Fault Attacks and Countermeasures for Elliptic Curve Cryptosystems

by

Silvana Medoš
Bachelor of Applied Science (Honours)
RMIT University

Dissertation submitted in fulfilment of the requirements for the degree of Doctor of Philosophy in Mathematics

School of Mathematical and Geospatial Sciences
RMIT University
Melbourne
Australia
March 29, 2009
DECLARATION

The candidate hereby declares that the work in this thesis, presented for the award of the Doctor of Philosophy in Mathematics and submitted in the School of Mathematical and Geospatial Sciences, RMIT University:

- has been done by the candidate alone and has not been submitted previously, in whole or in part, in respect of any other academic award and has not been published in any form by any other person except where due reference is given, and

- has been carried out under the supervision of Associate Professor Serdar Boztaş.

........................................
Silvana Medoš

Certification

This is to certify that the above statements made by the candidate are correct to the best of our knowledge.

........................................
Associate Professor Serdar Boztaş
Supervisor
Acknowledgements

I welcome this opportunity to thank my supervisor, Associate Professor Serdar Boztas, for his assistance and support during my research, as well as always believing in me.

Very special thanks to my husband, Milan, for his comments and feedback on my work, as well as his criticisms that made me stronger, and as ever, for his love and support. Also, special thanks to my family for their support, generosity and kindness.

My research was supported by ARC Linkage grant LP0455324, entitled “Developing a Scalable E-Security Incorporating Cryptography Biometric Authentication”, for which I’m very thankful.

Moreover, I wish to thank to all my colleges from School of Mathematics and Geospatial Sciences at RMIT University for having great time while doing my research.

Finally, I’m grateful and indebted to every single person that made my research time enjoyable.
# Contents

1 Introduction

1.1 Context of the Research and Thesis Outline .......................... 4

1.2 Cryptography .............................................................. 6

1.2.1 Symmetric-key Cryptography ....................................... 8

1.2.2 Public-key Cryptography ............................................ 9

1.2.3 Symmetric key cryptosystem vs. private key cryptosystem ...... 10

1.3 Number Theoretic Problems ........................................... 11

1.3.1 Computational Complexity ......................................... 11

1.3.2 Integer Factorization Problem .................................... 11

1.3.3 The RSA problem (RSAP) ........................................... 12

1.3.4 Discrete Logarithm Problem ....................................... 12

1.3.5 The Diffie-Hellman problem (DHP) ................................. 13

1.3.6 Elliptic Curve Discrete Logarithm Problem (ECDLP) .......... 14

1.3.7 Elliptic Curve Diffie-Hellman Problem (ECDHP) .............. 14

1.3.8 Elliptic Curve Decision Diffie-Hellman Problem (ECDDHP) .. 14

1.4 Security of Public Key Cryptosystems .................................. 15

1.4.1 Smartcards .............................................................. 16

1.5 Conclusion ................................................................. 18

2 Elliptic Curve Cryptography (ECC) ........................................ 19

2.1 Abstract Algebra .......................................................... 20

2.1.1 Groups ................................................................. 20

2.1.2 Rings and Fields ....................................................... 21
2.2 Introduction to Elliptic Curves ........................................ 24
  2.2.1 Simplified Weierstrass equations .............................. 25
  2.2.2 Group law for Elliptic Curves ................................. 26
  2.2.3 Point Representation in the Projective Coordinates ............. 29
2.3 Attacks on Elliptic Curve Cryptography. .......................... 31
  2.3.1 Pohlig-Hellman attack. .................................. 31
  2.3.2 Pollard’s Rho Attack. .................................. 32
  2.3.3 Isomorphism attacks. .................................. 33
2.4 Elliptic Curve Domain Parameters ................................. 34
2.5 Elliptic Curve Cryptographic Schemes ............................ 35
  2.5.1 Signature Scheme - ECDSA .................................. 35
  2.5.2 Public-key encryption - ECIES ......................... 37
  2.5.3 Key Establishment - ECMQV ............................... 38
2.6 Conclusion ...................................................... 40

3 Fault Models ....................................................... 41
  3.1 Fault Attacks .................................................... 42
    3.1.1 Physical Methods to Induce Faults .......................... 43
  3.2 Fault Models .................................................... 45
    3.2.1 Random Fault Model (RFM) .................................. 47
    3.2.2 Arbitrary Fault Model (AFM) .................................. 48
    3.2.3 Single bit Fault Model (SFM) .................................. 49
  3.3 Conclusion ...................................................... 50

4 Fault Attacks on Elliptic Curves Cryptosystems .................. 51
  4.1 Differential Fault Attacks on ECC ............................... 52
  4.2 Invalid-Curve Attacks ............................................ 56
    4.2.1 Invalid-Curve Attack on One-Pass ECDH .................... 56
    4.2.2 Invalid-Curve Attack on ECIES ............................. 56
  4.3 Point Addition - Undetectable faulty points ...................... 57
  4.4 Attacks targeting \( \lambda = (y_1 + y_2)/(x_1 + x_2) \) .................. 58
4.5 Attacks targeting $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$ 
4.6 Fault attacks on $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$ 
4.7 Conclusion

5 Fault-Tolerant Computation in Finite Fields by Embedding in Rings
5.1 CRT based Fault-Tolerant Computation
5.1.1 CRT based Finite Field Encoding
5.1.2 Fault-Tolerant Computation
5.1.3 Error correction and detection
5.1.4 Decoding based on the Euclid Algorithm
5.2 Lagrange Interpolation (LI) based Fault-Tolerant Computation
5.2.1 Lagrange Interpolation Finite Field Encoding
5.2.2 Fault-Tolerant Computation
5.2.3 Complexity of Interpolation and Evaluation
5.2.4 Error Detection and Correction
5.2.5 Decoding of the Output
5.3 Conclusion

6 Montgomery FTC in Finite Fields by Embedding in Rings
6.1 Montgomery Residue Representation FTC in Finite Fields
6.1.1 Multiplication in $GF(2^k)$
6.1.2 Montgomery Residue Representation FTC
6.1.3 Complexity
6.1.4 Error Detection and Correction
6.2 Montgomery Lagrange Representation FTC in Finite Fields
6.2.1 Montgomery LR Computation in the $GF(2^k)$
6.2.2 Montgomery Lagrange Representation FTC
6.2.3 Error Correction and Detection
6.2.4 LR Inversion in GF($2^k$)
6.3 Conclusion
List of Figures

1.1 Example of a smartcard. ................................................. 17

2.1 Point addition on an elliptic curve: $R = P + Q$. ...................... 27
2.2 Point doubling on an elliptic curve: $R = P + P$. ...................... 28

5.1 Fault tolerant computation over the finite field $GF(2^k)$. ............... 93
5.2 Fault tolerant computation of the finite field $GF(2^k)$ in the ring $R'$. 104

6.1 Fault tolerant computation over the finite field $GF(2^k)$. ............... 128
6.2 Fault tolerant multiplication over the finite field $GF(2^k)$. ............. 142
# List of Tables

2.1 Recommended parameters for elliptic curves over $GF(2^k)$. 36

5.1 Irreducible pentanomials $f(x) = x^k + x^m + x^n + x^h + 1$ over $GF(2)$, where $h < n < m < k/2$, $k \in [160 \ldots 450]$, suitable to use in Elliptic Curve Cryptography. 90

5.2 Irreducible trinomials $f(x) = x^k + x^m + 1$ over $GF(2)$, where $m < k/2$, $k \in [160..450]$, suitable to use in Elliptic Curve Cryptography. 91

6.1 Iterations of LR Inversion Algorithm 151
**List of Algorithms**

<table>
<thead>
<tr>
<th></th>
<th>Algorithm Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ECDSA signature generation</td>
<td>35</td>
</tr>
<tr>
<td>2</td>
<td>ECDSA signature verification</td>
<td>37</td>
</tr>
<tr>
<td>3</td>
<td>ECIES encryption</td>
<td>38</td>
</tr>
<tr>
<td>4</td>
<td>ECIES decryption</td>
<td>38</td>
</tr>
<tr>
<td>5</td>
<td>Chinese Remainder Algorithm</td>
<td>92</td>
</tr>
<tr>
<td>6</td>
<td>Euclid’s Decoding Algorithm</td>
<td>98</td>
</tr>
<tr>
<td>7</td>
<td>Welch-Berlekamp Decoding of the Output Vector</td>
<td>112</td>
</tr>
<tr>
<td>8</td>
<td>Montgomery Multiplication in $GF(2^k)$</td>
<td>118</td>
</tr>
<tr>
<td>9</td>
<td>Montgomery Multiplication over $GF(p^k)$</td>
<td>119</td>
</tr>
<tr>
<td>10</td>
<td>Montgomery Multiplication in $GF(p^k)$</td>
<td>119</td>
</tr>
<tr>
<td>11</td>
<td>Residue Representation Modular Multiplication</td>
<td>120</td>
</tr>
<tr>
<td>12</td>
<td>Fault Tolerant RR Modular Multiplication</td>
<td>123</td>
</tr>
<tr>
<td>13</td>
<td>Euclid’s Decoding Algorithm</td>
<td>131</td>
</tr>
<tr>
<td>14</td>
<td>LR Modular Multiplication</td>
<td>134</td>
</tr>
<tr>
<td>15</td>
<td>Fault tolerant LR Modular Multiplication</td>
<td>138</td>
</tr>
<tr>
<td>16</td>
<td>Welch-Berlekamp Decoding of the Output Vector</td>
<td>145</td>
</tr>
<tr>
<td>17</td>
<td>Leading term $LT(U, m)$</td>
<td>148</td>
</tr>
<tr>
<td>18</td>
<td>Inversion over $GF(2^k)$ in LR</td>
<td>150</td>
</tr>
</tbody>
</table>
Summary

Elliptic Curve Cryptography (ECC) security relies on the hardness of solving the elliptic curve discrete logarithm problem (ECDLP) but the security of cryptosystems does not only depend on the mathematical properties; an adversary can attack the implementation rather than algorithmic specification. In many practical applications of ECC the secret key is stored inside a tamper-resistant device, i.e., a smartcard, where differential fault attacks (DFA) can be used to enforce bit errors into the decryption (or signing) process which is performed inside the smartcard, so that information on the secret key can leak out. In order to prevent fault attacks hardware countermeasures can be applied, or new algorithms can be developed.

In this doctoral thesis the focus of our research is development of algorithmic countermeasures against active side-channel attacks, in particular fault attacks. Our aim is to protect elliptic curve computation in a tamper-resistant device, i.e., a smartcard, by protecting the computation in the finite binary extension field $GF(2^k)$ against these attacks.

The following are the main problems addressed of the thesis:

1. Evaluation of fault attacks through the following fault models: Random Fault Model, Arbitrary Fault Model, and Single Bit Fault Model. These fault models are justified by the physical attacks.

2. Proposal of new schemes for fault tolerant computation in the elliptic curve cryptosystems, i.e., Chinese Remainder Theorem based fault tolerant computation (FTC), as well as Lagrange interpolation based FTC, both based on embedding the field in larger rings.

3. Construction of new algorithmic countermeasures based on the proposed schemes, i.e., fault tolerant residue representation modular multiplication algorithm and fault tolerant Lagrange representation modular multiplication algorithm, which are immune against the error propagation of the proposed fault models;
4. Evaluation and characterization of possible error correcting algorithms, i.e., *Euclid’s decoding algorithm* and *Welch-Berlekamp decoding algorithm*.

5. Analysis of possible masked errors in elliptic curve point addition.

The main results of this thesis are:

1. New schemes for FTC are developed. Based on these schemes new algorithmic countermeasure against fault attacks are developed.

2. New algorithmic countermeasures provide protection of the elliptic curve computation in a tamper proof device by protection of the finite field computation against active side channel attacks, i.e., fault attacks. Our method where field elements are represented by the *redundant residue representation/redundant Lagrange representation* enables us to overcome the problem if one, or both coordinates $x, y \in GF(2^k)$ of the point $P \in E / \mathbb{F}_{2^k}$ are corrupted.

3. Computation of the field elements is decomposed into parallel, mutually independent, modular/identical channels, so that in case of fault at one channel, errors will not distribute to others.

4. Different fault models can be used on different channels.

5. We give proof that our algorithmic countermeasures work against fault attacks, and that the attacker can not break a system and recover secret key. However, arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed. It is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost.

6. If faults effects are detected, they can be corrected by applying error correction algorithms, i.e., *Euclid’s decoding algorithm/Welch-Berlekamp decoding algorithm* at the output vector of the corresponding computation.

7. Our proposed algorithms can have masked errors and will not be immune against attacks which can create such errors, but it is a difficult problem to counter masked errors since any anti-fault attack scheme will have some masked errors.
The work performed in this thesis has resulted in the following publications:

1. Our scheme for fault tolerant computation in public key cryptosystems based on the Lagrange interpolation and evaluation of fault attacks through fault models has been published:


2. Our new algorithmic countermeasure, i.e., fault tolerant Lagrange representation modular multiplication algorithm has been published:


3. Our fault tolerant residue representation modular multiplication algorithm has been published:

Chapter 1

Introduction

1.1 Context of the Research and Thesis Outline

Elliptic Curve Cryptography (ECC) recently has started to receive commercial acceptance and has been included in numerous standards. Its security relies on the hardness of solving the elliptic curve discrete logarithm problem (ECDLP), but the security of cryptosystems does not only depend on the mathematical properties; an adversary can attack the implementation rather than algorithmic specification. Cryptosystems are used in the real world where cryptographic protocols are implemented in software or hardware, obeying laws of physics. The circuits used leak information, e.g., power and timing information, over side-channels. Thus, one has a gray box, where an adversary has access to several side-channels.

In many practical applications of ECC the secret key is stored inside a tamper-resistant device, i.e., a smartcard. If an adversary can inflict some sort of physical stress on the smartcard, he can induce faults into the circuitry or memory. These faults become manifest in the computation as errors. If an error occurs, a faulty final result is computed. If the computation depends on some secret key, a comparison between correct data and faulty data may allow one to conclude facts about the secret key.

Operations on elliptic curves relies on computation in very large finite fields (with more that $2^{160}$ elements), where a single fault in computation can yield an erroneous output, which can then be used by an adversary to break cryptosystem. In order to prevent fault
attacks, either hardware countermeasures can be applied, or new algorithms can be develop. Usually countermeasures against one side-channel attack does not protect against other side-channel attacks. Until now, the scientific community has not been able to developed a theoretical framework to allow general security proofs for algorithms secure against fault attacks. To prevent passive side channel attacks it is important that the available information to the adversary running a side channel attack is independent of the secret key. There have been attempts to develop algorithmic countermeasures, but none are really satisfying. Therefore, there is a need to develop algorithmic countermeasures against fault attacks which are used in real world. Countermeasures against such attacks form the main focus of this thesis.

The focus of our research are algorithm countermeasures against active side-channel attacks, in particular fault attacks. Our aim is to protect elliptic curve computation in a tamper-resistant device, i.e., a smartcard, by protecting computation in the finite binary extension field $GF(2^k)$.

**Organization of the Thesis.** Chapter 1 is the introductory chapter, where we provide an overview of the cryptography including number theoretic problems and security of cryptosystems. In Chapter 2, we discuss the algebraic, cryptographic and mathematical background of elliptic curves that is required to understand the discussion of fault attacks and countermeasures. In Chapter 3 we provide the mathematical aspect of fault attacks through characterization of the fault models, i.e., Random Fault Model, Arbitrary Fault Model and Single bit Fault Model, that model physical behavior of attacked device. In Chapter 4, by investigating each variable used in the affine addition formula of a non-supersingular elliptic curve over $GF(2^k)$ we derive conditions that the inflicted error needs to have in order to yield an undetectable faulty point on the curve. In Chapter 5, we propose two countermeasures schemes; Chinese Remainder Theorem based fault tolerant computation, and Lagrange Interpolation based fault tolerant computation. In Chapter 6, we develop new algorithmic countermeasures that protect elliptic curve computation by protecting computation of the finite binary extension field, against fault attacks. In
Chapter 7, we conclude with the results of our research.

1.2 Cryptography

Cryptography has a long and fascinating history. Initial and limited use was by the Egyptians some 4000 years ago, and it played a crucial role in the outcome of both world wars. Cryptography was used mainly by those associated with the military, the diplomatic service and government in general. It was used as a tool to protect national secrets and strategies. The development of computers and communications systems in the 1960s brought with it a demand from the private sector for means to protect information in digital form and to provide security services. In 1976 Diffie and Hellman published *New Directions in Cryptography* [29]. This paper introduced the revolutionary concept of public-key cryptography and also provided a new method for key exchange, the security of which is based on the intractability of the discrete logarithm problem. In 1978 Rivest, Shamir, and Adleman [82] discovered the first practical public-key encryption and signature scheme, known as RSA. The RSA scheme is based on the intractability of factoring large integers. This application of a hard mathematical problem to cryptography revitalized efforts to find more efficient methods to factor integers. The 1980s saw major advances in this area but none which rendered the RSA system insecure. Another class of powerful and practical public-key schemes was found by ElGamal in 1985, which are based on the discrete logarithm problem. In 1985 Neal Koblitz and Victor Miller proposed the use of elliptic curves for public key cryptosystems. Today, cryptography is in the core of every day life, securing almost all its aspects, i.e., e-commerce, online banking, communication via internet.

**Definition 1.2.1 ([68]).** Cryptography is about design and analysis of mathematical techniques that enable secure communications in the presence of malicious adversaries. Cryptography is the study of mathematical techniques related to aspects of information security such as confidentiality, data integrity, entity authentication, and data origin authentication. Cryptography is not the only means of providing information security, but rather one set of techniques.
The objectives of secure communications are (A and B are authorized parties, E is unauthorized):

1. **Confidentiality** - keeping data secret from all but those authorized to see it, i.e., a message sent by A to B should not be readable by E;

2. **Data Integrity** - ensuring that data has not been altered by unauthorized means, i.e., B should be able to detect when the data sent by A has been modified by E;

3. **Data origin authentication** - corroborating the source of data, i.e., B should be able to verify that the data sent by A indeed originated from A;

4. **Entity authentication** - corroborating the identity of an entity, i.e., B should be convinced of the identity of the other communicating entity;

5. **Non-repudiation** - preventing an entity from denying previous commitments, or actions, i.e., when B receives a message purportedly from A, not only is B convinced that the message originated from A, but B can convince a neutral third party of this, so that A can not deny having sent the message to B.

A fundamental goal of cryptography is to adequately address these four areas in both theory and practice. Cryptography is about the prevention and detection of cheating and other malicious activities.

*One-way* and *trapdoor one-way functions* are the basis for public-key cryptography.

**Definition 1.2.2** ([68]). A function $f : X \rightarrow Y$ is called a one-way function if $f(x)$ is easy to compute for all $x \in X$, but for essentially all elements $y \in \text{Im}(f)$ it is computationally infeasible to find any $x \in X$ such that $f(x) = y$.

**Definition 1.2.3** ([68]). A trapdoor one-way function is a one-way function $f : X \rightarrow Y$ with the additional property that given some extra information (called the trapdoor information) it becomes feasible to find for any given $y \in \text{Im}(f)$, an $x \in X$ such that $f(x) = y$. 
Computing a one-way function corresponds to encrypting a message, while inverting the function corresponds to decrypting a ciphertext. The secret key serves as “helper” information which helps invert a one-way function easily. The existence of one-way functions and trapdoor one-way functions is still unclear, but there are a number of good candidates for one-way and trapdoor one-way functions.

Cryptographic techniques are divided into two types: symmetric-key and public-key.

1.2.1 Symmetric-key Cryptography

Definition 1.2.4 ([68]). Consider an encryption scheme consisting of the sets of encryption and decryption transformations \( \{E_e : e_k \in K\} \) and \( \{D_d : d_k \in K\} \), respectively, where \( K \) is the key space. The encryption scheme is said to be symmetric-key if for each associated encryption/decryption key pair \((e_k; d_k)\), it is computationally easy to determine \( d_k \) knowing only \( e_k \), and to determine \( e_k \) from \( d_k \). Since \( e_k = d_k \) in most practical symmetric-key encryption schemes, the term symmetric key becomes appropriate.

In symmetric encryption, the encryption key \( e_k \) and decryption key \( d_k \) are the same, or one can be easily deduced from the other. Popular symmetric encryption methods include the Data Encryption Standard (DES) and the Advanced Encryption Standard (AES). One of the major issues with symmetric-key systems is to find an efficient method to agree upon and exchange keys securely. This problem is referred to as the key distribution problem. In symmetric-key encryption, key \( e_k \) must be kept secret, as \( d_k \) can be deduced from \( e_k \). There are two classes of symmetric-key encryption schemes: block ciphers and stream ciphers.

Advantages of symmetric-key cryptography include:

1. Symmetric-key ciphers can be designed to have high rates of data throughput.

2. Keys for symmetric-key ciphers are relatively short.

3. Symmetric-key ciphers can be employed as primitives to construct various cryptographic mechanisms, i.e., pseudorandom number generators, hash functions, computationally efficient digital signature schemes.
4. Symmetric-key ciphers can be composed to produce stronger ciphers.

5. Symmetric-key encryption is perceived to have an extensive history.

Disadvantages of symmetric-key cryptography include:

1. In a two-party communication, the key must remain secret at both ends.

2. In a large network, there are many key pairs to be managed.

3. In a two-party communication between entities $A$ and $B$, key have to be changed frequently, and perhaps for each communication session.

4. Digital signature mechanisms arising from symmetric-key encryption typically require either large keys for the public verification function or the use of a trusted third party (TTP).

1.2.2 Public-key Cryptography

**Definition 1.2.5 ([68]).** Consider an encryption scheme consisting of the sets of encryption and decryption transformations \( \{E_e : e \in K\} \) and \( \{D_d : d \in K\} \), respectively. The encryption method is said to be a public-key encryption scheme, or asymmetric encryption if for each associated encryption/decryption pair \((e_k; d_k)\), one key \(e_k\) (the public key) is made publicly available, while the other \(d_k\) (the private key) is kept secret. For the scheme to be secure, it must be computationally infeasible to compute \(d_k\) from \(e_k\).

The encryption transformation \(E_e\) is being viewed here as a trapdoor one-way function, where \(d_k\) is the trapdoor information necessary to compute the inverse function and hence allow decryption. The main objective of public-key encryption is to provide privacy or confidentiality. Advantages of public-key cryptography include:

1. Only the private key must be kept secret (authenticity of the public key must, however, be guaranteed).

2. The administration of keys on a network requires the presence of only a functionally trusted TTP as opposed to an unconditionally trusted TTP. Depending on the mode
of usage, the TTP might only be required in an off-line manner, as opposed to in real time.

3. Depending on the mode of usage, a private key/public key pair may remain unchanged for considerable periods of time, e.g., many sessions (even several years).

4. Many public-key schemes yield relatively efficient digital signature mechanisms.

5. The key used to describe the public verification function is typically much smaller than for the symmetric-key counterpart. In a large network, the number of keys necessary may be considerably smaller than in the symmetric-key scenario.

Disadvantages of public-key encryption include:

1. Data throughput rates for the most popular public-key encryption methods are several orders of magnitude slower than the best known symmetric-key schemes.

2. Key sizes are typically much larger than those required for symmetric-key encryption, and the size of public-key signatures is larger than that of tags providing data origin authentication from symmetric-key techniques.

3. No public-key scheme has been proven to be secure (the same can be said for block ciphers). The most effective public-key encryption schemes found to date have their security based on the presumed difficulty of a small set of number-theoretic problems.

4. Public-key cryptography does not have as extensive a history as symmetric-key encryption.

1.2.3 Symmetric key cryptosystem vs. private key cryptosystem

Public-key encryption techniques may be used to establish a key for a symmetric-key. In practice public-key cryptography facilitates efficient signatures, i.e., non-repudiation, and key management, while symmetric-key cryptography is efficient for encryption and some
data integrity applications. Private keys in public-key systems are larger (e.g., 1024 bits for RSA) than secret keys in symmetric-key systems (e.g., 64 or 128 bits). For equivalent security, symmetric keys have bit lengths smaller than that of private keys in public-key systems, e.g., by a factor of 10 or more. For a detailed discussion on the issue of key length, see Lenstra, et. al [58].

1.3 Number Theoretic Problems

The security of many public-key cryptosystems relies on the intractability of the relevant computational problems. The true computational complexities of these problems are not known, i.e., it is widely believed that they are intractable, although no proof of this is known. Before listing computational problems, we will briefly overview computational complexity.

1.3.1 Computational Complexity

Definition 1.3.1. An algorithm to perform a computation is said to be a polynomial time algorithm if there exists an integer $d$ such that the number of bit operations required to perform the algorithm on integers of total length at most $k$ is $O(k^d)$.

The class of exponential time algorithms are very far from polynomial time. These have time estimate of $O(e^{ck})$, where $c$ is a constant, $k$ is the total binary length of the integers to which the algorithm is being applied. Subexponential time algorithms have time estimate of the form $O(e^{f(k)})$, where $k$ is the input length and $f(k) = o(k)$, i.e., $\lim_{k \to \infty} \frac{f(k)}{k} = 0$.

1.3.2 Integer Factorization Problem

The security of many cryptographic techniques depends on the intractability of the integer factorization problem, i.e., the RSA public-key encryption scheme, the RSA signature scheme, and the Rabin public-key encryption scheme.
Definition 1.3.2 ([68]). The integer factorization problem is the following: given a positive integer \( n \), find its prime factorization, that is, write \( n = p_1^{e_1} p_2^{e_2} \cdots p_k^{e_k} \) where the \( p_i \) are pairwise distinct primes and each \( e_i \geq 1 \).

The special-purpose factoring algorithms are tailored to perform better when the integer \( n \) being factored is of a special form, examples include: trial division, Pollard’s rho algorithm, Pollard’s \( p - 1 \) algorithm, the elliptic curve algorithm, and the special number field sieve. The running time of these algorithms depends on the certain properties of the factors of \( n \).

The general purpose factoring algorithms include the quadratic sieve and the general number field sieve. The running time of these algorithms depends on the size of the \( n \).

1.3.3 The RSA problem (RSAP)

The intractability of the RSA problem forms the basis for the security of the RSA public-key encryption scheme and the RSA signature scheme.

Definition 1.3.3 ([68]). The RSA problem (RSAP) is the following: given a positive integer \( n \) that is a product of two distinct large odd primes \( p \) and \( q \), a positive integer \( e \) such that \( \gcd(e, (p - 1)(q - 1)) = 1 \), and an integer \( c \), find an integer \( m \) such that \( m^e \equiv c \pmod{n} \).

If the factors of \( n \) are known then the RSA problem can be easily solved. There is no proof that the RSA and the integer factorization problems are computationally equivalent, but this is strongly believed to be the case.

1.3.4 Discrete Logarithm Problem

The security of many cryptographic techniques depends on the intractability of the discrete logarithm problem, i.e., Diffie-Hellman key agreement and its derivatives, ElGamal encryption, and the ElGamal signature scheme and its variants.

Let \( G \) be a finite cyclic group of order \( n \) with generator \( \alpha \). It is convenient to think of \( G \) as the multiplicative group \( \mathbb{Z}_p^* \) of order \( p - 1 \), where the group operation is simply multiplication modulo \( p \).
**Definition 1.3.4 ([68])**. Let $G$ be a finite cyclic group of order $n$. Let $\alpha$ be a generator of $G$, and let $\beta \in G$. The discrete logarithm of $\beta$ to the base $\alpha$, denoted $\log_\alpha \beta$, is the unique integer $x$, $0 \leq x \leq n - 1$, such that $\beta = \alpha^x$.

**Definition 1.3.5 ([68])**. The discrete logarithm problem (DLP) is the following: given a prime $p$, a generator $\alpha$ of $\mathbb{Z}_p^*$, and an element $\beta \in \mathbb{Z}_p^*$, find the integer $x$, $0 \leq x \leq p - 2$, such that $\beta \equiv \alpha^x \pmod{p}$.

**Definition 1.3.6 ([68])**. The generalized discrete logarithm problem (GDLP) is the following: given a finite cyclic group $G$ of order $n$, a generator $\alpha$ of $G$, and an element $\beta \in G$, find the integer $x$, $0 \leq x \leq n - 1$, such that $\beta = \alpha^x$.

The known algorithms for the DLP can be categorized as follows:

1. Algorithms that work in arbitrary groups, e.g., exhaustive search, the baby step giant step algorithm, Pollard’s rho algorithm;

2. Algorithms that work in arbitrary groups, but are especially efficient if the order of the group has only small prime factors, e.g., Pohlig-Hellman algorithm; and

3. The index-calculus algorithms which are efficient only in certain groups.

### 1.3.5 The Diffie-Hellman problem (DHP)

The security of many cryptographic schemes depends on the intractability of the Diffie-Hellman problem, i.e., Diffie-Hellman key agreement and its derivatives, and ElGamal public-key encryption. The DHP reduces to the DLP in polynomial time.

**Definition 1.3.7 ([68])**. The Diffie-Hellman problem (DHP) is the following: given a prime $p$, a generator $\alpha$ of $\mathbb{Z}_p^*$, and elements $\alpha^a \pmod{p}$ and $\alpha^b \pmod{p}$, find $\alpha^{ab} \pmod{p}$.

**Definition 1.3.8 ([68])**. The generalized Diffie-Hellman problem (GDHP) is the following: given a finite cyclic group $G$, a generator $\alpha$ of $G$, and group elements $\alpha^a$ and $\alpha^b$, find $\alpha^{ab}$.
1.3.6 Elliptic Curve Discrete Logarithm Problem (ECDLP)

The hardness of the ECDLP is essential for the security of all elliptic curve cryptography schemes.

**Definition 1.3.9** ([41]). The ECDLP is the following: given an elliptic curve $E(\mathbb{F}_q)$ defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order $n$, and a point $Q \in <P>$, find the integer $l \in [0, n-1]$ such that $Q = lP$. The integer $l$ is called the discrete logarithm of $Q$ to the base $P$, denoted $l = \log_P Q$.

There is no mathematical proof that the ECDLP is intractable. Also there is no theoretical evidence that the ECDLP is intractable. The best known attack on ECDLP uses a combination of the Pohlig-Hellman and Pollard’s rho algorithms, whose time $O(\sqrt{p})$ is fully exponential, where $p$ is the largest prime divisor of $n$, while the most naive algorithm is exhaustive search. Subexponential-time attacks for some versions of the ECDLP are: isomorphism attacks, Weil and Tate pairing attacks, attacks on the prime-field anomalous curves, and Weil descent.

1.3.7 Elliptic Curve Diffie-Hellman Problem (ECDHP)

**Definition 1.3.10** ([41]). The computational ECDHP is the following: given an elliptic curve $E(\mathbb{F}_q)$ defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order $n$, and points $A = aP, B = bP \in <P>$, find the point $C = AB$.

The ECDLP is no harder than the ECDHP, since if the ECDLP can be solved in $<P>$, by first finding $a$ from $(P, A)$ and then computing $C = aB$. It is not known if the ECDLP is equally as hard as the ECDHP.

1.3.8 Elliptic Curve Decision Diffie-Hellman Problem (ECD-DHP)

**Definition 1.3.11** ([41]). The ECDDHP is the following: given an elliptic curve $E(\mathbb{F}_q)$ defined over a finite field $\mathbb{F}_q$, a point $P \in E(\mathbb{F}_q)$ of order $n$, and points $A = aP, B = bP \in <P>$, find the point $C = AB$. It is not known if the ECDLP is equally as hard as the ECDHP.
\[ bP, C = cP \in < P >, \text{ determine whether } C = abP, \text{ or equivalently, whether } c \equiv ab \pmod{n}. \]

The ECDDHP is no harder than the ECDHP, since if the ECDHP in \( < P > \) can be solved then the ECDDHP in \( < P > \) can be efficiently solved by finding \( C' = abP \) from \((P, A, B)\), and then comparing \( C' \) with \( C \).

### 1.4 Security of Public Key Cryptosystems

To quantify the security of public key cryptosystems, the concept of computational security is widely used. Here an adversary is assumed to have limited computation time and memory available (polynomial in the input parameters). The security of the cryptosystem is then based on the fact that the problem of breaking the system is reducible to solving a problem that is strongly believed to be computationally hard, such as factoring a product of two large random primes and taking discrete logarithms in a large finite field.

A cryptographic method is said to be provably secure if the difficulty of defeating it can be shown to be essentially as difficult as solving a well-known and supposedly difficult (typically number-theoretic) problem, such as integer factorization or the computation of discrete logarithms.

Ad hoc security consists of any variety of convincing arguments that every successful attack requires a resource level (e.g., time and space) greater than the fixed resources of a perceived adversary. Cryptographic primitives and protocols which survive such analysis are said to have heuristic security, with security here typically in the computational sense.

Such claims of security generally remain questionable and unforeseen attacks remain a threat. As a rule, one assumes that an adversary always has access to all data being transmitted by two communicating parties and exact knowledge of every aspect of the used cryptographic scheme, except for the secret key. This is referred to as the Kerchoff’s Principle. Moreover, an adversary can request encryptions of polynomially many (in the
size of the input parameters) chosen messages to achieve his objective. This scenario is usually referred to as a black-box assumption, since it allows purely theoretical proofs on paper.

However, cryptosystems are used in the real world where cryptographic protocols are implemented in software or hardware, obeying laws of physics. The circuits used leak information, e.g., power and timing information, over side channels. Thus, one has a gray box, where an adversary has access to several side-channels. Security in the real world depends on some assumptions on which cryptographers silently agree on, i.e., it is assumed that all parties play by the rules and follow specifications (which may be reasonable assumption). Furthermore, it is assumed that a cryptosystem running on some device can not be attacked by other means than described by the black box model. This is not a realistic assumption, since the major problem for security in the real world is the environment in which the cryptographic algorithm is executed. An adversary can attack an implementation instead of the algorithmic specification. Thus there are a lot of ways to break a “provably secure” cryptosystem. The device running an algorithm and operating system has to be secure from unwanted influences, otherwise the cryptographic protocol cannot be secure.

