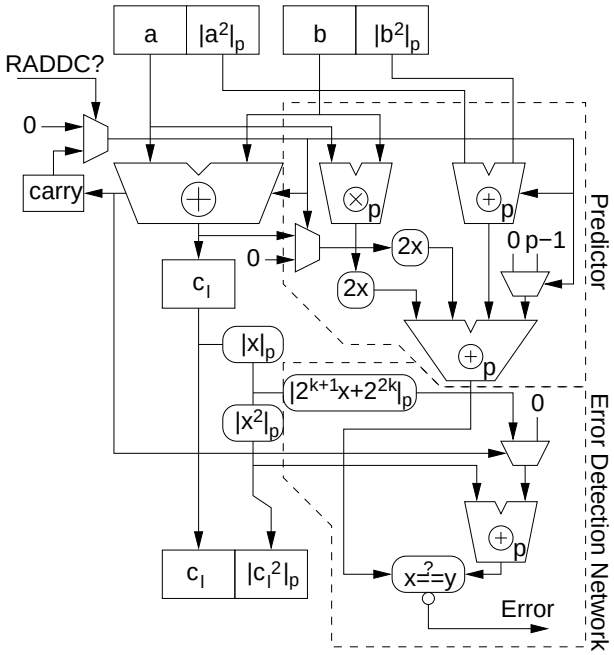


Figure 6.1: Robust Addition

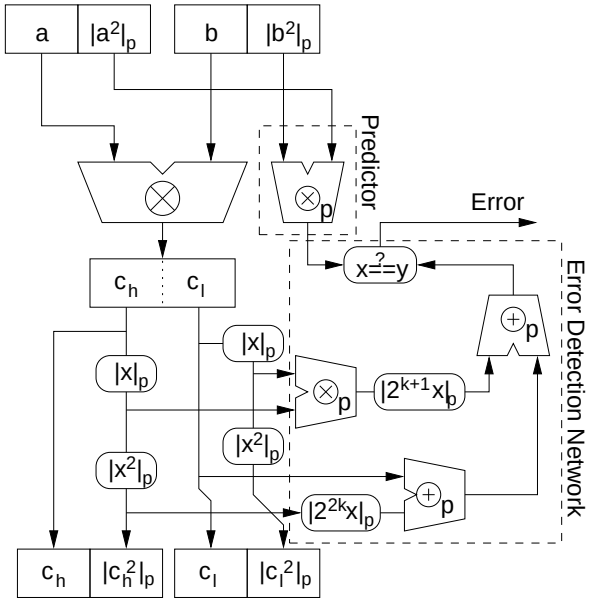


Figure 6.2: Robust Multiplication

Shifts, Subtraction, Logic Operations: We can apply similar re-computation techniques for other operations. Out of space considerations and since we do not need these other operations for the next section, we skip their details at this point.

Error Detection: The comparison between the predictor output and the re-computed joint check symbol is an easy target for an attack if carelessly implemented. We therefore require implementation as a totally self-checking circuit [Pra86]. The same holds for any other integrity checks.

6.4 Robust Montgomery Multiplication

We now show how to apply our robust code in a digit serial Montgomery Multiplication scheme. A good overview over several variants of the Montgomery algorithm is given in [KAK96]. In this example we will refer to the finely integrated operand scanning (FIOS) variant. It is the most suitable one for hardware implementations since it can be used in a pipelined fashion offering some degree of parallelization [Gau02].

Algorithm 6.1 k-bit Digit-Serial FIOS Montgomery Multiplication

Require: $d = \{0, \ldots, 0\}$, $M'_0 = -M_0^{-1} \bmod 2^k$

1: **for** $j = 0$ to $e - 1$ **do**
2: $(C, S) \Leftarrow a_0 b_j + d_0$
3: $U \Leftarrow S M'_0 \bmod 2^k$
4: $(C, S) \Leftarrow (C, S) + M_0 U$
5: **for** $i = 1$ to $e - 1$ **do**
6: $(C, d_{i-1}) \Leftarrow C + a_i b_j + M_i U + d_i$
7: **end for**
8: $(d_e, d_{e-1}) \Leftarrow C$
9: **end for**

In the following we require some basic familiarity on part of the reader with

the way of how Montgomery multiplication works. To review briefly: the objective is to compute the modular product of two N-bit numbers with respect to the N-bit modulus M. Montgomery's algorithm requires the initial transformation of all operands into residues of the form $\hat{x} = xR \bmod M$ and some final transformation back $x = \hat{x}R^{-1} \bmod M$. Here R is the Montgomery radix, usually $2^{k \cdot e}$, where $e = \lceil N/k \rceil$ represents the number of digits per operand. Without loss of generality we assume that the transformation into the Montgomery residue system has already taken place and we operate entirely within the residue system, so in order to simplify notation we will refer to a residue $\hat{x}$ simply as x.

The k-bit digit serial FIOS Montgomery algorithm takes as its inputs the e-digit vectors a and b, and computes the product $\mathrm{MM}(a, b) = a \cdot b \cdot R^{-1} \bmod M$. The value M'_0 is pre-computed whenever the modulus changes. In terms of notation, a pair (C, S) represents the concatenation of two variables as the destination for the result of an operation. Furthermore, the variable C is slightly larger than the other variables, i.e. $k+1$ bits. This is so to efficiently handle extra carries from the accumulation of C, d_i and the two products $a_i b_j$ and $M_i U$. The division by R is handled by the algorithm implicitly. For example, in line 4 the sum $(C, S) + M_0 U$ is assigned to (C, S), but in the following step S is dropped. This shift to the right by k bits, repeated e times, results in division by R.

As one can easily verify, Algorithm 6.1 consists of only very basic addition and multiplication steps. We may therefore obtain a robust digit-serial Montgomery algorithm (Alg. 6.2) simply by mapping all arithmetic steps to our robust arithmetic primitives introduced in the previous section. Additionally we insert intermediate checks during which we verify the integrity of operand values and their check symbols. This is indicated by a call to the pseudofunction $\mathrm{Check}((x, |x^2|_p), \ldots)$. Although not indicated in the algorithm description, we further assume that the error signal generated by the the internal integrity check within the arithmetic primitives RADD,

RADDC and RMUL, is also constantly evaluated. In the case of an error the algorithm is aborted with an exception.

Some comments about the robust algorithm: Algorithm 6.1 appears much shorter than Algorithm 6.2, since it combines multiple arithmetic operations into a single step. Also, while it handles carries implicitly using a larger width variable C, the robust algorithm is restricted to a digit size of exactly k bits. Thus, extra carry handling steps are required. In Alg. 6.2, line 6, the destination of the top half of the result is not assigned: $(-,-)$. This is equivalent to computing the result modulo 2^k, as in Alg. 6.1, line 3. A similar thing happens in Alg. 6.2, line 8, where the lower half of the result is dropped due to the implicit shift to the right. The point of performing the addition is purely to determine whether or not a carry is generated.

6.5 Analysis and Discussion

In this chapter we have presented a novel systematic non-linear arithmetic code which is robust against adversarial injection of faults and statistically occurring random faults (soft-errors). Based on this code we have introduced arithmetic primitives for robust computation over encoded digits. We have further used the example of digit serial Montgomery modular multiplication to demonstrate how robust arithmetic can be deployed for fault-secure multi-precision public-key computations.

Quite naturally the robustness of our scheme adds overhead, which has a negative impact on performance. This is a price we have to pay for the non-linearity that enables robustness. On the other hand this scheme offers unprecedented and quantifiable robustness against even the most motivated and capable adversaries.

In terms of critical path delay we estimate that multiplication incurs a performance hit of a little less than 100%, compared to a linear scheme, due to the re-computation of the quadratic check symbol. For addition, the absolute overhead is roughly the

same, however, in terms of relative overhead it fares much worse.

There are a couple of ways to reduce the complexity, based on the properties of the check modulus p, which is constant for a given implementation. Future research will quantify more precisely the performance and area overhead, and compare a variety of system parameters, i.e. digit size k, check modulus p, degree of parallelism, etc. For example, for certain values of k there exist Mersenne prime check moduli, which enable very efficient implementations for check symbol computation.

Our scheme scales reasonably well, since once the digit size is determined, the complexity of the predictor and error detection networks remain constant. We would like to emphasize that we clearly prioritize robustness over performance. Given the increased vulnerability level of mobile and ubiquitous security devices, and the progress in adversarial fault analysis techniques, we believe that this is a sensible argument.

Algorithm 6.2 Robust Montgomery Multiplication

Require: $d = \{(0,0), \ldots, (0,0)\}$, $M'_0 = -M_0^{-1} \bmod 2^k$

1: **for** $j = 0$ to $e-1$ **do**
2: **if** Check$((a_0, |a_0^2|_p), (b_j, |b_j^2|_p), (d_0, |d_0^2|_p), (M'_0, |(M'_0)^2|_p), (M_0, |M_0^2|_p))$ **then**
3: $((T_1, |T_1^2|_p), (T_0, |T_0^2|_p)) \Leftarrow \text{RMUL}((a_0, |a_0^2|_p), (b_j, |b_j^2|_p))$
4: $(T_0, |T_0^2|_p) \Leftarrow \text{RADD}((T_0, |T_0^2|_p), (d_0, |d_0^2|_p))$
5: $(T_1, |T_1^2|_p) \Leftarrow \text{RADDC}((T_1, |T_1^2|_p), (0,0))$
6: $((-,-), (U, |U^2|_p)) \Leftarrow \text{RMUL}((T_0, |T_0^2|_p), (M'_0, |M_0'^2|_p))$
7: $((T_3, |T_3^2|_p), (T_2, |T_2^2|_p)) \Leftarrow \text{RMUL}((M_0, |M_0^2|_p), (U, |U^2|_p))$
8: $(-,-) \Leftarrow \text{RADD}((T_0, |T_0^2|_p), (T_2, |T_2^2|_p))$
9: $(T_0, |T_0^2|_p) \Leftarrow \text{RADDC}((T_1, |T_1^2|_p), (T_3, |T_3^2|_p))$
10: $(T_1, |T_1^2|_p) \Leftarrow (\text{carry}, \text{carry})$
11: **for** $i = 1$ to $e-1$ **do**
12: **if** Check$((a_i, |a_i^2|_p), (b_j, |b_j^2|_p), (d_i, |d_i^2|_p), (U, |U^2|_p), (M_i, |M_i^2|_p))$ **then**
13: $(T_0, |T_0^2|_p) \Leftarrow \text{RADD}((T_0, |T_0^2|_p), (d_i, |d_i^2|_p))$
14: $(T_1, |T_1^2|_p) \Leftarrow \text{RADDC}((T_1, |T_1^2|_p), (0,0))$
15: $((T_4, |T_4^2|_p), (T_3, |T_3^2|_p)) \Leftarrow \text{RMUL}((a_i, |a_i^2|_p), (b_j, |b_j^2|_p))$
16: $(T_0, |T_0^2|_p) \Leftarrow \text{RADD}((T_0, |T_0^2|_p), (T_3, |T_3^2|_p))$
17: $(T_1, |T_1^2|_p) \Leftarrow \text{RADDC}((T_1, |T_1^2|_p), (T_3, |T_3^2|_p))$
18: $(T_2, |T_2^2|_p) \Leftarrow (\text{carry}, \text{carry})$
19: $((T_4, |T_4^2|_p), (T_3, |T_3^2|_p)) \Leftarrow \text{RMUL}((M_i, |M_i^2|_p), (U, |U^2|_p))$
20: $(d_{i-1}, |d_{i-1}^2|_p) \Leftarrow \text{RADD}((T_0, |T_0^2|_p), (T_3, |T_3^2|_p))$
21: $(T_0, |T_0^2|_p) \Leftarrow \text{RADDC}((T_1, |T_1^2|_p), (T_3, |T_3^2|_p))$
22: $(T_1, |T_1^2|_p) \Leftarrow (\text{carry}, \text{carry})$
23: **else**
24: ABORT
25: **end if**
26: **end for**
27: $(d_{e-1}, |d_{e-1}^2|_p) \Leftarrow (T_0, |T_0^2|_p)$
28: $(d_e, |d_e^2|_p) \Leftarrow (T_1, |T_1^2|_p)$
29: **else**
30: ABORT
31: **end if**
32: **end for**

Chapter 7

Low Cost Techniques for Robust Finite Field Arithmetic

Two important aspects in the selection of an error detection scheme are the overhead that is imposed by the scheme, and its capability of detecting errors (coverage). In an adversarial setting it is desirable to achieve error coverage close to 100%, simply because the probabilities of different classes of errors to occur are unknown. Such a high degree of coverage may come at a cost too high to be justifiable. Depending on the threat level and underlying error model, however, it may be possible to accommodate a slight decrease in coverage if it significantly reduces the cost of implementation. In this chapter we show how a special class of systematic non-linear codes with strong error detection properties [KT04] may be applied to elliptic curves defined over a specialized class of finite fields, dubbed optimal extension fields (OEF). We define robust versions of common components, i.e. adder, multiplier, shifter etc. with non-linearly encoded check-symbols, which are required for the realization of elliptic curve cryptosystems. Our design drastically reduces the overhead in both area and speed while maintaining a high level of confidence for detecting errors induced by an active adversary.

7.1 Optimal Extension Fields and their Arithmetic

Optimal extension fields were introduced by Bailey and Paar in [BP98]. The main idea is to use a generating polynomial of the form $f(x) = x^k - w$ to construct the extension field $GF(q^k)$, where q is selected as a *pseudo-Mersenne prime* given in the form $2^m \pm c$ with $\log_2 c < \lfloor \frac{m}{2} \rfloor$. The pseudo-Mersenne form allows efficient reduction in the ground field. The following Theorem [LN83] provides a simple means to identify irreducible binomials that can be used in OEF construction:

Theorem 7.1.1. *Let $k \geq 2$ be an integer and $w \in GF(q)^*$. Then the binomial $x^k - w$ is irreducible in $GF(q)[x]$ if and only if the following three conditions are satisfied:*

1. *each prime factor of k divides the order e of w in $GF(q)^*$;*
2. *the prime factors of k do not divide $\frac{q-1}{e}$;*
3. *$q = 1 \bmod 4$ if $k = 0 \bmod 4$.*

The representation of OEF elements utilizes the standard basis. An element $A \in GF(q^k)$ is represented as

$$A = \sum_{i=0}^{k-1} a_i x^i = a_0 + a_1 x + a_2 x^2 + \ldots + a_{k-1} x^{k-1}$$

where $a_i \in GF(q)$. The OEF arithmetic operations are performed as follows.

Addition/Subtraction:

The addition/subtraction of two field elements $A, B \in GF(q^k)$ is performed in the usual way, by adding/subtracting the polynomial coefficients in $GF(q)$ as follows.

$$A \pm B = \sum_{i=0}^{k-1} a_i x^i \pm \sum_{i=0}^{k-1} b_i x^i = \sum_{i=0}^{k-1} (a_i \pm b_i) x^i$$

Multiplication:

Let $A, B \in GF(q^k)$. Their product $C = A \cdot B$ is computed in two steps:

1. Polynomial multiplication:

$$C' = A \cdot B = \sum_{i=0}^{2k-2} c'_i x^i$$

2. Modular reduction:

$$\begin{aligned} C &= C' \pmod{f(x)} \\ &= \sum_{i=0}^{2k-2} c'_i x^i \pmod{x^k - w} \\ &= \sum_{i=0}^{k-2} (c'_i + w c'_{i+k}) x^i + c'_{k-1} x^{k-1} \end{aligned}$$

In the first step the ordinary product of two polynomials is computed. In the reduction step the binomial $f(x) = x^k - w$ facilitates efficient reduction. The reduction may be realized by only $k-1$ constant coefficient multiplications by w, and $k-1$ additions.

Inversion:

Standard algorithms to compute the multiplicative inverse of field elements are based either on the extended Euclidean algorithm, or Fermat's little theorem. These methods, however, are usually computationally expensive. Alternatively, there exists an elegant method for inversion in binary extension fields was introduced by Itoh and Tsujii [IT88], and later generalized by Guajardo and Paar [GP02]. For certain classes of fields, but particularly optimal extension fields, this method is the most efficient algorithm known to date. We will address some specifics of the algorithm a bit later and refer the reader to [GP02] for full details.

7.2 Robust Block Codes

A class of non-linear systematic error detecting codes, so-called "robust codes", were proposed by Karpovsky and Taubin [KT04]. They can achieve optimality according

to the minimax criterion, that is, minimizing over all (n, k) codes the maxima of the fraction of undetectable errors $Q(e)$ for $e \neq 0$. The following definition from [KT04] rigorously defines a particular class of non-binary codes.

Definition 7.2.1. Let V be a linear p-ary (n, k) code ($p \geq 3$ is a prime) with $n \leq 2k$ and check matrix $H = [P|I]$ with $\text{rank}(P) = n - k$. Then $C_V = \{(x, w)|x \in GF(q^k), w = (Px)^2 \in GF(q^r)\}$

To quantify the performance of C_V we need a metric. The error masking probability for a given non-zero error $e = (e_x, e_w)$ may be quantified as

$$Q(e) = \frac{|\{x|(x + e_x, w + e_w) \in C_V\}|}{|C_V|} \ . \tag{7.1}$$

Note that we call the code C_V *robust*, if it minimizes maxima of $Q(e)$ over all possible non-zero errors. Reference [KT04] provides the following theorem which quantifies the error detection performance of the non-linear code C_V.

Theorem 7.2.1. *For C_V the set $E = \{e|Q(e) = 1\}$ of undetected errors is a $(k - r)$-dimensional subspace of V, $p^k - p^{k-r}$ errors are detected with probability 1 and remaining $p^n - p^k$ errors are detected with probability $1 - p^{-r}$.*

These codes achieve total robustness for the case $r = k$, when the subspace of undetectable errors collapses to the zero codeword and all non-zero error patterns can be detected with a probabilty of either $1 - p^{-r}$ or 1. The reduced overhead for the case $r < k$ is obtained at the expense of the loss of total robustness. Nonetheless, some important properties of robust codes are retained: Contrary to linear encoding schemes, the probability of missing an error is largely data-dependent. This has one very important consequence. An active adversary trying to induce an undetected error in the data would need to a) know the value of the data *a priori* in order to compute an undetectable error pattern, and b) to induce a fault with sufficient spatial and temporal accuracy such as to successfully re-create the matching error

in the device. In a linear scheme, *any* error pattern that is a codeword itself will lead to a successful compromise. Although there also exist some error patterns in the robust scheme which will escape detection, their number is significantly smaller than in linear schemes.

7.3 Suitable Codes for OEF Arithmetic

Our objective is to protect OEF arithmetic against a sufficiently large class of error patterns, while keeping the overhead in performance low. As mentioned in the previous section, full robustness can only be achieved with $r = k$, resulting in a duplication of the operand size and likely more than 100% overhead. Furthermore, as we will show in the next section, only very specific choices of the sub-matrix P allow us to define arithmetic operations in such a way that the check-symbol of the result can be predicted efficiently based on the input operands' check-symbol.

An alternative method for robust multi-precision arithmetic was presented in [GSK06], which achieves total robustness by encoding single digits separately. Such a high degree of protection, however, can only be obtained by accepting a substantial amount of overhead on the predictor and error detection networks. Depending on the anticipated threat level, such rigor may not be required. As long as the probability of error detection is sufficiently high to render malicious fault insertions infeasible, a lighter-weight scheme will work as well.

For our purposes we need a linear code over $GF(q)$ with $\text{rank}(P) = n - k$, which we will transform into a robust code. The error correcting properties and ease of decoding are irrelevant in this setting. For our purposes it suffices to select a simple linear code which will allow us to build robust versions of the arithmetic operations (as described in the next section) in an efficient manner. We therefore pick a simple parity code of length $n = k + 1$. Note that by choosing $r = 1$ the $r \times n$ error

check matrix $H = [P|I_r]$ becomes simply $H = [1\ 1\ 1 \cdots 1\ 1]$. Hence, for a vector a representing an element of $GF(q^k)$, the matrix-vector product Pa may be computed by simply summing the coefficients of a, i.e. $Pa = \sum_{i=0}^{k-1} a_i$. Thus the robust encoded form of $a \in GF(q^k)$ is simply

$$\left(a, (Pa)^2\right) = \left(< a_0, a_1, a_2, \ldots, a_{k-1} >, \left(\sum_k a_i \right)^2 \right)$$

7.4 Robust OEF Arithmetic

To build an error detection network we need to provide robust realizations of the basic OEF arithmetic operations, i.e. addition and multiplication. We wish to minimize the overhead associated with computing and verifying the check-symbol. Hence, we choose the linear code such that P maps elements of $GF(q^k)$ to $GF(q)$. Note that field inversions and elliptic curve point operations are implemented through various compositions of these basic operations and hence there is no need to consider them here explicitly.

The general architecture for implementation of robust OEF arithmetic is depicted in Figure 7.1 and can be described as follows. A robust arithmetic operation consists of three major components: a) the main data-path, b) the check-symbol predictor and c) the error detection network (or, in the case of software implementation, the error detection procedure).

Main Data-Path: The basic arithmetic operation which computes the result $c = c_0 + c_1 x + c_2 x^2 + \ldots + c_{k-1} x^{k-1}$ without the check-symbol.

Predictor: Computes the expected check-symbol from input data. Whenever possible only the input operands' quadratic check-symbols shall be used, i.e. $(Pa)^2$ or $(Pb)^2$. When this is not feasible, a linear check-symbol Pa or some portion of the operands' data may be used as well.

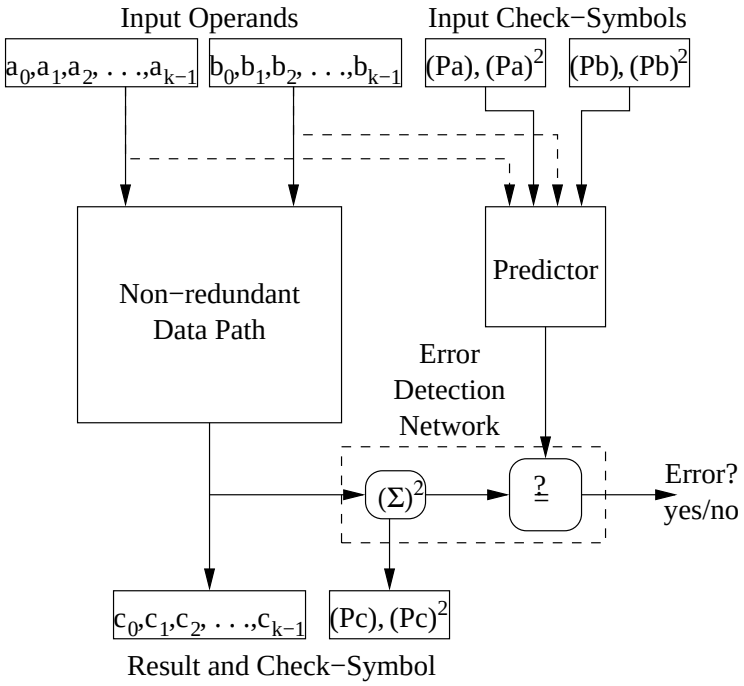


Figure 7.1: General Error Detection Architecture

Error Detection Network: Computes the actual check-symbol $(Pc)^2$ from the result of the operation and compares it to the output of the predictor. If both match, the result is deemed correct. If not, an alarm is raised and the operation is aborted. In hardware this error detection network is typically assumed to be free of errors and thus may need to be implemented as a self-checking architecture.

The intermediate result Pc that is computed in the error detection network as a by-product of the check-symbol should be saved as part of an extended check-symbol, since in many cases it is required for efficient realization of predictors, as we will see in the following sections.

7.4.1 Initial Encoding and Detection Network

Operands entering the device are typically not encoded in redundant form. We require an initial encoding operation to establish a relation between the operand and its (extended) check-symbol. As mentioned above, there is a clear benefit to retaining the intermediate linear sum of coefficients as part of the extended check-symbol. We thus represent encoded operands in the extended form

$$A = \left[< a_0, a_1, \ldots, a_{k-1} >, Pa, (Pa)^2 \right] \ .$$

Initial encoding of an operand a and integrity checks of intermediate results can both be realized in the same way. It requires $k-1$ additions in the ground field GF(q) to produce the linear sum Pa, followed by a squaring which yields the non-linear check-symbol $(Pa)^2$. The only difference lies in the subsequent comparison with a predicted check-symbol, which is absent in the initial encoding.

7.4.2 Addition

Given two encoded inputs $A = (a, Pa, (Pa)^2)$ and $B = (b, Pb, (Pb)^2)$ we want to compute $S = (s, Ps, (Ps)^2)$ such that $s = a + b \in GF(q^k)$. This is done by first computing the component-wise addition of the two field elements to form the sum $s = a + b$, then computing the associated check-symbol $w_s = (Ps)^2$.

However, since we want to protect against faults that may occur during the computation, we need a reference value that predicts the value of the new check-symbol. The output of this predictor should ideally be based exclusively on the input operands' check-symbols, but intermediate values obtained during their computation may be used as well. By using the extended check-symbols for a and b, $Pa, (Pa)^2$

and $Pb, (Pb)^2$, we can predict the value of the sum's check-symbol $(Ps)^2$ as follows.[1]

$$(Ps)^2 = (P(a+b))^2 = (Pa)^2 + 2(Pa)(Pb) + (Pb)^2$$

Since the computation of $(Ps)^2$ takes place entirely in the ground field $GF(q)$, it is relatively inexpensive compared to the addition in $GF(q^k)$, thereby adding just a minor amount of overhead. In the final step of the computation the error detection network compares the output of the predictor to the actual check-symbol $(Ps)^2$ of the sum s that was computed by summing the coefficients s_i and squaring the result. If they match, then the result $(s, Ps, (Ps)^2)$ is considered free of errors and passed on to the next stage of the computation. In case of a mismatch an alarm signal is raised.

7.4.3 Multiplication

The check-symbol of the product $c = ab \in GF(q^k)$ is computed directly as $Pc, (Pc)^2$. The predicted check-symbol needs to be computed as

$$(Pc)^2 = (P(ab))^2 \ .$$

Normally we would be stuck at this point, since in general multiplication by matrix P does not associate with the multiplication operation in $GF(q^k)$. Note, however, that due to our particular choice of P the check-symbol of a is explicitly computed as $\left(\sum_{i=0}^{k-1} a_i\right)^2$. Furthermore, observe the identity

$$(Pa)(Pb) = \left(\sum_{i=0}^{k-1} a_i\right)\left(\sum_{i=0}^{k-1} b_i\right) = \sum_{i,j=0,1,\ldots,k-1} a_i b_j \ . \tag{7.2}$$

[1] Here we are abusing the notation slightly by using juxtaposition for both matrix-vector products as well as for multiplication in $GF(q^k)$. Also we use vector and polynomial representations interchangeably.

On the other hand, assuming a modulus of $f(x) = x^k - w$ is used for the construction of $GF(q^k)$, the actual value of the computed check-symbol for $ab \bmod f(x)$ is

$$\begin{aligned}(Pc)^2 &= (P(ab))^2 = \left(\sum_{0\le i+j<k} a_i b_j + \sum_{k\le i+j} w a_i b_j\right)^2 \\ &= \left(\sum_{0\le i,j<k} a_i b_j + \left((w-1)\sum_{k\le i+j} a_i b_j\right)\right)^2\end{aligned}$$

After applying the identity from (7.2) and binomial expansion we obtain

$$(Pc)^2 = (Pa)^2(Pb)^2 + \left((w-1)\sum_{k\le i+j} a_i b_j\right)\left[2(Pa)(Pb) + \left((w-1)\sum_{k\le i+j} a_i b_j\right)\right]$$

which can be written in a more compact way as

$$(Pc)^2 = (Pa)^2(Pb)^2 + T_M\left[2(Pa)(Pb) + T_M\right] \ .$$

Hence, we can build a predictor with little cost. The key is to compute the term $T_M = (w-1)\sum_{k\le i+j} a_i b_j$ efficiently. The summation may be realized by collecting the b_j terms that are multiplied by the same a_i, i.e. a_1 is multiplied with b_{k-1}, a_2 with $(b_{k-1}+b_{k-2}), \ldots, a_{k-1}$ with $(b_{k-1}+b_{k-2}+\ldots+b_1)$. Together with the multiplication by the constant $(w-1)$ the computation requires k multiplications and $(k+1)(k-2)/2$ additions in $GF(q)$. In a digit serial architecture, however, we can compute even more efficiently. Note how the collections of b_j terms can be expressed as a partial sums $x_i = x_{i-1} + b_{k-i}$ for $0 < i < k$ and $x_0 = 0$. Thus it suffices to simply compute the current x_i from a buffered previous value x_{i-1} and accumulate the products $a_i x_i$ as depicted in Figure 7.2. This reduces the complexity to k multiplications and $2(k-2)$ additions. The overall complexity for a digit-serial predictor is thus $k+3$ multiplications and $2k-2$ additions in the ground field $GF(q)$.

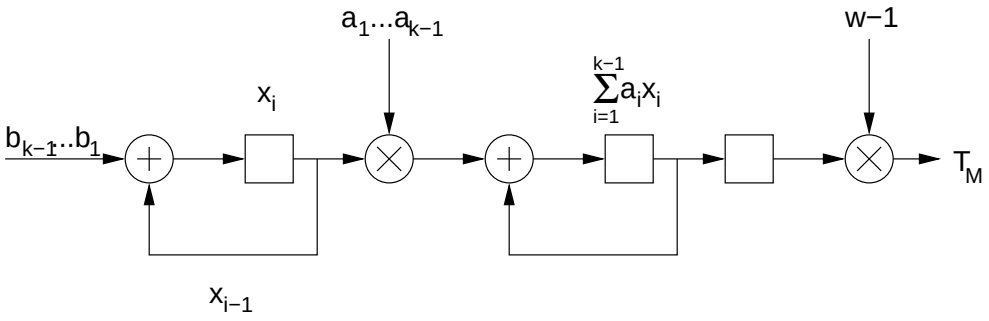


Figure 7.2: Data-flow Graph for the Digit-Serial Computation of T_M

7.4.4 Squaring

It is generally known that one can compute the square of a field element more efficiently than a general multiplication, by taking advantage of the symmetry of partial products [Par00]. For the computation of the check-symbol we can apply the same general procedure as for multiplication, only that now both operands are the same. This opens the door to further optimization:

$$\begin{aligned}(Pc)^2 &= (P(a^2))^2 = \left(\sum_{0\le i+j<k} a_i a_j + w \sum_{k\le i+j} a_i a_j\right)^2 \\ &= \left((Pa)^2 + \left((w-1)\sum_{k\le i+j} a_i a_j\right)\right)^2 \\ &= \left((Pa)^2 + T_S\right)^2\end{aligned}$$

As one can easily see, not only is the check-symbol computation shorter, but just like with general squaring, we can take advantage of partial product symmetries in the computation of the value T_S. This symmetry is twofold. Since the partial product $a_i a_j$ is the same as $a_j a_i$, we only need to compute it once for $i < j$. Once accumulated, we double the temporary result with a shift to the left, add the sum of the squares

a_i^2 and multiply by $w-1$, as follows.

$$T_S = (w-1)\left(2\sum_{k\leq i+j, i<j} a_i a_j + \sum_{i>k/2} a_i^2\right)$$

The second symmetry is due to the possibility of grouping partial products into products of sums, i.e. $(a_1 + a_{k-2})a_{k-1}$, $(a_2 + a_{k-3})(a_{k-1} + a_{k-2})$, $(a_3 + a_{k-4})(a_{k-1} + a_{k-2} + a_{k-3})$, and so forth. Here the first sum consists of two coefficients a_i and a_{k-1-i} whose indices are symmetric about $(k-1)/2$. Only if k is odd, the last iteration is the product of the single term $a_{\lfloor k/2 \rfloor}$ and the sum $(a_{k-1} + a_{k-2} + \ldots + a_{\lfloor k/2 \rfloor + 1})$. This grouping of terms can be easily translated into a digit-serial computation involving only k multiplications and $2k-5$ additions in GF(q). This brings the total complexity for computing the check-symbol of a squaring to $k+1$ multiplications and $2(k-2)$.