1.4.1 Smartcards

Smartcards can be used to securely store secret keys and perform the cryptographic operations, which require the secret key, namely signing and decryption. They are designed for purpose of a specific security related task. The most important characteristic of a smartcard is that it contains a computer with a CPU and a memory. Today’s smart cards have approximately the same computing power as the first IBM PC. Even cryptographic schemes that involve a significant amount of arithmetic operations such as elliptic curve cryptosystems can be run on them. The key characteristics of smartcards in today’s world are security, ease of use, mobility, and multi-functionality. One of the most important advantages of smart cards consists of the fact that their stored data can be protected against unauthorized access and tampering. Smartcards are tamper-resistant but not tamper-proof. The chip of a smart card consists of a microprocessor, ROM (Read Only
Memory), EEPROM (Electrical Erasable Programmable Read Only Memory), and RAM (Random Access Memory), as seen in Figure 1.1. By performing signature and decryption operations on the card itself, the user’s private key never needs to leave the secure storage of the card. The information stored in the ROM is written during production. It contains the card operating system and might also contain some applications. The operating system is provided by the manufacturer including the hardware. The EEPROM is used for permanent storage of data. Even if the smartcard is unpowered, the EEPROM still keeps the data. Some smartcards also allow the storage of additional application code, or application specific commands in the EEPROM. The RAM is the transient memory of the card and keeps the data only as long as the card is powered. More details on smartcards can be found in [78].

![Figure 1.1: Example of a smartcard.]

**Side-Channel Attacks**

Smartcards are electronic devices which obey the laws of physics. To compute a result, they require a certain amount of time, and energy, while electronic circuits emit a certain amount of radiation, energy, and sound, which can all be affected by their environment. Smartcards are not equipped with their own power source, or their own signal generator, so they have to be connected to the *smartcard reader*. This reader can measure time and power consumption of the smartcard. If this data is correlated to secret data, then
an adversary can obtain additional information. These additional sources of information are referred to as a \textit{side-channels}. It has been proven that large number of side-channels provide information which reveals important and compromising details about secret data. These details, can be used as a new trapdoors to invert a trapdoor one-way function without the secret key. Side-channel attacks can be:

- Passive - an adversary just listens to some side channel without interfering with the computation, e.g., \textit{power consumption} and \textit{power profile}, \textit{timing measurements}, \textit{electromagnetic emissions}, \textit{sound}, \textit{cache memory behavior}, \textit{presence and abuse of testing circuitry}, \textit{data gathered by probing circuitry}, or \textit{bus lines}.

- Active - an adversary tampers with an attacked device in order to create faults, i.e., induces faults into the device while it executes a known program, and observes the reaction, i.e., \textit{fault attacks}.

In this thesis we will concentrate on the \textit{active side-channel attacks}, i.e., \textit{fault attacks}.

\subsection*{1.5 Conclusion}

In this chapter we have provided context of our research and thesis outline, as well as an overview of the cryptography including number theoretic problems and security of cryptosystems.
Chapter 2

Elliptic Curve Cryptography (ECC)

Elliptic curve cryptography (ECC) was discovered independently by Neal Koblitz and Victor Miller in 1985. Elliptic curve cryptographic schemes provide the same functionality as RSA schemes. Their security is based on the hardness of the elliptic curve discrete logarithm problem (ECDLP). Currently the best algorithms known to solve the ECDLP have fully exponential running time, in contrast to the subexponential-time algorithms known for the integer factorization problem. Therefore, a desired security level can be attained with significantly smaller keys in elliptic curve cryptosystems than with their RSA counterparts. A 160-bit elliptic curve key provides the same level of security as a 1024-bit RSA key. The advantages that can be gained from smaller key sizes include speed, efficient use of power, bandwidth, and storage.

In this chapter, in Section 2.2 we provide an overview of elliptic curves by describing the group operations of addition and doubling, and projective coordinate representation of non-supersingular elliptic curves. Since elliptic curve operations are performed using arithmetic operations in the underlying field, and the points of elliptic curve together with addition operation form an abelian group, we provide short overview of groups, rings and fields in Section 2.1. Moreover, we will mention attacks on elliptic curve cryptosystems in Section 2.3, as well as some elliptic curve cryptographic schemes in Section 2.5.
2.1 Abstract Algebra

Here we provide standard definitions of algebraic structures which will be needed in the rest of the thesis. Please see [59], or [87] for more details.

**Definition 2.1.1.** A binary operation $\ast$ on a set $S$ is a mapping from $S \times S$ to $S$. That is, $\ast$ is a rule which assigns to each ordered pair of elements from $S$ an element of $S$.

### 2.1.1 Groups

**Definition 2.1.2.** A group $(G, \ast)$ consists of a set $G$ with a binary operation $\ast$ on $G$ satisfying the following three axioms:

(i) The group operation is associative. That is, $a \ast (b \ast c) = (a \ast b) \ast c$ for all $a, b, c \in G$.

(ii) There is an element $e \in G$, called the identity element, such that $a \ast e = e \ast a = a$ for all $a \in G$.

(iii) For each $a \in G$ there exists an element $a' \in G$, called the inverse of $a$, such that $a \ast a' = a' \ast a = 1$.

A group $G$ is abelian (or commutative) if, furthermore,

(iv) $a \ast b = b \ast a$ for all $a, b \in G$.

**Definition 2.1.3.** A group $G$ is finite if $|G|$ is finite. The number of elements in a finite group is called its order.

**Definition 2.1.4.** A non-empty subset $H$ of a group $G$ is a subgroup of $G$ if $H$ is itself a group with respect to the operation of $G$. If $H$ is a subgroup of $G$ and $H \neq G$, then $H$ is called a proper subgroup of $G$.

**Definition 2.1.5.** A group $G$ is cyclic if there is an element $\alpha \in G$ such that for each $b \in G$ there is an integer $i$ with $b = \alpha^i$. Such an element $\alpha$ is called a generator of $G$.

**Theorem 2.1.6.** If $G$ is a group and $a \in G$, then the set of all powers of $a$ forms a cyclic subgroup of $G$, called the subgroup generated by $a$, and is denoted by $\langle a \rangle$. 
Definition 2.1.7. Let $G$ be a group and $a \in G$. The order of $a$ is defined to be the least positive integer $t$ such that $a^t = 1$, provided that such an integer exists. If such a $t$ does not exist, then the order of $a$ is defined to be $\infty$.

Theorem 2.1.8. Let $G$ be a group, and let $a \in G$ be an element of finite order $t$. Then $|<a>|$, the size of the subgroup generated by $a$, is equal to $t$.

Theorem 2.1.9 (Lagrange’s theorem). If $G$ is a finite group and $H$ is a subgroup of $G$, then $|H|$ divides $|G|$. Hence, if $a \in G$, the order of $a$ divides $|G|$.

Theorem 2.1.10. Every subgroup of a cyclic group $G$ is also cyclic. In fact, if $G$ is a cyclic group of order $n$, then for each positive divisor $d$ of $n$, $G$ contains exactly one subgroup of order $d$.

2.1.2 Rings and Fields

Definition 2.1.11. A ring $(R,+,\cdot)$ is a set $R$, together with two binary operations, denoted by $+$ and $\cdot$, such that:

1. $R$ is an abelian group with respect to $+$.

2. $\cdot$ is associative; that is, $(a \cdot b) \cdot c = a \cdot (b \cdot c)$ for all $a,b,c \in R$.

3. The distributive laws hold; that is for all $a,b,c \in R$ we have $a \cdot (b + c) = a \cdot b + a \cdot c$ and $(b + c) \cdot a = b \cdot a + c \cdot a$.

Definition 2.1.12. Let $R$ be a ring:

1. A ring is called a ring with identity if the ring has a multiplicative identity; that is if there is an element $e$ such that $ae = ea = a$ for all $a \in R$.

2. A ring is called commutative if $\cdot$ is commutative.

3. A ring is called an integral domain if it is a commutative ring with identity $e \neq 0$ in which $ab = 0$ implies $a = 0$, or $b = 0$.

4. A ring is called a division ring (or skew field) if the nonzero elements of $R$ form a group under $\cdot$. 

21
5. A commutative division ring is called a field.

**Definition 2.1.13.** A subset $S$ of a ring $R$ is called a subring of $R$ provided $S$ is closed under $+$ and $\cdot$ and forms a ring under these operations.

**Definition 2.1.14.** A subset $J$ of a ring $R$ is called an ideal provided $J$ is a subring of $R$ and for all $a \in J$ and $r \in R$ we have $ar \in J$ and $ra \in J$.

Ideals are normal subgroups of the additive group of a ring, therefore an ideal $J$ of the ring $R$ defines a partition of $R$ into disjoint cosets called residue classes modulo $J$. Elements $a, b \in R$ are called congruent modulo $J$, written $a \equiv b \, (\text{mod} \, J)$, if they are in the same residue class modulo $J$, or equivalently, if $a - b \in J$. The set of residue classes of a ring $R$ modulo an ideal $J$ forms a ring which is called residue class ring of $R$ modulo $J$, denoted $R/J$, with respect to the operations

\[
(a + J) + (b + J) = (a + b) + J,
\]

\[
(a + J)(b + J) = ab + J.
\]

**Definition 2.1.15.** Let $R$ be an arbitrary ring. The ring formed by polynomials over $R$ with operations

\[
f(x) + g(x) = \sum_{i=0}^{n} (a_i + b_i)x^i, \quad f(x) = \sum_{i=0}^{n} a_i x^i, \quad g(x) = \sum_{i=0}^{n} b_i x^i,
\]

\[
f(x)g(x) = \sum_{k=0}^{n+m} c_k x^k, \quad c_k = \sum_{\substack{i+j=k \\ 0 \leq i \leq n \\ 0 \leq j \leq m}} a_i b_j, \quad f(x) = \sum_{i=0}^{n} a_i x^i, \quad g(x) = \sum_{j=0}^{m} b_j x^j
\]

is called the polynomial ring over $R$ and denoted $R[x]$.

**Theorem 2.1.16.** Let $R$ be a ring. Then:

1. $R[x]$ is commutative if and only if $R$ is commutative.
2. $R[x]$ is a ring with identity if and only if $R$ has an identity.
3. $R[x]$ is an integral domain if and only if $R$ is an integral domain.
Theorem 2.1.17 (Division Algorithm). Let \( g \neq 0 \) be a polynomial in \( \mathbb{F}[x] \). Then for any \( f \in \mathbb{F}[x] \) there exist polynomials \( q, r \in \mathbb{F}[x] \) such that \( f = qg + r \), where \( \deg(r) < \deg(g) \).

Theorem 2.1.18. For \( f \in \mathbb{F}[x] \), the residue class ring \( \mathbb{F}[x]/\langle f \rangle \) is a field if and only if \( f \) is irreducible over \( \mathbb{F} \).

Definition 2.1.19. An element \( b \in \mathbb{F} \) is a root of the polynomial \( f \in \mathbb{F}[x] \) if and only if \( x - b \) divides \( f(x) \).

Theorem 2.1.20. Let \( \mathbb{F} \) be a finite field. Then \( \mathbb{F} \) has \( p^n \) elements, where the prime \( p \) is the characteristic of \( \mathbb{F} \) and \( n \) is the degree of \( \mathbb{F} \) over its prime subfield.

Lemma 2.1.21. If \( \mathbb{F} \) is a finite field with \( q \) elements, then every \( a \in \mathbb{F} \) satisfies \( a^q = a \).

Lemma 2.1.22. If \( \mathbb{F} \) is a finite field with \( q \) elements and \( \mathbb{K} \) is a subfield of \( \mathbb{F} \), then the polynomial \( x^q - x \) in \( \mathbb{K}[x] \) factors in \( \mathbb{F}[x] \) as \( x^q - x = \prod_{a \in \mathbb{F}} (x - a) \) and \( \mathbb{F} \) is a splitting field of \( x^q - x \) over \( \mathbb{K} \).

Theorem 2.1.23 (Existence and Uniqueness of Finite Fields). For every prime \( p \) and every positive integer \( n \) there exist a finite field with \( p^n \) elements. Any finite field with \( q = p^n \) elements is isomorphic to the splitting field of \( x^q - x \) over \( \mathbb{F}_p \).

Theorem 2.1.24 (Subfield Criterion). Let \( \mathbb{F}_q \) be the finite field with \( q = p^n \) elements. Then every subfield \( \mathbb{F}_q \) has order \( p^m \), where \( m \) is a positive divisor of \( n \). Conversely, if \( m \) is a positive divisor of \( n \), then there is exactly one subfield of \( \mathbb{F}_q \) with \( p^m \) elements.

Theorem 2.1.25. For every finite field \( \mathbb{F}_q \) the multiplicative group \( \mathbb{F}_q^* \) of nonzero elements of \( \mathbb{F}_q \) is cyclic.

Definition 2.1.26. A generator of the cyclic group \( \mathbb{F}_q^* \) is called a primitive element of \( \mathbb{F}_q \).

Corollary 2.1.27. For a finite field \( \mathbb{F}_q \) and every positive integer \( n \) there exist an irreducible polynomial in \( \mathbb{F}_q[x] \) of degree \( n \).

Theorem 2.1.28. If \( f \) is an irreducible polynomial in \( \mathbb{F}_q[x] \) of degree \( m \) then \( f \) has a root \( \alpha \) in \( \mathbb{F}_{q^m} \). Furthermore, all the roots of \( f \) are simple and are given by the \( m \) distinct elements \( \alpha, \alpha^q, \alpha^{q^2}, \ldots, \alpha^{q^{m-1}} \) of \( \mathbb{F}_{q^m} \).
There are two special types of bases of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$:

- **polynomial basis** - made up of the powers of a defining element $\alpha$ of $\mathbb{F}_{q^m}$ over $\mathbb{F}_q$, i.e., $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$, where $\alpha$ is often primitive element of $\mathbb{F}_q[x]$; and

- **normal basis** - consists of a suitable element $\alpha \in \mathbb{F}_{q^m}$ and its conjugates with respect to $\mathbb{F}_q$, i.e., $\{\alpha, \alpha^q, \ldots, \alpha^{q^{m-1}}\}$.

### 2.2 Introduction to Elliptic Curves

**Definition 2.2.1 ([41]).** An elliptic curve $E$ over a field $K$ is defined by an equation

$$E : \ y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6 \quad (2.1)$$

where $a_1, a_2, a_3, a_4, a_6 \in K$ and $\Delta \neq 0$, where $\Delta$ is the discriminant of $E$ and is defined as follows:

$$\Delta = -d_2^2d_8 - 8d_4^3 - 27d_6^2 + 9d_2d_4d_6,$$

$$d_2 = a_1^2 + 4a_2,$$

$$d_4 = 2a_4 + a_1a_3,$$

$$d_6 = a_3^2 + 4a_6,$$

$$d_8 = a_1^2a_6 + 4a_2a_6 - a_1a_3a_4 + a_2a_3^2 - a_4^2.$$  

The set of points $(x, y) \in K \times K$ that satisfy equation (2.1) together with the point at infinity $\infty$ is denoted by

$$E(K) = \{(x, y) \in K \times K : y^2 + a_1xy + a_3y - x^3 - a_2x^2 - a_4x - a_6 = 0\} \cup \{\infty\}.$$  

**Remark 2.2.2.** on Definition 2.2.1

- Equation (2.1) is a Weierstrass equation.
• The condition $\Delta \neq 0$ ensures that elliptic curve is “smooth”, i.e., there are no points at which the curve has two or more distinct tangent lines.

• We say that $E$ is defined over the underlaying field $K$, because the coefficients $a_1, a_2, a_3, a_4, a_6$ of its defining equation are elements of $K$. If $E$ is defined over $K$, then $E$ is also defined over any extension field containing $K$.

### 2.2.1 Simplified Weierstrass equations

**Definition 2.2.3** ([41]). Two elliptic curves $E_1$ and $E_2$ defined over $K$ and given by the Weierstrass equations

\[
E_1 : y^2 + a_1 xy + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6 \\
E_2 : y^2 + a_1 x y + a_3 y = x^3 + a_2 x^2 + a_4 x + a_6
\]

are said to be isomorphic over $K$ if there exist $u, r, s, t \in K$, $u \neq 0$, such that the change of variables

\[
(x, y) \rightarrow (u^2 x + r, u^3 y + u^2 sx + t)
\]

(2.2) transforms equation $E_1$ into $E_2$. The transformation (2.2) is called an admissible change of variables.

A Weierstrass equations (2.1) can be simplified considerably by applying admissible changes of variables. We consider separately the cases where the underlaying field $K$ has characteristic different from 2 and 3, or equal to 2, or 3.

1. If $\text{char}(K) \neq 2, 3$, then by

\[
(x, y) \rightarrow \left( \frac{x - 3a_1^2 - 12a_2}{36}, \frac{y - 3a_1 x}{216} - \frac{a_1^3 + 4a_1 a_2 - 12a_3}{24} \right)
\]

$E$ is transformed to the curve $y^2 = x^3 + ax + b$, $a, b \in K$, where $\Delta = -16(4a^3 + 27b^2)$.

2. If $\text{char}(K) = 2$, then if
• $a_1 \neq 0$, then by

$$(x, y) \rightarrow \left(\frac{a_1^2 x + a_3}{a_1}, a_1^3 y + \frac{a_1^2 a_4 + a_2^3}{a_1^3}\right)$$

$E$ is transformed to the curve $y^2 + xy = x^3 + ax^2 + b$, $a, b \in K$. Such a curve is said to be non-supersingular, and $\Delta = b$.

• $a_1 = 0$ then by

$$(x, y) \rightarrow (x + a_2, y)$$

transforms $E$ to the curve $y^2 + cy = x^3 + ax + b$, $a, b, c \in K$. Such a curve is said to be supersingular, and $\Delta = c^4$.

3. If $\text{char}(K) = 3$ then if

• $a_1^2 \neq -a_2$ then by

$$(x, y) \rightarrow \left(x + \frac{d_4}{d_2}, y + a_1 x + a_1 \frac{d_4}{d_2} + a_3\right)$$

where $d_2 = a_1^2 + a_2$ and $d_4 = a_4 - a_1 a_3$, then $E$ is transformed to the curve $y^2 = x^3 + ax^2 + b$, $a, b \in K$. Such curve is said to be non-supersingular, and $\Delta = -a^3 b$.

• $a_1^2 = -a_2$, then by

$$(x, y) \rightarrow (x, y + a_1 x + a_3),$$

$E$ is transformed to the curve $y^2 = x^3 + ax + b$, $a, b \in K$. Such curve is said to be supersingular, and it has $\Delta = -a^3$.

2.2.2 Group law for Elliptic Curves

Let $E$ be an elliptic curve defined over $K$. There is a chord-and-tangent rule for adding two points in $E(K)$. Together with this addition operation, the set of points $E(K)$ forms an abelian group with $\infty$ serving as its identity. The addition rule is best explained geometrically, i.e., see Fig. 2.1. Let $P(x_1, y_1)$ and $Q(x_2, y_2)$ be two distinct points on an elliptic curve $E$. Then the sum $R$, of $P$ and $Q$ is defined as follows. First draw a line through $P$ and $Q$; this line intersects the elliptic curve at a third point. Then $R$ is the
reflection of this point about the $x$–axis.

The *double* $R$, of $P$, is defined as follows. First draw the tangent line to the elliptic curve at $P$, see Fig. 2.2. This line intersects the elliptic curve at a second point. Then $R$ is reflection of this point about $x$–axis. Algebraic formulas for the group law are derived from the geometric description as follows.

**Group law for** $E/K : y^2 = x^3 + ax + b$, $\text{char}(K) \neq 2, 3$

1. **Identity.** $P + \infty = \infty + P = P$, $P \in E(K)$.

2. **Negatives.** Let $P = (x, y) \in E(K)$, then $(x, y) + (x, -y) = \infty$, where $-P = (x, -y)$ is negative of $P$, $-P \in E(K)$, and $-\infty = \infty$.

3. **Point addition.** Let $P = (x_1, y_1), Q = (x_2, y_2) \in E(K)$, where $P \neq \pm Q$. Then $P + Q = (x_3, y_3)$, such that

   $x_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right)^2 - x_1 - x_2$ \quad and \quad $y_3 = \left( \frac{y_2 - y_1}{x_2 - x_1} \right) (x_1 - x_3) - y_1$.

4. **Point Doubling.** Let $P = (x_1, y_1) \in E(K)$, where $P \neq -P$, then $2P = (x_3, y_3)$,
where \( x_3 = \left( \frac{3x_1^2 + a}{2y_1} \right)^2 - 2x_1 \) and \( y_3 = \left( \frac{3x_1^2 + a}{2y_1} \right) (x_1 - x_3) - y_1 \).

**Group law for non-supersingular** \( E/\text{GF}(2^k) : y^2 = x^3 + ax^2 + b \)

1. **Identity.** \( P + \infty = \infty + P, \ P \in E(\text{GF}(2^k)) \).

2. **Negatives.** If \( P = (x, y) \in E(\text{GF}(2^k)) \), then \( (x, y) + (x, x + y) = \infty \), where \( -P = (x, x + y) \in E(\text{GF}(2^k)) \), and it is called **negative** of \( P \), and \( -\infty = \infty \).

3. **Point addition.** Let \( P = (x_1, y_1), Q = (x_2, y_2) \in E(\text{GF}(2^k)) \), where \( P \neq \pm Q \), then \( P + Q = (x_3, y_3) \), where

\[
    x_3 = \lambda^2 + \lambda + x_1 + x_2 + a \quad \text{and} \quad y_3 = \lambda (x_1 + x_3) + x_3 + y_1
\]

with \( \lambda = (y_1 + y_2)/(x_1 + x_2) \).

4. **Point doubling.** Let \( P = (x_1, y_1) \in E(\text{GF}(2^k)) \), \( P \neq -P \), then \( 2P = (x_3, y_3) \), where

\[
    x_3 = \lambda^2 + \lambda + a = x_1^2 + \frac{b}{x_1^2}, \quad y_3 = x_1^2 + \lambda x_3 + x_3
\]

with \( \lambda = x_1 + y_1/x_1 \).
Group law for supersingular $E_{GF}(2^k) : y^2 + cy = x^3 + ax + b$

1. **Identity.** $P + \infty = \infty + P, P \in E_{GF}(2^k)$

2. If $P = (x, y) \in E_{GF}(2^k)$, then $(x, y) + (x, y + c) = \infty$, where $-P = (x, x + c) \in E_{GF}(2^k)$, and it is called negative of $P$, and $-\infty = \infty$.

3. **Point addition.** Let $P = (x_1, y_1), Q = (x_2, y_2) \in E_{GF}(2^k)$, where $P \neq \pm Q$, then $P + Q = (x_3, y_3)$, where

$$x_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)^2 + x_1 + x_2 \quad \text{and} \quad y_3 = \left(\frac{y_1 + y_2}{x_1 + x_2}\right)(x_1 + x_3) + y_1 + c.$$

4. **Point doubling.** Let $P = (x_1, y_1) \in E_{GF}(2^k)$, $P \neq -P$, then $2P = (x_3, y_3)$, where

$$x_3 = \left(\frac{x_1^2 + a}{c}\right)^2 \quad y_3 = \left(\frac{x_1^2 + a}{c}\right)(x_1 + x_3) + y_1 + c.$$

### 2.2.3 Point Representation in the Projective Coordinates

The formulas for point addition and point doubling in affine coordinates require a field inversion and several field multiplications. If inversion in the field $K$ is significantly more expensive than multiplication, then it is advantageous to represent points using projective coordinates instead of affine coordinates.

Let $K$ be a field, and let $c$ and $d$ be positive integers, then one can define an equivalence relation $\sim$ on the set $K^3/\{(0,0,0)\}$ of nonzero triplets over $K$ by

$$(X_1, Y_1, Z_1) \sim (X_2, Y_2, Z_2) \quad \text{if} \quad X_1 = \lambda^c X_2, Y_1 = \lambda^d Y_2, Z_1 = \lambda Z_2, \lambda \in K^*.$$

The equivalence class containing $(X, Y, Z) \in K^3/\{(0,0,0)\}$ is

$$(X : Y : Z) = \{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K^*\}.$$

$(X : Y : Z)$ is called projective point, and $(X, Y, Z)$ is called a representative of $(X : Y : Z)$. The set of all projective points is denoted by $\mathbb{P}(K)$. Any element of an equivalence class can serve as its representative, i.e., if $Z \neq 0$, then $(X/Z^c, Y/Z^d, 1)$ is a representative of the projective point $(X : Y : Z)$. There is a $1 - 1$ correspondence between the set of projective points

$$\mathbb{P}(K)^* = \{(X : Y : Z) : X, Y, Z \in K, Z \neq 0\}.$$
and the set of affine points

\[ \mathbb{A}(K) = \{(x, y) : x, y \in K \}. \]

The set of projective points

\[ \mathbb{P}(K)^0 = \{(X : Y : Z) : X, Y, Z \in K, Z = 0\} \]

is called the line at infinity, since its points do not correspond to any of the affine points.

The projective form of the Weierstrass equation (2.1) of an elliptic curve \( E \) defined over \( K \) is obtained by replacing \( x \) by \( X/Z^c \), and \( y \) by \( Y/Z^d \), and clearing denominators.

Several types of projective coordinates have been proposed for the non-supersingular elliptic curve \( E : y^2 + xy = x^3 + ax^2 + b \) over a binary field \( K \), i.e., with \( \text{char}(K) = 2 \):

1. **Standard projective coordinates.** Here \( c, d = 1 \). The projective point \( (X : Y : Z) \), \( Z \neq 0 \), corresponds to the affine point \( (X/Z, Y/Z) \). The projective equation of the elliptic curve is

\[ Y^2Z + XYZ = X^3 + aX^2Z + bZ^3. \]

The point at infinity \( \infty \) corresponds to \((0 : 1 : 0)\), while negative of \( (X : Y : Z) \) is \( (X : X + Y : Z) \).

2. **Jacobian projective coordinates.** Here \( c = 2, d = 3 \). The projective point \( (X : Y : Z) \), \( Z \neq 0 \), corresponds to the affine point \( (X/Z^2, Y/Z^3) \). The projective equation of the elliptic curve is:

\[ Y^2 + XYZ = X^3 + aX^2Z^2 + bZ^6. \]

The point at infinity \( \infty \) corresponds to \((1 : 1 : 0)\), while the negative of \( (X : Y : Z) \) is \( (X : X + Y : Z) \).

3. **López-Dahab (LD) projective coordinates.** Here \( c = 1, d = 2 \). The projective point \( (X : Y : Z) \), \( Z \neq 0 \), corresponds to the affine point \( (X/Z, Y/Z^2) \). The projective equation of the elliptic curve is:

\[ Y^2 + XYZ = X^3Z + aX^2Z^2 + bZ^4. \]
The point at infinity $\infty$ corresponds to $(1 : 0 : 0)$, while the negative of $(X : Y : Z)$ is $(X : X + Y : Z)$. Formulas for computing double $(X_3 : Y_3 : Z_3)$ of $(X_1 : Y_1 : Z_1)$ are

$$Z_3 \leftarrow X_1^2Z_1^2, \quad X_3 \leftarrow X_1^4 + bZ_1^4, \quad Y_3 \leftarrow bZ_1^4Z_3 + X_3(aZ_3 + Y_1^2 + bZ_1^4) .$$

Formulas for computing sum $(X_3 : Y_3 : Z_3)$ of $(X_1 : Y_1 : Z_1)$ and $(X_2 : Y_2 : 1)$ are

$$A \leftarrow Y_2Z_1^2 + Y_1, \quad B \leftarrow X_2Z_1 + X_1, \quad C \leftarrow Z_1B, \quad D \leftarrow B^2(C + aZ_1^2), \quad Z_3 \leftarrow C^2,$$

$$E \leftarrow AC, \quad X_3 \leftarrow A^2 + D + E, \quad F \leftarrow X_3 + X_2Z_3, \quad G \leftarrow (X_2 + Y_2)Z_3^2, \quad Y_3 \leftarrow (E + Z_3)F + G .$$

### 2.3 Attacks on Elliptic Curve Cryptography.

The security of all elliptic curve cryptographic schemes is based on the hardness of the elliptic curve discrete logarithm problem (ECDLP), see Subsection 1.3.6. The most naive algorithm for solving the ECDLP is exhaustive search, where one computes the sequence of points $P, 2P, 3P, 4P, \ldots$ until $Q$ is encountered. The running time is approximately $n$ steps in the worst case, and $n/2$ on average, where $n$ is order of $P$. The countermeasure for this type of attack is to choose $n$ sufficiently large, i.e., $n \geq 2^{80}$. The best general-purpose attack known on the ECDLP is the combination of the Pohlig-Hellman algorithm and Polard’s rho algorithm, which has a fully-exponential running time of $O(\sqrt{p})$, where $p$ is the largest prime divisor of $n$. To resist this attack, $n$ should be chosen such that it is divisible by a prime number $p$ sufficiently large, so that $\sqrt{p}$ with $(p > 2^{160})$ steps is an infeasible amount of computation.

#### 2.3.1 Pohlig-Hellman attack.

The Pohlig-Hellman algorithm [74] reduces the computation of $l = \log_P Q$ to the computation of the discrete logarithm in the prime order subgroup of $\langle P \rangle$. It follows that the ECDLP in $\langle P \rangle$ is no harder then the ECDLP in its prime order subgroups. Therefore, as a countermeasure we select $P$ such that the order $n$ of $P$ is divisible by a large prime.
2.3.2 Pollard’s Rho Attack.

The rho algorithm for computing discrete logarithms was invented by Pollard [75]. The main concept of the Pollard’s rho algorithm is to find distinct pairs \((c', d')\) and \((c'', d'')\) of integers modulo \(n\) such that

\[c'P + d'Q = c''P + d''Q.\]

Then

\[(c' - c'')P = (d'' - d')Q = (d'' - d')lP,

and so

\[(c' - c'') \equiv (d'' - d')l \pmod{n}.

Therefore, \(l = log_PQ\) can be obtained by computing

\[l = (c' - c'')(d'' - d')^{-1} \pmod{n}.

A naive method for finding such pairs \((c', d')\) and \((c'', d'')\) is to select random integers \(c, d \in [0, n - 1]\) and store the triplets \((c, d, cP + dQ)\) in a table sorted by the third component until a point \(cP + dQ\) is obtained for a second time - such an occurrence is called a collision. By the birthday paradox, the expected number of iterations before a collision is obtained is approximately \(\sqrt{\pi n/2}\), but the drawback of this algorithm is the storage requirement for the \(\sqrt{\pi n/2}\) triplets. Pollard’s rho algorithm finds \((c', d')\) and \((c'', d'')\) in roughly the same expected time as the naive method, but with negligible storage requirements. The idea is to define an iterating function \(f : \langle P \rangle \rightarrow \langle P \rangle\), so that given \(X \in \langle P \rangle\) and \(c, d \in [0, n - 1]\) with \(X = cP + dQ\), it is easy to compute \(\overline{X} = f(X)\) and \(\overline{c}, \overline{d} \in [0, n - 1]\) with \(\overline{X} = \overline{c}P + \overline{d}Q\), where \(f\) should have the characteristics of a random function.

The Pollard’s rho method takes about \(\sqrt{\pi n/2}\) steps, where each step is an elliptic addition. Also, this method it can be parallelized ([72]), so that if \(m\) processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is \((\sqrt{\pi n/2}/m)\). The Pollard-\(\lambda\) method takes about \(2\sqrt{n}\) steps. Also, it can be parallelized so that if \(m\) processors are used, then the expected number of steps by each processor before a single discrete logarithm is obtained is about \((2\sqrt{n})/m\).
2.3.3 Isomorphism attacks.

Let $E$ be an elliptic curve defined over a field $\mathbb{F}_q$, and let $P \in E(\mathbb{F}_q)$ have prime order $n$, and $G$ be group of order $n$. Since $n$ is prime, $\langle P \rangle$ and $G$ are both cyclic and hence isomorphic. If one could efficiently compute an isomorphism

$$\psi : \langle P \rangle \rightarrow G,$$

then ECDLP instances in $\langle P \rangle$ could be efficiently reduced to instances of the DLP in $G$. Given $P$ and $Q \in \langle P \rangle$, we have

$$\log_P Q = \log_{\psi(P)} \psi(Q).$$

Isomorphism attacks reduce the EDLP to the DLP in groups $G$ for which subexponential-time (or faster) algorithms are known. They result in ECDLP solvers that are faster than the Pollard’s rho algorithm, but only for special classes of elliptic curves. The following attacks have been devised: attack on the prime-field-anomalous curves, the Weil and Tate pairing attacks, and Weil descent attacks.

Attack on prime-field anomalous curves

The attack on the prime-field-anomalous curves reduces the ECDLP in an elliptic curve of order $p$ defined over the prime field $\mathbb{F}_p$ to the DLP in the additive group $\mathbb{F}_p^+$ of integers modulo $p$.

**Definition 2.3.1.** Let $E$ be elliptic curve defined over a prime field $\mathbb{F}_p$. It is said to be prime-field-anomalous if $\#E(\mathbb{F}_p) = p$.

If $E$ is a prime-field-anomalous curve, the group $E(\mathbb{F}_p)$ is cyclic, since it has prime order, therefore $E(\mathbb{F}_p)$ is isomorphic to the additive group $\mathbb{F}_p^+$ of integers modulo $p$:

$$\psi : E(\mathbb{F}_p) \rightarrow \mathbb{F}_p^+. \quad (2.3)$$

Araki and Satoh [83], Smart [90] and Semaev [85] showed that the isomorphism (2.3) can be efficiently computed for prime-field anomalous elliptic curves. Therefore, these curves must not be used in cryptographic protocols.
Weil and Tate Pairing Attacks

The Weil and Tate Pairing Attacks are due to Menezes, Okamoto and Vanstone [67], and Frey and Ruck [33]. Let \( P \in E(\mathbb{F}_p) \) be of prime order \( n \), and let \( k \) be the smallest positive integer such that \( q^k \equiv 1 \pmod{n} \) and \( k \mid n - 1 \). Since \( n \mid q^k - 1 \), the multiplicative group \( \mathbb{F}_{q^k}^* \) of the extension field \( \mathbb{F}_{q^k} \) has a unique subgroup \( G \) of order \( n \). The Weil pairing attack constructs an isomorphism from \( <P> \) to \( G \) when \( n - (q - 1) \) is satisfied, while the Tate pairing attack constructs an isomorphism between \( <P> \) and \( G \) without this additional constraint. The integer \( k \) is called embedding degree.

This attack is only practical if \( k \) is small. The special class of elliptic curves with small embedding degree include supersingular curves, and elliptic curves of trace 2 (\( \#E(\mathbb{F}_q) = q - 1 \)), i.e., these curves have \( k \leq 6 \).

To ensure that an elliptic curve \( E \) defined over \( \mathbb{F}_q \) is immune to the Weil and Tate pairing attacks, it is sufficient to check that \( n \), the order of the base point \( P \in E(\mathbb{F}_q) \), does not divide \( q^k - 1 \) for all small \( k \) for which the DLP in \( \mathbb{F}_q^* \) is considered tractable. For example, if \( n > 2^{160} \), then it is enough to check this condition for all \( k \in [1, 20] \).