7.4.5 Multiplication by Scalar

The multiplication of a field element $a \in GF(q^k)$ by a scalar, i.e. a single ground field element $c \in GF(q)$, is required in various situations arising in elliptic curve cryptography. It is, for example, required in modular reduction and as part of inversion operations based on the extended euclidean and similar algorithms. Further it serves as the foundation for simple intra-coefficient shifting, i.e. multiplication by a scalar which is a power of two. The coefficients of the field elements all have to be multiplied by the same scalar $ca = ca_0 + ca_1 x + ca_2 x^2 + \ldots + ca_{k-1} x^{k-1}$. The predictor output, however can be computed very easily:

$$(P(ca))^2 = \left(\sum_{i=0}^{k-1} ca_i\right)^2 = c^2 (Pa)^2$$

All that is needed to compute the new check-symbol, therefore, is one squaring and one multiplication operation in $GF(q)$. If the scalar is an operand in memory, i.e. not hard-wired or -coded, then in all likelihood it is already protected by a check symbol

$(Pc)^2 = c^2$, and thus the predictor requires only a single multiplication. This is the case, e.g. during a step in the Itoh-Tsujii inversion algorithm, where the outcome of an operation is an element in the ground field, i.e. with all higher degree coefficients set to zero.

7.4.6 Shifting of Coefficients

While shifting is not (directly) an arithmetic operation it is sometimes needed in the implementation of arithmetic functions. In many cases, shifting is only needed when the least significant coefficient of the $GF(q^k)$ element has already been made zero by simply adding the properly scaled version of the modulus polynomial. Note the following simple yet useful property: As long as the shifted coefficients have a value of zero the check-symbol for an element $a \in GF(q^k)$ which is defined as the linear sum of the coefficients, i.e. $(Pa)^2 = \left(\sum_{i=0}^{k-1} a_i\right)^2$ is not altered. Hence, shifting does not affect the check-symbol and comes for free.

7.4.7 Frobenius Maps

Iterates of the Frobenius automorphism are of particular interest to OEF arithmetic. They play an important role in the inversion algorithm that was developed for normal basis representation by Itoh and Tsujii [IT88] and later adapted for use with standard basis by Guajardo and Paar [GP02]. The e-th iterate of the Frobenius map on the field element $a = \sum_k a_i x^i$ computes the exponentiation $\sigma^e(a) = a^{q^e}$ in a very efficient manner. This efficiency is based on the fact that OEFs are defined over irreducible binomials of the form $f(x) = x^k - w$. Note that exponentiation to a power of the field characteristic q, yields a much simpler expression for the result than for the general case:

$$a^{q^e} = \left(\sum_{i<k} a_i x^i\right)^{q^e} = \sum_{i<k} a_i x^{iq^e}$$

Reduction of this polynomial with respect to $f(x)$ gives the result

$$a^{q^e} = \sum_{i<k} a_i w^{\lfloor iq^e/k \rfloor} x^{iq^e \bmod k} = \sum_{i<k} a_i w^t x^s \ .$$

Observe that for a given parameter e the values w^t and x^s are constant and can be pre-computed. Furthermore, note that the value of s is unique for each $i = 0 \ldots k-1$, and we may express the resulting re-ordering of coefficients as a permutation $a' = \pi_e(a)$, defined by the indexing function $s_e(i) = iq^e \bmod k$. Hence, we can view the e-th iterate of the Frobenius automorphism $\sigma^e(a)$ as the component-wise multiplication (indicated here by the operator symbol $\odot$) of field element a with a pre-computed element $d^{(e)} = d_0^{(e)} + d_1^{(e)} x + \ldots + d_{k-1}^{(e)} x^{k-1}$, followed by a permutation of the resulting coefficients:

$$\begin{aligned} c &= \sigma^e(a) = \pi_e(a \odot d^{(e)}) && (7.3) \\ &= a_0 d_0^{(e)} + a_1 d_1^{(e)} x^{s_e(1)} + a_2 d_2^{(e)} x^{s_e(2)} + \ldots + a_{k-1} d_{k-1}^{(e)} x^{s_e(k-1)} && (7.4) \\ &= c_0 + c_1 x + \ldots + c_{k-1} x^{k-1} && (7.5) \end{aligned}$$

We now want to incorporate Frobenius maps into our error detection scheme. For the sake of a less confusing notation we will from now on refer to $d^{(e)}$ and its coefficients simply as d. Observe that we only need to build a predictor for the component-wise multiplication, since permutation of the coefficients is completely transparent and does not affect the check-symbol. Therefore we only need to predict the check-symbol $(Pc)^2 = (P(a \odot d))^2 = (\sum_{i=0}^{k-1} a_i d_i)^2$. Unfortunately, we cannot simply multiply the check-symbols $(Pa)^2$ and $(Pd)^2$ to do so. Rather take a subtractive synthesis approach by first computing the product $(Pa)(Pd) = \sum_{i,j<k} a_i d_j$ and then subtracting a term $T_F = \sum_{i \neq j} a_i d_j$. Thus the predicted check-symbol is computed as follows:

$$(Pc)^2 = (Pa)^2 (Pd)^2 - T_F \left(2(Pa)(Pd) - T_F\right) \tag{7.6}$$

We will now demonstrate how to efficiently compute T_F. Again, observe the identity from (7.2). We may visualize the partial products $a_i d_j$ as a $k \times k$ matrix, with

$(Pa)(Pd)$ being the sum of all entries. The check-symbol of the component-wise multiplication $a_i \odot d_i$ is the square of the sum of all entries on the main diagonal. Thus T_F is the sum of all matrix coefficients *other than* the main diagonal, in other words, the upper and lower triangular sub-matrices. We can compute these partial products and their summation efficiently by grouping expressions with common terms. For example, the upper triangular sub-matrix can be computed as the summation of $a_1 d_0$, $a_2(d_0 + d_1)$, $a_3(d_0 + d_1 + d_2)$, etc. Similarly, we can compute the lower sub-matrix in the exact same manner, only with the roles of a and d reversed. Figure 7.3 shows the data-flow graph for computing T_F in a digit-serial fashion. This operation requires $2(k-1)$ multiplications and $4(k-2)+1$ additions in GF(q). Together with the operations in (7.6) the total cost of prediction is $2k+1$ multiplications and $4(k-1)$ additions.

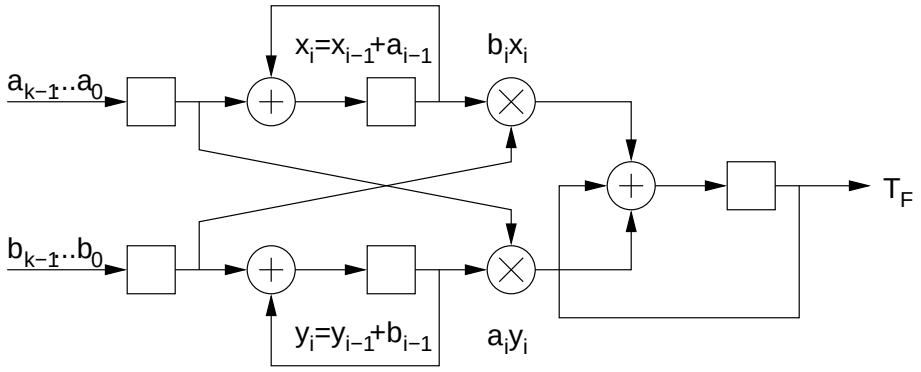


Figure 7.3: Data-Flow Graph for the Digit-Serial Computation of T_F

7.4.8 Further Notes

Some remarks regarding our assumptions for the implementation of arithmetic: We did not include fast multiplication algorithms like that of Karatsuba and Ofman[KO63] in our complexity estimate, since their performance advantage depends on the latency ratio of multiplication and addition operations. If both operations issue in a single

clock cycle, as is the case for many modern RISC based processors, the increased cost of more complicated control flow and irregular memory access patterns is not offset by the little performance advantage[Har04]. The lower asymptotic complexity of KOA mainly pays off for large operand sizes, as in RSA or Diffie-Hellman cryptosystems, but not so much for the short operands used in elliptic curve schemes. Despite being the slowest algorithm, Schoolbook multiplication is also relatively easy to implement.

7.5 Software Implementation on Embedded Processors

The motivation for elliptic curve cryptosystems over optimal extension fields stems primarily from the relatively weak performance of binary polynomial and large prime field arithmetic in software based implementations. Most general purpose processors do not support the arithmetic primitives that are required for efficient implementation, most importantly fast polynomial multiplication over GF(2). Instruction set extensions that would help overcome these limitations have been proposed in [GS04], but are unlikely to see implementation in mainstream processors in the near future.

OEF based implementations achieve a major speed-up due to the fact that the size of field element coefficients and their arithmetic operations can be mapped near perfectly to the native word size and ALU capabilities of microprocessors. The restriction of coefficients to a modulus smaller than or equal to the wordsize also obviates the need for elaborate carry-handling mechanisms as encountered in typical multiple precision arithmetic.

The extension field operations we considered in the previous section can be realized almost exclusively by using elementary ground field arithmetic, i.e. addition and multiplication in GF(q). We will consider complexity only in terms of these basic operations, since the individual implementation details are architecture dependent.

Table 7.1 lists the complexity of OEF operations for standard implementation, plus the additional overhead required for robust implementation. Naturally these expressions depend on the field extension degree k, while the size of operands in the ground field is irrelevant in most cases, as long as they fit in the native word size of the processor.

Table 7.1: Complexity of Various Extension Field Operations

	Standard Implementation	
Field op.	#Muls in GF(q)	#Add/Subs in GF(q)
Addition/Subtr.	-	$\mathcal{A}_A = k$
Multiply†	$\mathcal{M}_M = k^2 + k - 1$	$\mathcal{A}_M = k^2 - k$
Square†	$\mathcal{M}_S = k(k+3)/2 - 1$	$\mathcal{A}_S = k(k-1)/2$
Scalar Product	$\mathcal{M}_P = k$	-
Frobenius Map	$\mathcal{M}_F = k$	-
Fermat Inverse*	$\mathcal{M}_{FI} = (\delta - 1)\mathcal{M}_S + \frac{\delta}{2}\mathcal{M}_M$	$\mathcal{A}_{FI} = (\delta - 1)\mathcal{A}_S + \frac{\delta}{2}\mathcal{A}_M$
Itoh-Tsujii Inverse	$\mathcal{M}_{ITI} = \Gamma\mathcal{M}_M + \Delta\mathcal{M}_F + \mathcal{M}_P$	$\mathcal{M}_{ITI} = \Gamma\mathcal{A}_M$
	Robustness Overhead†	
	#Muls	#Adds
Addition/Subtr.	$\mathcal{M}_A^R = 1$	$\mathcal{A}_A^R = 2$
Multiply†	$\mathcal{M}_M^R = k + 4$	$\mathcal{A}_M^R = 2k - 2$
Square†	$\mathcal{M}_S^R = k + 1$	$\mathcal{A}_S^R = 2k - 4$
Scalar Product	$\mathcal{M}_P^R = 1$	-
Frobenius Map	$\mathcal{M}_F^R = 2k + 1$	$\mathcal{A}_F^R = 4(k-1)$
Fermat Inverse*	$\mathcal{M}_{FI}^R = (\delta - 1)\mathcal{M}_S^R + \frac{\delta}{2}\mathcal{M}_M^R$	$\mathcal{A}_{FI}^R = (\delta - 1)\mathcal{A}_S^R + \frac{\delta}{2}\mathcal{A}_M^R$
Itoh-Tsujii Inverse	$\mathcal{M}_{ITI}^R = \Gamma\mathcal{M}_M^R + \Delta\mathcal{M}_F^R + \mathcal{M}_P^R$	$\mathcal{A}_{ITI}^R = \Gamma\mathcal{A}_M^R + \Delta\mathcal{M}_F^R$
*Based on Fermat's Little Theorem, $\delta = \lceil \log_2(q^k - 2) \rceil$ †Overhead for predictor. Overhead for detection is 1 Mul., $k-1$ Adds. per Oper.		
$\Gamma = \lfloor \log_2(k-1) \rfloor + \mathrm{HW}(k-1)$		
$\Delta = \begin{cases} \Gamma & \text{if } k \text{ is odd} \\ \log_2 k & \text{if } k \text{ is a power of 2} \\ \lfloor \log_2(k-1) \rfloor + \mathrm{HW}(k) - 1 & \text{otherwise.} \end{cases}$		

7.6 Analysis of the Performance Overhead in ECC Point Multiplication

In this section we show how robust implementations of OEF cryptosystems on embedded processors can be realized efficiently with a minimum of performance overhead. We will base all of our performance analyses on the basic elliptic curve scalar point multiplication, as this is the central and most complex operation of all ECC based protocols. Based on the complexity analysis of OEF arithmetic in standard and robust implementations in the previous section, we want to derive robust point multiplication overhead estimates for a couple of example fields. We begin by breaking down the number of extension field operations required for a given set of parameters.

Table 7.2: Overhead of Robust OEF Arithmetic for ECC Point Multiplication

Field extension k	5	7	11	14	17
#MULs	40.27%	26.05%	14.88%	11.15%	8.89%
#ADDs	84.71%	63.44%	42.02%	33.49%	27.84%
#Instructions	62.49%	44.74%	28.45%	22.32%	18.37%

Table 7.3: Probability of Missing an Error in Robust OEF Arithmetic

size of ground field $\log_2 q$	45	32	21	16	14
k	5	7	11	14	17
$\Pr[e\|Q(e)=1]$	$< 2^{-89}$	$< 2^{-63}$	$< 2^{-41}$	$< 2^{-31}$	$< 2^{-27}$
$\Pr[e\|Q(e)=0]$	$< 2^{-44}$	$< 2^{-31}$	$< 2^{-20}$	$< 2^{-15}$	$< 2^{-13}$
$\Pr[e\|Q(e)=q^{-1}] \approx 1,\ q^{-1}$	$< 2^{-44}$	$< 2^{-31}$	$< 2^{-20}$	$< 2^{-15}$	$< 2^{-13}$

An elliptic curve over $\mathrm{GF}(q^k)$ is specified by its defining equation $y^2 = x^3 + ax + b$, and the parameters $a, b \in \mathrm{GF}(q^k)$. In this dissertation, as in [BHLM01], we will fix one parameter $a = -3$, since it allows faster arithmetic without sacrificing much

flexibility. Let $s \in \mathbb{Z}_{2^\ell}$ be an ℓ-bit scalar value with $\ell \approx k \log_2 q$, and $\mathcal{P}$ a base point represented in affine coordinates by the tuple $(x_\mathcal{P}, y_\mathcal{P})$. Furthermore let $\mathcal{Q}$ be the point resulting from multiplication of $\mathcal{P}$ with the scalar s, represented in Jacobian projective coordinates by the triple $(x_\mathcal{Q}, y_\mathcal{Q}, z_\mathcal{Q})$. We opted to make use of mixed Jacobian-affine coordinate representation for reasons of efficiency and popularity, but note that other coordinate systems may be used in the same way.

Scalar point multiplication consists of repeated applications of point addition, subtraction and doubling algorithms. For a detailed description of these algorithm we refer the reader to [HMV04]. By representing s in non-adjacent form (NAF), a scalar point multiplication requires $\ell - 1$ point doublings and on average $\ell/3$ point additions/subtractions. The NAF repeated-doubling algorithm, however, is susceptible to sign-change fault attacks [BOS06]. We therefore recommend the use of a side-channel resistant algorithm like the modified Montgomery-Ladder algorithm [FV06]. It requires $2(\ell - 1)$ point doublings and $\ell - 1$ point additions. We refer to [BHLM01] for the operation count of point doubling (4 multiplications, 4 squarings, 5 additions/subtractions, and 4 multiplications with a scalar[2]) and point addition and/or subtraction (8 multiplications, 3 squarings, and 7 additions/subtractions), without going into too much detail. Conversion from Jacobian to affine coordinates at the end of a point multiplication requires a single field inversion, one squaring and three multiplications.

These instruction counts allow us to analyze the overall performance impact that robust implementation of OEF arithmetic has on the performance of elliptic curve point multiplication for various parameters, listed in Table 7.2. Since the relative overhead only depends on the extension degree k, the table does not include the size of q. The robustness of the scheme, i.e. the error detection probability, on the other hand depends exclusively on the size of the ground field $\mathrm{GF}(q)$, a relationship that

[2]By scalar multiplication we refer to an operation $M : \mathrm{GF}(q) \times \mathrm{GF}(q^k) \mapsto \mathrm{GF}(q^k)$.

is illustrated in Table 7.3. We would like to emphasize briefly that the security of an elliptic curve system relies largely (but not exclusively!) on the size of the finite field over which it is defined, as long as one can find a curve of prime order, approximately the size of the field. The relatively large number of OEF parameters for field sizes of practical interest [BP01], thus provides us with a straight-forward mechanism for trading off the level of robustness and the acceptable performance overhead, without sacrificing cryptographic strength. For example, the following two OEFs are 224 bits in size: GF($(2^{16} - 15)^{14}$) and GF($(2^{32} - 5)^7$). The overhead for robustness with the former field is around 10% in the number of GF(q) multiplications and 17% in GF(q) additions, while it is 21% and 33% for the latter. By the same token, the chance of missing an error in an operand of GF($(2^{32} - 5)^7$) with non-negligible probability is approximately 2^{32} times smaller than that in the field with higher extension degree, and missing a random error is 2^{16} times less likely.