### 2.4 Elliptic Curve Domain Parameters

Domain parameters for an elliptic curve scheme describe an elliptic curve \( E \) defined over finite field \( \mathbb{F}_q \), a base point \( P \in E(\mathbb{F}_q) \), and its order \( n \).

**Definition 2.4.1.** Domain parameters \( D = (q, FR, S, a, b, P, n, h) \) are comprised of:

1. The field order \( q \).
2. Field representation \( FR \) used for the elements of \( \mathbb{F}_q \).
3. Seed \( S \), if the elliptic curve was randomly generated.
4. Coefficients \( a, b \in \mathbb{F}_q \) that define equation of the elliptic curve \( E \) over \( \mathbb{F}_q \), i.e., \( y^2 + xy = x^3 + ax^2 + b \) in the case of binary field.
5. Base point \( P = (x, y) \in E(\mathbb{F}_q) \).
6. The order $n$ of $P$.

7. The cofactor $h = \#E(\mathbb{F}_q)/n$

Elliptic curve domain parameters are specified by several standards. Table 2.1 gives overview of the properties of the elliptic curves defined over binary fields $GF(2^k)$ recommended by: IEEE P1363-2000 [43], American National Standards Institute (ANSI) [3], Standards for Efficient Cryptography Group (SECG) [84], the Wireless Transport Layer Security Specification [98] by the WAP forum, New European Schemes for Signature, Integrity and Encryption (NESSIE) [71], the specification for the Financial Services Markup Language (FSML) [34] by the eCheck initiative, Federal Information Processing Standard (FIPS) 186-2 [32] issued by the NIST, and from IPSec [45].

### 2.5 Elliptic Curve Cryptographic Schemes

#### 2.5.1 Signature Scheme - ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) is the elliptic curve analogue of the Digital Signature Algorithm (DSA). It appears in the standards such as ANSI X9.62, FIPS 186-2 [32], IEEE 1363-2000 [43] and ISO/IEC 15946-2, as well as several draft standards. Here, $H$ represents a cryptographic hash function whose outputs have bit length no more then that of $n$. In order for ECDSA to be secure, it is necessary

**Algorithm 1** ECDSA signature generation

**Input:** Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key $d$, message $m$.

**Output:** Signature $(r, s)$.

1. Select $k \in_R [1, n - 1]$.
2. Compute $kP = (x_1, y_1)$, and convert $x_1$ to an integer $\overline{x_1}$.
3. Compute $r = \overline{x_1} \mod n$. If $r = 0$, then go to step 1.
4. Compute $e = H(m)$.
5. Compute $s = k^{-1}(e + dr) \mod n$. If $s = 0$ then go to step 1.
6. Return $(r, s)$.
<table>
<thead>
<tr>
<th>Standard</th>
<th>Binary Curves</th>
</tr>
</thead>
<tbody>
<tr>
<td>IEEE</td>
<td>size: &gt; 161 bits</td>
</tr>
<tr>
<td></td>
<td>any “small” cofactor</td>
</tr>
<tr>
<td></td>
<td>avoid known weak curves</td>
</tr>
<tr>
<td></td>
<td>gives no specific curves</td>
</tr>
<tr>
<td>SECG</td>
<td>size: 113, 131, 163, 193, 233, 239, 283, 409, 571</td>
</tr>
<tr>
<td></td>
<td>cofactor 2, or 4</td>
</tr>
<tr>
<td></td>
<td>curve $E$ is chosen verifiably at random as specified</td>
</tr>
<tr>
<td></td>
<td>in ANSI X9.62 from the seed</td>
</tr>
<tr>
<td>ANSI X9.63</td>
<td>size: &gt; 160</td>
</tr>
<tr>
<td></td>
<td>any “small” cofactor</td>
</tr>
<tr>
<td></td>
<td>gives no specific curves</td>
</tr>
<tr>
<td>NIST</td>
<td>size: 163, 232, 283, 409, 571</td>
</tr>
<tr>
<td></td>
<td>cofactor 2, (or 4 for Koblitz curve with $a = 0$)</td>
</tr>
<tr>
<td></td>
<td>pseudo random curve + Koblitz curve</td>
</tr>
<tr>
<td>WAP</td>
<td>size: 113, 163, 233</td>
</tr>
<tr>
<td></td>
<td>cofactor 2</td>
</tr>
<tr>
<td></td>
<td>(some curves are equal to SECG)</td>
</tr>
<tr>
<td>FSML</td>
<td>size: 163, 283</td>
</tr>
<tr>
<td></td>
<td>cofactor 2</td>
</tr>
<tr>
<td></td>
<td>(curves are subset of SECG)</td>
</tr>
<tr>
<td>NESSIE</td>
<td>size: &gt; 160</td>
</tr>
<tr>
<td>IPSec</td>
<td>size: 155 - 571</td>
</tr>
<tr>
<td></td>
<td>cofactor 2, 4, or 12</td>
</tr>
<tr>
<td></td>
<td>(most of the curves are equal to SECG)</td>
</tr>
</tbody>
</table>

Table 2.1: Recommended parameters for elliptic curves over $GF(2^k)$. 
Algorithm 2 ECDSA signature verification

**Input:** Domain parameters $D = (q, FR, S, a, b, P, n, h)$, public key $Q$, message $m$, signature $(r, s)$.

**Output:** Acceptance, or rejection of the signature.

1. Verify that $r$ and $s$ are integers in the interval $[1, n - 1]$. If any verification fails then return (“Reject the signature”).
2. Compute $e = H(m)$.
3. Compute $w = s^{-1} \mod n$.
4. Compute $u_1 = ew \mod n$ and $u_2 = rw \mod n$.
5. Compute $X = u_1 P + u_2 Q$.
6. If $X = \infty$ then return (“Reject the signature”).
7. Convert the $x$-coordinate $x_1$ of $X$ to an integer $x_1$; compute $v = x_1 \mod n$.
8. If $v = r$ then return (“Accept the signature”); else return (“Reject the signature”).

that the ECDLP is intractable, and that hash function $H$ is cryptographically secure, i.e., that it is preimage resistant and collision resistant. Informally, a function $f : X \to Y$ is pre-image resistant if given $y \in \text{Im}(f) \subset Y$ it is infeasible to find an $x \in X$ such that $f(x) = y$. Similarly, $f$ is collision resistant if it is infeasible to find $x \neq x' \in X$ such that $f(x) = f(x')$.

### 2.5.2 Public-key encryption - ECIES

The Elliptic Curve Integrated Encryption Scheme (ECIES) is a variant of the ElGamal public-key encryption scheme, which was proposed by Bellare and Rogaway [15]. It is standardized in ANSI X9.63 [3], ISO/IEC 15946-3 and in IEEE P1363a draft standard. The following cryptographic primitives are used:

- **KDF** - a key derivation function that is constructed from a hash function $H$.
- **ENC** - the encryption function for a symmetric-key encryption scheme such as the AES, and DEC is the decryption function.
- **MAC** - is a message authentication code.
Algorithm 3 ECIES encryption

**Input:** Domain parameters $D = (q, FR, S, a, b, P, n, h)$, public key $Q$, plaintext $m$.

**Output:** Ciphertext $(R, C, t)$.

1. Select $k \in_R [1, n - 1]$.
2. Compute $R = kP$ and $Z = h kQ$. If $Z = \infty$ then go to step 1.
3. $(k_1, k_2) \leftarrow KDF(x_Z, R)$, where $x_Z$ is the $x$-coordinate of $Z$.
4. Compute $C = ENC_{k_1}(m)$ and $t = MAC_{k_2}(C)$.
5. Return $(R, C, t)$.

Algorithm 4 ECIES decryption

**Input:** Domain parameters $D = (q, FR, S, a, b, P, n, h)$, private key $d$, ciphertext $(R, C, t)$.

**Output:** Plaintext $m$, or rejection of the ciphertext.

1. Perform an embedded public key validation of $R$. If the validation fails then return (“Reject the ciphertext”).
2. Compute $Z = hdR$. If $Z = \infty$ then return (“Reject the ciphertext”).
3. $(k_1, k_2) \leftarrow KDF(x_Z, R)$, where $x_Z$ is the $x$-coordinate of $Z$.
4. Compute $t' = MAC_{k_2}(C)$. If $t' \neq t$ then return (“Reject the ciphertext”).
5. Compute $m = DEC_{k_1}(C)$.
6. Return $(m)$.

ECIES is proven to be secure under the assumption that the symmetric-key encryption scheme and MAC algorithm are secure, and that some non-standard variants of the computational and decision Diffie-Hellman problems are intractable.

### 2.5.3 Key Establishment - ECMQV

ECMQV is a three pass key agreement protocol that has been standardized in ANSI X9.63 [3], IEEE 1363-2000 [43], and ISO/IEC 15946-3. Elliptic curve domain parameters are $D = (q, FR, S, a, b, P, n, h)$, A’s key pair is $(Q_A, d_A)$, B’s key pair $(Q_B, d_B)$, KDF is a key derivation function, MAC is a message authentication code. If $R$ is an elliptic curve
point then $\overline{R}$ is defined to be an integer $(\pi \mod 2^{[f/2]}) + 2^{f/2}$, where $\pi$ is the integer representation of the $x$-coordinate of $R$, and $f = \lceil \log_2 n \rceil + 1$ in the bitlength of $n$.

**Protocol 5 ECMQV key agreement**

**Goal:** $A$ and $B$ establish a shared secret key.

**Protocol messages:**

$A \rightarrow B : A, R_A$

$A \leftarrow B : B, R_B, t_B = MAC_{k_1}(2, B, A, R_B, R_A)$

$A \rightarrow B : t_A = MAC_{k_1}(3, A, B, R_A, R_B)$

1. $A$ selects $k_A \in_R [1, n - 1]$, computes $R_A = k_A P$, and sends $A, R_A$ to $B$.

2. $B$ does the following:
   2.1 Perform an embedded public key validation of $R_A$.
   2.2 Select $k_B \in_R [1, n - 1]$ and compute $R_B = k_B P$.
   2.3 $s_B = (k_B + \overline{R_B d_B}) \mod n$ and $Z = h s_B(\overline{R_A Q_A})$, and verify $Z \neq \infty$.
   2.4 $(k_1, k_2) \leftarrow KDF(x_Z)$, where $x_Z$ is the $x$-coordinate of $Z$.
   2.5 Compute $t_B = MAC_{k_1}(2, B, A, R_B, R_A)$.
   2.6 Send $B, R_B, t_B$ to $A$.

3. $A$ does the following:
   3.1 Perform an embedded public key validation of $R_B$.
   3.2 $s_A = (k_A + \overline{R_A d_A}) \mod n$ and $Z = h s_A(\overline{R_B Q_B})$, and verify $Z \neq \infty$.
   3.3 $(k_1, k_2) \leftarrow KDF(x_Z)$, where $x_Z$ is the $x$-coordinate of $Z$.
   3.4 Compute $t = MAC_{k_1}(2, B, A, R_B, R_A)$, and verify that $t = t_B$.
   3.5 Compute $t_A = MAC_{k_1}(3, A, B, R_A, R_B)$ and send $t_A$ to $B$.

4. $B$ computes $t = MAC_{k_1}(3, A, B, R_A R_B)$, and verifies that $t = t_A$.

5. The session key is $k_2$.  

2.6 Conclusion

In this chapter we have discussed the algebraic, cryptographic and mathematical background to elliptic curves that is required to understand the discussion of fault attacks and countermeasures which are taken up in the rest of the thesis.
Chapter 3

Fault Models

Theoretical fault attacks are based on the fault models which model physical behavior of an attacked device. It is important to base all fault models on the real world, since fault attacks are implemented on real devices in the real physical world. In this thesis, we are not concerned with the physical realization of inducing faults; rather physical events will be translated into mathematical form. The mathematical aspect of fault attacks is necessary to prove that algorithmic countermeasures work and whether an attack can break a system, and recover secret information. To derive reasonable fault models so that they are justified by physical attacks, parameter settings from known actual attacks are combined with hardware countermeasures on the card. The most prominent fault models are provided by [73] and [20], and are used throughout the literature. In this chapter we will justify and modify these models so that we can later use them on proposed schemes in Chapter 5 and Chapter 6.

In Chapter 5 and Chapter 6, we propose decomposed computation in the parallel, mutually independent channels, and we assume that during each run of an attacked algorithm, in one single attack, an adversary can apply any of the proposed fault models, i.e., Random Fault Model (RFM), Arbitrary Fault Model (AFM), or Single bit Fault Model (SFM) per different channel. This way more channels can be targeted, where different fault models can be used on different channels. Also till now, no publication has implied that different fault models can be used in one single attack. We propose that if one single attack means targeting more channels, then different fault models can indeed be used in
one single attack. Therefore, in Section 3.1 we discuss fault attacks and possible physical ways of inducing fault. In Section 3.2 we present characterization of the fault models, as well as, possible fault models for schemes/algorithms that we propose later on in Chapter 5 and Chapter 6.

### 3.1 Fault Attacks

Fault attacks have been used before the cryptography community became aware of them, i.e., pay TV card hackers used *clock glitches* [1]. The first successful fault attack has been reported by Boneh et al. [22]. They have presented two important attacks on variants of RSA, used for computing digital signatures. These results triggered extensive research in the field of fault attacks, several authors extended these ideas to other cryptosystems using other fault models and different means of physical attacks.

Fault attacks are based on tampering with a device so that device performs abnormally. They exploit physical properties of devices. From the reaction of the device which can be a *faulty result*, an *error message*, or some form of *security reset*, including destruction of the device, an adversary wishes to learn about the secret key hidden in a device. Since cryptographic algorithms are public, adversary can determine what variables are used and what values they have depending on the secret key. This allows them to determine what kind of error will provoke a certain reaction, which may be observable by the adversary, e.g., if a single bit in the secret key is flipped during an attack, and the device does not detect this fault, a faulty result with a specific pattern is returned. By comparing this faulty result with the correct one, an adversary might be able to deduce one bit of the secret key. An adversary may also target the flow of operations, such that certain operations are repeated or skipped. To achieve and exploit a desired effect, he needs to have knowledge about how a certain physical attack will affect the logical flow of the attacked algorithm. Only then he will be able to bound his success probability and to compute secret data from a faulty output. Fault attacks can exploit faults in two different ways:
• an adversary causes the attacked device to malfunction and to output a faulty result; then the faulty result is used to derive secret information;

• *oracle attacks* - does not use the actual faulty result for computations, only the information whether the result was faulty or not.

### 3.1.1 Physical Methods to Induce Faults

There are numerous ways to induce faults into physical devices, but since the focus of this thesis are the theoretical and algorithmic aspects of fault attacks, we will briefly mention some of them:

• *Power Spikes* - short massive variations of the power supply, which are called *spikes*, can be used to induce errors into the computation of the smartcard. Spikes allow to induce both memory faults, as well as faults in the execution of a program (code change attacks). Both can be used to affect an arbitrary number of bits, starting with single affected bits. Experimental results of inducing faults by spikes are in detail described in [6].

• *Clock Glitches* - smartcards do not create their own clock signal, they use a randomized clock, they only randomize the clock signal provided by the external card reader. Since the adversary may replace the card reader by laboratory equipment, he may provide the card with a clock signal, which incorporates short massive deviations from the standard signal, which are beyond the required tolerance bounds. Such signals are called *glitches*. They can be used to both induce, memory faults, as well to cause a faulty execution behavior (code change attacks). More details can be found in [2], [1],[52].

• *Heat/Infrared Radiation* - electronic equipment only works reliably in a certain range of temperature. If the outside temperature is too low, or too high, faults occur. A heating source can be easily focused at devices such as smartcards. Changes in memory due to heat usually affect a large area, i.e., many bits.
• *Focused Ion Beams* - can be used to drill holes in the passivation layer of a smartcard, which can then be filled by a conducting material in order to access individual elements, or bus lines with measuring equipment, only if card is unpacked. Also, it can be tuned finely enough to ionize silicon locally, which may be interpreted as a signal by the circuit. Destructive faults are possible by destroying circuit elements, or bus lines. More details can be found in [52].

• *External Electrical Field Transients/Eddy Currents* - Changes in the external electrical field can induce faults into smartcard by placing device in an electromagnetic field, which may influence the transistors and memory cells. The main problem using such an approach is to target specific bits, or variables stored on the card. In [77] different approach was presented, i.e., given a coil, which is placed near a conducting surface, a magnetic field can be created if the coil is subject to an alternating current. This magnetic field induces *eddy currents* on the surface of the near conducting material. Eddy currents can be used to measure cracks in a surface, as well as electromagnetic emissions, also can be used to heat a material in a uniform way till is melted. Hence, can induce heat in a *transient, permanent, or destructive* way in a smartcard. Also, inducing eddy current does not require to unpack a chip, and can induce faults very precisely.

**Countermeasures**

Manufactures have developed large variety of the hardware countermeasures, which are usually specifically constructed for different means of physical attacks, i.e.,

- *sensors and filters* - e.g., anomalous frequency detectors, anomalous voltage detectors, or light detectors;

- *redundancy* - doubled memory, hardware, where result is computed twice in parallel, or doubled computation;

- *randomized clock* - used to achieve an unstable internal frequency, bus line, memory encryption, dummy random cycles and active and passive shields protecting the internal circuits.
Hardware countermeasures have a disadvantage, since highly reliable countermeasures are expensive, while other detect only specific attacks. Also, they are beyond the scope of this thesis, and will not be discussed in detail.

The scope of this thesis is to develop new software countermeasures, since they are easier to deploy, and they are cost efficient. Current software countermeasures include: masking, checksums, randomization, counters, baits and variable redundancy.

### 3.2 Fault Models

We assume a strong adversary, and a smartcard with several countermeasures in effect, the most important of which are:

(i) randomized clock which blurs timing behavior of a device,

(ii) memory and data encryption is incorporated to ensure that large number of bits is affected if a single bit is flipped by fault, and

(iii) address and bus line scrambling is employed such that the physical layout of the memory cells does not reflect the logical layout.

As fault attacks, we consider methods, approaches and algorithms which when applied to the attacked processor return the desired effect. We assume that the fault attack induces faults into processors by some physical set up, such that processor is exposed to some physical stress (cosmic rays, heat/infrared radiation, power spikes, clock glitches, etc.). An adversary can run the processor several times while inducing faults into structural elements of an attacked processor, until the desired effects occur. As a reaction the attacked processor malfunctions, i.e., memory cells change their current, bus lines transmit different signals, or structural elements are damaged. It is faulty (does not compute the correct output given its input), and its output is erroneous such that computation assigned to the faulty processor is disturbed, and its channel is affected. Our concern is the effect of a fault as it manifests itself in a modified data, or a modified program execution. We identify memory cells with their values, and we say that faults are induced into variables, or bits. Note that any fault induced in a variable $x$ can be described by means of an
additive error term \( x \mapsto x' = x + e(x) \), but the error term \( e(x) \) can itself take on quite different characteristics, depending on the type of the fault. The fault type describes the effect of the fault on an arbitrary set of bits, and in the literature is classified as:

**Stuck-at Faults.** Let \( b \) be an arbitrary bit stored in memory. Assume that \( b \) is modified by a stuck-at fault. Then \( b \mapsto b' = c \), where the constant \( c = 1 \) or \( c = 0 \). The value of the affected bit is not changed any more, even if a variable \( x \), which uses these bits, is overwritten. Clearly stuck-at faults will have a noticeable effect only if the variable is overwritten at some point. The effect is permanent, but not necessarily destructive.

**Bitflip Faults.** Let \( b \) be an arbitrary bit stored in memory. Assume that \( b \) is modified by a bitflip fault. Then \( b \mapsto b' = b + 1 \) (mod 2). The effect may be transient, permanent or destructive. A bitflip fault is easy to visualize, and always results in a fault on a variable using the bit which is faulty.

**Random Faults.** Let \( b \) be an arbitrary bit stored in memory. Assume that \( b \) is modified by a random fault. Then \( b \mapsto b' \) where \( b' \) is a random variable taking on the values 0 or 1. The effect may be transient, permanent, or destructive. Since several physical methods of fault induction are difficult to control precisely, random faults are considered to be the most realistic fault type. The random variable which models the fault may be uniform or non-uniform.

**Bit Set or Reset Fault.** Let \( b \) be an arbitrary bit stored in memory. Assume that \( b \) is modified by a bit set, or reset fault. Then \( b \mapsto b' = c_i \) where \( c_i \in \{0, 1\} \). The values \( c_i \) are known and are usually chosen by the adversary. The effect may be transient, permanent, or destructive.

Note that the above faults can be considered for an arbitrary, but unknown set of bits \( B \), where assumptions about how the adversary controls the choice of \( B \) can also model different attack scenarios. A full characterization of the different parameters needed to fully describe known fault attacks is presented in [73]. Therefore, fault attacks differ in:

- power to locate - a selected variable can be targeted, a selected bits can be targeted, or no power to locate;
• the timing of the induced faults - no control on timing, a exact time can be met, a
  fault is induced in a block of few operations;

• the number of bits affected - single faulty bit, few faulty bits, random number of
  faulty bits.

• the character of the fault - stuck-at fault, bit flip fault, random fault;

• the probability of the implied effect of an induced fault;

• the duration of an effect - depending on the type of the physical stress, fault effects
  can be of different duration:

  1. *Destructive faults* - an adversary destroys a physical structure on the chip,
     which causes a certain bit, or variable to be fixed at a specific value for all
     successive runs of the device, it cannot be reversed;

  2. *Permanent faults* - change an affected variable until that variable is explicitly
     overwritten, e.g., by a reset at the start of the next run;

  3. *Transient faults* - the induced fault is only short-lived, such that after a given
     amount of time, the effect ceases to exist and the correct value is present again.

Contrary to the the *destructive faults, permanent* and *transient faults* do not modify the
hardware of an attacked device. In the literature, it is assumed that *transient faults* only
affect the next request for the affected variable. *Permanent* and *transient faults* have the
same effect if a faulty variable is used only once. Otherwise, in case of *permanent faults*
all next request yield the faulty value, while in the case of *transient faults* immediate next
request gives faulty value, subsequent requests yield correct value. Therefore, we assume
the following fault models (inspired by [73]).

### 3.2.1 Random Fault Model (RFM)

Assume that an adversary does not know much about his induced faults to know its effect,
but he knows the affected variable at specific channel. Therefore, we assume that affected
variable \( r_j \in GF(2^k) \) (or \( r_j(x) \in GF(2)[x]/ < m_j(x) > \)) at specific channel \( j \) is changed to some random value from the finite field \( GF(2^k) \) (or \( GF(2)[x]/ < m_j(x) > \)), where all values can occur with the same probability.

Therefore, we assume that specific variable at specific channel used in Algorithm 12 and Algorithm 15 in Chapter 6, can be targeted, where an adversary can not target specific point of time, every line of code, iteration of a loop, is hit with the uniform probability. All bits of the targeted variable are affected, i.e., random number of bits, where fault type are random faults. Probability of an implied effect of an induced fault is certain, while duration of an effect is transient, or permanent.

This model is used if the attacker knows that an induced fault at specific channel will set the affected variable to a random value from \( GF(2^k) \) (or \( GF(2)[x]/ < m_j(x) > \)) according to the uniform distribution, or if his fault attack does not depend on some special values that have to appear at some time, or with specific probability. In addition, we can assume following two cases:

- an adversary may be is able to hit a targeted variable at a specific channel in a reasonable small interval of operation, or loop iterations, where this interval is derived from other sources of information;

- the affected variable at a specific channel is not known with certainty, i.e., an adversary may be able to hit a variable from a set of a few different variables.

In practice we have assumed a very weak adversary, where fault induced into memory, or the CPU at different point will leave the attacker with at most information that a certain variable is faulty.

### 3.2.2 Arbitrary Fault Model (AFM)

Here we have assumed that an adversary can target a specific line of code at a specific channel, but no specific variable in that line, i.e., adversary has limited control over induced faults, does not know much about his induced faults to know its type, or error distribution. Therefore, any number of bits can be affected, according to an unknown
probability distribution. Specific point of time can not be targeted, but if the affected line of the code at specific channel is from a loop, then every iteration of the loop is hit with uniform probability. The probability of the implied effect is certain, while duration of an effect is transient, or permanent. In AFM, transient faults on any variable, or operation in the affected line of code at specific channel is the same as if the result of the targeted line of code is changed by some fault at specific channel. In situation of permanent fault, we assume that all variables used in the targeted line at specific channel of code are hit with the same uniform probability. This attack is successful if attacker does not need the assumptions about the distribution of the error value, or does not need to be able to guess the error term to get information. Also, we can assume that an adversary can not target specific line of code at specific channel, but will hit any line at specific channel with known probability.

Mathematically, the effect of an attack using these fault models can be modeled as an addition of an unknown error $e_i \in GF(2^k)$ (or $e_j(x) \in GF(2)[x]/<m_j(x)>$). In case of RFM we assume that a variable $r_j$ at specific channel $j$ is changed to some random value $r_j + e_j$, where $e_j \in GF(2^k)$ (or $r_j(x) \mapsto r_j(x) + e_j(x)$ where $e_j(x) \in GF(2)[x]/<m_j(x)>$) with the same uniform probability, i.e., fault may result in any faulty value, while for AFM if we let $r_i$ be component to which is assigned the result of the affected line of code at specific channel, then the faulty value is $r_i + e_i$, where $e_i \in GF(2^k)$ (or $r_j(x) \mapsto r_j(x) + e_j(x)$ where $e_j(x) \in GF(2)[x]/<m_j(x)>$), and whose probability distribution is arbitrary and unknown.

### 3.2.3 Single bit Fault Model (SFM)

Here we assume a very strong adversary who can target at a specific point of time (or if affected variable is used in the loop, then a specific iteration) a bit of a specific variable used at a specific line of code at specific channel. Also, we can assume that at a specific channel at an unspecified time, a specific variable used in a specific line of code can be targeted, but not a specific bit of the variable. If the affected variable is used in a loop then every iteration is hit with the same probability.

Mathematically, the effect of an attack using these fault models can be modeled as an
addition of a single bit, i.e., \( x \mapsto x + 2^i \), \( 0 \leq i \leq k - 1 \), where in the first case, bit position \( i \) is chosen by an adversary, while in second case we assume that is chosen according to the uniform distribution.

### 3.3 Conclusion

In this chapter we have presented the mathematical aspect of fault attacks through fault models, i.e., *Random Fault Model*, *Arbitrary Fault Model* and *Single bit Fault Model*. These fault models have been motivated by real devices and real physical world, where power of an adversary is compared with power of the countermeasures present on the smartcard. *Single bit Fault Model* is the strongest fault model between all. If an adversary is able to target at specific point of time a bit of a specific variable used at specific line of code at specific channel, then he is also able to recover any bit of any other variable used in the algorithm at the specific channel. This fault model may seem unrealistic when one considers the countermeasures of today’s smartcards. Given the countermeasures of today’s smartcards, the most realistic fault model is *Random Fault Model*, and especially *Arbitrary Fault Model*, since countermeasures of today’s smartcards ensure that they are immune against really strong fault models such as *Single bit Fault Model*. The weakest adversary is assumed in the *Arbitrary Fault Model*, since it can be realized by any physical attacks which can be applied with high success probability.

In Chapter 5 and Chapter 6 we propose schemes/algorithms where computation is decomposed into parallel, mutually independent channels and we assume that an adversary can use either RFM, AFM, or SFM per channel. Also, we assume that more channels can be targeted at the same time, where either RFM, AFM, or SFM can be used per different channel.
Chapter 4

Fault Attacks on Elliptic Curve Cryptosystems

Security of elliptic curve cryptosystems is based on the difficulty of the discrete logarithm problem (DLP) in the group of points on an elliptic curve. However, it has been proven that security of cryptosystems does not only depend on the mathematical properties. Side channel attacks provide information which reveals important and compromising details about secret data. Some of these details can be used as a new trapdoor to invert a trapdoor one-way function without the secret key. This allows an adversary to break a cryptographic protocol, even if it has been proved to be secure in the mathematical sense. Specifically, in the case of fault attacks which are active attacks, an adversary has to tamper with an attacked device in order to create faults. E.g., if an adversary can inflict some physical stress on the smartcard, he can induce faults into circuitry or memory, as a result these faults are manifested in computation as errors. Therefore, a faulty final result is computed. Moreover, if the computation depends on some secret key, facts about the secret key can be concluded.

In Sections 4.1 and 4.2 we present the fault attacks given in [16], [27], [4]. These references assume that a faulty result is not on the original elliptic curve with overwhelming probability, and that a faulty result is easily detected by simply checking if the final result is a valid point on the original curve. Otto in [73] pointed out that any attack that yields a faulty result which is a valid point on the original curve, would be undetectable by
their proposed countermeasures, and that those undetectable points can be used for a new attacks. A Sign Change Attack [21] is a fault attack on scalar multiplication which does not change the original curve $E$, and works with points on the curve $E$ over $GF(p)$.

The authors of [21] claim that a sign change attack does not apply to elliptic curves of the characteristic 2, but we will show that sign change faults can be created in the affine addition formula of the non-supersingular elliptic curves over $GF(2^k)$. In Section 4.3 we show that it is possible to create faulty points that are valid points on the original non-supersingular elliptic curve over $GF(2^k)$ by inducing faults into the affine addition formula. In Section 4.4, 4.5 and 4.6 we present an analysis by investigating the possibilities of attacks which induce faults into variables used in the affine addition formula of non-supersingular elliptic curve. We derive conditions that the inflicted error has to satisfy in order to yield an undetectable faulty point, by investigating each variable used in the affine additions. Since our aim is to show that undetectable faulty points for non-supersingular elliptic curves over $GF(2^k)$ can be created and that point validation is not enough, we avoid analysis of the affine doubling formula, and other point representations, since analysis of affine addition proves our claim.

4.1 Differential Fault Attacks on ECC

Biehl et al. [16] extended the idea of differential fault attacks (DFA) on the RSA cryptosystem [23] to schemes using elliptic curves. The basic idea of DFA is the enforcement of bit errors into the decryption, or the signing process which is performed inside the smartcard, so that information on the secret key can leak out. The authors of [16] assume a cryptographically strong elliptic curve which is publicly known, and a secret key $d \in \mathbb{Z}$ which is stored inside a “tamper-proof” device and which is unreadable for outside users. Also they assume that we have access to the “tamper-proof” device such that we can compute $dP$ for arbitrary input points $P$. The common idea in these papers is that by inserting, or by disturbing the representation of a point by means of a random register fault we can force the device to apply its point addition (respectively multiplication) algorithm to a value which is not a point on the given curve, but on some different curve. A
crucial observation is that the result of this computation is a point on the new (probably
cryptographically less strong) curve, which can be exploited to compute $d$. These attacks
work by misusing the “tamper-proof” device to execute its computation steps on group
structures not originally intended by the designer of the cryptosystem. They present three
different types of attacks that can be used to derive information about the secret key, if
bit errors can be inserted into the elliptic curve computation in a “tamper-proof” device:

1. No correctness check for input points - it is assumed that the device does not explicit-
ily check whether an input point $P$, (or the result of the computation) is a point
on the cryptographically strong elliptic curve $E = (a_1, a_2, a_3, a_4, a_6)$ defined as in
(2.1). An adversary chooses the input pair $P = (x, y)$ to the “tamper-proof” device
such that the tuple $(a_1, a_2, a_3, a_4, \tilde{a}_6)$, where $\tilde{a}_6 = y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2 - a_4 x$
defines an elliptic curve $\tilde{E}$, whose order has a small divisor $l$, such that $\text{ord}(\tilde{P}) = l$. The output of the “tamper-proof” device with input $\tilde{P}$ is $d\tilde{P}$ on $\tilde{E}$. By repeating
this procedure with different choices of $\tilde{P}$, $l$ can be computed by use of the Chinese
Remainder Theorem.

2. Placing register faults properly - it is assumed that an adversary can enforce register
faults inside the “tamper-proof” device at some precise moment at the beginning of
the multiplication, after the device checks whether the given input point is a point
in the group of points of the cryptographically strong elliptic curve $E$. Therefore,
the device internally computes with $\tilde{P}$, and if it does not check whether the output
point is point on a $E$, outputs $l\tilde{P}$. We determine $a_6$ such that $l\tilde{P}$ satisfies the curve
equation $\tilde{E} = (a_1, a_2, a_3, a_4, \tilde{a}_6)$. By checking all possible candidates $\tilde{P}$, we find one
that is on $\tilde{E}$.

3. Faults at random moments of the multiplication - here an adversary can introduce
register faults during the computation of an a-priori chosen specific block of multi-
pliers bits, i.e., it is assumed that an adversary can repeatedly input some point $P$
on $E$ into the “tamper-proof” device and enforce a register fault during $m$ successive
iterations of the fast multiplication algorithm. Also, even if one cannot influence
at which block the register fault happens, one can deduce the secret key after an
expected number of polynomially many enforced random register faults.

Similar ideas have been described in [60], where a key recovery attack is presented on the discrete logarithm based protocols working in a prime order subgroup. The attack uses small order subgroups in $\mathbb{Z}_p^*$ to compute part of the secret key in a protocol working in a subgroup of a prime order $q$.

M. Ciet and M. Joy [27] generalize results from [16] by relaxing their assumptions. They analyze the implications of permanent faults (intentional, or accidental) in non-volatile memory, where the system parameters are stored. They consider permanent faults in the representation of point $P$, in the definition of the field $\mathbb{F}_q$, and in the curve parameters. Also, they extend the analysis to transient faults, originating from a perturbation of the reading in non-volatile memory, and resulting in faulty values for the system parameters used in working memory, throughout computation.