7.7 Discussion

We have presented a strong scheme for error detection in optimal extension field arithmetic, based on the theory of robust codes. Such schemes are necessary in defeating ever new attack vectors on cryptographic hardware that keep being developed. Special emphasis has been placed on keeping the overhead of error detection low and tolerable for performance critical applications. The method is easily applicable to both software and hardware implementations, and allows an easy trade-off between error detection overhead and robustness.

Chapter 8

Tamper Resilient Control Structures

In the previous chapters we focused exclusively on the protection of the data-path of cryptographic circuits. Like any other computing machinery, however, these circuits do not consist exclusively of a data path, but also of control structures like state machines. Since any system is only as secure as its weakest component, it is important to not leave Achilles' proverbial heel uncovered and protect the control logic of cryptographic circuits against fault attacks, as well.

Motivated by a hypothetical, yet realistic, fault analysis attack that in principle could be mounted against any modular exponentiation engine, even one with appropriate data path protection, we set out to close this remaining gap. In the following sections we present guidelines for the design of multi-fault resilient sequential control logic based on standard Error Detecting Codes (EDC) with large minimum distance. We introduce a metric which measures the effectiveness of the error detection technique in terms of the effort the attacker has to make in relation to the area overhead spent in implementing the EDC. Our comparison shows that the proposed EDC based technique provides superior performance when compared against regular N-modular

redundancy techniques. Furthermore, our technique scales well and does not affect the critical path delay.

8.1 Introduction

The goal of this chapter is to provide the control logic of a cryptographic circuit with a level of protection that is on par with other mechanisms protecting its data path. As stated in Chapter 4 we consider an adversarial situation in which we want to protect a system under worst-case conditions, rather than maximizing average case reliability. As such it is important to provide each and every component of the system with a uniform level of resilience against attacks. A secondary goal is to quantify this resilience and provide some practical design considerations. We would like to briefly remark on the issue of complexity and cost of implementation: compared to the data path the control logic tends to be fairly small in size. The overhead is therefore secondary to the improvement in fault-resilience and in most cases even irrelevant. That said, we will show in Section 8.5 that our solution exhibits much better performance when compared to traditional fault tolerant techniques of the same resilience level. Specifically, the contributions contained in this chapter may be summarized as follows:

- We discuss how a hypothetical attack targeting the control unit can be mounted utilizing an adversarial error model that was defined in Chapter 4.

- We define a useful metric for measuring the effectiveness of the error detection technique in terms of the effort the attacker has to make to mount a successful attack with respect to the area overhead spent in building the EDC.

- We provide a detailed analysis of the complexity of the proposed EDC and show that it provides superior error detection capabilities when compared against tra-

ditional modular redundancy based techniques, while not affecting the critical path delay.

- We provide evidence that our technique provides better scalability compared to traditional modular redundancy based techniques.

This chapter is structured as follows: Section 8.2 contains an example of a fault attack on the control circuit of a hypothetical modular exponentiation based public key accelerator. Section 8.3 will introduce notation and definitions of sequential circuits, as well as classes of faults. State encoding schemes based on error detecting codes (EDC) are introduced in Section 8.4 and used in Section 8.5 to design and analyze fault resilient sequential circuits, complete with an example. Section 8.6 quantifies the error detection capabilities of various linear codes and explains the importance of design diversity to counter certain classes of faults.

8.2 Motivation

We will motivate our research with an example attack scenario. We will demonstrate the theoretical feasibility of an attack on a slightly simplified public key cryptographic accelerator as it might be found in a ubiquitous security device such as a smart card. For sake of simplicity let us assume a basic modular exponentiation algorithm without CRT, using the secret private key, e.g. an RSA decryption or signature generation. It should, however, be straightforward to adapt the attack to a CRT-based implementation. We consider a regular standard-cell ASIC implementation with sufficient protection of the data path, but not the control logic, along with countermeasures to prevent timing and power analysis (balanced power consumption, constant run-time). We furthermore assume that the attacker has the ability to unpack the chip and induce bit-flips on the state registers with some temporal precision [SA02], even though it is our understanding that tamper-proof coating and/or packaging has become some-

thing of a standard practice among smart card manufacturers. The work of Anderson and Kuhn [AK96] gives reason to question the effectiveness of such measures against determined attackers.

A particular algorithm that meets the requirements set above is Joye and Yen's [JY02] variant of the Montgomery Ladder (Alg. 8.1), computing $y = x^e \bmod N$. The state machine in Figure 8.1 implements the algorithm on a data path with a single multiplier and three registers R_0, R_1 and R_2. Each register is n-bits wide; the size of the modulus N. This highly simplified state machine consist of seven states: the six active states, which are listed in Algorithm 8.1 right besides their respective instruction, and the idle state in which the circuit is inactive. The initial state after activation of the circuit sets up constants and counters. The two load states read the variables x and e via the input bus, while the result state returns the computed value y on the output bus. The multiply and square states are self-explanatory from the algorithm description. The control signals a and b for selecting between registers R_0 and R_1 are determined by scanning the exponent in R_2 bit-wise, as indicated by the superscript bit-index in parentheses.

The complete public-key accelerator circuit consists of a modular multiplication unit, a memory block for storage of operands and temporary results, as well as the exponentiation state machine which coordinates the movement of data between multiplier and memory. The multiplier can also be used for squaring operations. Although we will mainly focus on the state machine itself, we want to give a brief overview of the multiplier component, for a more complete picture. It is a pipelined, scalable Montgomery multiplier as described in [Gau02], which can perform arithmetic on both integer and binary polynomial operands. Scalable here refers to the ability to perform arithmetic with operands of nearly arbitrary precision only limited by the amount of memory, as well as the design time configurability of parameters like the number of pipeline stages and data path width. The multiplier computes the

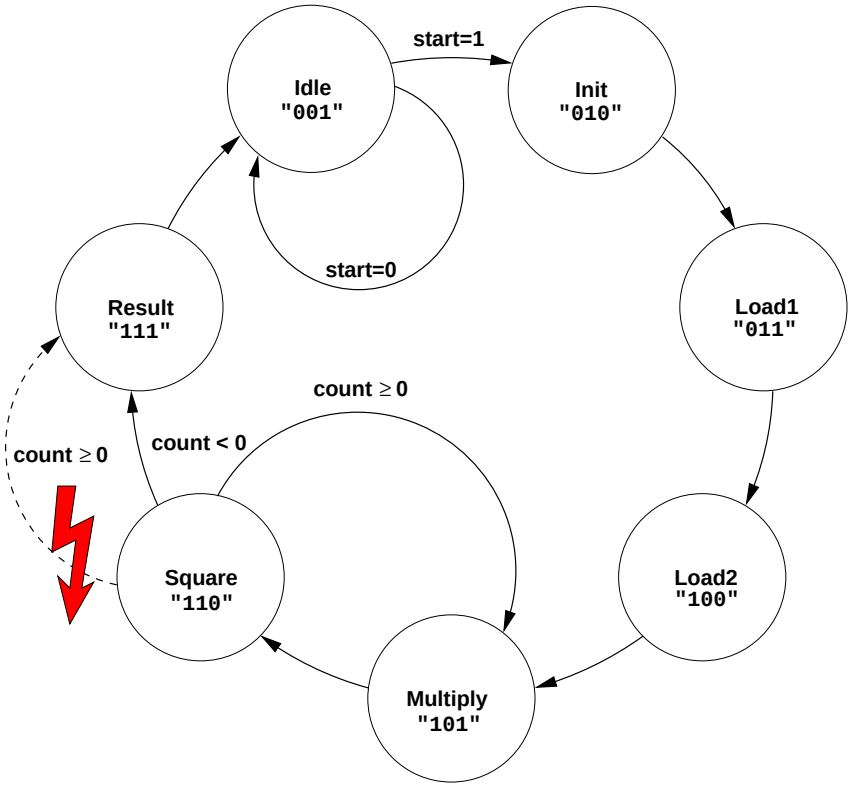


Figure 8.1: State Transition Diagram for Montgomery Ladder with Attack

Algorithm 8.1 Joye and Yen's Montgomery Ladder Exponentiation [JY02]

Require: x, e

Ensure: $y = x^e \bmod N$

$\mathrm{R}_0 \leftarrow 1$	`INIT`
$\mathrm{R}_1 \leftarrow x$	`LOAD1`
$\mathrm{R}_2 \leftarrow e$	`LOAD2`
for $i = n - 1$ downto 0 **do**	
$a \leftarrow \mathrm{R}_2^{(i)}$; $b \leftarrow \neg a$	
$\mathrm{R}_b \leftarrow \mathrm{R}_b \cdot \mathrm{R}_a \bmod N$	`MULTIPLY`
$\mathrm{R}_a \leftarrow \mathrm{R}_a^2 \bmod N$	`SQUARE`
end for	
$y \leftarrow \mathrm{R}_0$	`RESULT`

Montgomery product $C = A \cdot B \cdot R^{-1}$ using the word-level "finely integrated operand scanning" (FIOS) algorithm, which consists of two nested loops. To reflect a typical application scenario in an embedded system, we chose to implement a rather small variant of the multiplier with an 8-bit data path and three pipeline stages. Each stage consists of two 8×8-bit parallel multipliers and two adders, and can process one complete inner loop of the algorithm. Intermediate results are computed LSW first and can be passed from stage to stage. Delay registers with bypass-logic maintain proper scheduling of data words between the pipeline stages. Each pipeline stage has its own inner loop state machine, and the scheduling of each stage is controlled by a single outer loop state machine. Since typically there are fewer pipeline stages than outer loop iterations, a FIFO stores the result of the last stage and passes it back to the first stage when it becomes available again. Table 8.5 in Section 8.5 gives the exact area breakdown of the multiplier components.

We would like to point out that the multiplier currently has no fault detection or other side-channel attack countermeasures. We use it mainly to demonstrate the

rather small percentage of control logic in relation to the overall area of the design. Therefore the total increase in area will stay relatively small, even if the error detection mechanism in the control logic adds significant overhead. This will become visible later on in the synthesis results of our example.

Our main argument, however, is the following: Even if the data path of the circuit were protected by attack countermeasures, the system would still be vulnerable to fault attacks, unless the state machine is protected by an error detection mechanism. The following is a brief outline of such a hypothetical attack, which reveals the n-bit secret exponent e in short time:

1. Measure the total time of an (arbitrary) exponentiation and determine the computation time required per exponent bit: $t_{\text{bit}} \approx t_{\text{total}}/n$.

2. Set the round index $k = 1$, select a known ciphertext x and set the result of the previous round as $(\mathrm{R}_0, \mathrm{R}_1) = (1, x)$.

3. Compute the triple $(A, B, C) = (\mathrm{R}_0^2, \mathrm{R}_0 \cdot \mathrm{R}_1, \mathrm{R}_1^2) \bmod N$ and make a hypothesis H_0 about the exponent bit $e^{(n-k)} \in \{0, 1\}$.

4. Start exponentiating x.

5. After $k \times t_{\text{bit}}$ time steps, force the state machine into the state `111:RESULT`, by inducing a fault either on the least significant bit of the state or on the input signal `count < 0`. This will cause the circuit to reveal its intermediate result in R_0.

6. Verify the hypothesis by comparing the actual value of R_0 to A. If it matches $(e^{(n-k)} = 0)$, set $(\mathrm{R}_1, \mathrm{R}_0) = (A, B)$, else $(e^{(n-k)} = 1)$ set $(\mathrm{R}_0, \mathrm{R}_1) = (B, C)$.

7. Repeat steps 3 through 6 for all other bits of the exponent $(k = 2 \ldots n)$.

This attack works because there is no way to distinguish between a valid or a faulty transition (cf. Figure 8.1: solid vs. dashed arc) from the `110:SQUARE` state to the final state `111:RESULT`. Without a doubt, an actual attack on real hardware will probably be much more involved and not as simple as described here in our naïve scheme. Our basic point is, however, that the control logic of any cryptographic circuit should be identified as a potential point of attack. This is especially true, if other countermeasures against fault attacks on the data path are put into place, since the control logic becomes the weakest element in the system and as such an obvious target. To this end we propose an all-encompassing strategy to protect all the components of embedded cryptographic devices with a certain minimum level of guaranteed resilience to fault attacks.

8.3 Preliminaries and Definitions

A finite state machine can be formally defined by a six-tuple $(S, I, O, \Delta, \Lambda, \mathbf{s}_0)$. S is the set of valid states (the state space), which has dimension $k = \lceil \log_2 |S| \rceil$, and $\mathbf{s} \in S$ denotes the current state. In a similar manner I and O define the input and output sets, $\mathbf{i} \in I$ and $\mathbf{o} \in O$ the current inputs and outputs. The number of input and output signals are $\iota = \lceil \log_2 |I| \rceil$ and $\omega = \lceil \log_2 |O| \rceil$ bits. The sets Δ and Λ represent the next state and output logic functions, which take $\mathbf{s}$ and $\mathbf{i}$ as their arguments. They consist of a collection of boolean functions $\delta_j(\mathbf{s}, \mathbf{i})$ and $\lambda_l(\mathbf{s}, \mathbf{i})$, each of which compute a single bit in the next state or output vector. Finally, $\mathbf{s}_0$ indicates the initial state. For simplicity and without loss of generality we will assume that all 2^k states are valid throughout the remainder of this article. In the following we will refer to state machines that encode the state in a vector with $n > k$ bits as redundant state machines.

A regular (read: non-fault resilient) state machine, Figure 8.2, encodes the current

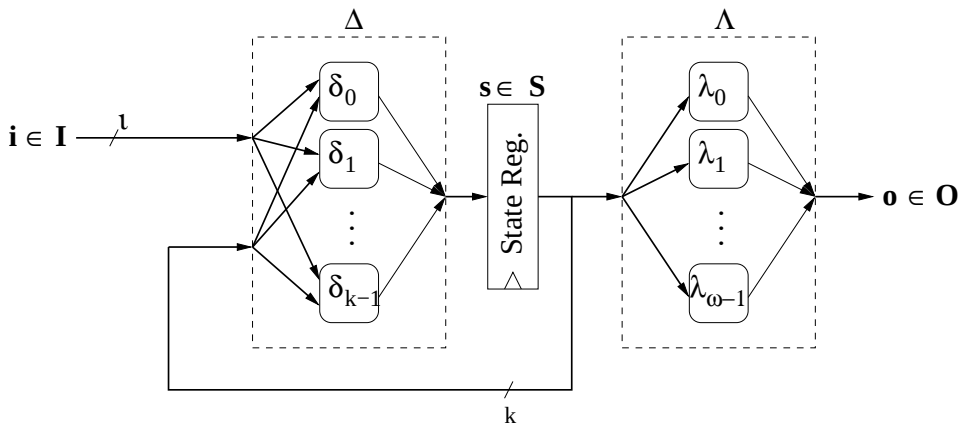


Figure 8.2: Non-redundant State Machine

state as a k-bit binary vector $\mathbf{s} = (s_0, s_1, \ldots, s_{k-1})$. In linear algebra terminology the state space S is a vector space over the field GF(2), spanned by a set of k linearly independent basis vectors $\nu_0 \ldots \nu_{k-1}$. The current state is a linear combination $\mathbf{s} = s_0\nu_0 + s_1\nu_1 + \ldots + s_{k-1}\nu_{k-1}$, with addition in GF(2) being equivalent to an exclusive OR operation. In the simplest case the basis vectors form a standard basis with orthogonal unit vectors:

$$\begin{array}{rcl} \nu_0 & = & (1 \quad 0 \quad \ldots \quad 0) \\ \nu_1 & = & (0 \quad 1 \quad \ldots \quad 0) \\ \vdots & & \vdots \quad \vdots \quad \ddots \quad \vdots \\ \nu_{k-1} & = & (0 \quad 0 \quad \ldots \quad 1) \end{array}$$

The next state $\mathbf{s}'$ is determined by evaluating the set of next state functions:

$$\mathbf{s}' = \mathbf{\Delta}_k(\mathbf{s}, \mathbf{i}) = (\delta_0(\mathbf{s}, \mathbf{i}), \delta_1(\mathbf{s}, \mathbf{i}), \ldots, \delta_{k-1}(\mathbf{s}, \mathbf{i}))$$

Similarly, the ω-bit output vector $\mathbf{o}$ is computed by evaluating the set of output logic functions $\Lambda_\omega(\mathbf{s}) = (\lambda_0(\mathbf{s}), \lambda_1(\mathbf{s}), \ldots, \lambda_{\omega-1}(\mathbf{s}))$. At each clock-edge the current state vector takes on the value of the next state vector $\mathbf{s} = \mathbf{s}'$. Mealy type state machines are different only in that their output functions also depend on the input vector. In

the following we will focus on Moore type machines, but the same principles also apply to Mealy machines.

8.3.1 Attacker Model

In the following we will assume that an attacker is somehow able to induce a fault into the device that results in the flipping of a bit. The attacker has control over the location of the bit flip, but there is a cost (or effort) attached, which increases with the number of bit flips. For example, if the attacker tries to change the state of a circuit by attacking the state register such that $\tilde{\mathbf{s}} = \mathbf{s} \oplus e$, the cost of the attack depends on the hamming weight of the induced error pattern e.

8.3.2 A Novel Effectiveness Metric: Attack Effort per Area

We need a metric by which to compare the effectiveness of various error detection mechanisms. An intuitive approach is to compare the cost or effort of a successful fault attack to the cost of preventing it. Before we can define this metric we need to introduce a couple of further definitions:

Resilience or Resiliency is a measure of how many errors the circuit can withstand. More concretely, this is the number of errors that can be accurately detected by the error detection network. Throughout this chapter we will refer to resilience as the parameter t and to a fault tolerant sequential circuit as t-resilient.

Fault Insertion Effort is a measure for the difficulty of inducing a fault into the system that manifests itself as one or more errors. It can not be given in absolute terms, since an attacker's effort depends on a large variety of factors. We will therefore only give the effort in relative terms, compared to the cost for induction of a single bit-error. There exist multiple types of faults all of which

warrant their own effort values. For simplicity we only model single bit faults, which manifest themselves also as single bit errors at the gate level.

We now define the total attack cost as the sum of the total effort required for a successful attack. We arbitrarily assign a nominal effort of 100% for a single bit error as the basis of comparison for various countermeasures. If the circuit is more resilient, then also the effort for a successful attack increases.

Definition 8.3.1. Let E_{bf} denote the effort required for inducing a change of value for a single bit in the circuit (bit fault). Let t denote the degree of worst-case fault-resilience of the circuit, i.e. the maximum number of errors that can be tolerated when induced simultaneously. The total cost for the attack is defined as $C = (t + 1) \cdot E_{\text{bf}}$.

In Definition 8.3.1, we implicitly assume that the fault insertion effort increases at least linearly with resilience, t. Inducing change of values for more than one bit in the circuit may not be much harder than changing one bit, especially when those bits are stored in places close to each other (e.g. several consecutive bits in a register). However, our proposed scheme necessitates changing a number of certain bits, which are not particularly close to each other, if the fault induction attack is to be successful. Many successful fault induction attacks do not necessarily need a high spatial precision, and inadvertently changing several other bits than the target bits is acceptable in these attack scenarios. The proposed technique, on the other hand, forces the attacker to be very precise when changing certain bits lest the attack be detected. Therefore, changing several bits, not necessarily next to each other in the circuit, without changing other bits is indeed a daunting task for the attacker. The exact relation of fault insertion effort to the resilience, t depends largely on the specific implementation, and therefore is difficult to formulate. Nevertheless, our assumption of a linearly increasing effort for fault insertion with respect to t may be justifiable at least as a first order approximation, since it is our intuition that the relation is even more complex and the effort should increase faster than just linearly.

Based on Definition 8.3.1 and the area overhead which is required to implement a specific countermeasure, we can define an effectiveness metric η which allows us to compare the different approaches. The figure of merit is the ratio between cost and area. Additionally we define a normalized effectiveness H which can be useful for comparing countermeasures on circuits with completely different functionality.

Definition 8.3.2. Let A denote the circuit area and C the cost for successfully implementing a fault attack. We define the effectiveness of the circuit as the ratio $\eta = C/A$ between the cost of the attack (in terms of effort) and the cost of the countermeasure (in terms of circuit area). Let further $\eta_0 = C_0/A_0$ be the effectiveness of the unprotected circuit. The ratio $H = \eta/\eta_0$ denotes the normalized effectiveness of a countermeasure with respect to a particular basis circuit.

In Section 8.5 we will use the effectiveness metric to compare traditional fault tolerance approaches with our method.

8.4 Redundant State Encoding

Looking at typical finite state machine implementations of cryptographic algorithms, one may notice that oftentimes only relatively few different states are necessary, or if a more complex algorithm is required the control logic can be broken down hierarchically. Typically only a few bits are necessary to store the state, depending on the encoding scheme used. Compared to the size of the data path in a typical embedded cryptographic system, the size of the control logic is often relatively small, even when a highly redundant encoding scheme is being used. As an example we refer to the Montgomery multiplier introduced above, which has a total area of 10,410 equivalent gates, excluding the area for the data storage. The control logic occupies an absolute area of 2,015 equivalent gates, around 20%; the remaining 80% belongs to the data path.

Electronic design automation tools often allow specification of a preferred state encoding scheme for the automatic synthesis of finite state machines. The typical styles to choose from are "one-hot", "gray" or "binary" encoding, but they only allow a trade-off between the speed and the area of the resulting circuit. We propose to investigate a third trade-off alternative, i.e. the degree of fault tolerance or, put differently, the resilience against fault attacks. Although fault tolerant sequential circuits have been the subject of research in the context of reliable system design, there are a couple of important aspects that require further investigation in the context of cryptographic circuits, due to the fundamentally different adversarial fault model.

We place a special emphasis on the detection, rather than correction, of faults. Our argument is quantitative in that the number of detectable faults $(d-1)$ is larger than the number of correctable faults $(\lfloor (d-1)/2 \rfloor)$, based on the minimum Hamming distance d of the coding scheme. Since there is no way of telling whether the multiplicity of the fault is strictly less than $d/2$ or not, there is a chance that the error correction will produce a valid, but ultimately incorrect next state. On the other hand we also argue qualitatively: due to the adversarial nature of faults in the cryptographic context and their potentially devastating effects on the security of the system, the detection of errors is far more important than producing a result under all circumstances. Detected faults should rather be dealt with under a suitable exception handling mechanism, that prevents side-channel leakage, e.g. by erasing sensitive key material. As a side benefit, the area overhead for detection is typically much less than that of correction and the circuit does not have to sit on the critical path.

One-hot encoding is preferred by digital logic designers whenever speed and a simple next state logic are desired. Despite its large amount of redundancy, its minimum distance properties are weak, since valid codewords only differ by two bits. In the case of a double error which resets the active bit and sets another one there is not enough information in the encoding to detect the error, and thus one-hot en-

coding is not suitable for our purposes. Since performance is less of a priority in security-critical applications, we can trade off the speed of one-hot encoding for the capability to detect errors. One way of doing so is by choosing a state encoding based on *error detecting codes* (EDC) with large minimum distance. For example, if we had a state machine with $m = 15$ states, one-hot encoding would require a 15-bit state vector. Encoding the states with a (15,4)-Simplex code, which is the dual of a (15,11)-Hamming code, would require a state vector of exactly the same size. One-hot encoding only has a minimum distance of 2, while the Simplex code has a distance of 8, which allows detection of all errors with a Hamming weight of up to 7.

8.5 Fault-Resilient Sequential Circuits

A fault-resilient state machine (Figure 8.3) must necessarily incorporate redundancy—not only in the state encoding, but also in the combinational logic of next state and output functions. In a state machine protected by a linear (n, k) code, the redundant state vector $\mathbf{s}^{(n)} = (s_0, s_1, \ldots, s_{n-1})$ now consists of n bits. In the following we will also implicitly assume that the input signal $\mathbf{i}$ is given in redundant form as well. Its next-state logic is a set $\mathbf{\Delta}_n(\mathbf{s}, \mathbf{i})$ of n functions δ_j, that each determine one bit of $\mathbf{s}$. In general, since there are only 2^k states (but encoded in redundant form), each δ_j is a boolean function over $k + \iota_k$ variables (ι_k denotes the number of non-redundant bits in the input vector $\mathbf{i}$).

Formally speaking, the state space $S^{(n)}$ is a k-dimensional subspace of an n-dimensional vector space $\{0, 1\}^n$. The basis vectors of this subspace are the n-bit row vectors of the generator matrix $\mathbf{G}$ of the code. Any value of the state vector $\mathbf{s}$ which is not a linear combination of these basis vectors is therefore treated as an error. While this interpretation nicely illustrates the mapping of k-bit states into redundant n-bit form, it is not useful for the definition of next state functions, because we want to keep

the state only in redundant form. If we were to decode the current state, compute the next state in only k bits and re-encode it to n bits, this would leave the system with a single point of failure in the next state function rendering it vulnerable to attacks. Rather, we keep the system state in redundant form at all times, to enhance its resilience against attackers probing for the weakest spot of the system.

An alternative interpretation that is more suitable for our definition of a redundant state machine is to view the columns of $\mathbf{G}$ as n redundant basis vectors of the non-redundant k-dimensional state space S, which is isomorphic to $S^{(n)}$. Any k *linearly independent* basis vectors form a complete basis for S and span the entire state space. Thus, those k (out of n) state variables that are associated with the respective vectors of such a basis can be used for specification of a next state or output function. In the following we will let σ_j and ξ_j denote such sets of k state variables associated with linearly independent basis vectors (columns of $\mathbf{G}$), for the next state and output functions, δ_j respectively λ_j. Additionally we introduce non-redundant subsets κ_j of input variables (one per next state function). Each κ_j selects ι_k input variables from the redundant input vector $\mathbf{i}$. If the columns of $\mathbf{G}$ were not linearly independent, then this would introduce an ambiguity about the current state and certain functions could not be implemented. It usually depends on the exact construction of the code to determine which of all $\binom{n}{k}$ possible sets of k state variables are suitable.

If enough many sets $\sigma_j,\ \kappa_j,\ \xi_j$ are available, then the next state and output logic functions can be defined over a variety of such sets. It is prudent to make use of such alternative definitions, so as to minimize the hazardous effect that a single faulty state variable has on the overall system.

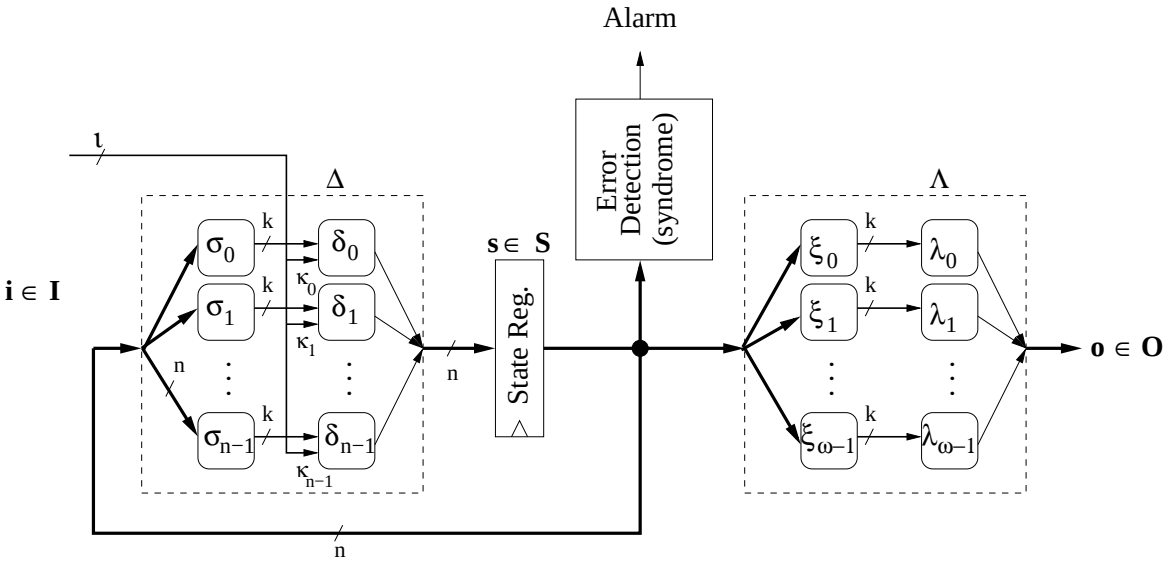


Figure 8.3: Fault-resilient State Machine

Example

An example will help to illustrate the last point: Let $\mathbf{G}$ be the generator matrix of a systematic $(7, 3)$ Simplex code (dual of the (7,4)-Hamming).

$$\mathbf{G} = \begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}$$

The rows of $\mathbf{G}$ form the k-dimensional basis for the state space S as a subspace of $\{0, 1\}^n$. Let $\mathbf{s}^{(n)} = \mathbf{s}^{(k)}\mathbf{G}$ be the redundant state vector. Using the modular exponentiation example from Section 8.2, the states of a redundant state machine would be encoded as in Table 8.1. Alternatively we can interpret the columns of $\mathbf{G}$ as redundant basis vectors $\nu_0, \nu_1, \ldots, \nu_6$ spanning S, and associated with $s_0, s_1, \ldots, s_6$, respectively.

In a non-redundant state machine the next-state logic functions would be specified as $\delta_j(\mathbf{s}^{(k)}, \mathbf{i})$. In the redundant case, we still have the same number of states, but now we have $n = 7$ state variables $(s_0, s_1, \ldots, s_6)$ that we can use to specify the next-state

Table 8.1: Simplex State Encoding for the Modular Exponentiation Example

State	$\mathbf{s}^{(k)} = (s_0, s_1, s_2)$	$\mathbf{s}^{(n)} = (s_0, s_1, s_2, s_3, s_4, s_5, s_6)$
`Idle`	001	0010111
`Init`	010	0101011
`Load1`	011	0111100
`Load2`	100	1001101
`Multiply`	101	1011010
`Square`	110	1100110
`Result`	111	1110001

functions. Due to systematic encoding we have the following relationships between redundant $(s_0, s_1, s_2, s_3, s_4, s_5, s_6)^{(n)}$ and non-redundant state variables $(s_0, s_1, s_2)^{(k)}$:

$$\begin{aligned}
s_0^{(n)} &= s_0^{(k)} \\
s_1^{(n)} &= s_1^{(k)} \\
s_2^{(n)} &= s_2^{(k)} \\
s_3^{(n)} &= s_0^{(k)} \oplus s_1^{(k)} \\
s_4^{(n)} &= s_0^{(k)} \oplus s_2^{(k)} \\
s_5^{(n)} &= s_1^{(k)} \oplus s_2^{(k)} \\
s_6^{(n)} &= s_0^{(k)} \oplus s_1^{(k)} \oplus s_2^{(k)}
\end{aligned}$$

Let for example $\sigma_j = (s_1, s_3, s_6)$. Since columns 1, 3 and 6 of $\mathbf{G}$ are linearly independent, it is generally possible to specify an arbitrary next state or output function over this set of state variables, e.g. $\delta_j(\sigma, \mathbf{i}) = \delta_j(s_1, s_3, s_6, i_0)$. This is always possible and does not depend on what the actual application of the state machine is[1]. The boolean function definition of δ_j, however, does depend on the application.

[1]Once a specific function is defined, it might not need the complete state information in k variables and the logic implementation may be minimized.