1. **Faults in the base point** - Let the point $P = (x, y)$ be a system parameter that is stored in the non-volatile memory of the cryptographic device. It is read from that memory for computation of $dP$. Assume that only the $x$ coordinate of point $P$ is corrupted (or only $y$ is corrupted). The cryptographic device then computes $\tilde{Q} = d\tilde{P}$, where $\tilde{P} = (\tilde{x}, y)$ is unknown, but fixed. It is easy to recover value of $\tilde{P}$ from output value $Q = d(\tilde{x}, y) = (\tilde{x}_d, \tilde{y}_d)$. Point $\tilde{Q}$ defines a curve $\tilde{E}(a_1, a_2, a_3, a_4, \tilde{a}_6)$ with

$$\tilde{a}_6 = \tilde{y}_d^2 + a_1\tilde{x}_d\tilde{y}_d + a_3\tilde{y}_d - \tilde{x}_d^3 - a_2\tilde{x}_d^2 - a_4\tilde{x}_d.$$  

Since $\tilde{P} = (\tilde{x}, y) \in \tilde{E}$, $\tilde{x}$ is a root of the polynomial

$$X^3 + aX^2 + (a_4 - a_1)X + (\tilde{a}_6 - y^2 - a_3y).$$  

By assuming that (4.1) has a unique root $\tilde{x}$, where $l = \text{ord}_{\tilde{E}}(\tilde{P})$ is small enough so that discrete logarithm of $\tilde{Q}$ is computable, the value of $d \mod l$ can be recovered. Otherwise, there are 2, or 3 candidates for $\tilde{x}$, since $\tilde{x}$ is a root of (4.1). In the permanent-fault model it is assumed that only a portion of $x$ is corrupted, so the candidate having the most bits matching those of $x$ is likely to be $\tilde{x}$. In the transient fault model, the whole value of $x$ is likely to be corrupted.

If both coordinates $x, y$ are corrupted, such that $\tilde{P} = (\tilde{x}, \tilde{y})$ is the corresponding point, the output value $\tilde{Q} = d\tilde{P} = (\tilde{x}_d, \tilde{y}_d)$ yields the value $\tilde{a}_6$. We only know that
the point $\tilde{P}$ lies on the curve $\tilde{E}(a_1, a_2, a_3, a_4, \tilde{a}_6)$. Further assumptions are needed to completely recover $\tilde{P}$.

2. **Faults in the definition of the field $GF(2^k)$** - The finite field $GF(2^k)$ is regarded as a quotient $GF(2)[x]/ < f(x) >$, where $f(x)$ is an irreducible polynomial of degree $k$ over $GF(2)$, which is stored in non-volatile memory as a binary string $(a_0, \ldots, a_k)$ corresponding to $f(x) = \sum_{i=0}^{k} a_i X^i$, with $a_k = 1$. A non-singular elliptic curve over $GF(2^k)$ is given by $E: y^2 + xy = x^3 + ax^2 + b$. Assume that there is an error in the representation of $f(x)$, so that computation is performed modulo $\tilde{f}(x) = \sum_{i=0}^{k} \tilde{a}_i X^i$, instead of modulo $f(x)$. The polynomial $\tilde{f}(x)$ can be recovered by observing that

$$\tilde{b} \equiv y^2 + xy + x^3 + ax^2 \equiv \tilde{y}_d^2 + \tilde{x}_d \tilde{y}_d + \tilde{a}_d \tilde{x}_d + ax^2 (mod \ f(x)),$$

where $d\tilde{P} = (\tilde{x}_d, \tilde{y}_d)$ and $\tilde{P} \equiv P (mod \ \tilde{f}(x))$. Let

$$\Delta(X) = y^2 + xy + x^3 + ax^2 + \tilde{y}_d^2 + \tilde{x}_d \tilde{y}_d + \tilde{a}_d \tilde{x}_d + ax^2,$$

then it follows that $\tilde{f}(x)|\Delta(X)$. Given the factorization of $\Delta$, trying all possible combinations yields $\tilde{f}(x)$ as the polynomial whose representation best matches the representation of $f(x)$. In case of permanent fault, scalar multiplication, $Q' = d'P$ with $d' \neq d$, eases the recovery of $\tilde{P}$ as a factor of $gcd(\Delta, \Delta')$ where $\Delta'$ is defined from $Q'$. If the fault is transient, it is always possible to distinguish $\tilde{f}(x)$.

3. **Faults in the curve parameters** - Modification of the parameter $a_6$ does not affect the computation of $dP$, since $a_6$ is not needed in the addition formula. Assume that the parameter $a_4$ is faulty, i.e., $\tilde{a}_4$ is introduced into the computation. Then, computation is performed over the curve $\tilde{E}(a_1, a_2, a_3, \tilde{a}_4, \tilde{a}_6)$. Since, $P = (x, y)$ and $\tilde{Q} = dP = (\tilde{x}_d, \tilde{y}_d)$ lie on the curve $\tilde{E}$, we have the system of equations:

$$\tilde{a}_4 x + \tilde{a}_6 = y^2 + a_1 xy + a_3 y - x^3 - a_2 x^2$$

$$\tilde{a}_4 \tilde{x}_d + \tilde{a}_6 = \tilde{y}_d^2 + a_1 \tilde{x}_d \tilde{y}_d + a_3 \tilde{y}_d - \tilde{x}_d^3 - a_2 \tilde{x}_d^2.$$

After solving system for $\tilde{a}_4$ and $\tilde{a}_6$, we compute the logarithm of $\tilde{Q}$ with respect to $P$ in $< P > \subseteq \tilde{E}(a_1, a_2, a_3, \tilde{a}_4, \tilde{a}_6)$, and get the value $d mod l$, where $l = ord_{\tilde{E}}(P)$.  

55
4.2 Invalid-Curve Attacks

Adrian et al. [4] combined the small subgroup attack of Lim and Lee [60] and DFA of Bieh et al. [16] that are effective on the one-pass ECDH, ECIES, and one-pass ECMQV protocols if the receiver of an elliptic curve point does not verify that the point does indeed lie on the elliptic curve specified by the domain parameters. If $E$ is an elliptic curve defined over $\mathbb{F}_q$ whose equation differs from $E'$'s equation only in the coefficient $a_6$, then the addition laws for $E$ and $E'$ are the same and elliptic curve $E$ is called an invalid curve relative to $E$. The only way to prevent the invalid-curve attack is to check that a received point does indeed lie on the legitimate elliptic curve. Now, we will mention some invalid-curve attacks.

4.2.1 Invalid-Curve Attack on One-Pass ECDH

Suppose that one-pass ECDH is used by entity $A$ to establish a shared secret key $k$ with entity $B$, where $k$ is subsequently used by $B$ to send messages authenticated with a message authentication algorithm (MAC) to $A$. $A$ selects an invalid curve $E$, such that it contains a point $\bar{Q}$ of small order $l$, and sends $\bar{Q}$ to $B$. $B$ computes $Z = k_B \bar{Q}$ and $k = H(x(Z))$. When $B$ sends $A$ a message $m$ and its tag $t = MAC_k(m)$, $A$ can determine the correct $Z$, by finding $Z' \in \langle \bar{Q} \rangle$ satisfying $t = MAC_{k'}(m)$ where $k' = H(x(Z'))$. Since $\bar{Q}$ has order $l$, the expected number of trials before $A$ succeeds is $l/2$, where $A$ has determined $k_l \equiv \pm k_B \mod l$. Therefore, $A$ knows that $k_l^2 \equiv k_B^2 \mod l$. By repeating the attack with point $\bar{Q}$ with pairwise relatively prime orders, entity $A$ can recover $z = k_B^2 \mod n$ for $N > n^2$ by the Chinese Remainder Theorem. Since $k_B^2 < n^2 < N$ we have that $z = k_B^2$, and $A$ can compute $k_B = \sqrt{z}$, while $B$ is unaware that the attack has taken the place.

4.2.2 Invalid-Curve Attack on ECIES

Entity $A$ selects a point $\bar{Q}$ of order $l$ on an invalid curve $E$. Entity $A$ makes a guess $k_l \in [0, l - 1]$ for $k_B \mod l$ and computes $Z = k_l \bar{Q}$. $A$ transmits $\bar{Q}$ (instead of $R_A$) to $B$, who computes $Z' = k_B \bar{Q}$, with overwhelming probability that key $k_B'$ derived from $Z'$
satisfies \( t = MAC_k(c) \) if and only if \( k_i \equiv \pm k_B \mod l \). If \( A \) is able to determine whether or not \( B \) accepts the chipertext, then \( A \) can be expected to determine \( \pm k_B \mod l \), after about \( l/4 \) iterations on the average. The attack can be repeated to recover \( k_B \). Here, the victim \( B \) may be aware that an invalid-curve attack has been lunched on the ECIES, if he notices that he is receiving many invalid ciphertexts from \( A \).

The attacks presented in [16], [27] and [4] assume that with overwhelming probability a faulty result is not on the original elliptic curve with overwhelming probability, and if that the smartcard checks if the final result is a valid point on the original curve then the faulty point is captured. By this argument, any attack that yields a faulty result which is a valid point on the original curve, would be undetectable by their proposed countermeasures. Therefore, those undetectable points can be used for new attacks. In the next section we demonstrate that it is possible to create faulty points that are valid points of the original non-supersingular elliptic curve over \( GF(2^k) \). We derive necessary conditions for the errors which cause undetectable faulty results. Our analysis is performed through use of computer the algebra system MAPLE.

### 4.3 Point Addition - Undetectable faulty points

A fault induced into the affine point addition formula of non-supersingular elliptic curve over \( GF(2^k) \) might be useful to recover secret data. We assume that the error induced into variable \( x \) can be written as a sum \( x + e, e \in GF(2^k) \), where \( e \) is random variable, whose distribution depends on a chosen fault model. We will consider permanent and transient faults, since some variables are used more than once. Let \( P_1, P_2 \) be two points on non-supersingular elliptic curve \( E \) over \( GF(2^k) \) defined by

\[
E : y^2 + xy = x^3 + ax^2 + b.
\]
We assume that $P_1 \neq P_2$ in point addition. This implies that $P_3 = (x_3, y_3) = P_1 + P_2$ is computed as:

$$
\begin{align*}
    x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a \\
    y_3 &= \lambda(x_1 + x_3) + x_3 + y_1, \quad \text{where} \\
    \lambda &= \frac{y_1 + y_2}{x_1 + x_2}.
\end{align*}
$$

(4.2)

Also, let $P = (x, y) \in GF(2^k)$, then $-P = (x, x + y)$ is the negative of the point $P$.

The variables that are used are the point coordinates $x_1, y_1, x_2, y_2$, the variable $\lambda$ and the output coordinates $x_3, y_3$, and each of them can be targeted by fault attacks. We describe in detail the conditions required for an error to be undetectable.

### 4.4 Attacks targeting $\lambda = (y_1 + y_2)/(x_1 + x_2)$

The parameter $\lambda$ is computed using Equation (4.2). We investigate faults in parameters $y_1, y_2, x_1, x_2$.

**Faults induced into $y_2$**

Assume that an adversary induces faults into $y_2$, i.e., $y_2 \mapsto y_2 + e, e \in GF(2^k)$. Since $y_2$ is only used once, we have that permanent and transient fault have same effect and do not need to be considered separately. The induced fault forces faulty values $\tilde{\lambda}, \tilde{x}_3, \tilde{y}_3$ in the computation, i.e.,

$$
\begin{align*}
    \tilde{\lambda} &= \frac{y_1 + y_2 + e}{x_1 + x_2} = \frac{y_1 + y_2}{x_1 + x_2} + \tilde{e} = \lambda + \tilde{e}, \quad \text{where} \quad \tilde{e} = \frac{e}{x_1 + x_2}, \quad x_1 \neq x_2, \quad (4.3) \\
    \tilde{x}_3 &= \tilde{\lambda}^2 + \tilde{\lambda} + x_1 + x_2 + a = (\lambda + \tilde{e})^2 + (\lambda + \tilde{e}) + x_1 + x_2 + a \\
    &= x_3 + \tilde{e}((\tilde{e} + 1), \quad (4.4) \\
    \tilde{y}_3 &= \tilde{\lambda}(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = (\lambda + \tilde{e})(x_1 + x_3 + \tilde{e}(e + 1)) + x_3 + \tilde{e}(e + 1) + y_1 \\
    &= \lambda(x_1 + x_3) + \lambda\tilde{e}(e + 1) + \tilde{e}(x_1 + x_3) + \tilde{e}^2(e + 1) + x_3 + \tilde{e}(\tilde{e} + 1) + y_1 \\
    &= y_3 + \tilde{e}((\tilde{e} + 1)(\lambda + \tilde{e} + 1) + x_1 + x_3). \quad (4.5)
\end{align*}
$$
A faulty point \( \tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3) \) is a valid faulty point only if
\[
\tilde{y}_3^2 + \tilde{x}_3 \tilde{y}_3 + \tilde{x}_3^2 + a \tilde{x}_3^2 + b = 0
\]
over \( GF(2^k) \). Therefore,
\[
\tilde{c} (x_1 + x_2) \left( \tilde{c} + \frac{x_2}{x_1 + x_2} \right) \left( \tilde{c}^2 + \tilde{c} + \frac{y_1 x_2 + x_1 y_2 + x_3^2 + x_1^2 x_2}{(x_1 + x_2)^2} \right) + T = 0,
\]
(4.6)
where
\[
T = y_3^2 + x_3 y_3 + x_3^2 + a x_3^2 + b + \frac{x_1 (x_3^2 + x_1^2 a + x_1 y_1 + y_1^2 + a_2 x_2^2 + y_2^2 + x_2^3 + x_2 y_2)}{(x_1 + x_2)^2} \tilde{c}.
\]
(4.7)
Since, \( P_1, P_2 \) are valid points, and the correct result \( P_3 \) is a valid point on the elliptic curve, by applying the Weierstrass equation
\[
y_i^2 + x_i y_i = x_i^3 + x_i^2 a + b, \quad i \in \{1, 2, 3\}
\]
we have that \( T = 0 \). Therefore,
\[
\tilde{c} (x_1 + x_2) \left( \tilde{c} + \frac{x_2}{x_1 + x_2} \right) \left( \tilde{c}^2 + \tilde{c} + \frac{y_1 x_2 + x_1 y_2 + x_3^2 + x_1^2 x_2}{(x_1 + x_2)^2} \right) = 0.
\]
(4.8)
It follows that
\[
\tilde{c} = 0 \Rightarrow e = 0,
\]
\[
\tilde{c} = \frac{x_2}{x_1 + x_2} \Rightarrow e = x_2.
\]
The first value \( e = 0 \) represents error free computation, and it can be neglected. To solve
\[
\tilde{c}^2 + \tilde{c} + \frac{y_1 x_2 + x_1 y_2 + x_3^3 + x_1^2 x_2}{(x_1 + x_2)^2} = 0,
\]
(4.9)
we will use formulas for the solution of quadratic equations over \( GF(2^k) \) given in [25].

Let
\[
x^2 + x + m = 0, \quad m \in GF(2^k),
\]
(4.10)
and let \( T_4(m) \) be trace of \( m \) for \( k \) even such that
\[
T_4(m) = \sum_{i=0}^{(k-2)/2} m^{2i}, \quad \text{then}
\]

**Theorem 4.4.1** ([25]). Assume that (4.10) has a solution \( x_1 \) in \( GF(2^k) \), and \( k \) is odd, then solution can be expressed as
\[
x_1 = \sum_{j \in J} m^{2j}
\]
\[
= \sum_{i \in I} m^{2i},
\]
where \( I = \{1, 3, 5, \ldots, k-2\} \), \( J = \{0, 2, 4, \ldots, k-1\} \).
Theorem 4.4.2 ([25]). Assume that (4.10) has a solution $x_1$ in $GF(2^k)$ and $k \equiv 2 \pmod{4}$, then $x_1$ can be expressed as follows:

$$x_1 = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \quad \text{for} \quad T_4(m) = 0,$$

$$x_1 = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \quad \text{for} \quad T_4(m) = 1,$$

where $\alpha^2 + \alpha = 1$.

Theorem 4.4.3 ([25]). If (4.10) has a solution $x_1$ in $GF(2^k)$, $k \equiv 0 \pmod{4}$, and $T_4(m) = 1$, then $x_1$ can be expressed as

$$x_1 = S + S^2 + m^{2k-1} \left(1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}}\right),$$

where

$$S = \sum_{j=1}^{(k/4)-1} \sum_{i=j}^{(k/4)-1} m^{2^{2i-1+k/2+2j-2}}. \quad (4.11)$$

Therefore, let $m = \frac{y_1x_2 + x_1y_2 + x_1^2x_2}{(x_1 + x_2)^2}$, $x_1 \neq x_2$ in (4.9), then depending on the degree $k$ of the finite field we have

- if $k$ is odd then

  $$\tilde{e} = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}. \quad (4.12)$$

  Therefore, the faulty point is valid if

  $$e = (x_1 + x_2) \sum_{i \in I} m^{2i}. \quad (4.13)$$

- if $k \equiv 2 \pmod{4}$ then

  - if $T_4(m) = 0$ then

    $$\tilde{e} = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}. \quad (4.14)$$

    Therefore, the faulty point is valid if

    $$e = (x_1 + x_2)^{\frac{(k-6)/4}{2}} \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}. \quad (4.15)$$
if \( T_4(m) = 1 \) and \( \alpha^2 + \alpha = 1 \) then
\[
\tilde{e} = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}}.
\] (4.16)

The faulty point is valid if
\[
e = (x_1 + x_2) \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}} \right).
\] (4.17)

- if \( k \equiv 0 \pmod{4}, T_4(m) = 1 \) and where \( S \) is as in (4.11) then
\[
\tilde{e} = S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right).
\] (4.18)

The faulty point is valid if
\[
e = (x_1 + x_2) \left( S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right).
\] (4.19)

Therefore,
\[
\tilde{e} \in \left\{ 0, \frac{x_2}{x_1 + x_2}, e^* \right\},
\]
where depending on the field degree \( k \), \( e^* \) can be as in (4.12), (4.14), (4.16), (4.18). Moreover,
\[
e \in \{ 0, x_2, e^* \},
\]
where depending on \( k \), \( e^* \) can be as in (4.13), (4.15), (4.17), (4.19).

The value \( e = x_2 \) represents the case when \( -P_2 = (x_2, y_2 + x_2) \) is used instead of \( P_2 = (x_2, y_2) \). Also, \( e^* \) has value that depends on the degree \( k \) of the finite binary extension field.

**Faults induced into \( y_1 \)**

Assume that an adversary induces faults into \( y_1 \), i.e., \( y_1 \mapsto y_1 + e, e \in GF(2^k) \). Since \( y_1 \) is used in computation of \( y_3 \) we need to investigate permanent and transient faults.
Transient Faults. Fault induced into \( y_1 \) yields exactly same values for \( \tilde{\lambda}, \tilde{x}_3, \tilde{y}_3 \), as when it is induced into \( y_2 \), i.e., see (4.3), (4.4), (4.5). Therefore,

\[
\tilde{e} \in \left\{ 0, \frac{x_2}{x_1 + x_2}, \tilde{e}^* \right\},
\]

where depending on the field degree \( k \), \( \tilde{e}^* \) can be as in (4.12), (4.14), (4.16), (4.18). Moreover,

\[
e \in \left\{ 0, x_2, e^* \right\},
\]

where depending on \( k \), \( e^* \) can be as in (4.13), (4.15), (4.17), (4.19). Therefore, the induced transient fault in \( y_1 \) which produces a valid faulty point depends on the value \( x_2 \), and value \( e^* \), which depends on the degree \( k \) of the finite binary extension field.

Permanent Faults. In case of a permanent fault induced into \( y_1 \), the computation of \( \tilde{\lambda} \) and \( \tilde{x}_3 \) is same as for when fault is induced into \( y_2 \), see (4.3), (4.4), and

\[
\tilde{y}_3 = \tilde{\lambda}(x_1 + \tilde{x}_3) + \tilde{x}_3 + \tilde{y}_1 = (\lambda + \tilde{e}) (x_1 + x_3 + \tilde{e} (\tilde{e} + 1) + x_3 + \tilde{e} (\tilde{e} + 1) + y_1 + e
\]

\[
= y_3 + \tilde{e}(\tilde{e} + 1) (\lambda + \tilde{e} + 1) + x_3 + x_2.
\]

A faulty point \( \tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3) \) is a valid faulty point only if \( \tilde{y}_3^2 + \tilde{x}_3 \tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0 \) over \( GF(2^k) \). Therefore,

\[
\tilde{e} (x_1 + x_2) \left( \tilde{e} + \frac{x_1}{x_1 + x_2} \right) \left( \tilde{e}^2 + \tilde{e} + \frac{y_1 x_2 + x_1 y_2 + x_3^3 + x_1 x_2^2}{(x_1 + x_2)^2} \right) + T = 0,
\]

where

\[
T = y_3^2 + x_3 y_3 + x_3^3 + a x_3^2 + b + \frac{x_2 (x_3^3 + x_1^2 a + x_1^2 y_1 + x_1 y_x + a x_2^2 + y_2^2 + x_2^3 + x_2 y_2)}{(x_1 + x_2)^2} \tilde{e}. \quad (4.21)
\]

Since, \( P_1, P_2 \) are valid points and the correct result \( P_3 \) is a valid point on the elliptic curve, then by applying the Weierstrass equation \( y_i^2 + x_i y_i = x_i^3 + x_i^2 a + b, \ i \in \{1, 2, 3\} \) we have that \( T = 0 \). Therefore,

\[
\tilde{e} (x_1 + x_2) \left( \tilde{e} + \frac{x_1}{x_1 + x_2} \right) \left( \tilde{e}^2 + \tilde{e} + \frac{y_1 x_2 + x_1 y_2 + x_3^3 + x_1 x_2^2}{(x_1 + x_2)^2} \right) = 0. \quad (4.22)
\]

It follows that

\[
\tilde{e} = 0 \Rightarrow e = 0,
\]

\[
\tilde{e} = \frac{x_1}{x_1 + x_2} \Rightarrow e = x_1.
\]
The first value $e = 0$ represents error free computation, therefore it can be neglected. To solve
\[ \tilde{e}^2 + \tilde{e} + \frac{y_1 x_2 + x_1 y_2 + x_2^3 + x_1 x_2^2}{(x_1 + x_2)^2} = 0 \] (4.23)
we will use formulas for the solution of quadratic equations over $GF(2^k)$ given in [25]. Let $m = \frac{y_1 x_2 + x_1 y_2 + x_2^3 + x_1 x_2^2}{(x_1 + x_2)^2}$, with $x_1 \neq x_2$, then the solution is
\[ \tilde{e} \in \left\{ 0, \frac{x_1}{x_1 + x_2}, e^* \right\}, \]
where depending on the field degree $k$, $e^*$ can be as in (4.12), (4.14), (4.16), (4.18). Moreover,
\[ e \in \{0, x_1, e^*\}, \]
where depending on $k$, $e^*$ can be as in (4.13), (4.15), (4.17), (4.19). Therefore, permanent fault in $y_1$ which produces a valid faulty point depends on the value $x_1$, which represents the case when $-P_1 = (x_1, y_1 + x_1)$ is used instead of $P_1 = (x_1, y_1)$, and value $e^*$, which depends on the degree $k$ of the finite field.

**Fault induced into $x_2$**

Assume that an adversary induces a fault into $x_2$, i.e., $x_2 \mapsto x_2 + e$, $e \in GF(2^k)$. Since $x_2$ is used in computation of $x_3$ we need to investigate permanent and transient faults.

**Transient faults.** Given transient fault we have
\[
\tilde{\lambda} = \frac{y_1 + y_2}{x_1 + x_2 + e} = \lambda + \tilde{e}, \text{ where } \tilde{e} = \frac{e(y_1 + y_2)}{(x_1 + x_2)^2 + e(x_1 + x_2)}; \]
\[
\tilde{x}_3 = (\lambda + \tilde{e})^2 + (\lambda + \tilde{e}) + x_1 + x_2 + a = \lambda^2 + \tilde{e}^2 + \lambda + \tilde{e} + x_1 + x_2 + a
\]
\[= x_3 + \tilde{e}(\tilde{e} + 1), \] (4.25)
\[
\tilde{y}_3 = \tilde{\lambda}(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = (\lambda + \tilde{e}) (x_1 + x_3 + \tilde{e}(\tilde{e} + 1)) + x_3 + \tilde{e}(\tilde{e} + 1) + y_1
\]
\[= y_3 + \tilde{e}((\tilde{e} + 1)(\lambda + \tilde{e} + 1) + x_1 + x_3). \] (4.26)

We assume that $e \neq (x_1 + x_2) \pmod{2}$, since otherwise an arithmetic error will be caused. A faulty point $\tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0$
over \( GF(2^k) \). Therefore,

\[
\bar{e} (x_1 + x_2) \left( \bar{e} + \frac{x_2}{x_1 + x_2} \right) \left( \bar{e}^2 + \bar{e} + \frac{y_1 x_2 + x_1 y_2 + x_1^3 + x_1^2 x_2}{(x_1 + x_2)^2} \right) + T = 0,
\]

where

\[
T = y_3^2 + x_3 y_3 + x_3^3 + a x_3^2 + b + \frac{x_1 \left( x_1^3 + x_1^2 a + x_1 y_1 + y_1^2 + a_2 x_2^2 + y_2^2 + x_2^3 - x_2 y_2 \right)}{(x_1 + x_2)^2} \bar{e}.
\]

Since, \( P_1, P_2 \) are valid points and the correct result \( P_3 \) is a valid point on the elliptic curve, by applying the Weierstrass equation \( y_i^2 + x_i y_i = x_i^3 + x_i^2 a + b, i \in \{1, 2, 3\} \) we have that \( T = 0 \). Therefore,

\[
\bar{e} (x_1 + x_2) \left( \bar{e} + \frac{x_2}{x_1 + x_2} \right) \left( \bar{e}^2 + \bar{e} + \frac{y_1 x_2 + x_1 y_2 + x_1^3 + x_1^2 x_2}{(x_1 + x_2)^2} \right) = 0. \tag{4.27}
\]

Since equation \( (4.27) \) is exactly the same as equation \( (4.6) \), it follows that

\[
\bar{e} \in \left\{ 0, \frac{x_2}{x_1 + x_2}, \bar{e}^* \right\},
\]

where \( x_1 \neq x_2 \). Depending on the field degree \( k \), \( \bar{e}^* \) can be as in \((4.12), (4.14), (4.16), (4.18) \). Moreover,

\[
e \in \left\{ 0, \frac{x_2(x_1 + x_2)}{y_1 + y_2 + x_2}, \bar{e}^* \right\}, \quad y_1 + y_2 + x_2 \neq 0,
\]

where depending on \( k \), \( \bar{e}^* \) can be as one of the following:

- if \( k \) is odd then

\[
e = \frac{(x_1 + x_2)^2 \sum_{i \in I} m^{2i}}{(y_1 + y_2) + (x_1 + x_2) \sum_{i \in I} m^{2i}}, \quad I \in \{1, 3, \ldots, k - 2\},
\]

if \((y_1 + y_2) + (x_1 + x_2) \sum_{i \in I} m^{2i} \neq 0\).

- if \( k \equiv 2 \pmod{4} \) then

\[
e = \frac{(x_1 + x_2)^2 \sum_{i=0}^{(k-6)/4} (m + m^2)^{2i+4i}}{(y_1 + y_2) + (x_1 + x_2) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2i+4i}},
\]

if \((y_1 + y_2) + (x_1 + x_2) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2i+4i} \neq 0\).
- if \( T_4(m) = 1 \) and \( \alpha^2 + \alpha = 1 \) then

\[
e = \frac{(x_1 + x_2)^2(\alpha + \sum_{i=0}^{(k-6)/4}(m + m^2)^2^{2+i/4})}{(y_1 + y_2) + (x_1 + x_2)(\alpha + \sum_{i=0}^{(k-6)/4}(m + m^2)^2^{2+i/4})},
\]

if \((y_1 + y_2) + (x_1 + x_2)(\alpha + \sum_{i=0}^{(k-6)/4}(m + m^2)^2^{2+i/4}) \neq 0\).

- if \( k \equiv 0 \mod 4 \), \( T_4(m) = 1 \) and \( S \) is as in (4.11) then

\[
e = \frac{(x_1 + x_2)^2 \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right)}{(y_1 + y_2) + (x_1 + x_2) \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right)},
\]

if \((y_1 + y_2) + (x_1 + x_2) \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \neq 0\).

**Permanent faults.** In the case of a permanent fault induced into \( x_2 \), the computation of \( \tilde{\lambda}, \tilde{x}_3, \tilde{y}_3 \) is as follows:

\[
\tilde{\lambda} &= \frac{y_1 + y_2}{x_1 + x_2 + e} = \tilde{e}(y_1 + y_2), \text{ where } \tilde{e} = \frac{1}{x_1 + x_2 + e};
\]

\[
\tilde{x}_3 = \tilde{\lambda}^2 + \lambda + x_1 + \tilde{x}_2 + a = (\tilde{e}(y_1 + y_2))^2 + \tilde{e}(y_1 + y_2) + x_1 + x_2 + e + a
\]

\[
= \tilde{e}^2(y_1 + y_2)^2 + \tilde{e}(y_1 + y_2) + \frac{1}{\tilde{e}} + a, \quad e \neq 0,
\]

\[
\tilde{y}_3 = \tilde{\lambda}(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1
\]

\[
= \tilde{e}(y_1 + y_2) \left( x_1 + \tilde{e}^2(y_1 + y_2)^2 + \tilde{e}(y_1 + y_2) + \frac{1}{\tilde{e}} + a \right) + \tilde{e}^2(y_1 + y_2)^2 + \tilde{e}(y_1 + y_2) + \frac{1}{\tilde{e}} + a + y_1, \quad \tilde{e} \neq 0
\]

\[
= \tilde{e}(y_1 + y_2) \left( x_1 + \tilde{e}^2(y_1 + y_2)^2 + a + 1 \right) + y_2 + \frac{1}{\tilde{e}} + a, \quad e \neq 0.
\]

We assume that \( e \neq (x_1 + x_2) \mod 2 \), since otherwise an arithmetic error will be caused.

A faulty point \( \tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3) \) is a valid faulty point only if \( \tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0 \) over \( GF(2^k) \). Therefore,

\[
A\tilde{e}^3 + B\tilde{e}^2 + C\tilde{e} + D = 0, \quad \text{where}
\]

\[
A = y_2^4 + y_1^3 x_1 + y_2^3 x_1 + y_1^2 y_2 x_1 + y_1 y_2^2 x_1 + y_1^4,
\]

\[
B = y_1^3 + y_1 y_2 + y_1^2 x_1 + y_2^2 x_1 + y_2 x_1 + y_1^2,
\]

\[
C = ay_1 x_1 + ay_2 x_1 + y_1 y_2 + y_2^2,
\]

\[
D = y_1 x_1 + y_1 a + y_2 x_1 + y_1^2 + b.
\]
We will solve equation (4.28) by the method proposed in [25]. We will transform the equation (4.28) into quadratic equation and we will apply formulas for solution of quadratic equations over $GF(2^k)$. Therefore, we multiply equation (4.28) by $1/A$, $A \neq 0$, i.e.,

$$
\tilde{e}^3 + \frac{B}{A} \tilde{e}^2 + \frac{C}{A} \tilde{e} + \frac{D}{A} = 0.
$$

(4.29)

Let $a = \frac{B}{A}$, $b = \frac{C}{A}$, $c = \frac{D}{A}$, then (4.29) can be rewritten as

$$
\tilde{e}^3 + ae^2 + be + c = 0.
$$

(4.30)

Now, substitute $a + x(a^2 + b)^{\frac{1}{2}}$ into (4.30), then (4.30) is rewritten as

$$
x^3 + x + \frac{ba + c}{(a^2 + b)\sqrt{a^2 + b}} = 0, \quad (a^2 + b)\sqrt{a^2 + b} \neq 0.
$$

(4.31)

Let $l = \frac{ba + c}{(a^2 + b)\sqrt{a^2 + b}}$, then (4.31) can be written as

$$
x^3 + x + l = 0.
$$

(4.32)

Now, substitute $x = \omega + \frac{1}{\omega}$, $\omega \neq 0$ in (4.32) then

$$
\frac{\omega^6 + 1 + l\omega^3}{\omega^3} = 0, \quad \omega^3 \neq 0.
$$

(4.33)

Let $\omega^3 = z$ in (4.33) then

$$
\frac{z^2 + 1 + lz}{z} = 0 \quad \Rightarrow \quad z^2 + lz + 1 = 0.
$$

(4.34)

Let $t = z/l$ in (4.34) then

$$
t^2 + t + \frac{1}{l^2} = 0, \quad t^2 \neq 0.
$$

(4.35)

Let $m = 1/l^2$, then depending on the field degree $k$ solution of the quadratic equation (4.35) is as follows: if $k$ is odd then

$$
t = \sum_{i \in I} (m)^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\}.
$$

Therefore,

$$
z = l \sum_{i \in I} (m)^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\},
$$

$$
\omega = \left( l \sum_{i \in I} (m)^{2i} \right)^{1/3}, \quad I \in \{1, 3, 5, \ldots, k - 2\},
$$

66
\[ x = \left( l \sum_{i \in I} (m_i)^2 \right)^{1/3} + \frac{1}{\left( l \sum_{i \in I} (m_i)^2 \right)^{1/3}}, \quad I \in \{1, \ldots, n-2\}, \quad \left( l \sum_{i \in I} (m_i)^2 \right)^{1/3} \neq 0, \]

\[ \overline{e} = a + (a^2 + b)^{1/2} \left( \left( l \sum_{i \in I} (m_i)^2 \right)^{1/3} + \frac{1}{\left( l \sum_{i \in I} (m_i)^2 \right)^{1/3}} \right), \quad I \in \{1, 3, 5, \ldots, n-2\}. \]

The faulty point is valid if

\[ e = \frac{1}{a + (a^2 + b)^{1/2} \left( \left( l \sum_{i \in I} (m_i)^2 \right)^{1/3} + \frac{1}{\left( l \sum_{i \in I} (m_i)^2 \right)^{1/3}} \right)} \cdot (x_1 + x_2), \quad I \in \{1, \ldots, n-2\}, \]

if \( a + (a^2 + b)^{1/2} \left( \left( l \sum_{i \in I} (m_i)^2 \right)^{1/3} + \frac{1}{\left( l \sum_{i \in I} (m_i)^2 \right)^{1/3}} \right) \neq 0. \]

If \( k \equiv 2 \pmod{4} \) then

- if \( T_4(m) = 0 \) then

\[ t = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}. \]

Therefore,

\[ z = l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}, \]

\[ \omega = \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3}, \]

\[ x = \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3} + \frac{1}{\left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3}}, \]

if \( \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3} \neq 0. \) Now it follows

\[ \overline{e} = a + (a^2 + b)^{1/2} \left( \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3} + \frac{1}{\left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3}} \right). \]
The faulty point is valid if
\[
e = \frac{1}{a + (a^2 + b)^{1/2} \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3} + \left( l \sum_{i=0}^{(k-6)/4} \frac{1}{(m + m^2)^{2^{2+4i}}} \right)^{1/3}} - (x_1 + x_2),
\]
and if \( \left( l \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right)^{1/3} + \left( l \sum_{i=0}^{(k-6)/4} \frac{1}{(m + m^2)^{2^{2+4i}}} \right)^{1/3} \neq 0. \)

- if \( T_4(m) = 1 \) and where \( \alpha^2 + \alpha = 1 \), then
\[
t = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}.
\]

Therefore,
\[
z = l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right),
\]
\[
\omega = \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3},
\]
\[
x = \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3} + \frac{1}{\left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3}},
\]
where \( \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3} \neq 0. \)

Now it follows
\[
\tilde{e} = a + (a^2 + b)^{1/2} \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3} + \frac{1}{\left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3}}.
\]

The faulty value is valid if
\[
e = \frac{1}{a + (a^2 + b)^{1/2} \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3} + \left( l \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right) \right)^{1/3}} + (x_1 + x_2),
\]
and if denominator is different than 0.
If \( k \equiv 0 \pmod{4} \) and \( S \) is as in (4.11), then
\[
    t = S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right).
\]

Therefore,
\[
    z = l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right),
\]
\[
    \omega = \left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{1/3},
\]
\[
    x = \frac{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{2/3} + 1}{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{1/3}},
\]
if denominator is different than 0. Now it follows
\[
    \tilde{e} = a + (a^2 + b)^{1/2} \left( \frac{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{2/3} + 1}{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{1/3}} \right).
\]

The faulty point is valid if
\[
    e = \frac{1}{a + (a^2 + b)^{1/2} \left( \frac{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{2/3} + 1}{\left( l \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right) \right)^{1/3}} \right)} + (x_1 + x_2), \quad (4.40)
\]
and if denominator is different than 0.