Table 8.2: State Transition Table of the Fault-Resilient FSM

State	s_0	s_1	s_2	s_3	s_4	s_5	s_6	start	cnt< 0	s'_0	s'_1	s'_2	s'_3	s'_4	s'_5	s'_6
(illegal)	0	0	0	0	0	0	0	x	x	0	0	0	0	0	0	0
idle	0	0	1	0	1	1	1	0	x	0	0	1	0	1	1	1
idle	0	0	1	0	1	1	1	1	x	0	1	0	1	0	1	1
init	0	1	0	1	0	1	1	x	x	0	1	1	1	1	0	0
load1	0	1	1	1	1	0	0	x	x	1	0	0	1	1	0	1
load2	1	0	0	1	1	0	1	x	x	1	0	1	1	0	1	0
mult	1	0	1	1	0	1	0	x	x	1	1	0	0	1	1	0
sqr	1	1	0	0	1	1	0	x	0	1	0	1	1	0	1	0
sqr	1	1	0	0	1	1	0	x	1	1	1	1	0	0	0	1
res	1	1	1	0	0	0	1	x	x	0	0	1	0	1	1	1

Conversely, it should immediately become clear that we should not specify a next-state function over the sets (s_0, s_1, s_3) or (s_0, s_2, s_4). The bases associated with the selection of state variables are $\{\nu_0, \nu_1, \nu_3\} = \{(1,0,0),(0,1,0),(1,1,0)\}$ and $\{\nu_0, \nu_2, \nu_4\} = \{(1,0,0),(0,0,1),(1,0,1)\}$, both of which are not linearly independent: $\nu_0 + \nu_1 + \nu_3 = \mathbf{0}$ and $\nu_0 + \nu_2 + \nu_4 = \mathbf{0}$ in GF(2). In both examples one coordinate is always zero in all basis vectors. If a hypothetical next state function depends on that coordinate to distinguish between two different states, the function cannot be implemented.

We will now give a concrete example of a fault-resilient state machine implementation using the error detecting $(7, 3)$ Simplex code. We will further compare our approach to N-modular redundancy, a concept from traditional fault tolerant computation, with respect to area overhead and fault-resilience. In N-modular redundancy functional modules are replicated N times, ideally employing design diversity techniques. Their outputs are either voted on by a majority logic (i.e. for error correction) or simply compared for consistency to detect any error that may occur in a module.

The most common configuration is *triple modular redundancy* (TMR), sometimes also referred to as triads [HJSI75], which allows either the detection of two errors or the correction of a single error. Since a fault-resilient system requires redundancy not only in the state machine but overall, redundant versions of the input and output signals need to be present as well. In a system with non-redundant inputs, for example, the attack described in Section 8.2 could be carried out by forcing the condition count < 0, which may be indicated to the state machine by a single status bit. Such an attack is harder to accomplish if input signals are available as multiple redundant and independent variables. The amount of redundancy should be chosen to protect against the same number of errors t as the state encoding scheme, to ensure a uniform level of resilience for the entire system. The level of resilience is measured by the minimum number t of signals an attacker needs to change (bit-flips) for a successful attack. For state encoding with linear codes over GF(2) we have $t = d - 1$, where d is the minimum distance (or non-zero Hamming weight) of all code words. In a modular redundant scheme the overall resilience level can be found by concatenating the state vectors of the individual modules and computing its total minimum distance D and resilience $t = D - 1$. They are determined by the degree of modular redundancy N and the distances d_j of each module's individual state encoding:

$$D = \sum_{j=1}^{N} d_j$$

In the following we will therefore require a D-fold redundant set of input signals to be available to the circuit, which in turn must provide an D-fold redundant set of output functions. This provides coverage for the corner case that all t faults occur on the inputs. For example, in the case of a TMR system with simple binary encoded states the minimum distance between words of the concatenated state vectors is 3 ($D = N$) with a worst case resilience of $t = 2$. Our EDC-based state machine implementation is a special case with $N = 1,\ d = 4$ and therefore requires an input set of distance 4

to maintain the same amount of resilience against $D - 1 = 3$ faults.

We will now provide an example using the Montgomery Ladder algorithm introduced earlier in section 8.2. A very simple state machine implementation requires two input signals, `start` and `count!=0`, and the D-fold redundant input vector thus consists of the signals $\mathbf{i} = (i_{0,0}, \ldots, i_{0,D-1}, i_{1,1}, \ldots, i_{1,D-1})$, with the mapping $i_{0,j}$ = `start` and $i_{1,j}$ = `count!=0`. It computes three output signals for controlling the data path (multiplier): `mul_op_a, mul_op_b` and `mul_res`. Again, these signals will be generated with D-fold redundancy and mapped to the output vector $\mathbf{o} = (o_{0,0}, o_{0,1}, o_{0,2}, \ldots, o_{2,D-1})$.

As with the redundant state vector, we will not use all elements of $\mathbf{i}$ for definition of the next state functions, but only subsets κ_j, each containing two input variables. Finally, we need ω sets ξ_j of k state variables each as inputs to the output functions λ_j, again with minimal overlap. For our example we chose:

$$
\begin{array}{lcllcl}
\sigma_0 &=& (s_0, s_2, s_3) & \kappa_0 &=& (i_{0,0}, i_{1,0}) \\
\sigma_1 &=& (s_1, s_3, s_4) & \kappa_1 &=& (i_{0,1}, i_{1,1}) \\
\sigma_2 &=& (s_2, s_4, s_5) & \kappa_2 &=& (i_{0,3}, i_{1,2}) \\
\sigma_3 &=& (s_3, s_5, s_6) \ , & \kappa_3 &=& (i_{0,0}, i_{1,3}) \ , \\
\sigma_4 &=& (s_0, s_4, s_6) & \kappa_4 &=& (i_{0,1}, i_{1,0}) \\
\sigma_5 &=& (s_0, s_1, s_5) & \kappa_5 &=& (i_{0,2}, i_{1,2}) \\
\sigma_6 &=& (s_1, s_2, s_6) & \kappa_6 &=& (i_{0,3}, i_{1,3})
\end{array}
$$

$$
\begin{array}{lcllcl}
\xi_0 &=& (s_0, s_1, s_2) & \xi_6 &=& (s_0, s_4, s_5) \\
\xi_1 &=& (s_0, s_3, s_5) & \xi_7 &=& (s_2, s_5, s_6) \\
\xi_2 &=& (s_3, s_4, s_6) & \xi_8 &=& (s_1, s_2, s_6) \\
\xi_3 &=& (s_2, s_3, s_4) & \xi_9 &=& (s_0, s_3, s_4) \\
\xi_4 &=& (s_1, s_2, s_4) & \xi_{10} &=& (s_1, s_5, s_6) \\
\xi_5 &=& (s_3, s_5, s_6) & \xi_{11} &=& (s_0, s_1, s_6)
\end{array}
$$

From the state transition table (Table 8.2) we can now derive the concrete next state

functions based on the state and input variable sets σ_j and κ_j:

$$\begin{aligned}
\delta_0(\sigma_0,\kappa_0) &= s_0 s_2' + s_2 s_3 \\
\delta_1(\sigma_1,\kappa_1) &= i_{0,1} s_1 s_3' s_4 + i_{0,1} s_3' s_4 i_{1,1} + s_1' s_3' s_4 i_{1,1} + s_3 s_4' \\
\delta_2(\sigma_2,\kappa_2) &= s_2 s_4' s_5' + s_2' s_4 + s_2' s_5 + s_4 s_5 i_{1,2}' \\
\delta_3(\sigma_3,\kappa_3) &= s_3' s_5 s_6' i_{0,0}' + s_3 s_5' + s_3 s_6 + s_5 s_6 i_{1,3} \\
\delta_4(\sigma_4,\kappa_4) &= s_0 s_4' + s_0' s_4 s_6' + s_0' s_4 i_{1,0}' + s_4' s_6 \\
\delta_5(\sigma_5,\kappa_5) &= i_{0,2}' s_0 + s_0 s_5' + s_1' s_5 \\
\delta_6(\sigma_6,\kappa_6) &= i_{0,3} s_1 s_6' + s_1 s_2 + s_2 s_6
\end{aligned}$$

We can derive the output logic equations in a similar fashion (omitted here due to space considerations).

8.5.1 Synthesis Results and Overhead Analysis

Due to the principal difference in error detection capability between an N-modular redundant and an EDC-based state machine, we decided to compare the non-redundant implementation to four different redundant implementations with slightly varying parameters:

1. a 2-fault resilient triple modular redundant (TMR, $N = 3$) version
2. a 3-fault resilient quadruple modular redundant (QMR, $N = 4$) version
3. an internally 3-fault resilient EDC version with a 3-fold redundant I/O set (3-EDC)[2].
4. a 3-fault resilient EDC version with a 4-fold redundant I/O set (4-EDC).

Our initial analysis consisted of using the UC Berkeley SIS logic synthesis tool and the MCNC synthesis script for mapping the five different circuit descriptions to the

[2]This effectively reduces the worst case fault resilience to 2 faults on the input set.

MCNC standard libraries `mcnc.genlib` and `mcnc_latch.genlib` and comparing the results (Table 8.3).

8.5.1.1 Area Overhead

Table 8.3: Initial Analysis of Various EDC and NMR Schemes

	Non-red	3-EDC	4-EDC	TMR	QMR
# Inputs	2	6	8	6	8
# Outputs	3	10	13	10	13
# SOP-Literals	42	155	155	135	179
Gate Count	19	56	60	59	77
# Latches	3	8	8	10	13
Total Area	64	207	210	210	276
Area Overhead (%)	0	223	228	228	331

Despite the higher literal count of 3- and 4-EDC compared to the TMR implementation, the actual circuit area is almost exactly the same. The lower resilience of TMR against faults, however, makes a compelling argument for building fault resilient state machines based on error detecting codes, especially since the degree of resilience will be even better for larger state space dimension k. A quadruple modular redundant design requires more than 100% more overhead for achieving the same level of resilience. In addition, while N-modular redundancy schemes have a constant overhead of approximately $(N-1)\cdot 100\%$ independent of the state space dimension k, the (storage) overhead of EDC based fault resilient state machines actually decreases with a larger state space (cf. Fig. 8.4). It appears reasonable to expect a similar trend for the complexity of the next state logic.

These initial results encouraged us to proceed with the analysis of a more detailed state machine model synthesized using Synopsys DesignCompiler and a 0.13μm ASIC

standard cell library. Special care was taken to direct the synthesis not to share logic between the next state functions δ_n. This is to avoid the manifestation of a single stuck-at fault as an error on multiple outputs of the next state function $\Delta_n(\mathbf{s}, \mathbf{i})$. The results[3] are given in Table 8.4. It shows some clear differences in area requirements

Table 8.4: Analysis of Detailed FSM Models (*10-state FSM with $k = 4$)

	Non-red	3-EDC	4-EDC	TMR	QMR	non-red.*	EDC
Total Area	48	137	143	189	249	71	361
Area Overhead	0.0%	188.1%	199.3%	295.8%	422.4%	0.0%	413.3%
Resilience t	0	2	3	2	3	0	7
Attack Effort	100%	300%	400%	300%	400%	100%	800%
Effectiveness η	2.098	2.184	2.804	1.590	1.606	1.422	2.216
Norm. Effect. H	1.000	1.041	1.336	0.758	0.766	1.000	1.559

between the codes based and the TMR implementation, which are not visible from the initial analysis. To some extent this is due to the more accurate modeling (we only used a simplified state machine design for the SIS flow), but the main reason is the fact that we used an industry-strength standard cell library with advanced complex gates of varying drive strength. The size of these gates is often more compact than a realization from simple gates, and given in precise decimal fraction area numbers. The size of gates in the MCNC library on the other hand was only given in integers.

8.5.1.2 Effectiveness and Scalability

Table 8.4 also gives the performance of our examples in terms of the effectiveness metric for fault resilience measures that we defined in Section 8.3, Definition 8.3.2. The value η describes the relative effort per gate, while H is the normalized effectiveness value relative to the effectiveness of the unprotected circuit. The results are quite

[3]The area is given in terms of equivalent gate size, i.e. the size relative to a 2-input NAND gate.

interesting: As expected, the EDC based fault resilient circuits show the best effectiveness in preventing fault attacks, however, the N-modular redundancy approach fares much worse than having no countermeasure at all. We also experimented with a slightly different implementation with 10 states and a much larger resilience $t = 7$. Here we used a (15,4)-Simplex code for state encoding. Our results indicate that at slightly more than 400% area overhead the EDC based scheme scales much better than N-modular redundancy, for which we expect an overhead of well beyond 800% at the same resilience level.

8.5.1.3 Critical Path Overhead

We could not determine any substantial critical path overhead with any of our examples. All designs synthesized at 1GHz clock frequency without problems, and we would expect the critical path to be on the data path much rather than the control logic. In the case of 3-EDC and 4-EDC the redundant state variables are computed in parallel, completely independent of each other, and therefore do not add to the critical path. Furthermore, the error detection network is not in the critical path either, since it operates in parallel with the next state and output logic. We conclude that there is no performance overhead associated with these fault resilience measures.

8.5.2 Influence of Control Logic Overhead on Total Circuit Area

Following the overhead analysis of the exponentiation state machine we want to extrapolate the total overhead of implementing fault resilient control logic throughout the entire modular exponentiation circuit. We start from the break-down of area requirements for the Montgomery multiplier described in Section 8.2. Table 8.5 shows the components of the multiplier sorted by type and their size in equivalent gates. For extrapolation we factored the overhead estimates from Table 8.4 into the area

Table 8.5: Data Path / Control Logic Area Comparison of Montgomery Multiplier

Component (Type)	Area (equiv. gates)	Percentage
Pipeline Stages (Data Path)	6,756	64.90%
Delay Registers (Data Path)	1,640	15.75%
Inner Loop FSM (Control Logic)	1,079	10.36%
Outer Loop FSM (Control Logic)	848	8.15%
FIFO control (Control Logic)	87	0.84%

requirement for control logic, but not for the data path. Due to the fact that control logic occupies less than 20% of the total area in a non-redundant implementation, the overhead due to fault resiliency techniques is limited to around 40% for our method, whereas for quadruple modular redundancy it exceeds 80% (Table 8.6).

Table 8.6: Total Overhead for Modular Exponentiation Circuit with FT Control

	Non-red	3-EDC	4-EDC	TMR	QMR
Data Path	8,396	8,396	8,396	8,396	8,396
Control Logic	2,062	5,942	6,173	8,163	10,773
Total Area	10,458	14,338	14,569	16,559	19,169
Overhead	0.00%	37.09%	39.30%	58.33%	83.29%

8.6 Selection of Codes for t-Fault Resilient State Machines

We have seen in the previous example how we can design a fault-resilient state machine with up to 8 states using an ordinary (7,3) Simplex code. Since this code has a minimum distance of four, it is able to detect up to three faults in the state vector

s. In general, the number of errors that can be detected by means of a linear block code with minimum distance d is $t = d - 1$. The selection of a particular code for a fault-resilient state machine implementation therefore depends on the desired level of resilience, i.e. the number of detectable errors t, as well as the dimension of the state space k. These two parameters determine the length n of the block code, as shown in Table 8.7. We obtained our numbers from the reference table of minimum-distance bounds for binary linear codes [HS73]. We restructured the table in order to display the minimum length n for a given dimension k and desired fault-resilience t.

Table 8.7: Minimum Code Lengths n for State Space of Dimension k and Fault Resilience t

t \ k	1	2	3	4	5	6	7
1	2	^{s}3	4	5	6	7	8
2	3	5	6	7	9	10	11
3	4	6	^{s}7	8	10	11	12
4	5	8	10	11	13	14	15
5	6	9	11	12	14	15	16
6	7	11	13	14	15	17	18
7	8	12	14	^{s}15	16	18	19

The first column of the table consists entirely of trivial repetition codes, because of $k = 1$. The first row, on the other hand, consists of all parity codes with no repetitions of column vectors, but with only a minimum distance of 2. These two are the extremes between which we can trade off the fault resilience versus the code length (or rather compactness). This is visualized in Figure 8.4, where EDC with different levels of fault resilience are juxtaposed with one-hot and binary encoding in a double log-scale graph. For a very small number of states adding resilience against multiple faults incurs some overhead penalty, even when compared to one-hot encoding. But

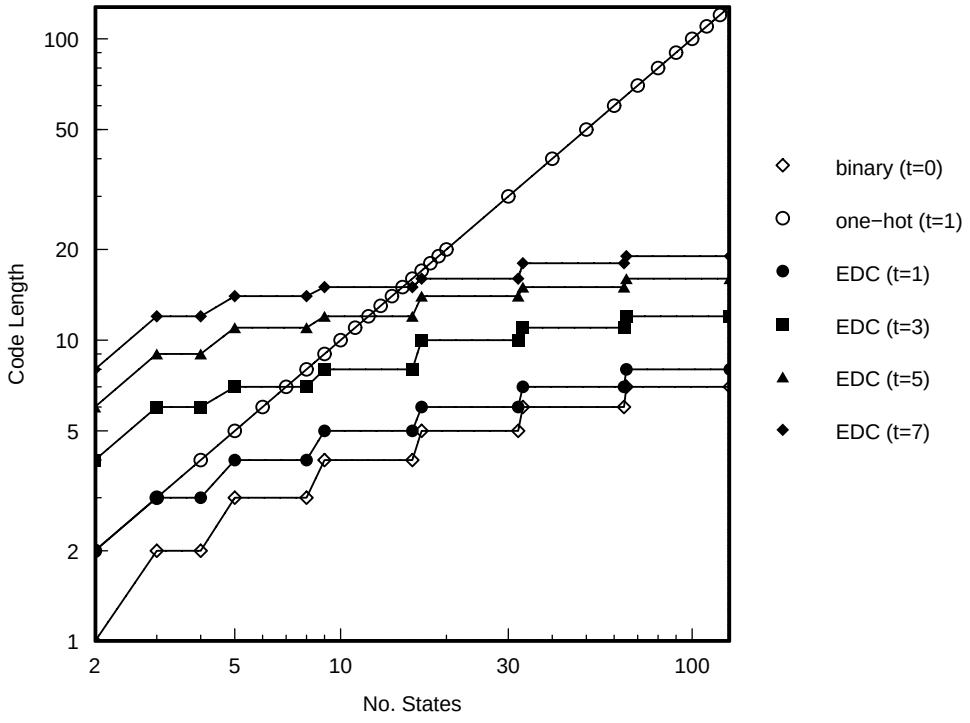


Figure 8.4: Lengths of Various t-Resilient Error Detecting Codes

as the state space grows, the complexity of one-hot encoding grows much more rapid with each additional state and eventually becomes infeasible. On the other hand, the relative overhead of error detecting codes over simple binary encoding decreases monotonically.