**Fault induced into \( x_1 \)**

Assume that an adversary induces a fault into \( x_1 \), i.e., \( x_1 \mapsto x_1 + e \), \( e \in GF(2^k) \). Since \( x_1 \) is used in computation of \( x_3 \) and \( y_3 \), we need to investigate permanent and transient faults.

**Transient faults.** Given a transient fault in \( x_1 \) the computed faulty values \( \tilde{x}, \tilde{x}_3, \tilde{y}_3 \) are same as in (4.24), (4.25), (4.26). Therefore, it follows exactly the same analysis as for the case when transient fault is induced into \( x_2 \).
Permanent faults. In case of permanent fault induced into $x_2$, the computation of $\tilde{\lambda}, \tilde{x}_3, \tilde{y}_3$ is as follows:

\[
\begin{align*}
\tilde{\lambda} &= \frac{y_1 + y_2}{x_1 + x_2 + e} = \tilde{e}(y_1 + y_2) \quad \text{where} \quad \tilde{e} = \frac{1}{x_1 + x_2 + e}, \\
\tilde{x}_3 &= \tilde{\lambda}^2 + \tilde{\lambda} + \tilde{x}_1 + x_2 + a = (\tilde{e}(y_1 + y_2))^2 + \tilde{e}(y_1 + y_2) + x_1 + e + x_2 + a \\
&= \tilde{e}^2(y_1 + y_2)^2 + \tilde{e}(y_1 + y_2) + \frac{1}{\tilde{e}} + a, \quad \tilde{e} \neq 0, \\
\tilde{y}_3 &= \tilde{\lambda}(\tilde{x}_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 \\
&= \tilde{e}(y_1 + y_2) \left( x_1 + e + \tilde{e}^2(y_1 + y_2)^2 + \tilde{e}(y_1 + y_2) + \frac{1}{\tilde{e}} + a \right) + \tilde{e}^2(y_1 + y_2)^2 + \\
&= \tilde{e}(y_1 + y_2) \left( x_2 + \tilde{e}^2(y_1 + y_2)^2 + a + 1 \right) + \frac{1}{\tilde{e}} + a + y_1, \quad \tilde{e} \neq 0.
\end{align*}
\]

We assume that $e \neq (x_1 + x_2) \pmod{2}$, since otherwise an arithmetic error will be caused. Faulty point $\tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + ax_3^2 + b = 0$ over $GF(2^k)$. Therefore,

\[
A\tilde{e}^3 + B\tilde{e}^2 + C\tilde{e} + D = 0, \quad \text{where} \quad (4.41)
\]

\[
\begin{align*}
A &= y_1^2y_2x_2 + y_1y_2^2x_2 + y_1^3x_2 + y_2^3x_2 + y_1^4 + y_1^4, \\
B &= y_2^3 + y_1y_2^2 + y_1^2x_2 + y_2^2x_2 + y_2^2x_2 + y_1^2x_2, \\
C &= ay_1x_2 + ay_2x_2 + y_1^2 + y_1y_2, \\
D &= y_1x_2 + y_2x_2 + y_2a + y_2^2 + b.
\end{align*}
\]

The arguments which one used to solve equation (4.41) is exactly the same as for equation (4.28). Therefore, depending on the degree of the finite field $k$, the error value $e \in GF(2^k)$ can be as (4.37), or (4.38), or (4.39), or (4.40).

### 4.5 Attacks targeting $x_3 = \lambda^2 + \lambda + x_1 + x_2 + a$

Here we are going to investigate faults in $x_1, x_2, a$ and $\lambda$. The values $x_2, a$ are not used in computation of $y_3$. Therefore, transient and permanent faults have the same effect,
while values $\lambda, x_1$ and $x_3$, are used again and transient and permanent faults have to be investigated separately. Faults induced into $x_3$ are analyzed later in Section 4.6.

**Fault induced into $x_1$**

Assume that an adversary induces a fault into $x_1$, i.e., $x_1 \mapsto x_1 + e$, $e \in GF(2^k)$. Since $x_1$ is used in the computation of $y_3$, we need to investigate permanent and transient faults.

**Transient faults.** Given a transient fault into $x_1$ the following values are computed:

$$\tilde{x}_3 = \lambda^2 + \lambda + x_1 + e + x_2 + a = x_3 + e,$$

$$\tilde{y}_3 = \lambda(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = \lambda(x_1 + x_3 + e) + x_3 + e + y_1 = y_3 + e(\lambda + 1).$$

A faulty point $\tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0$ over $GF(2^k)$. Therefore,

$$e(x_3(\lambda + 1 + y_3 + x_3^2) + e((\lambda + 1)^2 + \lambda + 1 + x_3 + a) + e^2) + T = 0,$$

where $T = y_3^2 + x_3y_3 + x_3^3 + ax_3^2 + b$. Since, $P_1, P_2$ are valid points and correct result $P_3$ is a valid point on the elliptic curve, by applying the Weierstrass equation $y_3^2 + x_3y_3 = x_3^3 + x_3^2a + b$ we have that $T = 0$. Therefore,

$$e(x_3(\lambda + 1 + y_3 + x_3^2) + e((\lambda + 1)^2 + \lambda + 1 + x_3 + a) + e^2) = 0.$$

The first solution $e = 0$ can be neglected, since it represents error free computation, while the other follows as a solution of the quadratic equation

$$e^2 + e((\lambda + 1)^2 + \lambda + 1 + x_3 + a) + x_3(\lambda + 1 + y_3 + x_3^2) = 0. \quad (4.42)$$

Now, let $y = \frac{x_3(\lambda + 1) + y_3 + x_3^2}{((\lambda + 1)^2 + \lambda + 1 + x_3 + a)^2}$, $(\lambda + 1)^2 + \lambda + 1 + x_3 + a \neq 0$, $x_1 \neq x_2$, then (4.42) can be written as

$$y^2 + y + \frac{x_3(\lambda + 1) + y_3 + x_3^2}{((\lambda + 1)^2 + \lambda + 1 + x_3 + a)^2} = 0. \quad (4.43)$$

Equation (4.43) can be solved by using formulas for solutions of quadratic equations over $GF(2^k)$ given in [25]. Let $m = \frac{x_3(\lambda + 1) + y_3 + x_3^2}{((\lambda + 1)^2 + \lambda + 1 + x_3 + a)}$; then depending on the degree $k$ of the finite field, the possible error values are:
• if $k$ is odd then

\[ y = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}, \quad \text{such that} \]

\[ e = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}. \]

• if $k \equiv 2 \pmod{4}$ then

  - if $T_4(m) = 0$ then

    \[ y = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}, \quad \text{such that} \]

    \[ e = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}}. \]

  - if $T_4(m) = 1$ and $\alpha^2 + \alpha = 1$ then

    \[ y = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \quad \text{such that} \]

    \[ e = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+4i}} \right). \]

• if $k \equiv 0 \pmod{4}$ and $T_4(m) = 1$ with $S$ as in (4.11), then

\[ y = S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \quad \text{such that} \]

\[ e = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right). \]

**Permanent faults.** In the case of a permanent fault induced into $x_1$ the computation of the value $\tilde{x}_3$ is the same as when transient fault is induced into $x_1$, while

\[ \tilde{y}_3 = \lambda(\tilde{x}_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = y_3 + e. \]

A faulty point $\tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0$ over $GF(2^k)$. Therefore,

\[ e \left( y_3 + x_3 + e(x_3 + a) + e^2 \right) + T = 0 \]
where

\[ T = y_3^2 + x_3y_3 + x_3^3 + ax_3^2 + b. \]

Since, \( P_1, P_2 \) are valid points and the correct result \( P_3 \) is a valid point on the elliptic curve, by applying the Weierstrass equation \( y_3^2 + x_3y_3 = x_3^3 + x_3^2a + b \) we have that \( T = 0 \). Therefore,

\[ e (y_3 + x_3 + x_3^2 + e (x_3 + a) + e^2) = 0. \]

The first solution \( e = 0 \) can be neglected, since it represents error free computation, while other follows as a solution of the quadratic equation

\[ e^2 + e (x_3 + a) + y_3 + x_3 + x_3^2 = 0. \quad (4.44) \]

Now, let \( y = \frac{e}{x_3+a} \), \( x_3 + a \neq 0 \), then the equation (4.44) can be written as

\[ y^2 + y + \frac{y_3 + x_3 + x_3^2}{(x_3 + a)^2} = 0. \quad (4.45) \]

The equation (4.45) can be solved by using formulas for solutions of quadratic equations over \( GF(2^k) \) given in [25]. Let \( m = \frac{y_3+x_3+x_3^2}{(x_3+a)^2} \), then depending on the degree \( k \) of the finite field possible error values are:

- if \( k \) is odd then

\[ y = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}, \quad \text{such that} \]

\[ e = (x_3 + a) \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}. \]

- if \( k \equiv 2 \mod 4 \) then

  - if \( T_4(m) = 0 \) then

    \[ y = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{i+4i}}, \quad \text{such that} \]

    \[ e = (x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{i+4i}}. \]
- if \( T_4(m) = 1 \) and \( \alpha^2 + \alpha = 1 \) then
\[
y = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i+4i}} \quad \text{such that} \quad e = (x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i+4i}} \right).
\]

- if \( k \equiv 0 \,(\text{mod} \,4) \) and \( T_4(m) = 1 \) with \( S \) as in (4.11), then
\[
y = S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \quad \text{such that} \quad e = (x_3 + a) \left( S + S^2 + m^{2k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2i+k/2} \right) \right).
\]

**Fault induced into \( x_2 \) (or \( a \), or \( \lambda^2 \))**

If adversary induces faults into \( x_2 \mapsto x_2 + e, (a \mapsto a + e, \text{or } \lambda^2 \mapsto \lambda^2 + e), e \in GF(2^k) \), we have the same situation as when a transient fault is induced into \( x_1 \). Since \( x_2 \) (or \( a \), or \( \lambda^2 \)) it is not used again after computation of \( x_3 \), we do not need to investigate transient and permanent faults separately. Therefore, for
\[
m = \frac{x_3(\lambda+1)+y_3+x_3^2}{(\lambda+1)^2+\lambda+1+x_3+a},
\]
\((\lambda + 1)^2 + \lambda + 1 + x_3 + a)^2, x_1 \neq x_2,\) depending on the degree \( k \) of the finite field possible error values are:

- if \( k \) is odd then
\[
y = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}, \quad \text{such that} \quad e = (\lambda + 1)^2 + \lambda + 1 + x_3 + a \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k-2\}.
\]

- if \( k \equiv 2 \,(\text{mod} \,4) \) then
  - if \( T_4(m) = 0 \) then
\[
y = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i+4i}} \quad \text{such that} \quad e = (\lambda + 1)^2 + \lambda + 1 + x_3 + a \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i+4i}}.
\]

74
\[ y = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^i+4i} \quad \text{such that} \]
\[ e = \left( (\lambda + 1)^2 + \lambda + 1 + x_3 + a \right) \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^i+4i} \right). \]

- if \( k \equiv 0 \mod 4 \) and \( T_4(m) = 1 \) with \( S \) as in (4.11), then
\[ y = S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \quad \text{such that} \]
\[ e = \left( (\lambda + 1)^2 + \lambda + 1 + x_3 + a \right) \left( S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right). \]

**Faults induced into \( \lambda \)**

Assume that an adversary induces a fault into \( \lambda \mapsto \lambda + e, e \in GF(2^k) \), during the computation of the \( x_3 \). Since, it is used in the computation of the \( y_3 \) we need to consider permanent and transient faults separately.

**Transient faults.** Assume that an adversary induces a transient fault, then:
\[
\tilde{x}_3 = (\lambda + e)^2 + \lambda + e + x_1 + x_2 + a = x_3 + e(e + 1), \quad (4.46)
\]
\[
\tilde{y}_3 = \lambda(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = \lambda(x_1 + x_3 + e(e + 1)) + x_3 + e(e + 1) + y_1
\]
\[
= y_3 + e(e + 1)(\lambda + 1). \quad (4.47)
\]

Let \( \tilde{e} = e(e + 1) \), then (4.46) can be rewritten as:
\[
\tilde{x}_3 = x_3 + \tilde{e},
\]
\[
\tilde{y}_3 = y_3 + \tilde{e}(\lambda + 1).
\]

A faulty point \( \tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3) \) is a valid faulty point only if \( \tilde{y}_3^2 + \tilde{x}_3 \tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0 \) over \( GF(2^k) \). Therefore,
\[
\tilde{e}(x_3(\lambda + 1) + y_3 + x_3^2 + \tilde{e}(\lambda + 1)^2 + \lambda + 1 + x_3 + a) + \tilde{e}^2 \right) + T = 0,
\]

75
where

\[ T = y_3^2 + x_3 y_3 + x_3^3 + ax_3^2 + b. \]

Since, \( P_1, P_2 \) are valid points and the correct result \( P_3 \) is a valid point on the elliptic curve, then by applying the Weierstrass equation

\[ y_3^2 + x_3 y_3 = x_3^3 + x_3^2 a + b \]

we have that \( T = 0 \).

Therefore,

\[ \tilde{e}(x_3 (\lambda + 1) + y_3 + x_3^2 + \tilde{e}((\lambda + 1)^2 + \lambda + 1 + x_3 + a) + \tilde{e}^2) = 0. \]

Therefore, \( \tilde{e} = 0 \), i.e., \( e = 0 \), \( e = 1 \). The solution \( \tilde{e} = 0 \) can be neglected, since it represents error free computation, while other follows as a solution of the quadratic equation

\[ \tilde{e}^2 + \tilde{e}((\lambda + 1)^2 + \lambda + 1 + x_3 + a) + x_3 (\lambda + 1) + y_3 + x_3^2 = 0. \quad (4.48) \]

Now, let \( y = \frac{\tilde{e}}{(\lambda + 1)^2 + \lambda + 1 + x_3 + a} \), \( (\lambda + 1)^2 + \lambda + 1 + x_3 + a \neq 0 \), \( x_1 \neq x_2 \), then the equation (4.48) can be written as

\[ y^2 + y + \frac{x_3(\lambda + 1) + y_3 + x_3^2}{(\lambda + 1)^2 + \lambda + 1 + x_3 + a} = 0. \quad (4.49) \]

Equation (4.49) can be solved by using formulas for solutions of quadratic equations over \( GF(2^k) \) given in [25]. Let \( m = \frac{x_3(\lambda + 1) + y_3 + x_3^2}{(\lambda + 1)^2 + \lambda + 1 + x_3 + a} \), then depending on the degree \( k \) of the finite field possible error values are:

- if \( k \) is odd then

\[ y = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\}. \]

Therefore,

\[ \tilde{e} = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\}. \]

Moreover,

\[ ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i \in I} m^{2i} = e(e + 1), \quad I \in \{1, 3, 5, \ldots, k - 2\}, \]

\[ e^2 + e + ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i \in I} m^{2i} = 0. \quad (4.50) \]

Let \( l = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i \in I} m^{2i} \) then (4.50) can be rewritten as

\[ e^2 + e + l = 0. \quad (4.51) \]

Depending on the degree \( k \) of the finite field it follows
- if $k$ is odd then
  \[ e = \sum_{i \in l} l^{2i} \quad I \in \{1, 3, 5, \ldots, k - 2\}. \quad (4.52) \]

- if $k \equiv 2 \ (mod \ 4)$ then
  
  * if $T_4(l) = 0$
    \[ e = \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^{2+i}}, \quad \text{i.e.,} \quad (4.53) \]
  
  * if $T_4(l) = 1$ and $\alpha^2 + \alpha = 1$ then
    \[ e = \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^{2+i}}, \quad \text{i.e.,} \quad (4.54) \]

- if $k \equiv 0 \ mod \ 4, T_4(l) = 1$ and $S$ is as in (4.11) then
  \[ e = S + S^2 + l^{2^{k-1}} \left(1 + \sum_{i=0}^{(k/4)-1} l^{2^{2i+1/2}}\right), \quad \text{i.e.,} \quad (4.55) \]

  - if $k \equiv 2 \ mod \ 4$ then

    - if $T_4(m) = 0$ then
      \[ y = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}}. \]

  Therefore,

  \[ \tilde{e} = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}}. \]

  Moreover,

  \[ ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}} = e(e + 1), \quad \text{i.e.,} \]

  \[ e^2 + e + ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}} = 0. \quad (4.56) \]

  Let $l = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2+i}}$, then (4.56) can be written as (4.51). Depending on the degree $k$ faulty point is valid if solution $e$ is as in (4.52), (4.53), (4.54), (4.55).
if $T_4(l) = 1$ and $\alpha^2 + \alpha = 1$ then

$$y = \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^2+4i}.$$

Therefore,

$$\bar{e} = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^2+4i} \right).$$

Moreover,

$$((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^2+4i} \right) = e(e+1), \text{ i.e.,}$$

$$e^2 + e + ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^2+4i} \right) = 0. \quad (4.57)$$

Let $l = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( \alpha + \sum_{i=0}^{(k-6)/4} (l + l^2)^{2^2+4i} \right)$, then (4.57) can be rewritten as (4.51). Depending on the degree $k$ of the finite field, a faulty point is valid if the solution $e$ is as in (4.52), (4.53), (4.54), (4.55).

- if $k \equiv 0 \pmod{4}$ and $T_4(m) = 1$ and $S$ is as in (4.11) then

$$y = S + S^2 + m^{k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right).$$

Therefore,

$$\bar{e} = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( S + S^2 + m^{k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right), \text{ i.e.,}$$

$$((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( S + S^2 + m^{k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right) = e(e+1), \text{ i.e.,}$$

$$e^2 + e + ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( S + S^2 + m^{k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right) = 0. \quad (4.58)$$

Let $l = ((\lambda + 1)^2 + \lambda + 1 + x_3 + a) \left( S + S^2 + m^{k-1} \left( 1 + \sum_{i=0}^{(k/4)-1} m^{2^{2i+k/2}} \right) \right)$, then (4.58) can be rewritten as (4.51). Depending on the degree $k$ of the finite field, a faulty point is valid if the solution $e$ is as in (4.52), (4.53), (4.54), (4.55).
Permanent faults. In case of a permanent fault induced into $\lambda$, the computation of the value $\tilde{x}_3$ is the same as when transient fault is induced into $\lambda$, while

\[
\tilde{x}_3 = x_3 + e(e + 1),
\]

\[
\tilde{y}_3 = \tilde{\lambda}(x_1 + \tilde{x}_3) + \tilde{x}_3 + y_1 = (\lambda + e)(x_1 + x_3 + e(e + 1)) + x_3 + e(e + 1) + y_1
\]

\[= y_3 + e ((e + 1)(\lambda + e + 1) + x_1 + x_3).\]

A faulty point $\tilde{P}_3 = (\tilde{x}_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0$ over $GF(2^k)$. Therefore,

\[
e(x_1 + x_2)(e + \frac{x_2}{x_1 + x_2})(e^2 + e + \frac{x_1^3 + x_1^2x_2 + y_1x_2 + y_2x_1}{(x_1 + x_2)^2}) + T = 0
\]

where

\[
T = y_3^2 + x_3y_3 + x_3^3 + ax_3^2 + b + \frac{x_1(ax_1^2 + x_1^3 + x_1y_1 + y_1^2 + y_2^2 + ax_2^2 + x_2^3 + x_2y_2)}{(x_1 + x_2)^2}.
\]

Since, $P_1, P_2$ are valid points and correct result $P_3$ is a valid point on the elliptic curve, then by applying the Weierstrass equation $y_3^2 + x_3y_3 = x_3^3 + x_3^2a + b$ we have that $T = 0$. Therefore,

\[
e(x_1 + x_2)(e + \frac{x_2}{x_1 + x_2})(e^2 + e + \frac{x_1^3 + x_1^2x_2 + y_1x_2 + y_2x_1}{(x_1 + x_2)^2}) = 0.
\]

It follows that

\[
e = 0,
\]

\[
e = \frac{x_2}{x_1 + x_2}, \quad x_1 \ne x_2.
\]

The first value $e = 0$ represents error free computation, therefore it can be neglected. Since, equation

\[
e^2 + e + \frac{x_1^3 + x_1^2x_2 + y_1x_2 + y_2x_1}{(x_1 + x_2)^2} = 0 \quad (4.59)
\]

is exactly the same as equation (4.9), the solutions of (4.9) are error values for equation (4.59).
4.6 Fault attacks on $y_3 = \lambda(x_1 + x_3) + x_3 + y_1$

The value of $y_3$ is computed using $\lambda$, $x_1$, $x_3$ and $y_1$. Since, $\lambda, x_1, y_1, y_3$ are not used again in the addition formula we do not need to investigate permanent and transient faults separately, while in case of variable $x_3$ which is returned by the addition formula this distinction is necessary.

Fault induced into $\lambda$

Assume that an adversary induces a fault into $\lambda \mapsto \lambda + e$, $e \in GF(2^k)$ such that the computed faulty value is:

$$\tilde{y}_3 = (\lambda + e)(x_1 + x_3) + x_3 + y_1$$

$$= y_3 + e(x_1 + x_3).$$

A faulty point $P_3 = (x_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^2 + ax_3^2 + b = 0$ over $GF(2^k)$. Therefore,

$$e(x_3(x_1 + x_3) + e(x_1 + x_3)^2) + T = 0,$$

where

$$T = \tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^2 + ax_3^2 + b = 0.$$

Therefore, a valid faulty point occurs only if

$$e \equiv 0 \pmod{2},$$

$$e \equiv \frac{x_3}{x_1 + x_3} \pmod{2} \quad \text{if} \quad x_1 \neq x_3.$$

Fault induced into $y_1$

Assume that an adversary induces faults into $y_1 \mapsto y_1 + e$, $e \in GF(2^k)$, then

$$\tilde{y}_3 = \lambda(x_1 + x_3) + x_3 + y_1 + e$$

$$= y_3 + e.$$

A faulty point $P_3 = (x_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^2 + ax_3^2 + b = 0$ over $GF(2^k)$. Therefore,

$$e(x_3 + e) + T = 0,$$

where
\[ T = \tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^3 + ax_3^2 + b = 0. \]

A valid faulty point will occur only if

\[ e = 0 \pmod{2}, \]
\[ e = x_3 \pmod{2}. \]

**Fault induced into** \( x_1 \)

Assume that an adversary induces a fault into \( x_1 \mapsto x_1 + e, e \in GF(2^k) \), then

\[
\tilde{y}_3 = \lambda(x_1 + e + x_3) + x_3 + y_1 = y_3 + \lambda e.
\]

A faulty point \( \tilde{P}_3 = (x_3, \tilde{y}_3) \) is a valid faulty point only if \( \tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^3 + ax_3^2 + b = 0 \) over \( GF(2^k) \). Therefore,

\[
e(x_3\lambda + \lambda^2e) + T = 0, \quad \text{where}
\]
\[
T = \tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^3 + ax_3^2 + b = 0.
\]

A valid faulty point will occur only if

\[
e = 0 \pmod{2}, \]
\[
e = \frac{x_3}{\lambda} \pmod{2} \quad \text{if} \quad \lambda \neq 0.
\]

**Fault induced into** \( y_3 \)

Assume that an adversary induces a fault into \( y_3 \mapsto y_3 + e, e \in GF(2^k) \). Therefore, \( \tilde{P}_3 = (x_3, \tilde{y}_3) \) is a valid point only if

\[
e \equiv x_3 \pmod{2}.
\]

**Fault induced into** \( x_3 \)

Assume that an adversary induces a fault into \( x_3 \mapsto x_3 + e, e \in GF(2^k) \). Since \( x_3 \) is used later we need to investigate both transient and permanent faults.
Transverse faults. Assume that an adversary induces a transient fault into $x_3$, then

\[
\tilde{y}_3 = \lambda(x_1 + x_3 + e) + x_3 + e + y_1
\]

\[
= y_3 + e(\lambda + 1).
\]

A faulty point $\tilde{P}_3 = (x_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^3 + ax_3^2 + b = 0$ over $GF(2^k)$. Therefore,

\[
e(x_3(\lambda + 1) + (\lambda + 1)^2 e) + T = 0, \quad \text{where}
\]

\[
T = \tilde{y}_3^2 + x_3\tilde{y}_3 + x_3^3 + ax_3^2 + b = 0.
\]

Therefore, a valid faulty point occurs only if

\[
e \equiv 0 \text{ (mod 2)},
\]

\[
e \equiv \frac{x_3}{\lambda + 1} \text{ (mod 2)} \quad \text{if} \quad \lambda \neq 1.
\]

Permanent faults. Assume that an adversary induces a permanent fault into $x_3$, then

\[
\tilde{x}_3 = x_3 + e
\]

\[
\tilde{y}_3 = y_3 + (\lambda + 1)e.
\]

A faulty point $\tilde{P}_3 = (x_3, \tilde{y}_3)$ is a valid faulty point only if $\tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0$ over $GF(2^k)$. Therefore,

\[
e(x_3(\lambda + 1) + y_3 + x_3^2 + ((\lambda + 1)^2 + 1 + \lambda + x_3 + a) e + e^2) + T = 0,
\]

\[
T = \tilde{y}_3^2 + \tilde{x}_3\tilde{y}_3 + \tilde{x}_3^3 + a\tilde{x}_3^2 + b = 0.
\]

The solution $e = 0$ can be neglected, since it is error free, while the equation

\[
e^2 + ((\lambda + 1)^2 + 1 + \lambda + x_3 + a) e + x_3(\lambda + 1) + y_3 + x_3^2 = 0 \quad (4.60)
\]

is solved by using formulas for the solution of the quadratic equation over $GF(2^k)$ given in [25]. Let $z = \frac{e}{(\lambda+1)^2+1+\lambda+x_3+a}$, $(\lambda + 1)^2 + 1 + \lambda + x_3 + a \neq 0$ in (4.60), then

\[
z^2 + z + m = 0, \quad \text{where} \quad m = \frac{x_3(\lambda + 1) + y_3 + x_3^2}{((\lambda + 1)^2 + 1 + \lambda + x_3 + a)^2}.
\]

Depending on the degree $k$ of the finite field we have
• if $k$ is odd then

$$z = \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\}.$$  

Therefore, the faulty value is valid if

$$e = \left( (\lambda + 1)^2 + 1 + \lambda + x_3 + a \right) \sum_{i \in I} m^{2i}, \quad I \in \{1, 3, 5, \ldots, k - 2\}.$$  

• if $k \equiv (2 \ mod \ 4)$ then

  - if $T_4(m) = 0$ then

    $$z = \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i}}.$$  

    Therefore, the faulty value is valid if

    $$e = \left( (\lambda + 1)^2 + 1 + \lambda + x_3 + a \right) \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i}}.$$  

  - if $T_4(m) = 1$ and $\alpha^2 + \alpha = 1$ then

    $$z = \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i}}.$$  

    Therefore, the faulty point is valid if

    $$e = \left( (\lambda + 1)^2 + 1 + \lambda + x_3 + a \right) \left( \alpha + \sum_{i=0}^{(k-6)/4} (m + m^2)^{2^{2i}} \right).$$  

• if $k \equiv 0 \ (mod \ 4)$, $T_4(m) = 1$ and $S$ is as in (4.11) then

$$z = S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} 2^{2i+k/2} \right).$$  

Therefore, the faulty point is valid if

$$e = \left( (\lambda + 1)^2 + 1 + \lambda + x_3 + a \right) \left( S + S^2 + m^{2^{k-1}} \left( 1 + \sum_{i=0}^{(k/4)-1} 2^{2i+k/2} \right) \right).$$  

83
4.7 Conclusion

In this chapter by investigating each variable used in the affine addition formula of a non-supersingular elliptic curve we have derived conditions that an inflicted error needs to have in order to yield an undetectable faulty point on a non-supersingular elliptic curve over $GF(2^k)$. Note that if an adversary can add the value $x_2$ to $y_2$ in computation of $\lambda$, or if the point $P_2 = (x_2, y_2)$ in affine point addition, can be set to $-P_2 = (x_2, y_2 + x_2)$ then valid a faulty point can be created. Similarly, if an adversary can permanently add the value of $x_1$ to $y_1$ in the computation of $\lambda$, or if the point $P_1 = (x_1, y_1)$ in affine point addition can permanently be set to $-P_1 = (x_1, y_1 + x_1)$ then a valid faulty point can be created. Moreover, if an adversary can induce faults such that $x_2$ (or permanently $x_1$) is added to $y_1 + y_2$ in the computation of $\lambda$, the same effect is obtained as if $x_2$ is added to $y_2$ (or permanent $x_1$ to $y_1$) in the computation of $\lambda$. Also, after the correct computation of $y_3$, if an adversary mounts the attack such that value of $x_3$ is added to $y_3$, then a valid faulty point can be achieved. Finally, most of the induced faults that produce undetectable faulty points have error values depending on the degree $k$ of the finite field. Since it is possible to create faulty points that are valid points on the original non-supersingular elliptic curve by inducing faults into the affine addition formula of the non-supersingular curve, the common countermeasure of checking if the point is on the curve fails, therefore other possible countermeasures are required.
Chapter 5

Fault-Tolerant Computation in Finite Fields by Embedding in Rings

The arithmetic structure of finite fields is utilized in public key cryptography (smartcard technology, e-commerce, and internet security), as well as in coding theory (error-free communications and data storage). The security of cryptosystems does not only depend on the mathematical properties; an adversary can attack the implementation rather than algorithmic specification. Cryptosystems are used in the real world where cryptographic protocols are implemented in software, or hardware, obeying laws of physics. The circuits used leak information, e.g., power and timing information, over side channels. Thus, one has a gray box, where an adversary has access to several side-channels. Elliptic curve cryptographic applications rely on computation in a very large finite (with more than $2^{160}$ elements). Unfortunately, a single fault in computation can yield an erroneous output, which can then be used by an adversary to break cryptosystem. Since we require high reliability and robustness, fault tolerant finite field computation in elliptic curve cryptosystems is crucial.

Imbert et al. in [44] present fault-tolerant computation over the integers based on the modulus replication residue number system, which allows modular arithmetic computations over identical channels. Rayhani-Masoleh et al. in their work [81] present multipliers for fields $GF(2^k)$ whose operations are resistant to errors caused by certain faults. They can correct single errors caused by one, or more faults in the multiplier circuits. Gaubatz
and Sunar in [37] introduce scaled embedding for Fault-Tolerant Public Key Cryptography based on *arithmetic codes* and *binary cyclic codes* in order to achieve robust fault tolerant arithmetic in the finite field. Also, Gaubatz et al. in [36] present a scheme for robust multi-precision arithmetic over positive integers, protected by a *non-linear arithmetic residue codes*. Karpovsky et al. in [48] present a method that uses *systematic nonlinear (cubic) robust error detecting codes* to protect a hardware implementation of the *Advanced Encryption Standard* (AES) against a side-channel attack known as *Differential Fault Analysis attack* [42].

In this chapter we are concerned with protection of elliptic curve cryptosystems by protecting finite field computation against active side channel attacks, i.e., fault attacks - where an adversary induces faults into a device, while it executes the correct program. In Section 5.1 we propose a Chinese Remainder Theorem (CRT) based fault tolerant computation (FTC) in finite field for use in elliptic curve cryptosystems, as well as Lagrange Interpolation (LI) based fault tolerant computation in Section 5.2. Computation is decomposed into parallel, mutually independent, channels, so that fault effects do not spread to the other channels. By assuming fault models from Chapter 3 we test error correcting/detecting capability of our proposed schemes. We provide analysis of the error detection and correction capabilities of our proposed schemes in Subsection 5.1.3 and 5.2.4, as well as an error correction algorithms in Subsection 5.1.4 and 5.2.5. Our approach is based on the error correcting codes, i.e., *redundant residue polynomial codes* [24] and *Reed-Solomon codes* [80].

### 5.1 CRT based Fault-Tolerant Computation

In this section we provide CRT based FTC. This approach is based on the use of well known error correcting codes, i.e., *redundant residue polynomial codes* [24] which are generalization of *Reed-Solomon codes* [80].

Let $GF(q)$ be a finite field and $GF(q)[x]$ the ring of polynomials over $GF(q)$. Let $m_1(x), m_2(x), \ldots, m_n(x)$ be $n$ moduli in $GF(q)[x]$ which are relatively prime, and let
\( M(x) = \prod_{i=1}^{n} m_i(x) \). Furthermore, assume that the degree of each \( m_i(x) \) is \( d \), and \( kd \) information signals \( u = (u_0, u_1, \ldots, u_{kd-1}) \) are represented by polynomial

\[
a(x) = u_0 + u_1 x + \ldots + u_{kd-1} x^{kd-1}.
\]

Then a redundant residue polynomial code is the residue representation of \( a(x) \), i.e.,

\[
v = (a_1(x), a_2(x), \ldots, a_n(x)), \quad \text{where} \quad a_i(x) \equiv a(x) \mod m_i(x), \quad i = 1, \ldots, n,
\]

where \( \deg(a_i(x)) < d \). Polynomial \( a(x) \) is recovered by the Chinese remainder theorem.

Now, we will present application of this code to the fault tolerant computation in the finite binary extension field.

### 5.1.1 CRT based Finite Field Encoding

We want to protect computation over the field \( GF(2^k) \), which can be represented as the set of polynomials modulo a irreducible polynomial \( f(x) \), \( \deg(f(x)) = k \), i.e.,

\[
GF(2)[x]/ < f(x) > = \{ a_0 + \ldots + a_{k-1} x^{k-1} | a_i \in GF(2) \}.
\]

The inputs to the computation are elements from the binary extension field \( GF(2^k) \) represented as a polynomials of degree \( \leq k - 1 \) whose coefficients are from \( GF(2) \).

Let \( n \) be the expected degree of the output, which is not reduced modulo irreducible polynomial \( f(x) \), then computation of the finite field \( GF(2^k) \) can be performed with encoded operands in the larger polynomial ring \( R[x] = GF(2)[x]/ < m(x) > \), where \( \deg(m(x)) > n \). Operations in the ring \( R[x] \) are polynomial addition and multiplication modulo ring modulus \( m(x) \).

Let the ring modulus \( m(x) \) be product of the \( v \) pairwise relatively prime polynomials, i.e.,

\[
m(x) = m_1(x)m_2(x) \cdots m_v(x).
\]

Then, there is an isomorphism between polynomial ring \( R[x] \) and direct product ring \( R' \) of \( v \) smaller rings \( GF(2)[x]/ < m_i(x) > \), i.e.,

\[
R[x] \cong GF(2)[x]/ < m_1(x) > \times \cdots \times GF(2)[x]/ < m_v(x) >.
\]
Let $\phi$ be a bijective mapping, i.e.,

$$\phi : GF(2)[x]/ < f(x) > \rightarrow R',$$

then for all polynomials $a(x) \in GF(2)[x]/ < f(x) >$ we have

$$\phi(a(x)) = (a_1(x), \ldots, a_v(x)),$$

where $a_i(x) \equiv a(x) \pmod{m_i(x)}$ for $i = 1, \ldots, v$. Inverse mapping $\phi^{-1}$ is computed by Chinese Remainder Theorem (CRT) for polynomials,

$$a(x) \equiv \left( \sum_{i=1}^{v} a_i(x)T_i(x)M_i(x) \right) \pmod{m(x)}, \quad (5.1)$$

where $M_i(x) = \frac{m(x)}{m_i(x)}$, and polynomials $T_i(x)$ are computed by solving congruences $T_i(x)M_i(x) \equiv 1 \pmod{m_i(x)}$.

**Computation in the larger ring**

In general, let $\ast$ represent any of the two operations of the finite field $GF(2)[x]/ < f(x) >$, i.e., (addition, or multiplication), and $\odot$ represent any of the two operations of the polynomial ring $R'$, i.e., (componentwise addition, or componentwise multiplication). Given input polynomials $g(x), h(x) \in GF(2)[x]/ < f(x) >$ we want to compute

$$(g(x) \ast h(x)) \pmod{f(x)},$$

where $\max\{\deg(g \ast h)\} = n$. Computation is performed with encoded field elements in the ring $R' \cong R[x]$. Let $r = g(x) \ast h(x)$ without the modulo $f(x)$ reduction. Then

$$r = (g_1(x), \ldots, g_v(x)) \odot (h_1(x), \ldots, h_v(x))$$

$$= (g_1(x) \odot h_1(x), \ldots, g_v(x) \odot h_v(x)),$$

where $g_i(x) \equiv g(x) \pmod{m_i(x)}$, $h_i(x) \equiv h(x) \pmod{m_i(x)}$ and $g_i(x), h_i(x) \in GF(2)[x]/ < m_i(x) >$ for $i = 1, \ldots, v$, and vector $r \in R'$. By CRT, vector $r \in R'$ will determine a unique polynomial $r(x) \in R[x] = GF(2)[x]/ < m(x) >$ of degree $n$, which is then reduced modulo irreducible polynomial $f(x)$, so that $r(x)(mod f(x)) \in GF(2)[x]/ < f(x) >$. Next we will demonstrate how this computation can be protected by adding redundancy.
5.1.2 Fault-Tolerant Computation

To protect computation in the finite field we add redundancy by adding more parallel, modular channels than the minimum required to represent the output polynomial of a certain expected degree, i.e., see Fig. 5.1. Added redundant moduli $m_{v+1}(x), \ldots, m_c(x)$ have to be relatively prime to each other and to the non-redundant moduli $m_1(x), \ldots, m_v(x)$. Therefore, now computation happens in the larger direct product ring

$$R'' = GF(2)[x]/<m_1(x) > \times \ldots \times GF(2)[x]/<m_v(x) > \times \ldots \times GF(2)[x]/<m_c(x)>,$$

where

$$m'(x) = m_1(x) \cdot \ldots \cdot m_v(x) \cdot m_{v+1}(x) \cdot \ldots \cdot m_c(x),$$

such that

$$R'' \cong GF(2)[x]/<m'(x)>.$$

The redundant polynomial moduli have to be of degree larger then the largest degree of the non-redundant moduli, i.e.,

$$\deg(m_{v+j}(x)) > \max\{\deg\{m_1(x), \ldots, m_v(x)\}\}, \quad j = 1, \ldots, c - v,$$

and

$$\deg\left(\frac{m'(x)}{m_{j_1}(x) \cdot \ldots \cdot m_{j_{c-v}}(x)}\right) > n, \quad (5.3)$$

where $c - v$ is added redundancy.

Computational efficiency

To have efficient reduction in the smaller polynomial rings $GF(2)[x]/<m_i(x)>$, $i = 1, \ldots, v, \ldots, c$, modulus $m'(x)$ have to be chosen as a product of the pairwise relatively prime polynomials which are of the special low Hamming weight, leading to efficient modular reduction. Therefore, a smaller ring modulus can be chosen to be in the Mersenne form $x^n - 1$, or pseudo-Mersenne form $x^n + u(x)$, where polynomial $u(x)$ is of low weight. In $GF(2^k)$, the reduction is relatively inexpensive if the field is constructed by choosing the reduction polynomial to be a trinomial, i.e., $x^k + x^m + 1$ with $m < k/2$, or a pentanomial.
Table 5.1: Irreducible pentanomials $f(x) = x^k + x^m + x^n + x^h + 1$ over $GF(2)$, where $h < n < m < k/2$, $k \in [160 \ldots 450]$, suitable to use in Elliptic Curve Cryptography.

(if no trinomial is available) $x^k + x^m + x^n + x^h + 1$ with $h < n < m < k/2$, see Table 5.1, Table 5.2.

Algorithm 5 is only applied at the end of the computation, and its complexity is:

**Theorem 5.1.1** ([35]). Let $GF(2)[x]$ be polynomial ring over a field $GF(2)$, $m_1(x), \ldots, m_c(x) \in GF(2)[x]$, $d_i = \deg(m_i(x))$ for $1 \leq i \leq c$, $l = \deg(m'(x)) = \sum_{1 \leq i \leq c} d_i$, and $r_i(x) \in GF(2)[x]/ < m_i(x) >$ with $\deg(r_i(x)) < d_i$. Then the unique solution $r'(x) \in GF(2)[x]$ with $\deg(r'(x)) < l$ of the Chinese Remainder Problem $r'(x) \equiv r_i(x)(\mod m_i(x))$ for $1 \leq i \leq c$ for polynomials can be computed using $O(l^2)$ operations in $GF(2)$.

**Proof.** In step 1 of the Algorithm 5 we compute $m_1(x), m_1(x)m_2(x), \ldots, m_1(x)\cdots m_c(x)$
Table 5.2: Irreducible trinomials $f(x) = x^k + x^m + 1$ over $GF(2)$, where $m < k/2$, $k \in [160..450]$, suitable to use in Elliptic Curve Cryptography.

In step 2 we compute $m'(x)/m_i(x)$, $1 \leq i \leq c$, where each division takes at most $(d_i + 1)(l - d_i + 1)$ operations, therefore

$$
\sum_{1 \leq i \leq c} (2d_i + 1)(l - d_i + 1) \leq 2n \sum_{1 \leq i \leq c} (d_i + 1) = 2l(l + c) \in O(l^2).
$$

In step 2 we fix $i = 1, \ldots, c$. Then, the Extended Euclidean Algorithm with inputs
Algorithm 5 Chinese Remainder Algorithm.

Inputs: $m_1(x), \ldots, m_c(x) \in GF(2)[x], r_i(x) \in GF(2)[x] / < m_i(x) >, 1 \leq i \leq c.$

Outputs: $r'(x) \in GF(2)[x] / < m'(x) >$ where $r(x) \equiv r_i(x)(mod m_i(x)), 1 \leq i \leq c.$

1. $m'(x) \leftarrow m_1(x) \cdots m_c(x)$

2. for $1 \leq i \leq c$ do

3. compute $m'(x)/m_i(x)$

4. call the Extended Euclid Algorithm to compute $s_i(x), t_i(x) \in GF(2)[x]$ with

5. $s_i(x)m'(x) + t_i(x)m_i(x) = 1, v_i \leftarrow r_i(x)s_i(x) mod m_i(x)$

6. return $\sum_{1 \leq i \leq c} v_i \frac{m'(x)}{m_i(x)}$

$m'(x)/m(x)$ takes $O(d_i(l - d_i))$ operations. Since $\deg(s_i(x)) < \deg(m_i(x)) = d_i$, then to compute $v_i(x)s_i(x)(mod m_i(x))$ it takes $O(d_i^2)$ operations. Therefore, there is $O(ld_i)$ operations for each $i$, and $O(l^2)$ for step 2. In step 3, $v_i(x)(m'(x)/m_i(x)), i = 1, \ldots, c$ takes $O(d_i(l - d_i))$ operations, and $O(cl)$ additions of all products. Therefore, the total cost is $O(l^2)$.

5.1.3 Error correction and detection

Assume that there is one processor per independent channel as in Fig. 5.1. Let us assume that we have $c$ processors, where each processor computes the $i$-th polynomial residue and $i$-th residue operations. Also, we assume that Chinese Remainder Algorithm (CRA) at the end of the computation is error free. We assume that a fault attack induces faults into processors by some physical means. As a reaction, the attacked processor malfunctions, and it does not compute the correct output given its input. We are concerned with the effect of a fault as it manifests itself in a modified data, or a modified program execution. Therefore, we consider the fault models presented in Chapter 3. Since computation is decomposed into parallel, mutually independent, modular channels, the adversary can use either RFM, or AFM, or SFM per channel. Assume that at most $c - v$ channels have faults.
Figure 5.1: Fault tolerant computation over the finite field $GF(2^k)$. 
Let \( r' \in R'' \) be computed vector with \( c \) components, where \( e_j(x) \in GF(2)[x]/ < m_j(x) > \) is the error polynomial at \( j \)-th position; then the computed component at the \( j \)-th positions is \( b_j = (r(x) + e_j(x))(mod m_j(x)) \), and each processor will have as an output component
\[
b_j = \begin{cases} 
(r(x) + e_j(x))(mod m_j(x)), & j \in \{j_1, \ldots, j_\lambda\}, \\
r(x) \mod m_j(x), & \text{else}.
\end{cases}
\]

Here, we have assumed that the set of error positions are \( \{j_1, \ldots, j_\lambda\} \). By CRT the computed vector \( r' \in R'' \) with corresponding set of \( c \) moduli gives as a output polynomial \( r'(x) \in GF(2)[x]/ < m'(x) > \),
\[
r'(x) \equiv \left( \sum_{1 \leq i \leq c} r_i(x)T_i(x)M_i(x) \right) (mod m'(x))
\]
\[
= \left( \sum_{1 \leq i \leq c} r_i(x)T_i(x)M_i(x) + \sum_{1 \leq i \leq \lambda} e_{j_i}(x)T_{j_i}(x)M_{j_i}(x) \right) (mod m'(x))
\]
\[
= (r(x) + e(x))(mod m'(x)), \tag{5.4}
\]

where \( M_i(x) = \frac{m'(x)}{m_i(x)}, T_i(x) \) is computed by solving congruences \( T_i(x)M_i(x) \equiv 1 \mod m_i(x), \)
and \( M_{j_i}(x) = \frac{m'(x)}{m_{j_i}(x)}, \) while \( T_{j_i}(x) \) is computed by solving congruences \( T_{j_i}(x)M_{j_i}(x) \equiv 1 \mod m_{j_i}(x). \) Polynomial \( r(x) (mod m'(x)) \) in (5.4) is correct expected polynomial of degree \( \leq n \) and \( e(x)(mod m'(x)) \in GF(2)[x]/ < m'(x) > \) is the error polynomial such that:

**Theorem 5.1.2.** Let \( e_{j_i}(x) \in GF(2)[x]/ < m_{j_i}(x) > \) be error polynomial at positions \( j_i, \)
\( i \in \{1, \ldots, \lambda\}, \lambda \leq c - v \) then \( \deg(e(x)) > n. \)

**Proof.** We have that
\[
e(x) = \left( \sum_{1 \leq i \leq \lambda} e_{j_i}(x)T_{j_i}(x)M_{j_i}(x) \right) (mod m'(x))
\]
\[
= e_{j_1}(x)T_{j_1}(x)M_{j_1}(x) + \ldots + e_{j_\lambda}(x)T_{j_\lambda}(x)M_{j_\lambda}(x)
\]
\[
= \frac{m'(x)}{m_{j_1}(x) \cdot \ldots \cdot m_{j_\lambda}(x)} \sum_{i=1}^{\lambda} \frac{m_{j_i}(x) \cdot \ldots \cdot m_{j_\lambda}(x)}{m_{j_i}(x)} T_{j_i}(x) e_{j_i}(x). \tag{5.5}
\]

Since,
\[
\deg \left( \sum_{i=1}^{\lambda} \frac{\prod_{i=1}^{\lambda} m_{j_i}(x)}{m_{j_i}(x)} T_{j_i}(x) e_{j_i}(x) \right) < \deg \left( \frac{m'(x)}{\prod_{i=1}^{\lambda} m_{j_i}(x)} \right),
\]

94
and by (5.3), \( \deg \left( \frac{m'(x)}{\prod_{i=1}^{m} m_i} \right) > n \), we have that \( \deg(e(x)) > n \).

Therefore, faulty processors affect the result in an additive manner.

**Definition 5.1.3.** The set of correct results of computation, where \( n \) is expected degree of output polynomial of the computation without modulo \( f(x) \) reduction is

\[
C = \{ r'(x) \in GF(2)[x]/ < m'(x) > | \deg(r'(x)) \leq n \}.
\]

**Lemma 5.1.4.** The error is masked iff \( \deg(e(x)) \leq n \).

*Proof.* Let \( \deg(e(x)) \leq n \) in (5.4), then \( \deg(r'(x)) \leq n \), i.e., \( r'(x)(\mod f(x)) \in GF(2)[x]/ < f(x) > \).

**Lemma 5.1.5.** Let the expected degree of output polynomial without reduction modulo \( f(x) \) be \( n \), and let \( c > v \) be the number of parallel independent, modular channels (or number of processors). Then if up to \( c - v \) channels fail, the output polynomial \( r'(x) \notin C \).

*Proof.* By referring to (5.4), since if \( \deg(e(x)) > n \), the output polynomial \( r'(x) \) has to be such that \( \deg(r'(x)) > n \). Since, expected degree of the output polynomial of the field computation is \( n \) we have that \( r'(x) \notin C \).

**Lemma 5.1.6.** Let the expected degree of the output polynomial without reduction modulo \( f(x) \) be \( n \), and let \( c > v \) be number of parallel, independent, modular channels (or number of processors). If there is no faulty processors then \( r'(x)(\mod f(x)) \in GF(2)[x]/ < f(x) > \).

*Proof.* If there are no faulty processors, then clearly no errors occurred, and \( \deg(r'(x)) \leq n \), so that \( r'(x) = r(x), r'(x) \in C \). Therefore, \( r'(x)(\mod f(x)) \in GF(2)[x]/ < f(x) > \).

It is straightforward to appeal to the standard coding theory result below, to state the error detection and correction capability of our set up:

**Theorem 5.1.7.** (i) If the number of parallel, mutually independent, modular, redundant channels is \( d + t \leq c - v \ (d \geq t) \), then up to \( t \) faulty processors can be corrected, and up to \( d \) simultaneously detected. (ii) By adding \( 2t \) parallel, redundant, independent modular channels at most \( t \) faulty processors can be corrected.
Proof. To prove this we will use the fact that channels can be discarded if the suitable dynamic range is retained without affecting result. Assume that the expected degree of output polynomial is $n$ without modulo $f(x)$ reduction, with coefficients from finite field $GF(2)$. There are $v$ non-redundant channels. By adding one extra redundant, parallel, independent, modular channel, we can detect one faulty processor, since by CTR output vector will be of degree $> n$. By adding one more parallel, independent, modular channel, this faulty processor can be corrected, or up to two faulty processors can be detected. By removing one channel out of $v + 2$ channels, there are still $v + 1$ channels to detect the fault. If the removed channel was one that is not faulty, then faulty channel is present in the remaining $v + 1$ channels, and by CRT, the output polynomial is of degree $> n$. Since the expected degree is $n$, the error is detected. If the removed channel is one that is faulty, then the remaining $v + 1$ channels define correct output polynomial of degree $n$, unless the error is masked. Similarly, if we add 4 redundant channels than it is possible to detect up to 4 faults, or detect up to two and correct up to two faults. In general, by adding $d + t$, $d \geq t$ channels if we remove $t$ correct channels than since $d \geq t$, there is the capability to detect up to $d$ errors. In general, by adding $2t$ redundant, parallel, independent, modular channels at most $t$ faulty processors can be corrected.

5.1.4 Decoding based on the Euclid Algorithm

Let

\begin{align*}
s_{-1}(x) &= 1, \quad t_{-1}(x) = 0, \quad d_{-1}(x) = m'(x), \\
s_0(x) &= 0, \quad t_0(x) = 1, \quad d_0(x) = r'(x),
\end{align*}

and let

\begin{equation*}
s_n(x)m'(x) + t_n(x)r'(x) = d_n(x)
\end{equation*}

be consecutive steps in Euclidean algorithm for calculating $gcd(m'(x), r'(x))$ where $m'(x)$ is given by (5.2), and $r'(x) \in GF(2)[x]/ < m'(x) >$ is output polynomial of the computation. Also, let $j_i, i \in \{1, \ldots, \lambda\}, \lambda \leq c - v$ be positions of error polynomial $e(x)$, and let

\begin{equation*}
m^+(x) = \prod_{i=1}^{\lambda} m_{j_i}(x),
\end{equation*}
\[\nu = \sum_{i=1}^{v} \deg(m_i(x)) + \sum_{i=1}^{\lambda} \deg(m_{ji}(x)) - 1,\]
\[u = \sum_{i=1}^{\lambda} \deg(m_{ji}(x)),\]
then it follows:

**Lemma 5.1.8** ([86]). If \(\nu \geq \deg(\text{gcd}(m'(x), r'(x)))\), \(u + \nu = \deg(m'(x)) - 1\), then there is a unique index \(j\) in the algorithm such that \(\deg(t_j) \leq u\), \(\deg(d_j) \leq \nu\).

**Theorem 5.1.9** ([86]). If \(t(x), d(x)\) are nonzero and \(t(x)r'(x) \equiv d(x) \mod m'(x), \deg(t(x)) + \deg(d(x)) < \deg(m'(x))\), then there is a unique index \(j\) and a polynomial \(\zeta\) such that \(t(x) = \zeta(x)t_j(x), d(x) = \zeta(x)d_j(x)\).

By (5.4) and (5.5) we have

\[r'(x) = r(x) + \frac{m'(x)}{m^+(x)} \sum_{i=1}^{\lambda} \frac{m^+(x)}{m_{ji}(x)} T_{ji}(x)e_{ji}(x), \text{ i.e.,}\]
\[r'(x)m^+(x) - m'(x) \sum_{i=1}^{\lambda} \frac{m^+(x)}{m_{ji}(x)} T_{ji}(x)e_{ji}(x) = r(x)m^+(x).\]  
(5.6)

Using (5.6) we apply Theorem 5.1.9 with \(t(x) = m^+(x)\), and \(d(x) = r(x)m^+(x)\), then correct output residue is

\[r(x) = \frac{r(x)m^+(x)}{m^+(x)} = \frac{d(x)}{t(x)} = \frac{d_j(x)}{t_j(x)},\]

where \(j\) is the first index for which

\[\deg(d_j(x)) < \sum_{i=1}^{v} \deg(m_i(x)) + \sum_{i=1}^{\lambda} \deg(m_{ji}(x)).\]

If \(\deg(d_j(x)) - \deg(t_j(x)) \geq \sum_{i=1}^{v} \deg(m_i(x)), \text{ or } t_j(x) \nmid d_j(x)\), then more than \(\lambda\) errors occurred.
Algorithm 6 Euclid’s Decoding Algorithm

**Input:** output vector of computation \( r' = (r_1(x), r_2(x), \ldots, r_c(x)) \in R'' \).

**Output:** \( r(x) \in GF(2)[x]/ < f(x) >. \)

1. By CRT algorithm compute \( r'(x) \)
2. if \( \text{deg}(r'(x)) \leq n \) then
3. \( r(x) = r'(x), r(x) \text{ mod } f(x) \)
4. else
5. \( t_{-1}(x) = 0, t_0(x) = 1, d_{-1}(x) = m'(x), d_0(x) = r'(x) \)
6. \( j = 1 \)
7. while \( \text{deg}(d_j(x)) > \sum_i^{\nu} \text{deg}(m_i(x)) + \sum_i^{\lambda} \text{deg}(m_{ji}(x)) \) do
8. \( d_{j-2}(x) = q_j(x)d_{j-1}(x) + d_j(x); \text{deg}(d_j(x)) < \text{deg}(d_{j-1}(x)) \)
9. \( t_j(x) = t_{j-2}(x) - q_j(x)t_{j-1}(x) \)
10. \( j = j + 1 \)
11. return \( r(x) = \frac{d_j(x)}{t_j(x)}, r(x) \text{ mod } f(x) \)

**Example 5.1.10.** Assume that we want to protect computation in \( GF(2^3) \cong GF(2)[x]/ < x^3 + x + 1 >. \) Let the inputs to the computation be the following finite field elements: \( a(x) = x+1, b(x) = x^2 + 1. \) We want to compute following expression \( (a(x)b(x)) \text{ mod } f(x). \)

Let \( R[x] = GF(2)[x]/ < m(x) >, \) where

\[
\begin{align*}
m(x) &= x^{11} + x^8 + x^6 + x^4 + x^2 \\
&= (x^2 + x + 1)x^2(x^3 + x^2 + 1)(x^4 + x + 1) \\
&= m_1(x)m_2(x)m_3(x)m_4(x).
\end{align*}
\]

Now, \( v = 2, c - v = 2 \) and error correction capability is \( t = 1. \) Therefore, computation will happen with encoded field elements in the following direct product ring:

\[
GF(2)[x]/ < m_1(x) > \times \ldots \times GF(2)[x]/ < m_4(x) >,
\]

where

\[
a(x) \leftrightarrow a = (x+1, x+1, x+1, x+1) \\
b(x) \leftrightarrow b = (x, 1, x^2 + 1, x^2 + 1), \text{ so that}
\]

98
where $\odot$ is componentwise multiplication.

By applying the Chinese Remainder Theorem on (5.7), it follows $x^3 + x^2 + x + 1 \in \mathbb{F}[x]$.

Now, assume that an adversary induces faults into point $P \in E/\mathbb{F}(2^3)$ by inducing faults into one of 4 processors, by some physical set up, causing attacked processor to be faulty, such that erroneous output of the computation is

$$a \odot b = (1, x, x, x^3 + x^2 + x + 1),$$

so that by applying CRT on (5.8) it follows that:

$$r'(x) = x^9 + x^6 + x^4 + x^3 + x.$$

From the extended Euclidian algorithm for calculating $\gcd(r'(x), m(x))$ we have $d(x) = x^5 + x^4 + x^3 + x^2$, and $s(x) = x^3 + x^2 + 1$. Therefore $r(x) = d(x)/t(x) = x^3 + x^2 + x + 1$, i.e., $r(x)(\mod f(x)) = x^2 \in \mathbb{F}(2^k)$.

5.2 Lagrange Interpolation (LI) based Fault-Tolerant Computation

In this section we provide LI based FTC. This approach is based on the use of the first original approach of Reed-Solomon codes. Reed-Solomon codes are very effective in correcting random symbol errors, and random burst errors, and they are widely used for error control in communication and data storage systems, ranging from deep-space telecommunications to compact discs. Suppose that we have a block of $k$ information symbols $m_0, m_1, \ldots, m_{k-1} \in \mathbb{F}(q)$. These symbols can be used to construct a polynomial $P(x) = m_0 + m_1 x + \ldots + m_{k-1} x^{k-1}$. A Reed-Solomon codeword $c$ is formed by evaluating $P(x)$ at each of the $q$ elements in the finite field $\mathbb{F}(q)$, i.e.,

$$c = (c_0, c_1, \ldots, c_{q-1}) = (P(0), P(\alpha), \ldots, P(\alpha^{q-1})).$$
Upon receiving message \((P(0), P(\alpha), \ldots, P(\alpha^{q-1}))\) decoding can be done by solving simultaneously any \(k\) of \(q\) equations,

\[
P(0) = a_0 \\
P(\alpha) = a_0 + a_1 \alpha + a_2 \alpha^2 + \ldots + a_{k-1} \alpha^{k-1} \\
P(\alpha^2) = a_0 + a_1 \alpha^2 + a_2 \alpha^4 + \ldots + a_{k-1} \alpha^{2k-2} \\
\vdots \\
P(1) = a_0 + a_1 + a_2 + \ldots + a_{k-1}.
\]

Now, we will present application of the Reed-Solomon code to fault tolerant computation in the finite binary extension field.

### 5.2.1 Lagrange Interpolation Finite Field Encoding

Recall that we want to protect computation of the finite binary extension field \(GF(2^k)\), which can be represented as the set of polynomials modulo an irreducible polynomial \(f(x)\), \(\text{deg}(f(x)) = k\), i.e.,

\[
GF(2)[x]/ < f(x) >= \{a_0 + \ldots + a_{k-1} x^{k-1} | a_i \in GF(2)\},
\]

where the element

\[
a_0 + a_1 x + \ldots + a_{k-1} x^{k-1} \in GF(2)[x]/ < f(x) >
\]

can also be considered as a vector

\[
(a_0, a_1, \ldots, a_{k-1}) \in GF(2)^k, \quad a_i \in GF(2).
\]

Also, let the polynomial \(f(x)\) be primitive with \(f(\alpha) = 0\). Then

\[
GF(2^k) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^k-2}\}.
\]

The inputs to the computation are elements from the binary extension field \(GF(2^k)\) represented as a polynomials of degree \(\leq k - 1\) whose coefficients are from \(GF(2)\).
Clearly, polynomials can be represented by their coefficients, or by their values at sufficiently many points.

The input polynomials $g_i(x)$ from the finite field $GF(2)[x] < f(x) >$ are evaluated at the minimum required number of distinct elements from the set $T = \{\alpha_j | \alpha_j \in GF(2^k)\}$ such that there are enough values to represent the polynomial resulting from the computation. Evaluating input polynomials $g_i(x) \in GF(2)[x] < f(x) >$ at distinct elements $\alpha_j \in T$ is same as taking remainder modulo $x - \alpha_j$.

Let $n$ be the expected degree of the output, which is not reduced modulo irreducible polynomial $f(x)$. Then, there exist mapping $\phi$

$$\phi : GF(2)[x]/ < f(x) > \mapsto GF(2^k)[x]/ < x - \alpha_0 > \times \cdots \times GF(2^k)[x]/ < x - \alpha_n >,$$

such that each input polynomial $g_i(x) \in GF(2)[x]/ < f(x) >$ is evaluated at $n + 1$ distinct elements from the set $T = \{\alpha_j | \alpha_j \in GF(2^k)\}$, i.e.,

$$g_i(x) \leftrightarrow (g_i(\alpha_0), g_i(\alpha_1), \ldots, g_i(\alpha_n)),$$  \hfill (5.9)

where $g_i(\alpha_j) \in GF(2^k)$ (or equivalently $g_i(\alpha_j) \in GF(2^k)$) are evaluations of the input polynomials $g_i(x) \in GF(2)[x]/ < f(x) >$ at distinct elements from the set $T$. Equivalently, $g_i(\alpha_j)$ is remainder of $g_i(x)$ on division by linear polynomial $(x - \alpha_j)$, i.e., $g_i(x) \equiv g_i(\alpha_j) \mod (x - \alpha_j)$.

**Computation in the Larger Ring**

The computation of the finite field $GF(2^k)$ will be performed with encoded operands (as in (5.9)) in the direct product ring:

$$R = GF(2^k)[x]/ < x - \alpha_0 > \times \cdots \times GF(2^k)[x]/ < x - \alpha_n > \approx GF(2^k)^{n+1},$$ \hfill (5.10)

while preserving arithmetic structure.
Also, we have that

\[ R \cong GF(2^k)[x]/<m(x)>, \]  

(5.11)

where

\[ m(x) = \prod_{i=0}^{n} (x - \alpha_i), \]

such that \( \text{deg} (m(x)) = 1 + \max \{\text{deg} (g(x) * h(x))\} \), where \( g(x), h(x) \in GF(2)[x]/<f(x)> \) are input polynomials.

Operations in the ring \( R \) are \textit{componentwise} \textit{addition} \( \oplus \), i.e., for components \( c_i, b_i \in GF(2^k) \) we have

\[
(c_0, c_1, \ldots, c_n) \oplus (b_0, b_1, \ldots, b_n) = (c_0 + b_0, c_1 + b_1, \ldots, c_n + b_n),
\]

and \textit{componentwise} \textit{multiplication} \( \odot \), i.e., for components \( c_i, b_i \in GF(2^k) \) we have

\[
(c_0, c_1, \ldots, c_n) \odot (b_0, b_1, \ldots, b_n) = (c_0b_0, c_1b_1, \ldots, c_nb_n),
\]

where \( c_i + b_i, c_i b_i \in GF(2^k) \).

Thus, the computation is decomposed into \( n + 1 \) parallel, independent channels that are identical, such that computations in the each channel are mutually independent and happen over the same finite field \( GF(2^k) \).

In general, let \( * \) represent any of the two operations of the finite field \( GF(2)[x]/<f(x)> \), i.e., (addition, or multiplication), and \( \circ \) represent any of the two operations of the polynomial ring \( R \), i.e., (componentwise addition, or componentwise multiplication). Given input polynomials \( g(x), h(x) \in GF(2)[x]/<f(x)> \) we want to compute \( (g(x) * h(x)) (mod f(x)) \), where \( \max \{\text{deg} (g * h)\} = n \). Computation is performed with encoded field elements in the ring \( R \). Let \( r(x) = g(x) * h(x) \) without the modulo \( f(x) \) reduction. Then

\[
\begin{align*}
    r &= (g(\alpha_0), g(\alpha_1), \ldots, g(\alpha_n)) \circ (h(\alpha_0), h(\alpha_1), \ldots, h(\alpha_n)) \\
    &= (g(\alpha_0) \odot h(\alpha_0), g(\alpha_1) \odot h(\alpha_1), \ldots, g(\alpha_n) \odot h(\alpha_n)) \\
    &= (r(\alpha_0), r(\alpha_1), \ldots, r(\alpha_n)),
\end{align*}
\]  

(5.12)
where \( g(\alpha_i), h(\alpha_i) \) are polynomial evaluations of polynomials \( g(x), h(x) \in GF(2)[x]/ < f(x) > \) at \( n+1 \) distinct elements of \( GF(2^k) \).

The following is well known:

**Proposition 5.2.1.** (Lagrange Interpolation Formula [59]). For \( n \geq 0 \), let \( a_0, \ldots, a_n \) be \( n+1 \) distinct elements of \( F \), and let \( b_0, \ldots, b_n \) be \( n+1 \) arbitrary elements of \( F \). Then there exist exactly one polynomial \( f \in F[x] \) of degree \( \leq n \) such that \( f(a_i) = b_i \) for \( i = 0, \ldots, n \). This polynomial is given by

\[
f(x) = \sum_{i=0}^{n} b_i \prod_{k=0}^{n} (a_i - a_k)^{-1} (x - a_k).
\]

By Proposition 5.2.1, interpolating \( n+1 \) output components \( r(\alpha_j) \in GF(2^k) \) at distinct elements \( \alpha_j \in GF(2^k) \) will determine a unique polynomial \( r(x) \in GF(2^k)[x]/ < m(x) > \) of degree \( n \). If the coefficients \( a_i \) of the polynomial \( r(x) \) lie in \( GF(2) \), then \( r(x)(mod f(x)) \in GF(2)[x]/ < f(x) > \).

### 5.2.2 Fault-Tolerant Computation

To protect computation in the finite field we add redundancy by adding more parallel channels than the minimum required to represent the output polynomial of a certain expected degree, i.e., see Figure 5.2. Thus, input polynomials are evaluated at additional distinct elements, with the constraint that elements \( \alpha_j \in GF(2^k) \) at which polynomials are evaluated are all distinct elements. Let \( n \) be expected degree of the output polynomial without modulo \( f(x) \) reduction, then by Proposition 5.2.1, the minimum required number of non-redundant polynomial evaluations is \( n+1 \). We add \( c-n-1 \) extra redundant polynomial evaluations, \( c > n+1 \), so that computation now takes place in the even larger direct product ring

\[
R' = GF(2^k)[x]/ < x - \alpha_0 > \times \ldots \times GF(2^k)[x]/ < x - \alpha_{c-1} > \cong GF(2^k)^c.
\]

Also, we have that

\[
R' \cong GF(2^k)[x]/ < m'(x) >,
\]

103
Figure 5.2: Fault tolerant computation of the finite field $GF(2^k)$ in the ring $R'$. 
where
\[ m'(x) = \prod_{i=0}^{c-1} (x - \alpha_i), \]
and \( \deg(m'(x)) = c, \ c > n + 1. \)

Let \( n \) be the expected degree of the output, which is not reduced modulo irreducible polynomial \( f(x) \). Then each input polynomial \( g_i(x) \in GF(2)[x]/ < f(x) > \) is evaluated at \( c, c > n + 1 \) distinct elements from the set \( T = \{ \alpha_j \mid \alpha_j \in GF(2^k) \} \), i.e.,

\[ g_i(x) \rightarrow (g_i(\alpha_0), g_i(\alpha_1), \ldots, g_i(\alpha_n), g_i(\alpha_{n+1}), g_i(\alpha_{n+2}), \ldots, g_i(\alpha_{c-1})) \in R', \]

where \( g_i(\alpha_0), g_i(\alpha_1), \ldots, g_i(\alpha_n) \) are non-redundant components of \( i \)-th input polynomial, and \( g_i(\alpha_{n+1}), g_i(\alpha_{n+2}), \ldots, g_i(\alpha_{c-1}) \) are redundant components of \( i \)-th input polynomial.