Duplication of columns in $\mathbf{G}$ is an easy and efficient way to increase the minimum distance—and therefore the resilience—of an EDC. This is helpful against adversarial single-mode failures such as light attacks. In fact, whenever the code length n exceeds 2^k, $\mathbf{G}$ inevitably contains some duplicated column vectors. Take, for example, the $(10, 3)$ code of resilience four: With $k = 3$ there can only be seven unique non-zero column vectors in $\mathbf{G}$; hence there must be three duplicates. It turns out that Simplex codes achieve maximum fault resilience possible without duplication of columns. Their generator matrix consists of all non-zero column vectors of dimension

k. They are marked in Table 8.7 with the small letter s. If $\mathbf{G}$ contains duplicate columns, this means that two or more state variables associated with those columns will have the same value (under fault-free conditions), and could thus be computed by identical next state functions. The inherent problem of identical functions is their susceptibility to common-mode failure as defined earlier in Section 4. If the cause of a fault is not locally limited, it is likely to have the same effect on identical circuit implementations of a boolean function, and therefore increase the fault multiplicity.

8.6.1 Using Design Diversity to Counter CMF

We can counter the effects of CMF using a technique called *design diversity* [MM00]. By implementing the next state function in several different, but ultimately equivalent ways, the likelihood of combined failure under the same circumstances is reduced. Design diversity is naturally promoted by defining functions over alternative sets of k state variables, as described in Section 8.5. It is, however, important to not choose the σ_j naïvely. For example, if we were to define the next state functions δ_j over only those sets σ_j that all include the state variable s_0, then we would have biased the circuit. A fault in s_0 might spread out to cause subsequent errors in all other state variables. An active adversary could systematically test for such a bias and exploit it. We must therefore balance the use of each state variable across all σ_j. In order to reduce the influence of a single bit error in any of the state variables, the subsets σ_j should uniformly cover all state variables, with minimum overlap. In a k-dimensional state space the minimum cover implies that each state variable be used a maximum of k times. The goal is therefore to find n sets σ_j chosen uniformly from $\mathbf{s}_{(n)}$ such that each state variable occurs only k times. Oftentimes a minimum cover can be found relatively easy, simply by inspection. Automating the process should be fairly straightforward, but lies outside of the scope of this work. Returning to our previous

example with the (7,3) Simplex code we could choose the subsets as follows:

$$\begin{bmatrix} \sigma_0 \\ \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \sigma_4 \\ \sigma_5 \\ \sigma_6 \end{bmatrix} = \begin{bmatrix} (s_0, s_2, s_3) \\ (s_1, s_3, s_4) \\ (s_2, s_4, s_5) \\ (s_3, s_5, s_6) \\ (s_0, s_4, s_6) \\ (s_0, s_1, s_5) \\ (s_1, s_2, s_6) \end{bmatrix}$$

One can readily verify that the basis vectors associated with the sets σ_j are all linearly independent and no state variable gets used more than k times. In other words, there is no single state variable that stands out over others as a potential target for an attack.

8.6.2 Attack Resilience

Finally we would like to show intuitively how the fault resilient circuit holds up in different attack scenarios. We start with the most likely point of attack for an adversary with single-mode fault insertion capability, the state register, and continue from there towards the boundaries of the sequential circuit.

State register. A transient fault can completely change the register contents past the duration of the fault, since the flip-flops are made up from bistable devices. Here the structural design of the fault-resilient state machine comes into play: Since the error detection network is connected directly to the output of the state register, all faults up to multiplicity t will be detected. Note that this would not be the case if the detector were to directly follow the next state function. From our choices of σ_j one can see that each state variable has an influence on up to k variables of the next state and the attacker would only require a fraction $\lceil t/k \rceil$ of

faults to influence more than t state variables. With error detection right after the state register this threat is eliminated. Furthermore, the detection network does not add to the critical path, thereby avoiding a performance penalty.

Feedback path. Any fault on the feedback path from the state register to the inputs of the next state function would be detected by the error detection circuit as well.

Next state logic. Fault multiplicity stemming from shared logic between individual next state functions is not likely, due to the minimum cover policy of selecting the sets σ_j, i.e. there should not be any pair of state variables that is a member of more than one σ_j. But even if that could happen, it would be easy to simply disallow sharing of logic gates between individual functions during synthesis.

I/O set. The input and output vectors must also be considered as a potential point of failure, as mentioned earlier. It is not clear at this point, if the error detection mechanism of the sequential circuit alone can guarantee t-fault resilience. It might be necessary to have additional error detection outside of the state machine, checking for faults on the I/O set, but this is outside of the scope of this work. The minimum requirement is that the I/O set should contain a sufficiently high amount of redundancy to enable t-fault resilience, as described in the example in the previous section. One potential solution is to use weight-based codes as described in [DT99].

As we can see from this intuitive analysis, the architecture of our sequential circuit can withstand fault insertion of degree t, which was our design goal.

8.7 Discussion

In this chapter we addressed the importance of not only protecting the data path of cryptographic circuits against fault attacks, but also the control logic itself. Despite the abundance of fault tolerance techniques for sequential circuits in other application domains, those are based on a fundamentally different fault model and therefore not suitable for the task. We strongly believe that these techniques have to be re-evaluated and adapted to the new threat scenario that presents itself in the form of cryptographic fault analysis. A determined attacker can probe a system for its weaknesses and enforce a worst case scenario. For protection against an adversary with the capability to induce t faults simultaneously, the circuit must be resilient up to that degree. Most of the ideas proposed in the literature, however, are concerned with efficiency and the ability to detect two and correct at most a single error. This work is a first step towards fault resilient sequential circuits under an adversarial fault model. We gave a motivating example of a hypothetical attack on a modular exponentiation based crypto system that would succumb to this kind of attack, in spite of a protected data path and other algorithmic countermeasures. While it is impossible to ever completely protect against a determined adversary, we presented some general principles of how to improve the resilience against an attack, to make it infeasible beyond a certain level of effort.

We showed that the overhead incurred through the use of error detecting codes is less than that of N-modular redundant implementations, while providing better fault resilience properties. In addition to this we demonstrated that the percentage of control logic in a typical cryptographic application is only a small fraction of the data path. Thus, even a high amount of overhead due to fault resilience measures does not significantly influence the complexity of the overall system. It is, however important to understand the mechanisms of fault propagation to properly implement fault resilience. The rule for minimal overlap between state variables in the sets σ_n

is such an example. Furthermore, we defined a new metric which captures the effectiveness of a fault detection method in terms of the minimum effort the attacker has to make to mount a successful attack with respect to the area overhead spent in implementing the error detection scheme. Our analysis and implementation results have shown that the proposed coding based techniques provide far superior performance when compared against N-modular redundant error detection techniques. For proof of concept we implementated a Montgomery muliplier whose control logic was realized using various error detection techniques. Our implementation results show that the control logic occupies less than 20% of the total area in a non-redundant implementation, and the overhead due to fault resiliency techniques is limited to around 40% for our method, whereas for a comparable quadruple modular redundancy it exceeds 80%. Furthermore, the effectiveness metric gives a high value of 2.8 for the proposed technique whereas for QMR it is a mere 1.6. At around 75% effectiveness after normalization it is clear that N-modular redundancy performs even less effective than the unprotected circuit. For the same implementation, we found that the critical path delay was not affected by the EDC. We provided rationale to claim that this result would hold for even higher degrees of resiliency.

Further research is certainly required in this area. It is our hope that this first step will encourage others to further contribute to this important aspect of embedded cryptography. One open question, for example, is whether other types of codes, e.g. Reed-Solomon etc., would provide for higher effectiveness. For the time being we only investigated linear block codes due to the simpler error detection mechanism.

Chapter 9

Conclusion and Outlook

This dissertation we addressed the question of how to protect hardware implementations of public key cryptography against tampering attacks of a skilled adversary.

We presented three different ways to protect the finite field arithmetic of public key cryptosystems. Each technique has its unique applications, advantages and disadvantages, and their uses should be carefully evaluated on an individual basis. Additionally, we realized that it is insufficient to only protect the data path of a cryptographic hardware implementation. An attack on the control logic of the circuit may completely reveal the secret key without touching the data path. Consequently we developed design techniques for tampering resilient control structures, in our case finite state machines.

In the following we discuss each of these contributions and provide an outlook on future research opportunities.

Homomorphic Embedding

The homomorphic embedding of a finite field into a redundant ring provides the opportunity to detect errors that may be the result of tampering. It has many benefits that make it a good choice for performance critical applications with moderate

security needs. The overhead incurred by the larger operands and the additional operations is minor and can be determined by the designer. Additionally the low-weight form of the scaled modulus enables efficient modular reduction that can compete directly with other low-weight field moduli, for example those recommended by NIST for the elliptic curve digital signature algorithm [oSN00]. The error checking procedure consists of a simple modular reduction that can be executed outside of the critical path and thus does not affect the speed of the operation. The technique is in principle applicable to fields of arbitrary characteristic, but since binary and prime characteristic are the most commonly used field types, we only considered these here.

One of the limitations of homomorphic embedding is certainly the fact that not any arbitrary field may be embedded, and not with an arbitrary amount of redundancy, either. That said, a large choice of parameters exist that enable the designer to make fine-grained trade-offs between the security level and the amount of redundancy—we included a selection of suitable parameters in the appendix.

Future Research An interesting question regarding the generalization of cyclic codes would certainly be if we can make any statement about their worst case error detection capabilities, i.e. to provide a bound on their minimum distance properties. Secondly, in our research we have focused mainly on error detection, while we did not cover error correction (apart from an algorithm based fault tolerance perspective).

Nonlinear Arithmetic Codes

If we take a pessimistic point of view in terms of adversarial capabilities, linear codes can never provide for adequate protection against errors that are induced precisely and with a high degree of sophistication. In such a case it is absolutely necessary to apply a non-linear scheme such as the one that we presented in chapter 6.

Nonlinear digit-based coding schemes can be problematic when it comes to design-

ing a predictor for a linear arithmetic operation such as addition or multiplication. Particularly in the context of multi-precision arithmetic, carry propagation issues need to be considered. In our case we were able to work around this issue by employing a check-symbol recomputation strategy, which ensures the integrity of operands and their check-symbols at all times.

The powerful error detection capability that these codes possess comes at a relative high cost in terms of check-symbol predictor and error detection network, and thus may be only of interest to highly security-sensitive applications.

Future Research Our construction of nonlinear arithmetic codes can be applied to protect any digit based integer arithmetic scheme, for example, an RSA modular exponentiation in an integer ring. It does not currently apply to polynomial arithmetic that would be of interest in the implementation of an elliptic curve cryptosystem over binary extension fields. Another open question is if there exist other constructions for the check-symbol prediction and error detection networks with a lower overhead.

Low-Cost Codes

Finally we presented a set of low-cost tamper-proof arithmetic primitives for optimal extension fields. These are based on the original construction of robust codes by Karpovsky and Taubin [KT04] and represent a compromise between robustness and overhead for error detection. Optimal extension fields have been proposed in the context of fast field arithmetic on low-cost embedded processors. The techniques presented here may be applied to both hard- and software implementations. Our analysis of an elliptic curve scalar point multiplication shows that the overhead incurred is minimal.

Future Research Recently, Baktır [BS06] and Saldamlı [Sal05] introduced spectral techniques for fast multiplication in finite fields. Operands are converted into a

spectral representation by means of a number theoretic transform. Since after the transformation the coefficients of the spectral representation of the operands are all elements in GF(q), it appears to be possible to use robust codes to protect the spectral arithmetic against adversarial faults.

Tamper Resilient Control Structures

Motivated by a hypothetical attack on a modular exponentiation based public key accelerator, we explored the use of large minimum distance error detecting codes for state machine encoding, along with other design considerations. We introduced a novel design-metric and quantitively compared our approach to traditional N-modular redundancy, with the result that our approach is more effective and scales better. Following initial concerns of a reviewer about the overhead of our technique, we also analyzed the ratio of control logic to data path in a typical cryptographic circuit. We concluded that this ratio is typically very small and thus overhead in control logic does not have a huge impact on the overall cost of the system, while adding a valueable extra security margin.

Future Research Our work represents an initial step in the direction of fault resilient control structures in this application domain. So far we only considered finite state machines, but there are other control structures that present challenges of their own, such as pipeline control circuits for security processors. Another direction for future research may be to look into more suitable families of codes and potentially online error recovery techniques.

General Open Issues

Not all questions pertaining to the issue of adversarial faults in cryptographic systems could be addressed in this dissertation. In this section we would like to mention a couple of research objectives that are of general relevance in this context.

A major issue raised by our work is the question of accuracy of our error and attacker models. In general, this is a difficult question to answer, since it depends on a large number of factors. The literature on this subject is quite sparse, so that we had to rely on a various assumptions about the fault induction capabilities of a determined individual and how these faults may manifest themselves in the circuit.

Since we do not possess the infrastructure for experimentally verifying these assumptions, we had to extrapolate from only a few reports available, e.g. [SA02]. That being said, we provide one important argument that makes our assumptions more reasonable: in contrast to previous work on fault tolerant cryptographic circuits, which only assume random faults, we model a determined adversary with strong capabilities. Our rationale here is that it is generally better to err on the safe side. Nonetheless, more work on a realistic model (or better yet: several realistic models for a variety of attacker profiles) is urgently needed. A good first step in this direction was provided by Lemke-Rust and Paar in [LRP06].

Another open question that our work raises is the issue of fault simulation. So far, we determined the failure modes of our approaches analytically under either the logical or the arithmetic error model. We did not, however, verify the error models experimentally, e.g. by simulating random faults in a computer model of our circuit. Such an experiment could give valuable insights into the relative performance benefits of our linear and non-linear error detection schemes under a weaker, i.e. random fault capable, attacker model. Unfortunately, fault simulations require a certain infrastructure that was not available to us at the time, so we had to leave this issue open.

Bibliography

[ABF+02] C. Aumüller, P. Bier, W. Fischer, P. Hofreiter, and J.-P. Seifert. Fault attacks on RSA with CRT: Concrete results and practical countermeasures. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Proceedings of 4th International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*, number 2523 in Lecture Notes in Computer Science LNCS, pages 260–275, Heidelberg, 2002. Springer-Verlag.

[AK96] R. Anderson and M. Kuhn. Tamper resistance - a cautionary note. In *Proceedings of the Second Usenix Workshop on Electronic Commerce*, pages 1–11. USENIX Assoc., USENIX Press, Nov 1996.

[Avi67] A. Avizienis. Design of fault-tolerant computers. In *Proc. 1967 AFIPS Fall Joint Computer Conf.*, volume 31 of *AFIPS Conf. Proc.*, pages 733–743, 1967.

[BBK+02] G. Bertoni, L. Breveglieri, I. Koren, P. Maistri, and V. Piuri. On the propagation of faults and their detection in a hardware implementation of the advanced encryption standard. In M. Schulte, S. Bhattacharyya, N. Burgess, and R. Schreiber, editors, *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures, and Processors (ASAP)*, pages 303–314, San Jose, CA, USA, July 2002. IEEE Computer Society Press.