Now, let \( r' \in R' \) be an output vector of computation which is in the form

\[ r' = (r(\alpha_0), r(\alpha_1), \ldots, r(\alpha_n), r(\alpha_{n+1}), r(\alpha_{n+2}), \ldots, r(\alpha_{c-1})). \quad (5.13) \]

By Proposition 5.2.1, if there is no fault effect, \( c \) output components \( r(\alpha_j) \in GF(2^k) \) at distinct elements \( \alpha_j \in GF(2^k) \) will determine a unique polynomial \( r'(x) \in GF(2^k)[x]/ < m'(x) > \) of degree \( n \) with coefficients \( a_i \in GF(2) \), otherwise, \( \deg(r'(x)) > n \) with coefficients \( a_i \in GF(2^k) \).

Therefore, it follows that:

**Definition 5.2.2.** The set of correct results of computation, where \( n \) is expected degree of output polynomial of the computation without modulo \( f(x) \) reduction, is

\[ C = \{ r'(x) \in GF\left(2^k\right)[x]/ < m'(x) > \mid \deg(r'(x)) < n + 1, a_i \in GF(2) \}. \]

### 5.2.3 Complexity of Interpolation and Evaluation

**Remark 5.2.3.** Input polynomials are only evaluated at the beginning, while interpolation performed at on the end of the computation. Only if there is no errors we do modulo \( f(x) \) reduction of the interpolated polynomial.
Lemma 5.2.4. Computational complexity of evaluating input polynomials $g_i(x) \in GF(2)[x]/ < f(x) >$ at $c > n + 1$ distinct elements from set $T$, where $n$ is expected degree of the output polynomial without modulo $f(x)$ is $O(ck)$, since the required number of operations in $GF(2^k)$ is $2c(k - 1)$.

Proof. Let $g_i(x) = \sum_{i=0}^{k-1} a_i x^i \in GF(2^k)$, then by Horner’s rule
\[
g_i(x) = (\ldots (a_{k-1}x + a_{k-2})x + \ldots + a_1)x + a_0
\]
it can be evaluated at a single point $\alpha_i \in T$ by $k - 1$ addition and $k - 1$ multiplications. Therefore, evaluating $g_i(x)$ at $c > n + 1$ distinct elements from $T$ it will require $2c(k - 1)$ operations in $GF(2^k)$. So computation complexity of input polynomial evaluation is $O(ck)$.

Lemma 5.2.5. Computational complexity of interpolating output vector $r' \in R'$ is $O(c^2)$, $c > n + 1$.

Proof. Proof is as in [35]. Therefore, let $m_i = x - \alpha_i$, $0 \leq i \leq c$, $\alpha_i \in T$. First it is computed $m_0 m_1 m_2 \ldots, m' = m_0 m_1 \ldots m_{c-1}$. This takes
\[
\sum_{1 \leq i \leq c} (2i - 1) = c^2 - 2c + 1
\]arithmetic operations. For $0 \leq i \leq c - 1$ we divide $m'$ by $m_i$, taking $2c - 2$ operations. Evaluation of $m'/m_i$ at $\alpha_i$ takes at most $2c - 4$ operations, then $m'/m_i$ is divided by that value in order to obtain $l_i = \prod_{0 \leq j < c, j \neq i} \frac{x - \alpha_j}{\alpha_i - \alpha_j}, \alpha_i, \alpha_j \in GF(2^k)$. This adds to $4c^2 - 5c$ operations for $0 \leq i \leq c - 1$. Computing linear combination $\sum_{0 \leq i \leq c} f(\alpha_i)l_i$ takes $2c^2 - c$ operations. Therefore, computational complexity of interpolating output vector $r' \in R'$ is $O(c^2)$, $c > n + 1$.

Theorem 5.2.6. Total computational complexity of evaluating input polynomials $g_i(x) \in GF(2)[x]/ < f(x) >$ at the beginning of computation, and interpolation of the result of the computation at the end of computation is $O(c^2)$.

Proof. Since the computational complexity of evaluating inputs $g_i(x) \in GF(2)[x]/ < f(x) >$ is $O(ck)$, where $k < c$, $ck < c^2$ and complexity of interpolating result vector is $O(c^2)$, then total complexity is $O(c^2)$. 

106
5.2.4 Error Detection and Correction

Assume that there is one processor per independent channel, i.e., see Figure 5.2. Let us assume that we have \( c \) processors, where each processor computes \( i \)-th polynomial evaluation and operations of the finite field \( GF(2^k) \). Also, we assume that Lagrange interpolation at the end of the computation is error free. We assume that a fault attack induces faults into processors by some physical means. As a reaction, the attacked processor malfunctions, and it does not compute the correct output given its input. We are concerned with the effect of a fault as it manifests itself in a modified data, or a modified program execution. Therefore, we consider the fault models presented in Chapter 3. Since computation is decomposed into parallel, mutually independent, identical channels, the adversary can use either RFM, or AFM, or SFM per channel.

Assume that at most \( c - n - 1 \) channels have faults. Let \( r' \in R' \) be computed vector with \( c \) components, where \( e_j \in GF(2^k) \) is the error at \( j \)-th position; then the computed component at the \( j \)-th positions is

\[
r_j = r(\alpha_j) + e_j,
\]

and each processor will have as an output component

\[
r_j = \begin{cases} 
  r(\alpha_j) + e_j, & j \in \{j_1, \ldots, j_t\}, \\
  r(\alpha_j), & \text{else}.
\end{cases}
\]

Here, we have assumed that the set of error positions are \( \{j_1, \ldots, j_t\} \), i.e., \( e_j \) is the effect of the fault in the channel \( j_i \).

By Proposition 5.2.1, the computed vector \( r' \in R' \) with corresponding set of \( c \) distinct elements \( \alpha_j \in GF(2^k) \) gives as a output unique polynomial \( r'(x) \in GF(2^k)[x]/ < m'(x) > \),

\[
r'(x) = \sum_{0 \leq i \leq c-1} r_i \prod_{0 \leq j \leq c-1 \atop i \neq j} \frac{x - \alpha_j}{\alpha_i - \alpha_j}
\]

\[
= r(x) + \sum_{1 \leq i \leq t} e_{j_i} \prod_{0 \leq i \leq c-1 \atop j_i \neq i} \frac{x - \alpha_i}{\alpha_j - \alpha_i}
\]

\[
= r(x) + e(x),
\]

(5.15)
where \( r(x) \) is correct expected polynomial of degree \( \leq n \) with coefficients from the ground field \( GF(2) \), and \( e(x) \) is the error polynomial such that

**Theorem 5.2.7.** Let effects of the fault \( e_{j_1} \neq 0, \ldots, e_{j_t} \neq 0 \) be any set of \( 1 \leq t \leq c - n - 1 \) elements of \( GF(2^k) \), \( c > n + 1 \), then \( \text{deg}(e(x)) > n \) whose coefficients \( a_i \in GF(2^k) \).

**Proof.** We have that

\[
e(x) = \sum_{1 \leq i \leq t} e_{j_i} \prod_{0 \leq j \leq c-1, \ j \neq i} \frac{x - \alpha_j}{\alpha_{j_i} - \alpha_i}
\]

\[
= \prod_{0 \leq i \leq c-1} (x - \alpha_i) \left( \prod_{0 \leq j \leq c-1, \ j \neq i} e_{j_i} \frac{e_{j_i}}{(x - \alpha_{j_i})} \prod_{0 \leq j \leq c-1, \ j \neq i} \frac{1}{\alpha_{j_i} - \alpha_i} \right) + \ldots
\]

Since, \( \text{deg} \left( \prod_{0 \leq i \leq c-1} \frac{(x - \alpha_i)}{(x - \alpha_{j_i})} \right) = c - 1, \ldots, \text{deg} \left( \prod_{0 \leq i \leq c-1} \frac{e_{j_i}}{(x - \alpha_{j_i})} \right) = c - 1, c > n + 1 \), then \( \text{deg}(e(x)) = c - 1 > n \) with coefficients \( \frac{e_{j_i}}{\alpha_{j_i} - \alpha_i} \in GF(2^k) \).

Therefore, faulty processors affect the result in an additive manner.

**Theorem 5.2.8.** The error is masked iff error polynomial \( e(x) \) has coefficients from \( GF(2) \), and if \( \text{deg}(e(x)) \leq n \).

**Proof.** Let \( n \) be expected degree of the output polynomial without modulo \( f(x) \) reduction, and let \( r'(x) \) be computed polynomial as in (5.15). Since, \( \text{deg}(r(x)) \leq n \) with coefficients \( a_i \in GF(2) \), then if \( \text{deg}(e(x)) \leq n \) with coefficients \( a_i \in GF(2) \) we have that \( \text{deg}(r'(x)) \leq n \) with \( a_i \in GF(2) \) in which case error is masked.

**Theorem 5.2.9.** Let the expected degree of output polynomial be \( n \) without modulo \( f(x) \) reduction, and let \( c > n + 1 \) be the number of parallel, independent channels (or number of processors). Then if up to \( c - n - 1 \) channels fail, the output polynomial \( r'(x) \) is such that \( \text{deg}(r'(x)) > n \) with coefficients \( a_i \in GF(2^k) \), i.e., \( r'(x) \notin C \).

**Proof.** By referring to (5.15), since \( \text{deg}(e(x)) > n \) with coefficients \( a_i \in GF(2^k) \), and \( \text{deg}(r(x)) = n \) with coefficients \( a_i \in GF(2) \), the output polynomial \( r'(x) \) has to be such that \( \text{deg}(r'(x)) > n \), and \( a_i \in GF(2^k) \). Therefore, \( r'(x) \notin C \).
Theorem 5.2.10. Let the expected degree of the output polynomial be $n$ without modulo $f(x)$ reduction, and let $c > n + 1$ be number of parallel, independent channels (or number of processors). If there is no faulty processors then there are no errors and $\deg(r'(x)) \leq n$ and coefficients $a_i \in GF(2)$, i.e., $r'(x) \in C$. Therefore, $r'(x) \pmod{f(x)} \in GF(2)[x]/<f(x)>$.

Proof. If there are no faulty processors, then clearly no errors occurred, and $\deg(r'(x)) \leq n$ with coefficients $a_i \in GF(2)$, so that $r'(x) = r(x)$. Therefore $r'(x) \in C$ implies $r'(x) \pmod{f(x)} \in GF(2)[x]/<f(x)>$. \hfill \qed

It is straightforward to appeal to the standard coding theory result below, to state the error detection and correction capability of our set up:

Theorem 5.2.11. If the number of parallel, mutually independent, identical redundant channels is $d + t \leq c - n - 1$ ($d \geq t$), then up to $t$ faulty processors can be corrected, and up to $d$ simultaneously detected.

Proof. To prove this we will use the fact that channels can be discarded if the suitable dynamic range is retained without affecting result. Assume that the expected degree of output polynomial is $n$ without modulo $f(x)$ reduction, with coefficients from finite field $GF(2)$. There are $n + 1$ non-redundant channels. By adding one extra redundant parallel independent channel, we can detect one faulty processor, since by interpolating corresponding output vector of the computation the highest order coefficient of the output polynomial will be non-zero. By adding one more parallel independent channels, this faulty processor can be corrected, or up to two faulty processors can be detected. By removing one channel out of $n + 3$ channels, there are still $n + 2$ channels to detect the fault. If the removed channel was one that is not faulty, then one that is faulty is present in the remaining $n + 2$ channels, and by Proposition 5.2.1, the output polynomial is of degree $> n$ with coefficients $a_i \in GF(2^k)$. Since the expected degree is $n$, the error is detected. If the removed channel is one that is faulty, then the remaining $n + 2$ channels (by Proposition 5.2.1) define correct output polynomial of degree $n$ with coefficients $a_i \in GF(2)$, unless the error is masked. Similarly, if we add 4 redundant channels than it is possible to detect up to 4 faults, or detect up to two and correct up to two faults. In general, by adding $d + t$,
$d \geq t$ channels if we remove $t$ correct channels than since $d \geq t$, there is the capability to detect up to $d$ errors.

Similarly we have the following theorem:

**Theorem 5.2.12.** By adding $2t$ redundant independent channels at most $t$ faulty processors can be corrected.

**Proof.** The proof follows from the proof of the Theorem 5.2.11.

While it is true that arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed here, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost.

### 5.2.5 Decoding of the Output

The original approach of decoding *Reed-Solomon codes* can be applied to our case. If $t$ processors out of $c$ are faulty, we can enumerate all combinations of $n + 1$ equations, each combination solve and keep counter for each solution, so that the largest number of votes identifies correct solution. Since, this way of decoding is inefficient, the *Welch-Berlekamp decoding algorithm* [95] can be used instead. For completeness’ sake, we discuss the algorithm below and provide detailed proofs:

Let output vector $r' \in R'$ of computation be as in (5.13). We choose a set of $n + 1$ indices $K = \{0, 1, \ldots, n\}$, and $\overline{K} = \{0, \ldots, c - 1\} \setminus K$. By *Lagrange Interpolation* we determine polynomial

$$r'(x) = \sum_{j \in K} r_j \prod_{i \in K, i \neq j} \frac{x - \alpha_j}{\alpha_i - \alpha_j},$$

(5.16)

where $\deg(r'(x)) \leq n, j \in K$.

If the output component $r_j \in GF(2^k)$ includes an error, then output polynomial is as in (5.15). Let component $r_l$ be as in (5.14), and

$$\psi_j(x) = \prod_{\substack{i \in K \setminus \alpha_j \neq j}} \frac{x - \alpha_j}{\alpha_i - \alpha_j},$$

110
then by evaluating (5.16) at \( \alpha_l \in GF(2^k) \) we have

\[
r''(\alpha_l) = \sum_{j \in K} r(\alpha_j) \psi_j(\alpha_l) + \sum_{j \in K} e_j \psi_j(\alpha_l).
\]

Subtracting the interpolated values from the received values we have

\[
r_l - r''(\alpha_l) = r(\alpha_l) + e_l - \sum_{j \in K} r(\alpha_j) \psi_j(\alpha_l) - \sum_{j \in K} e_j \psi_j(\alpha_l).
\]

Since \( \deg(r(x)) \leq n \) then the syndromes are

\[
S_l = r_l - r''(\alpha_l) = e_l - \sum_{j \in K} e_j \psi_j(\alpha_l), \quad \text{for} \quad l \in \overline{K}. \tag{5.17}
\]

Let's define a polynomial \( g(x) \) as

\[
g(x) = \prod_{j \in K} (x - \alpha_j), \tag{5.18}
\]

where its formal derivative is \( g'(x) \) such that

\[
\psi_j(x) = \frac{g(x)}{(x - \alpha_j)g'(\alpha_j)}.
\]

Then the syndromes \( S_l \) can be expressed as

\[
S_l = e_l - \sum_{j \in K} e_j \frac{g(\alpha_j)}{(\alpha_l - \alpha_j)g'(\alpha_j)}.
\]

or

\[
\frac{S_l}{g(\alpha_l)} = \frac{e_l}{g(\alpha_l)} - \sum_{j \in S} \frac{e_l}{g'(\alpha_j) \cdot (\alpha_l - \alpha_j)} \tag{5.19}
\]

For \( l \in \overline{K} \) we let

\[
y_l = \frac{S_l}{g(\alpha_l)}. \tag{5.20}
\]

Now, let \( H \) be set of indices for which \( e_i \neq 0 \), and define

\[
d_K(x) = \prod_{i \in H \cap K} (x - \alpha_i),
\]

whose roots are those \( \alpha_i \) for which \( i \in K \), and \( e_i \neq 0 \), and

\[
d_{\overline{K}}(x) = \prod_{i \in H \cap \overline{K}} (x - \alpha_i),
\]

111
whose roots are those $\alpha_i$ for which $i \in \overline{K}$, and $e_i \neq 0$.

By writing equation (5.19) as

$$y_l = \frac{e_l}{g(\alpha_l)} - \frac{v(\alpha_l)}{d_K(\alpha_l)}, \quad (5.21)$$

and multiplying it by $d(\alpha_l) = d_K(\alpha_l)d_{\overline{K}}(\alpha_l)$, where for $l \in \overline{K}$, either $e_l = 0$, or $d_{\overline{K}}(\alpha_l) = 0$, we have the Welch-Berlekamp key equation

$$d(\alpha_l)y_l = r'(\alpha_l). \quad (5.22)$$

By rewriting (5.22) in the form

$$y_l = \frac{r'(\alpha_l)}{d(\alpha_l)}, \quad (5.23)$$

and using rational interpolation we can obtain rational function $\frac{r'(x)}{d(x)}$, whose computation complexity is $O(c^2)$, where $|\overline{K}| = c - n - 1$.

The above discussion demonstrates the correctness of the algorithm below:

**Algorithm 7** Welch-Berlekamp Decoding of the Output Vector.

**Inputs:** output vector of computation $r' = (r_0, \ldots, r_n, r_{n+1}, \ldots, r_{c-1})$, set of $c$ distinct points $T = \{\alpha_j \mid \alpha_j \in GF(2^k)\}$, set of indices $K = \{0, 1, \ldots, n\}$, $\overline{K} = \{0, \ldots, c - 1\} \setminus K$, polynomial $g(x) = \prod_{i \in K} (x - x_i)$.

**Outputs:** polynomials $d(x)$, $r'(x)$.

1. By Lagrange interpolation, interpolate output vector $r'$ in order to get polynomial $r'(x)$
2. if $\deg(r'(x)) \leq n$ and $a_i \in GF(2)$ then $r'(x)$
3. else
4. for $i \in K$ do find $r'(x)$, where $\deg(r') \leq n$
5. evaluate $r'(x)$, at $\alpha_l$, $l \in \overline{K}$, $|\overline{K}| = c - n - 1$
6. determine syndromes $S_l = r_l - r'(x_l)$, $l \in \overline{K}$
7. determine $y_l = \frac{S_l}{g(x_l)}$
8. solve key equation $d(x_l)y_l = r'(x_l)$
9. return $d(x)$, $r'(x)$.  

112
The error locations $\alpha_0, \ldots, \alpha_{t-1}$ are the roots of the polynomial $d(x)$, while the error values are obtained from the following relation:

$$r'(x)g(x) = S(x)d(x),$$

such that

$$e_i = S_i - S(\alpha_i), \quad i \in \{0, \ldots, t-1\}.$$

**Example 5.2.13.** Assume that we want to protect computation in the finite binary field $GF(2^3) \cong GF(2)[x]/ < x^3 + x + 1 >$, where $\alpha$ is primitive root of the primitive polynomial $x^3 + x + 1$, i.e., $\alpha^3 + \alpha + 1 = 0$.

Let the inputs to the computation be the following finite field elements in the polynomial representation $h_1(x), g_1(x), h_2(x), g_2(x) \in GF(2)[x]/ < x^3 + x + 1>$ where

$$h_1(x) = x + 1, \quad g_1(x) = x^2 + 1,$$

$$h_2(x) = x^2, \quad g_2(x) = x^2 + 1.$$

Assume that we want to compute following expression

$$[(h_1(x)g_1(x)) + (h_2(x)g_2(x))] \pmod{f(x)}.$$  \hfill (5.24)

Since, the maximum possible degree of expression (5.24) is 4, the minimum number of polynomial evaluations is 5. Also we want to correct single errors, so we add $c - n - 1 = 2$ extra channels. Therefore, we chose following set $T$ of distinct points

$$T = \{0, 1, \alpha, \alpha^2, \alpha^3, \alpha^4, \alpha^5\}$$

$$= \{\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6\},$$

where new encoded computation will happen in the ring $R' = GF(2^3)[x]/ \langle \prod_{i=0}^{6}(x - \alpha_i) \rangle$. Componentwise addition $\oplus$ and componentwise multiplication $\odot$ are operations in the ring $R'$.  

113
At elements of the set $T$ we evaluate input finite field elements in the polynomial representation $h_1(x), g_1(x), h_2(x), g_2(x) \in GF(2)[x]/ < x^3 + x + 1 >$. Therefore, we have

- $h_1(x)$ at $T$ is $u_1 = (1, 0, \alpha^3, \alpha^6, \alpha, \alpha^5, \alpha^4)$,
- $g_1(x)$ at $T$ is $u_2 = (1, 0, \alpha^6, \alpha^5, \alpha^2, \alpha^3, \alpha)$,
- $h_2(x)$ at $T$ is $v_1 = (0, 1, \alpha^2, \alpha^4, \alpha^6, \alpha, \alpha^3)$,
- $g_2(x)$ at $T$ is $v_2 = (1, 0, \alpha^6, \alpha^5, \alpha^2, \alpha^3, \alpha)$,

such that

$$(h_1(x) \circ g_1(x)) \oplus (h_2(x) \circ g_2(x)) = (1, 0, \alpha^4, \alpha, 1, \alpha^2, 1).$$

Assume that an adversary induces faults into point $P/ GF(2^3)$ by inducing faults into one of 7 processors by some physical set up, causing attacked processor to be faulty, such that erroneous output of computation is

$$r' = (1, 0, \alpha^4, \alpha^2, 1, \alpha^2, 1).$$

Now, we select set of $n + 1 = 5$ indices $K = \{0, 1, 2, 3, 4\}$, and we determine polynomial $r'(x)$ whose degree is at most 4.

Therefore, the by interpolating vector

$$(r') = (1, 0, \alpha^4, \alpha^2, 1) \text{ at } (0, 1, \alpha, \alpha^2, \alpha^3)$$

we get

$$r'(x) = \alpha^3 x^4 + x^3 + \alpha^6 x^2 + \alpha^4 x + 1.$$

Given index selection $K$ we evaluate $r'(x)$ at $\alpha_5 = \alpha^4$, $\alpha_6 = \alpha^5$, i.e., $r'(\alpha^4) = \alpha$, $r'(\alpha^5) = \alpha^5$, such that

$$S = r' - (1, 0, \alpha^4, \alpha^2, 1, \alpha, \alpha^5) = (0, 0, 0, 0, \alpha^4, \alpha^4).$$

Therefore, syndromes are $S_5 = \alpha^4$, $S_6 = \alpha^4$.

Let now define polynomial

$$g(x) = x(x - 1)(x - \alpha)(x - \alpha^2)(x - \alpha^3).$$
New interpolated data is given by $\alpha_5 = \alpha^4$, $\alpha_6 = \alpha^5$, and
$$y_5 = \frac{S_5}{g(\alpha_5)} = 1, \quad \text{and} \quad y_6 = \frac{S_6}{g(\alpha_6)} = \alpha^5.$$The problem is to determine polynomials $d(x), r'(x)$ from
$$d(x_5) y_5 = r'(x_5)$$
$$d(x_6) y_6 = r'(x_6).$$By rational interpolation at points $(1, \alpha^4), (\alpha^5, \alpha^5)$ we obtain that $d(x) = \alpha (\alpha^2 + x)$ and $r'(x) = \alpha^9$. Therefore, error locations are the roots of the polynomial $d(x)$, i.e., $\alpha_3 = \alpha^2$, while error values are obtained by
$$S(x) = \frac{r'(x) g(x)}{d(x)} = \alpha x^4 + \alpha^6 x^2 + \alpha^5 x,$$such that $e_3 = S_3 - S(\alpha^2) = \alpha^4$, so that $r_3 - e_3 = \alpha$. Therefore, the correct output vector of computation is
$$r' = (1, 0, \alpha^4, \alpha, 1, \alpha^2, 1).$$By interpolating $r'$ we have $r'(x) = x^4 + x^3 + x + 1$, i.e., $r'(x) \mod f(x) = x^2 + x$.\qed

5.3 Conclusion

In this chapter we have discussed fault tolerant computation in a finite field for use in elliptic curve cryptosystems. We have proposed a means of protecting computation of the finite field $GF(2^k)$ against active side-channel attacks, i.e., fault attacks. Computation is done in two stages, firstly in the larger polynomial ring by decomposing computation in the parallel, mutually independent, modular/identical channels, and secondly, result is reduced modulo irreducible polynomial $f(x)$. This offers a great advantage, since computations are mutually independent (fault effects do not spread to the other channels), and they are performed over modular channels (CRT based FTC), or over the same field (LI based FTC). Our approach is based on the use of well know error correcting codes, i.e., redundant residue polynomial codes which are generalization of the Reed-Solomon
codes and on the use of the first original approach of Reed-Solomon codes. By assuming proposed fault models from Chapter 3, our proposed schemes provide protection against their error propagation. Since the computation is decomposed into parallel, mutually independent channels, the adversary can use either RFM, or AFM, or SFM per channel. Fault-tolerant computation is obtained by the use of redundancy. By adding $d + t$, $d \geq t$ redundant modular/identical channels we can correct up to $t$ faulty processors, and simultaneously detect $d$ faulty processors. Also, efficient error correction is possible through the use of Euclid’s decoding algorithm for CRT based FTC and Welch-Berlekamp decoding algorithm for LI based FTC. Moreover, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost. Also, we can say that CRT based FTC is generalization of the LI based FTC. LI FTC is more efficient in the sense that computation is done over identical channels.

In the next chapter we construct the new algorithmic countermeasures that are based on those proposed schemes.
Chapter 6

Montgomery FTC in Finite Fields
by Embedding in Rings

Highly reliable hardware countermeasures against fault attacks are very expensive and most moderately priced countermeasures are only capable of detecting specific attacks. New fault attacks are being developed frequently, so detecting currently known forms of physical tampering will most probably not be sufficient against future developments. Although preventing an error is always the best countermeasure, this cannot be guaranteed by most hardware countermeasures. Therefore, algorithmic countermeasures are needed, which do not depend on the physical attack, only on the induced fault. Moreover, they are more cost efficient and easier to deploy.

In this chapter we construct new algorithms which are immune against fault attacks, i.e., fault tolerant residue representation (RR) modular multiplication algorithm in Subsection 6.1.2 and fault tolerant Lagrange representation (LR) modular multiplication algorithm in Subsection 6.2.2. These algorithms provide fault tolerant computation in $GF(2^k)$ for use in elliptic curve cryptosystems. By assuming the fault models from Chapter 3 we test the error correcting/detecting capability of our proposed algorithmic countermeasures. We provide analysis of the error detection and correction capabilities of our proposed algorithmic countermeasures in Subsection 6.1.4 and 6.2.3, as well as an analysis of an error correction algorithm.
6.1 Montgomery Residue Representation FTC in Finite Fields

In this section we propose a fault tolerant residue representation modular multiplication algorithm for fault tolerant computation in the finite field $GF(2^k)$ for use in elliptic curve cryptosystems.

6.1.1 Multiplication in $GF(2^k)$

Koc and Acar give in [54] a finite field $GF(2^k)$ analogue of the Montgomery multiplication for modular multiplication of integers [70]. Elements of the finite field are considered as a polynomials of degree $< k$, while $p(x) = x^k$ is used as a Montgomery factor, since reduction modulo $x^k$, and division modulo $x^k$ consist of ignoring the terms of degree larger than $k$ for the remainder operation, and shifting the polynomial to the right by $k$ places for the division. Instead of computing $a(x)b(x) \in GF(2^k)$ for $a(x), b(x) \in GF(2^k)$ it computes $a(x)b(x)p^{-1}(x) \mod f(x)$, where $f(x)$ is a irreducible polynomial of degree $k$ with coefficients in $GF(2)$, and $p^{-1}(x)$ is inverse of $p(x)$ modulo $f(x)$. The Montgomery multiplication method requires that $p(x)$ and $f(x)$ are relatively prime, i.e., $gcd(p(x), f(x)) = 1$, such that by an Extended Euclidean Algorithm $p(x)p^{-1}(x) + f(x)f'(x) = 1$. Bajard et al. The authors of [7] first remarked that Koc and Acar’s algorithm extends to any extension field $GF(p^k)$. In the polynomial basis representation, the elements of $GF(p^k)$ can be modeled as the polynomials in $GF(p)[x]$ of degree at most $k − 1$. Let $f(x)$ be
a monic irreducible polynomial of degree \( k \), and let \( p(x) = x^k \) be a *Montgomery factor* such that \( \gcd(p(x), f(x)) = 1 \). Then given \( a(x), b(x) \in GF(p)[x]/ < f(x) > \), Algorithm 9 can be used to compute \( a(x)b(x)p^{-1}(x) \mod f(x) \). Bajard et al. [9] modified Algorithm 9 by allowing the polynomial \( p(x) \) to be any polynomial of degree \( k \) satisfying \( \gcd(p(x), f(x)) = 1 \), and replacing division by \( p(x) \) in step 2 of Algorithm 9 by a multiplication by \( p^{-1}(x) \) modulo another given polynomial \( p'(x) \). This operation is only possible if \( \gcd(p(x), p'(x)) = 1 \). Algorithm 10 computes \( a(x)b(x)p^{-1}(x) \mod f(x) \) for any relatively prime polynomials \( p(x) \) and \( p'(x) \) satisfying \( \gcd(p(x), f(x)) = 1 \) and \( \gcd(p(x), p'(x)) = 1 \).

**Algorithm 9** Montgomery Multiplication over \( GF(p^k) \)

**Inputs:** \( a(x), b(x) \in GF(p)[x], \deg(a(x)), \deg(b(x)) \leq k - 1 \); irreducible polynomial \( f(x) \in GF(p)[x], \deg(f(x)) = k, p(x) = x^k \)

**Output:** \( a(x)b(x)p^{-1}(x) \mod f(x) \)

1. \( q(x) \leftarrow -a(x)b(x)f'(x) \mod p(x) \)
2. \( r(x) \leftarrow (a(x)b(x) + q(x)f(x))/p(x) \)

**Algorithm 10** Montgomery Multiplication in \( GF(p^k) \)

**Inputs:** \( a(x), b(x) \in GF(p)[x]/ < f(x) > \), irreducible polynomial \( f(x) \in GF(p)[x], \deg(f(x)) = k, \deg(a(x)), \deg(b(x)) \leq k - 1, \deg(p(x)) = \deg(p'(x)) \geq k \), s.t. \( \gcd(p(x), f(x)) = \gcd(p(x), p'(x)) = 1 \)

**Output:** \( a(x)b(x)p^{-1}(x) \mod f(x) \)

1. \( q(x) \leftarrow -a(x)b(x)f'(x) \mod p(x) \)
2. \( r(x) \leftarrow (a(x)b(x) + q(x)f(x))p^{-1}(x)(mod p'(x)) \)

**Lemma 6.1.1** ([9]). *Algorithm 10 is correct and returns* \( a(x)b(x)p^{-1}(x) \mod f(x) \).

*Proof.* In Step 1, \( q(x) \) is computed such that \( p(x) | (a(x)b(x) + q(x)f(x)) \). Indeed, \( a(x)b(x) + q(x)f(x) \equiv a(x)b(x) - a(x)b(x)f'(x)f(x) \equiv 0 \mod p(x) \), and \( \deg(a(x)b(x) + q(x)f(x)) = 1 \). Therefore, \( a(x)b(x)p^{-1}(x) \mod f(x) = a(x)b(x)p^{-1}(x) \mod f(x) \).
the case of residues in fault tolerant computation in the field based on trinomial residue arithmetic. We consider this algorithm, i.e., Algorithm 11 for \( \text{GF} \) field \( \text{GF}(q) \).

Algorithm 11 Residue Representation Modular Multiplication.

**Inputs:** \( a_i(x), b_i(x), a_{v+j}(x), b_{v+j}(x), f_{v+j}(x), i, j = 1, \ldots, v \). Precomputed: \( f'_i(x), p'_{v+j}(x), k_{v+j}(x), k_i(x), i, j = 1, \ldots, v, v \times v \) matrices \( w, w' \).