[BDL97] D. Boneh, R. A. DeMillo, and R. J. Lipton. On the importance of checking cryptographic protocols for faults. In W. Fumy, editor, *Advances in Cryptology - EuroCrypt'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 37–51, Heidelberg, 1997. Springer. Proceedings.

[Ber04] M. Berg. Fault tolerant design techniques for asynchronous single event upsets within synchronous finite state machine architectures. In *7th International Military and Aerospace Programmable Logic Devices (MAPLD) Conference*. NASA, Sep 2004.

[BHLM01] M. Brown, D. Hankerson, J. López, and A. Menezes. Software implementation of the NIST elliptic curves over prime fields. In D. Naccache, editor, *Topics in Cryptology — CT-RSA 2001*, volume 2020 of *Lecture Notes in Computer Science*, pages 250–265, Heidelberg, April 2001. Springer.

[BOS03] J. Blömer, M. Otto, and J.-P. Seifert. A new CRT-RSA algorithm secure against bellcore attacks. In *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*, pages 311–320, New York, NY, USA, 2003. ACM Press.

[BOS06] J. Blömer, M. Otto, and J.-P. Seifert. Sign change fault attacks on elliptic curve cryptosystems. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC '06)*, volume 4236 of *Lecture Notes in Computer Science*, pages 36–52, Heidelberg, Oct 2006. Springer.

[BP98] D. V. Bailey and C. Paar. Optimal extension fields for fast arithmetic in public-key algorithms. In H. Krawczyk, editor, *Advances in Cryptology - CRYPTO 98*, volume 1462 of *Lecture Notes in Computer Science*, pages 472–485, Heidelberg, August 1998. Springer.

[BP01] D. V. Bailey and C. Paar. Efficient arithmetic in finite field extensions with application in elliptic curve cryptography. *Journal of Cryptology*, 14(3):153–176, 2001.

[BS97] E. Biham and A. Shamir. Differential fault analysis of secret key cryptosystems. In B.S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO'97*, number 1294 in Lecture Notes in Computer Science, pages 513–525, Heidelberg, 1997. Springer-Verlag.

[BS06] S. Baktır and B. Sunar. Finite field polynomial multiplication in the frequency domain with application to elliptic curve cryptography. In *21st International Symposium on Computer and Information Sciences (ISCIS 2006)*, volume 4263 of *Lecture Notes in Computer Science LNCS*, pages 991–1001, Heidelberg, Oct 2006. Springer-Verlag.

[CT91] M. Chen and E. A. Trachtenberg. Permutation codes for the state assignment of fault tolerant sequential machines. In *Proc. 10th Digital Avionics Systems Conference*, pages 85–89. IEEE/AIAA, Oct 1991.

[CT05] H. Choukri and M. Tunstall. Round reduction using faults. 2nd International Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'05), Edinburgh, Scotland, September 2005.

[DT99] N. Das and N. A. Touba. Weight-based codes and their application to concurrent error detection of multilevel circuits. In *VTS'99*, 1999.

[FV06] G. Fumaroli and D. Vigilant. Blinded fault resistant exponentiation. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC '06)*, volume 4236 of *Lecture Notes in Computer Science*, pages 62–70, Heidelberg, Oct 2006. Springer.

[Gau02] G. Gaubatz. Versatile montgomery multiplier architectures. Master's thesis, Worcester Polytechnic Institute, Worcester, Massachusetts, May 2002.

[GHS02] P. Gaudry, F. Hess, and N. P. Smart. Constructive and destructive facets of weil descent on elliptic curves. *Journal of Cryptology*, 15(1):19–46, 2002.

[GMO01] K. Gandolfi, C. Mourtel, and F. Olivier. Electromagnetic analysis: Concrete results. In Ç. K. Koç, D. Naccache, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems–CHES 2001*, volume 2162 of *Lecture Notes in Computer Science LNCS*, pages 251–272, Heidelberg, May 2001. Springer.

[Gor87] D. M. Gordon. Perfect multiple error-correcting arithmetic codes. *Mathematics of Computation*, 49(180):621–633, Oct 1987.

[GP02] J. Guajardo and C. Paar. Itoh-tsujii inversion in standard basis and its application in cryptography. *Design, Codes, and Cryptography*, 25(2):207–216, Feb 2002.

[GS04] J. Großschädel and E. Savaş. Instruction set extensions for fast arithmetic in finite fields GF(p) and GF(2^m). In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems–CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 133–147, Heidelberg, Aug 2004. Springer.

[GS05] G. Gaubatz and B. Sunar. Robust finite field arithmetic for fault-tolerant public-key cryptography. In L. Breveglieri and I. Koren, editors, *2nd Workshop on Fault Diagnosis and Tolerance in Cryptography - FDTC 2005*, September 2005.

[GSK06] G. Gaubatz, B. Sunar, and M. G. Karpovsky. Non-linear residue codes for robust public-key arithmetic. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC '06)*, volume 4236 of *Lecture Notes in Computer Science*, pages 173–184, Heidelberg, Oct 2006. Springer.

[GSS07] G. Gaubatz, E. Savaş, and B. Sunar. Sequential circuit design for embedded cryptographic applications resilient to adversarial faults. *IEEE Transactions on Computers*, 2007.

[Har04] L. Hars. Long modular multiplication for cryptographic applications. Cryptology ePrint Archive, Report 2004/198, 2004.

[HJSI75] A. L. Hopkins Jr. and T. B. Smith III. The architectural elements of a symmetric fault-tolerant multiprocessor. *IEEE Transactions on Computers*, C-24:498–505, May 1975.

[HMV04] D. Hankerson, A. Menezes, and S. Vanstone. *Guide to Elliptic Curve Cryptography*. Springer, Heidelberg, first edition, 2004.

[HS73] H. J. Helgert and R. D. Stinaff. Minimum-distance bounds for binary linear codes. *IEEE Transactions on Information Theory*, 19(3):344–356, May 1973.

[IT88] T. Itoh and S. Tsujii. A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases. *Information and Computation*, 78:171–177, 1988.

[JLQ99] M. Joye, A.K. Lenstra, and J.-J. Quisquater. Chinese remaindering based cryptosystem in the presence of faults. *Journal of Cryptology*, 4(12):241–245, 1999.

[JWK04] N. Joshi, K. Wu, and R. Karri. Concurrent error detection schemes for involution ciphers. In M. Joye and J.-J. Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science LNCS*, pages 400–412, Heidelberg, Aug 2004. Springer.

[JY02] M. Joye and S. M. Yen. The montgomery powering ladder. In B.S. Kaliski Jr., Ç.K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems–CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 291–302. Springer-Verlag, 2002.

[KAK96] Ç.K. Koç, T. Acar, and B.S. Jr. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, June 1996.

[Ker83] A. Kerckhoff. La cryptographie militaire. *Journal de Sciences Militaires*, IX:5–83, Jan 1883.

[KJJ99] P. C. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In M. Wiener, editor, *Advances in Cryptology–CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science LNCS*, pages 388–397. Springer, Aug 1999.

[KKT04] M. G. Karpovsky, K. J. Kulikowski, and A. Taubin. Robust protection against fault-injection attacks of smart cards implementing the advanced encryption standard. In L. Simoncini, editor, *Proc. Int. Conf. Dependable Systems and Networks (DSN'04)*, pages 93–101. IEEE Computer Society, IEEE Press, 2004.

[KKT05] K. Kulikowski, M. G. Karpovsky, and A. Taubin. Robust codes for fault attack resistant cryptographic hardware. In L. Breveglieri and I. Koren,

editors, *2nd Int. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC'05)*, Sep 2005.

[KO63] A. Karatsuba and Y. Ofman. Multiplication of multidigit numbers by automata. *Soviet Physics-Doklady*, 7:595–596, 1963.

[Koc96] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO '96: Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, pages 104–113, London, UK, 1996. Springer-Verlag.

[KT04] M. Karpovsky and A. Taubin. New class of nonlinear systematic error detecting codes. *IEEE Transactions on Information Theory*, 50(8):1818–1820, August 2004.

[LN83] R. Lidl and H. Niederreiter. *Finite Fields*. Addison-Wesley, Reading, MA, 1st edition, 1983.

[LRP06] K. Lemke-Rust and C. Paar. An adversarial model for fault analysis against low-cost cryptographic devices. In L. Breveglieri, I. Koren, D. Naccache, and J.-P. Seifert, editors, *Fault Diagnosis and Tolerance in Cryptography 2006 (FDTC '06)*, volume 4236 of *Lecture Notes in Computer Science*, pages 131–143, Heidelberg, Oct 2006. Springer.

[Man67] D. Mandelbaum. Arithmetic codes with large distance. *IEEE Transactions on Information Theory*, 13(2):237–242, April 1967.

[MM00] S. Mitra and E. J. McCluskey. Which concurrent error detection scheme to choose? In *Proc. Int. Test Conference (ITC)*, pages 985–994. IEEE, IEEE Press, 2000.

[oSN00] National Institute of Standards and Technology (NIST). *Digital Signature Standard (DSS)*. Number 186-2 in Federal Information Processing Standards Publication. U.S. Department of Commerce, Jan 2000.

[ÖSS04] E. Öztürk, B. Sunar, and E. Savaş. Low-power elliptic curve cryptography using scaled modular arithmetic. In M. Joye and J.-J. Quisquater, editors, *Workshop on Cryptographic Hardware and Embedded Systems–CHES 2004*, volume 3156 of *Lecture Notes in Computer Science LNCS*, pages 92–106. Springer, Aug 2004.

[Par00] B. Parhami. *Computer Arithmetic: Algorithms and Hardware Designs*. Oxford University Press, 2000.

[PQ03] G. Piret and J.-J. Quisquater. A differential fault attack technique against SPN structures, with application to the AES and Khazad. In C. D. Walter, Ç. K. Koç, and C. Paar, editors, *Proc. Int. Workshop on Cryptographic Hardware and Embedded Systems (CHES'03)*, volume 2779 of *Lecture Notes in Computer Science*, pages 77–88, Heidelberg, 2003. Springer-Verlag.

[Pra57] E. Prange. Cyclic error correcting codes in two symbols. Technical Report AFCRC-TN-57-103, USAF Cambridge Research Laboratory, Bedford, Mass., 1957.

[Pra86] D.K. Pradhan, editor. *Fault Tolerant Computing – Theory and Techniques*, volume 1. Prentice-Hall, New Jersey, 1^{st} edition, 1986.

[Pro89] I.K. Proudler. Idempotent AN codes. In *IEE Colloquium on Signal Processing Applications of Finite Field Mathematics*, pages 8/1–8/5, London, June 1989. IEE, IEE.

[QS01] J.-J. Quisquater and D. Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In I. Attali and T. Jensen, editors, *Smart Card Programming and Security*, volume 2140 of *Lecture Notes in Computer Science LNCS*, pages 200–210, Berlin / Heidelberg, Sep 2001. Springer.

[Ren78] D.A. Rennels. Architectures for fault-tolerant spacecraft computers. *Proceedings of the IEEE*, 66(10):1255–1268, Oct 1978.

[RG71] T. R. N. Rao and O. N. Garcia. Cyclic and multiresidue codes for arithmetic operations. *IEEE Trans. Inf. Theory*, 17(1):85–91, Jan 1971.

[RH02] A. Reyhani-Masoleh and M.A. Hasan. Error detection in polynomial basis multipliers over binary extension fields. In B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 515–528, Heidelberg, 2002. Springer. 4th International Workshop, Redwood Shores, CA, USA.

[RMH04] A. Reyhani-Masoleh and M. A. Hasan. Towards fault-tolerant cryptographic computations over finite fields. *ACM Transactions on Embedded Computing Systems*, 3(3):593–613, August 2004.

[SA02] S. P. Skorobogatov and R. J. Anderson. Optical fault induction attacks. In B. S. Kaliski, Jr., Ç. K. Koç, and C. Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12, Berlin, Heidelberg, New York, August 2002. Springer-Verlag.

[Sal05] G. Saldamlı. *Spectral Modular Arithmetic*. PhD thesis, Department of Electrical and Computer Engineering, Oregon State University, Corvallis, OR, Jun 2005.

[Sha99] A. Shamir. Method and apparatus for protecting public key schemes from timing and fault attacks. US Patent No. 5,991,415, 1999.

[van92] J. H. van Lint. *Introduction to Coding Theory*, volume 86 of *Graduate Texts in Mathematics*. Springer, second edition, 1992.

[Wag04] D. Wagner. Cryptanalysis of a provably secure CRT-RSA algorithm. In *CCS '04: Proceedings of the 11th ACM Conference on Computer and Communications Security*, pages 92–97, New York, NY, USA, 2004. ACM Press.

[Wal92] C. D. Walter. Faster modular multiplication by operand scaling. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91: Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 313–323, Heidelberg, 1992. IACR, Springer.

[WHBG02] H. Wu, M. A. Hasan, I. F. Blake, and S. Gao. Finite field multiplier using redundant representation. *IEEE Transactions on Computers*, 51(11):1306–1316, November 2002.

[YJ00] S. M. Yen and M. Joye. Checking before output may not be enough against fault-based cryptanalysis. *IEEE Trans. Comp.*, 49(9):967–970, Sep 2000.

Appendix

For practical purposes we present a selection of useful parameters for scaled embedding. In all three tables the parameter n refers to the size in bits, respectively degree, of the scaled modulus $m = p \cdot s$, while the parameter k is indicative of the size (degree) of the field modulus of $\mathcal{F}$ to be embedded in the ring. Finally, the amount of redundancy due to the scaling factor s is quantified by the difference $n-k$. Table A.1 provides parameters for prime field embedding, with $k = \lceil \log_2 p \rceil$ and $n-k = \lceil \log_2 s \rceil$, while Tables A.2 and A.3 give parameters suitable for embedding binary extension fields, where $p(x)$ is of prime degree $\deg(p(x)) = k$ and the scaling factor $s(x)$ has degree $\deg(s(x)) = n - k$.

n	k	u	n-k	n	k	u	n-k	n	k	u	n-k	n	k	u	n-k
300	160	1	140	203	168	-3	35	206	177	3	29	229	203	1	26
205	161	1	44	261	168	-1	93	200	180	-3	20	256	206	1	50
211	162	3	49	208	171	3	37	221	181	-1	40	233	208	1	25
236	162	1	74	227	172	-1	55	223	184	1	39	241	217	-1	24
239	162	-1	77	205	173	3	32	259	184	-1	75	248	227	1	21
232	163	1	69	210	173	3	37	233	186	-1	47	251	232	1	19
209	166	-3	43	202	174	-3	28	210	193	-3	17				

Table A.1: Factorizations of $m = p \cdot s = 2^n + u$

n	k	n-k
173	163	10
190	163	27
202	163	39
264	163	101
209	179	30
235	191	44
308	191	117
334	191	143
239	193	46
306	211	95
390	211	179
371	239	132
391	239	152
452	251	201
412	263	149

n	k	n-k
482	269	213
419	277	142
495	277	218
587	311	276
605	311	294
470	313	157
355	337	18
446	337	109
544	337	207
604	337	267
669	337	332
578	349	229
590	349	241
674	349	325
468	353	115

n	k	n-k
475	359	116
662	359	303
456	367	89
728	373	355
407	379	28
401	389	12
626	389	237
724	389	335
492	397	95
559	397	162
623	397	226
715	401	314
458	409	49
827	419	408
746	443	303

n	k	n-k
862	443	419
786	449	337
831	457	374
920	461	459
630	463	167
760	463	297
618	467	151
577	479	98
748	491	257
763	503	260
764	503	261
849	503	346
957	503	454
553	521	32
779	521	258

Table A.2: Factorizations of $m(x) = p(x) \cdot s(x) = x^n + x + 1$

n	k	n-k
235	167	68
283	179	104
199	181	18
207	199	8
319	199	120
405	223	182
357	229	128
281	233	48

n	k	n-k
291	251	40
383	269	114
321	281	40
417	281	136
543	281	262
295	293	2
341	293	48
509	317	192

n	k	n-k
381	373	8
473	389	84
463	401	62
617	409	208
477	431	46
521	431	90
471	433	38
615	457	158

n	k	n-k
765	457	308
823	457	366
785	463	322
965	487	478
675	503	172

Table A.3: Factorizations of $m(x) = p(x) \cdot s(x) = x^n + x^2 + 1$