**Output:** \( (r_1(x), \ldots, r_v(x)) \).

1. \( (t_1(x), \ldots, t_v(x)) \leftarrow (a_1(x), \ldots, a_v(x)) \otimes (b_1(x), \ldots, b_v(x)) \)
2. \( (q_1(x), \ldots, q_v(x)) \leftarrow (t_1(x), \ldots, t_v(x)) \otimes (f'_1(x), \ldots, f'_v(x)) \)
3. Change of RR: \( (q_1(x), \ldots, q_v(x)) \rightarrow (q_{v+1}(x), \ldots, q_{2v}(x)) \)
4. \( (r_{v+1}(x), \ldots, r_{2v}(x)) \leftarrow [(t_{v+1}(x), \ldots, t_{2v}(x)) \oplus (q_{v+1}(x), \ldots, q_{2v}(x)) \otimes (f_{v+1}(x), \ldots, f_{2v}(x))] \otimes (p'_{v+1}(x), \ldots, p'_{2v}(x)) \)
5. Change of RR: \( (r_{v+1}(x), \ldots, r_{2v}(x)) \rightarrow (r_1(x), \ldots, r_v(x)) \).

field \( \text{GF}(2^k) \) is considered as the set of polynomials modulo a irreducible polynomial \( f(x), \deg(f(x)) = k \), i.e.,

\[
\text{GF}(2)[x]/ \langle f(x) \rangle = \{a_0 + \ldots + a_{k-1}x^{k-1} | a_i \in \text{GF}(2)\},
\]

and \( \{m_1(x), \ldots, m_v(x)\} \) is a set of \( v \) relatively prime polynomials from the polynomial ring \( \text{GF}(2)[x] \), such that

\[
n = \deg(m_1(x)) + \ldots + \deg(m_v(x)) \geq k,
\]
where
\[ m(x) = m_1(x) \cdot \ldots \cdot m_v(x), \quad m(x) \in GF(2)[x]. \]

Then by the Chinese Remainder Theorem (5.1) there exists a ring isomorphism between these two algebraic structures, i.e.,
\[ GF(2)[x]/< m(x)> \cong GF(2)[x]/< m_1(x)> \times \ldots \times GF(2)[x]/< m_v(x)> . \]

Therefore, all \( a(x) \in GF(2^k) \) have a corresponding residue representation, i.e.,
\[ a(x) \leftrightarrow a = (a_1(x), \ldots, a_v(x)), \]
where \( a_i(x) \equiv a(x) (mod m_i(x)) \) for \( i = 1, \ldots, v. \)

The Montgomery factor is then
\[ p(x) = \prod_{i=1}^{v} m_i(x), \]
such that \( gcd(p(x), f(x)) = 1 \). The computation is performed in parallel, i.e., \( q_i(x) = a_i(x) b_i(x) f'_i(x), \) \( i = 1, \ldots, v \) where \( f'(x) \equiv f^{-1}(x)(mod p(x)) \). Since the inverse modulo \( p(x) \) of \( p(x) \) does not exist, \( r(x) = (a(x) b(x) + q(x) f(x)) p^{-1}(x) \) is evaluated by choosing a polynomial
\[ p'(x) = \prod_{i=v+1}^{2v} m_i(x), \]
where \( gcd(p(x), p'(x)) = 1 \), and \( gcd(m_i(x), m_j(x)) = 1 \) for \( i \neq j, \) \( i, j = 1, \ldots, 2v \). Therefore, the change of the residue representation \( (q_1(x), \ldots, q_v(x)) \) to \( (q_{v+1}(x), \ldots, q_{2v}(x)) \) is performed by:
\[
\begin{pmatrix}
q_{v+1}(x) \\
q_{v+2}(x) \\
\vdots \\
q_{2v}(x)
\end{pmatrix} =
\begin{pmatrix}
w_{1,v+1}(x) & w_{2,v+1}(x) & \ldots & w_{v,v+1}(x) \\
w_{1,v+2}(x) & w_{2,v+2}(x) & \ldots & w_{v,v+2}(x) \\
\vdots & \vdots & \vdots & \vdots \\
w_{1,2v}(x) & w_{2,2v}(x) & \ldots & w_{v,2v}(x)
\end{pmatrix} \begin{pmatrix}
k_1(x) \\
\vdots \\
k_{v-1}(x) \\
k_v(x)
\end{pmatrix},
\]
where
\[
w_{i,v+j}(x) = \left( \frac{p(x)}{m_i(x)} \right) (mod m_{v+j}(x)), \quad i, j = 1, \ldots, v, \quad \text{and}
\]
\[
k_i(x) = \left( q_i(x) \left( \frac{p(x)}{m_i(x)} \right)^{-1} (mod m_i(x)) \right) mod m_i(x),
\]

121
where \( i = 1, \ldots, v \). Now, computation of step 4 of Algorithm 11 happens in the following direct product ring:

\[
GF(2)[x] / < m_{v+1}(x) > \times \cdots \times GF(2)[x] / < m_v(x) >.
\]

Change of the residue representation \((r_{v+1}(x), \ldots, r_{2v}(x))\) to \((r_1(x), \ldots, r_v(x))\) is achieved by:

\[
\begin{pmatrix}
r_1(x) \\
r_2(x) \\
\vdots \\
r_v(x)
\end{pmatrix} = \begin{pmatrix}
w'_{v+1,1}(x) & w'_{v+2,1}(x) & \cdots & w'_{2v,1}(x) \\
w'_{v+1,2}(x) & w'_{v+2,2}(x) & \cdots & w'_{2v,2}(x) \\
\vdots & \vdots & \ddots & \vdots \\
w'_{v+1,v}(x) & w'_{v+2,v}(x) & \cdots & w'_{2v,v}(x)
\end{pmatrix} \begin{pmatrix}
k_{v+1}(x) \\
k_{v+2}(x) \\
\vdots \\
k_{2v}(x)
\end{pmatrix},
\]

where

\[
w'_{v+i,j}(x) = \left( \frac{p'(x)}{m_{v+i}(x)} \right) (mod m_j(x)), \quad i, j = 1, \ldots, v, \quad \text{and} \quad k_{v+j} = \left( r_{v+j}(x) \left( p'(x)m_{v+j}(x) \right)^{-1} (mod m_{v+j}(x)) \right) m_{v+j}(x), \quad j = 1, \ldots, v.
\]

In the next subsection we extend use of Algorithm 11 to fault tolerant computation in the field \(GF(2^k)\) by use of redundancy.

### 6.1.2 Montgomery Residue Representation FTC

Depending on the security required, to protect computation in the finite field we add redundancy by adding more \((c > v)\) parallel, modular channels than the required minimum, i.e., see Fig. 6.1. The new redundant moduli

\[ m_{v+1}(x), \ldots, m_c(x) \in GF(2)[x] \]

have to be relatively prime to each other and to the non-redundant moduli \(m_1(x), \ldots, m_v(x)\).

Therefore, now computation happens in the larger direct product ring \(R''\)

\[
GF(2)[x] / < m_1(x) > \times \cdots \times GF(2)[x] / < m_v(x) > \times \cdots \times GF(2)[x] / < m_c(x) >,
\]

where

\[
m'(x) = m_1(x) \cdots m_v(x) \cdots m_c(x), \quad m'(x) \in GF(2)[x], \quad \deg(m'(x)) > n,
\]

122
Algorithm 12 Fault Tolerant RR Modular Multiplication.

**Inputs:** \( a_i(x), b_i(x), a_{c+j}(x), b_{c+j}(x), f_{c+j}(x), i, j = 1, \ldots, c > v. \)

**Precomputed:** \( f_i'(x), p_{c+j}^{-1}(x), k_{c+j}(x), k_i(x), i, j = 1, \ldots, c, c \times c \) matrices \( w, w', c > v. \)

**Output:** \( r(x) \in GF(2)[x]/ < m_1(x) > \times \cdots \times GF(2)[x]/ < m_c(x) >. \)

1. \((t_1(x), \ldots, t_c(x)) \leftarrow (a_1(x), \ldots, a_c(x)) \otimes (b_1(x), \ldots, b_c(x))\)
2. \((q_1(x), \ldots, q_c(x)) \leftarrow (t_1(x), \ldots, t_c(x)) \otimes (f_1'(x), \ldots, f_c'(x))\)
3. Change of RR: \((q_1(x), \ldots, q_c(x)) \rightarrow (q_{c+1}(x), \ldots, q_{2c}(x))\)
4. \((r_{c+1}(x), \ldots, r_{2c}(x)) \leftarrow [(t_{c+1}(x), \ldots, t_{2c}(x)) \oplus (q_{c+1}(x), \ldots, q_{2c}(x)) \otimes (f_{c+1}(x), \ldots, f_{2c}(x)) \otimes (p_{c+1}^{-1}(x), \ldots, p_{2c}^{-1}(x))\)
5. Change of RR: \((r_{c+1}(x), \ldots, r_{2c}(x)) \rightarrow (r_1(x), \ldots, r_c(x))\).
6. CRT interpolation: \(r(x) \leftarrow (r_1(x), \ldots, r_c(x))\).

such that

\[ R'' \cong GF(2)[x]/ < m'(x) >. \]

The redundant polynomial moduli have to be of degree larger than the largest degree of the non-redundant moduli, i.e.,

\[ \deg(m_{v+j}(x)) > \max \{ \deg \{ m_1(x), \ldots, m_v(x) \} \}, \quad j = 1, \ldots, c - v, \]

and

\[ \deg \left\{ \frac{m'(x)}{m_j_1(x) \cdots m_j_{c-v}(x)} \right\} \geq n \geq k, \quad (6.1) \]

where \( c - v \) is the added redundancy.

Therefore, all \( a(x) \in GF(2^k) \) have corresponding redundant residue representation, i.e.,

\[ a(x) \leftrightarrow a = (a_1(x), \ldots, a_v(x), \ldots, a_c(x)), \]

where \( a_i(x) = a(x) (\text{mod } m_i(x)) \) for \( i = 1, \ldots, c. \)

Now,

\[ p(x) = \prod_{i=1}^{c} m_i(x), \quad i = 1, \ldots, c \]
is a Montgomery factor, such that \( \gcd(p(x), f(x)) = 1 \), and computation is done in parallel, i.e., \( q_i(x) = a_i(x)b_i(x)f_i(x)(\mod m_i(x)), \ i = 1, \ldots, c \), where \( f'(x) \equiv f^{-1}(x)(\mod p(x)) \). Since, the inverse modulo \( p(x) \) of \( p(x) \) does not exist, \( r(x) = (a(x)b(x) + q(x)f(x))p^{-1}(x) \) is evaluated by choosing polynomial
\[
p'(x) = \prod_{i=c+1}^{2c} m_i(x),
\]
where \( \gcd(p(x), p'(x)) = 1 \), and \( \gcd(m_i(x), m_j(x)) = 1 \) for \( i \neq j, i, j = 1, \ldots, 2c \). Therefore, change of the residue representation \( (q_1(x), \ldots, q_c(x)) \) to \( (q_{c+1}(x), \ldots, q_{2c}(x)) \) is done by:
\[
\begin{pmatrix}
q_{c+1}(x) \\
q_{c+2}(x) \\
\vdots \\
q_{2c}(x)
\end{pmatrix} = \begin{pmatrix}
w_{1,c+1}(x) & w_{2,c+1}(x) & \ldots & w_{c,c+1}(x) \\
w_{1,c+2}(x) & w_{2,c+2}(x) & \ldots & w_{c,c+2}(x) \\
\vdots & \vdots & \ddots & \vdots \\
w_{1,2c}(x) & w_{2,2c}(x) & \ldots & w_{c,2c}(x)
\end{pmatrix} \begin{pmatrix}
k_1(x) \\
k_2(x) \\
\vdots \\
k_{c-1}(x) \\
k_c(x)
\end{pmatrix},
\]
where
\[
w_{i,c+j}(x) = \left( \frac{p(x)}{m_i(x)} \right)(\mod m_{c+j}(x)), \ i, j = 1, \ldots, c, \text{ and }
\]
\[
k_i(x) = q_i(x) \left( \frac{p(x)}{m_i(x)} \right)^{-1}(\mod m_i(x)) mod m_i(x),
\]
\( i = 1, \ldots, c \). Now, the computation of step 4 of Algorithm 4 happens in the following direct product ring:
\[
GF(2)[x]/ < m_{c+1}(x) > \times \ldots \times GF(2)[x]/ < m_{2c}(x) > .
\]
The change of the residue representation \( (r_{c+1}(x), \ldots, r_{2c}(x)) \) to \( (r_1(x), \ldots, r_c(x)) \) is achieved by:
\[
\begin{pmatrix}
r_1(x) \\
r_2(x) \\
\vdots \\
r_c(x)
\end{pmatrix} = \begin{pmatrix}
w'_{c+1,1}(x) & w'_{c+1,2}(x) & \ldots & w'_{c,1}(x) \\
w'_{c+2,1}(x) & w'_{c+2,2}(x) & \ldots & w'_{c,2}(x) \\
\vdots & \vdots & \ddots & \vdots \\
w'_{c+1,c}(x) & w'_{c+2,c}(x) & \ldots & w'_{c,c}(x)
\end{pmatrix} \begin{pmatrix}
k_{c+1}(x) \\
k_{c+2}(x) \\
\vdots \\
k_{2c}(x)
\end{pmatrix},
\]
where
\[
w'_{c+i,j}(x) = \left( \frac{p'(x)}{m_{c+i}(x)} \right)(\mod m_j(x)), \ i, j = 1, \ldots, c, \text{ and }
\]
\[
k_{v+j} = r_{v+j}(x) \left( \frac{p'(x)}{m_{v+j}(x)} \right)^{-1}(\mod m_{v+j}(x)) mod m_{v+j}(x).
\]
Theorem 6.1.2. If there are no fault effects, Algorithm 12 will determine a unique polynomial of degree $< n$ where $n \geq k$ with coefficients $a_i \in GF(2)$, otherwise, it will be of degree $\geq n \geq k$.

Proof. Assume that there is no fault induced, then the theorem follows. Given $a(x), b(x) \in GF(2)[x]/< f(x) >$, $\deg (f(x)) = k$ in step 1 of Algorithm 12 we compute $t(x) = a(x)b(x)$ of degree $\leq 2k - 2$. In step 2 of Algorithm 12 we compute $q(x) = t(x)f'(x) \mod p(x)$ of degree at most $\deg(p(x)) - 1$, where $\deg(p(x)) > n \geq k$. We have that $\deg(q(x)f(x)) \leq \deg(p(x)) - 1 + k$. Therefore, since $2k - 2 < \deg(p(x)) - 1 + k$ and $\deg(p(x)) = \deg(p'(x)) = k$ we have that $r(x) = (t(x) + q(x)f(x))p^{-1}(x)$ is of degree at most $k - 1$. Otherwise, it will be of degree $\geq n \geq k$. 

\[ \square \]

Definition 6.1.3. The set of correct results of computation is

\[ C = \{ r(x) \in GF(2)[x]/ < m'(x) > | \deg (r(x)) < n \geq k \} . \]

6.1.3 Complexity

The Chinese Remainder Algorithm 5 is only applied at the end of the computation, and its complexity is:

Theorem 6.1.4 ([35]). Let $GF(2)[x]$ be polynomial ring over a field $GF(2)$, $m_1(x), \ldots, m_c(x) \in GF(2)[x]$, $d_i = \deg(m_i(x))$ for $1 \leq i \leq c$, $l = \deg(m'(x)) = \sum_{1 \leq i \leq c} d_i$, and $r_i(x) \in GF(2)[x]$ with $\deg(r_i(x)) < d_i$. Then the unique solution $r'(x) \in GF(2)[x]$ with $\deg(r'(x)) < l$ of the Chinese Remainder Problem $r'(x) \equiv r_i(x) (\mod m_i(x))$ for $1 \leq i \leq c$ for polynomials can be computed using $O(l^2)$ operations in $GF(2)$.

Theorem 6.1.5. Computational complexity of the Algorithm 12 is $O(l^2)$.

Proof. Let $\deg (m_1(x)) = d_1, \ldots, \deg (m_c(x)) = d_c$, and $d_1 + \ldots + d_c = l$. Then the complexity of step 1 is $\sum_{i=1}^{c} O(d_i^2) < O((\sum_{i=1}^{c} d_i)^2) = O(l^2)$, same as of step 2. In step 3, matrix and vector are precomputed, and their multiplication has complexity $O(sc^2) + \sum_{i=1}^{c} O(d_{i+c}) < O(l^2)$, where $s$ is degree of two polynomials multiplied such that $s \leq \max_i (2d_i)$, $i = 1, \ldots, c$. The complexity of computing step 4 is
In step 5, \(O(s'(c^2)) + \sum_{i=1}^{c} O(d_i) < O(l^2)\), where \(s'\) is degree of two polynomials multiplied such that \(s' \leq \max_i (2d_i), i = c + 1, \ldots, 2c\). Step 6 has complexity \(O(l^2)\). Therefore, the complexity of the algorithm \(12\) is \(O(l^2)\).

Assume a polynomial residue representation using \(c\) degree \(d\) trinomials, such that \(cd > k\). We note that in Algorithm 12, steps 1, 2 and 4 are accomplished in parallel. In step 1 we perform \(c\) multiplications \(a_i(x)b_i(x) \mod m(x)\). By using Mastrovito’s algorithm for trinomials \[96\] we require \(d^2\) AND and \(d^2 - 1\) XOR operations. Therefore, the cost of step 1 is \(cd^2\) AND, and \(c(d^2 - 1)\) XOR. In steps 2 and 4 we perform \(3c\) constant multiplications expressed as \(3c\) matrix-vector products of the form \(ZU\), where \(Z\) is a \(d \times d\) precomputed matrix. The complexity is \(3cd^2\) AND, and \(3cd(d - 1)\) XOR, plus \(c\) additions in step 4. The complexity for steps 1, 2 and 4 is: \(4cd^2\) AND, and \(4cd^2 - 2cd - c\) XOR operations, with a latency \(4T_A + (1 + 4\lceil \log_2 d \rceil) T_X\), where \(T_A\) and \(T_X\) represent delay for one AND gate, and one XOR gate respectively. In steps 3, 5, and 6, CRT interpolation requires \((c^2 + c)\) modular multiplications modulo trinomial of degree \(d\), and precomputation of \((c^2 + c)\) matrices \(d \times d\). In case when there is no clue about coefficients of the matrices then upper bound for the cost of one matrix-vector product is \(d^2\) AND and \(d(d - 1)\) XOR operations, with latency \(T_A + \lceil \log_2 d \rceil T_X\). Therefore, the complexity of steps 3, 5, and 6 is \(3(c^2 + c)d^2\) AND, and \(3(c^2 + c)d(d - 1)\) XOR gates, with latency \(3T_A + 3\lceil \log_2 d \rceil T_X\). The gate count is:

\[
\#\text{AND} : 7cd^2 + 3c^2 d^2 , \\
\#\text{XOR} : cd (3c (d - 1) + 7d - 5) - c ,
\]

and the delay is equal to:

\[7T_A + (1 + 7\lceil \log_2 d \rceil) T_X .\]

Let assume that \(c = l^x\) and \(d = l^{1-x}\) \((cd = l)\), then

\[
\#\text{AND} : 3l^2 + 7l^{2-x} , \\
\#\text{XOR} : 3l^2 - 5l + 7l^{2-x} - 3l^{x+1} - l^x ,
\]

and the latency is

\[7T_A + (1 + 7\lceil \log_2 l^{1-x} \rceil) T_X .\]
The area complexity is usually given depending on the number of XOR gates. Therefore, best asymptotic area complexity reached for \( x = 2/5 \) is \( O(l^2) \), i.e.,
\[
3l^2 - 5l + 7l^{8/5} - 3l^{7/5} - l^{2/5}.
\]

*Computational efficiency.* To have efficient reduction in the smaller polynomial rings \( GF(2)[x]/ < m_i(x) >, i = 1, \ldots, c \), modulus \( m'(x) \) have to be chosen as a product of the pairwise relatively prime polynomials which are of the special low Hamming weight, leading to efficient modular reduction. Therefore, a smaller ring modulus can be chosen to be in the *Mersenne form* \( x^n - 1 \), or *pseudo-Mersenne form* \( x^n + u(x) \), where polynomial \( u(x) \) is of low weight. In \( GF(2^k) \), the reduction is relatively inexpensive if the field is constructed by choosing the reduction polynomial to be a *trinomial*, i.e., \( x^k + x^m + 1 \) with \( m < k/2 \), or a *pentanomial* (if no trinomial available) \( x^k + x^m + x^n + x^h + 1 \) with \( h < n < m < k/2 \), see Table 5.1, Table 5.2.

### 6.1.4 Error Detection and Correction

Let us assume that here is one processor per independent channel as in Fig. 6.1. Let us assume that we have \( c \) processors, where each processor computes \( i \)-th polynomial residue and \( i \)-th residue operations. Also, we assume that all precomputed inputs are error free, as well as the *Chinese Remainder Algorithm* 5 at the end of the computation. As before, we assume that a fault attack induces faults into processors by some physical means. As a reaction, the attacked processor malfunctions, and it does not compute the correct output given its input. We are concerned with the effect of a fault as it manifests itself in a modified data, or a modified program execution. Therefore, we consider the fault models presented in Chapter 3. Since computation is decomposed into parallel, mutually independent channels, the adversary can use either RFM, or AFM, or SFM per channel.

Assume that at most \( c - v \) channels have faults. Let \( r' \in R'' \) be the computed vector with \( c \) components, where \( e_j(x) \in GF(2)[x]/ < m_j(x) > \) is the error polynomial at \( j \)-th position; then the computed component at the \( j \)-th positions is \( b_j = r(x)(mod m_j(x)) + \)
Figure 6.1: Fault tolerant computation over the finite field $GF(2^k)$. 
Here, we have assumed that the set of error positions are computed vector $r$ polynomial $M$.

Since, 

\[ e(x) = \sum_{i=1}^{\lambda} \frac{m'(x)}{m_{j_i}(x)} \prod_{i=1}^{\lambda} m_{j_i}(x) T_{j_i}(x) \] 

and each processor will have as an output component 

\[ b_j = \begin{cases} 
(r(x) + e_j(x))(\text{mod } m_j(x)), & j \in \{j_1, \ldots, j_\lambda\}, \\
(r(x))(\text{mod } m_j(x)), & \text{else.} 
\end{cases} \]

Here, we have assumed that the set of error positions are $\{j_1, \ldots, j_\lambda\}$. By CRT the computed vector $r' \in \mathbb{R}^n$ with corresponding set of $c$ moduli $m_i(x)$ gives as a output polynomial $r'(x) \in GF(2)[x]/ < m'(x) >$,

\begin{align*}
  r'(x) &\equiv \left( \sum_{1 \leq i \leq c} r_i(x) T_i(x) M_i(x) \right) (\text{mod } m'(x)) \\
  &= \left( \sum_{1 \leq i \leq c} r_i(x) T_i(x) M_i(x) \right) (\text{mod } m'(x)) + \left( \sum_{1 \leq i \leq \lambda} e_{j_i}(x) T_{j_i}(x) M_{j_i}(x) \right) (\text{mod } m'(x)) \\
  &= (r(x) + e(x)) (\text{mod } m'(x)) ,
\end{align*}

(6.2)

where $M_i(x) = \frac{m'(x)}{m_i(x)}$, polynomials $T_i(x)$ are computed by solving congruences $T_i(x) M_i(x) \equiv 1 (\text{mod } m_i(x))$, $M_{j_i}(x) = \frac{m'(x)}{m_{j_i}(x)}$, polynomials $T_{j_i}(x)$ are computed by solving congruences $T_{j_i}(x) M_{j_i}(x) \equiv 1 (\text{mod } m_{j_i}(x))$. Moreover, $r(x) (\text{mod } m'(x))$ is correct output polynomial of degree $< n$ and $e(x) (\text{mod } m'(x)) \in GF(2)[x]/ < m'(x) >$ is the error polynomial such that:

**Theorem 6.1.6.** Let $e_{j_i}(x) \in GF(2)[x]/ < m_{j_i}(x) >$ be error polynomial at positions $j_i$, $i \in \{1, \ldots, \lambda\}$, $\lambda \leq c - \nu$ then $\deg(e(x)) \geq n \geq k$.

**Proof.** We have that

\[ e(x) = \left( \sum_{1 \leq i \leq \lambda} e_{j_i}(x) T_{j_i}(x) M_{j_i}(x) \right) (\text{mod } m'(x)) \\
  = e_{j_1}(x) T_{j_1}(x) M_{j_1}(x) + \ldots + e_{j_\lambda}(x) T_{j_\lambda}(x) M_{j_\lambda}(x) \\
  = \frac{m'(x)}{m_{j_1}(x) \cdot \ldots \cdot m_{j_\lambda}(x)} \sum_{i=1}^{\lambda} \frac{m_{j_i}(x) \cdots \cdot m_{j_\lambda}(x)}{m_{j_i}(x)} T_{j_i}(x) e_{j_i}(x).
\]

(6.3)

Since,

\[ \deg \left( \sum_{i=1}^{\lambda} \frac{m_{j_i}(x)}{m_{j_i}(x)} T_{j_i}(x) e_{j_i}(x) \right) < \deg \left( \frac{m'(x)}{\prod_{i=1}^{\lambda} m_{j_i}(x)} \right), \]

and by (6.1), $\deg \left( \frac{m'(x)}{\prod_{i=1}^{\lambda} m_{j_i}(x)} \right) \geq n \geq k$, we have that $\deg(e(x)) \geq n \geq k$. 

\[ \square \]
Therefore, the faulty processors affect the result in an additive manner.

**Lemma 6.1.7.** The error is masked iff \( \text{deg}(e(x)) < n \) where \( n \geq k \).

**Proof.** Let \( \text{deg}(e(x)) < n, n \geq k \) in (6.2), then \( \text{deg}(r'(x)) < n \), i.e., \( r'(x) \in C \). \( \square \)

**Lemma 6.1.8.** Let the degree of the ring modulus \( m'(x) \) be \( n \geq k \), and let \( c > v \) be the number of parallel, independent, modular channels (or number of processors). Then if up to \( c - v \) channels fail, the output polynomial \( r'(x) \notin C \).

**Proof.** By referring to (6.2), since if \( \text{deg}(e(x)) \geq n \), the output polynomial \( r'(x) \) has to be such that \( \text{deg}(r'(x)) \geq n \). By Definition 6.1.3, \( r'(x) \notin C \). \( \square \)

**Lemma 6.1.9.** Let the degree of the ring modulus \( m'(x) \) be \( n \geq k \), and let \( c > v \) be number of parallel, independent, modular channels (or number of processors). If there is no faulty processors then \( r'(x) \in GF(2)[x]/<f(x)> \).

**Proof.** If there are no faulty processors, then clearly no errors occurred, and \( \text{deg}(r'(x)) \leq n \), so that \( r'(x) = r(x), r'(x) \in C \). Therefore, \( r'(x) \in GF(2)[x]/<f(x)> \). \( \square \)

It is straightforward to appeal to the standard coding theory result below, to state the error detection and correction capability of our set up:

**Theorem 6.1.10.** (i) If the number of parallel, mutually independent, modular, redundant channels is \( d + t \leq c - v \) (\( d \geq t \)), then up to \( t \) faulty processors can be corrected, and up to \( d \) simultaneously detected. (ii) By adding \( 2t \) redundant independent channels at most \( t \) faulty processors can be corrected.

While it is true that arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed here, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost.

Decoding is based on the *Extended Euclidean Algorithm* discussed in Subsection 5.1.4, here we present the algorithm as Algorithm 13 for completeness.
Algorithm 13 Euclid’s Decoding Algorithm

Input: output vector of computation $r' = (r_1(x), r_2(x), \ldots, r_c(x)) \in R^c$

Output: $r(x) \in GF(2)[x]/ < f(x) >$

1. By CRT algorithm compute $r'(x)$

2. if $\deg(r'(x)) < n \geq k$ then

3. $r(x) = r'(x)$

4. else

5. $t_{-1}(x) = 0$, $t_0(x) = 1$, $d_{-1}(x) = m'(x)$, $d_0(x) = r'(x)$

6. $j = 1$

7. while $\deg(d_j(x)) > \sum_{i=1}^v \deg(m_i(x)) + \sum_{i=1}^\lambda \deg(m_{ji}(x))$ do

8. $d_{j-2}(x) = q_j(x)d_{j-1}(x) + d_j(x)$; $\deg(d_j(x)) < \deg(d_{j-1}(x))$

9. $t_j(x) = t_{j-2}(x) - q_j(x)t_{j-1}(x)$

10. $j = j + 1$

11. return $r(x) = \frac{d_j(x)}{t_j(x)}$

Example 6.1.11. Assume that we want to protect computation in the finite binary extension field $GF(2^3) \cong GF(2)[x]/ < x^3 + x + 1 >$. Let the inputs to the computation be the following finite field elements: $a(x) = x$, $b(x) = x + 1$. We want to compute following expression $\left(a(x)b(x)\right) \mod f(x)$, where $f(x) = x^3 + x + 1$. Let $R[x] = GF(2)[x]/ < m'(x) >$, where

\[
m'(x) = x^2(x^2 + x + 1)(x^3 + x^2 + 1)(x^4 + x + 1)
= m_1(x)m_2(x)m_3(x)m_4(x).
\]

Now, $v = 2$, $c - v = 2$ and the error correction capability is $t = 1$. Therefore, computation will happen with encoded field elements in the following direct product ring:

\[
GF(2)[x]/ < m_1(x) > \times \cdots \times GF(2)[x]/ < m_4(x) >,
\]

where

\[
a(x) \leftrightarrow a = (x, x, x, x),
\]

\[
b(x) \leftrightarrow b = (x + 1, x + 1, x + 1, x + 1),
\]
\[ f'(x) \leftrightarrow f' = (x + 1, x + 1, x, x^2 + 1), \]
such that
\[ a \odot b = (x, 1, x^2 + x, x^2 + x), \]
\[ q = (x, x + 1, 1, x^3 + x^2 + 1), \]
where \( \odot \) is componentwise multiplication. Also, by \( \oplus \) we denote componentwise addition. Let \( \{m_5(x), m_6(x), m_7(x), m_8(x)\} \) be new set of residues, such that \( \gcd(m_i(x), m_j(x)) = 1 \), \( i, j = 1, \ldots, 8 \), and \( p'(x) = \prod_{i=5}^{8} m_i(x), \gcd(m'(x), p'(x)) = 1 \), i.e.,
\[
p'(x) = (x^4 + x^3 + 1) (x^5 + x^3 + x^2 + x + 1) (x^6 + x^5 + x^2 + x + 1) (x^7 + x + 1) = m_5(x)m_6(x)m_7(x)m_8(x).
\]
Therefore, change of residue representation of \( q \) is done by:
\[
w = \begin{pmatrix}
x^2 + x & x^3 + x^2 + x & x^3 + 1 & x^3 + x + 1 \\
 x^4 + x^3 + 1 & x^2 + x + 1 & x^3 + x^2 + x + 1 & x^4 + x^3 + x^2 + x + 1 \\
x^5 + x^4 + x^3 + x^2 + x + 1 & x^5 + x^2 & x^4 + x^3 + x^2 + x + 1 & x^5 + x^2 + 1 \\
x^6 + x^4 + x^3 + 1 & x^4 + x^2 + x & x^6 + x^5 + 1 & x^3 + x^2 + x + 1
\end{pmatrix}
\]
\[
k = \begin{pmatrix}
x \\
x \\
x^2 + x \\
x^3
\end{pmatrix},
\]
such that \( q' = wk = (x + 1, x^2, x^3 + x^2 + x, x^5 + x^2 + 1) \). Now, the computation happens in the new ring:
\[ GF(2)[x]/ < m_5(x) > \times \ldots \times GF(2)[x]/ < m_8(x) >, \]
with,
\[ a(x) \leftrightarrow a' = (x, x, x), \]
\[ b(x) \leftrightarrow b' = (x + 1, x + 1, x + 1, x + 1), \]
\[ f(x) \leftrightarrow f = (x^3 + x + 1, x^3 + x + 1, x^3 + x + 1, x^3 + x + 1), \]
\( p(x)^{-1} = \left( 1, x^3 + 1, x^5 + x + 1, x^5 + x^4 + x^3 + x \right), \) such that,
\[
a' \odot b' = (x^2 + x, x^2 + x, x^2 + x, x^2 + x),
\]
\[
q' \odot f = (x^2, x + 1, x^2 + 1, x^6 + 1),
\]
\[
a' \odot b' \odot q' \odot f = (x, x^2 + 1, x + 1, x^6 + x^2 + x + 1),
\]

i.e., result of computation \( r' = (a' \odot b' \odot q' \odot f) \odot p^{-1} \) is
\[
r' = (x, x, x, x).
\]

Now, we do change of residue representation by:

\[
\begin{pmatrix}
    r_1(x) \\
    r_2(x) \\
    r_3(x) \\
    r_4(x)
\end{pmatrix} = \begin{pmatrix}
    x + 1 & 1 & 1 & 1 \\
    x + 1 & x + 1 & x + 1 & 1 \\
    x^2 & 1 & x^2 + x & x^2 + x \\
    1 & x^2 + x + 1 & x^3 + x^2 + x & x^3 + x^2
\end{pmatrix} \begin{pmatrix}
    1 \\
    x^2 + x + 1 \\
    x^5 + x^3 + x + 1 \\
    x^6 + x^5 + x^4 + x^2 + 1
\end{pmatrix},
\]

so that final result of computation is:
\[
r = (x, x, x, x).
\] (6.4)

By applying CRT interpolation on (6.4) we get \( r(x) = x \). Since, \( \deg(r(x)) < 4 \), \( r(x) \in C \), i.e., \( r(x) \in GF(2)[x]/<f(x)> \).

Now assume that an adversary induces faults into point \( P \in E/\text{GF}(2^3) \) by inducing faults into one of 8 processors by some physical set up, causing attacked processor to be faulty, such that erroneous output of the computation is
\[
r'(x) = (x + 1, x, x).
\] (6.5)

By applying CRT interpolation on (6.5) we get
\[
r'(x) = x^9 + x^6 + x^4 + x^2 + x + 1.
\] (6.6)

Since \( \deg(r'(x)) > 4 \) we detect an error, and by the extended Euclid’s algorithm for \( \gcd (r'(x), m'(x)) \) we have that at \( j = 1 \), \( d(x) = x^3 \), and \( t(x) = x^2 \). Therefore, correct residue output is \( r(x) = d(x)/t(x) = x \). □

133
6.2 Montgomery Lagrange Representation FTC in Finite Fields

In this section we propose a fault tolerant Lagrange representation (LR) modular multiplication algorithm for fault tolerant computation in the $GF(2^k)$ for use in elliptic curve cryptosystems.

6.2.1 Montgomery LR Computation in the $GF(2^k)$

Bajard et al. in [7] have proposed a new multiplication algorithm for the implementation of elliptic curve cryptography over an optimal extension field $GF(p^k)$, where $p > 2^k$. We will show that their proposed algorithm is correct for $p = 2$, i.e., Algorithm 14.

Our aim is to extend use of this algorithm for fault tolerant computation in the field $GF(2^k)$ by use of redundancy. Operations used in Algorithm 14 are: componentwise addition $\oplus$ and componentwise multiplication $\otimes$.

Algorithm 14 LR Modular Multiplication.

**Inputs:** $a(x), b(x) \in GF(2)[x]/ < f(x) >$, irreducible polynomial $f(x) \in GF(2)[x]$, $p(x) = \Pi_{i=1}^{k}(x - \alpha_i), p'(x) = \Pi_{i=k+1}^{2k}(x - \alpha_j), \alpha_i \in T, \alpha_j \in T', \gcd(p, p') = \gcd(p, f) = 1$, $T = \{\alpha_i \in GF(2^k) | i \in \{1, \ldots, k\}\}$, $T' = \{\alpha_j \in GF(2^k) | j \in \{k+1, \ldots, 2k\}\}$, $\alpha_i \neq \alpha_j$.

**Precomputed:** $f'(x), \xi(x) \in GF(2^k)[x], k \times k$ matrices $w, w'$.

**Output:** $(r_1, \ldots, r_k)$.

1. $(t_1, \ldots, t_k) \leftarrow (a_1, \ldots, a_k) \otimes (b_1, \ldots, b_k)$
2. $(q_1, \ldots, q_k) \leftarrow (t_1, \ldots, t_k) \otimes (f'_1, \ldots, f'_k)$
3. Change of LR: $(q_1, \ldots, q_k) \rightarrow (q_{k+1}, \ldots, q_{2k})$
4. $(r_{k+1}, \ldots, r_{2k}) \leftarrow [(t_{k+1}, \ldots, t_{2k}) \oplus (q_{k+1}, \ldots, q_{2k}) \otimes (f_{k+1}, \ldots, f_{2k})] \otimes (\xi_{k+1}, \ldots, \xi_{2k})$
5. Change of LR: $(r_{k+1}, \ldots, r_{2k}) \rightarrow (r_1, \ldots, r_k)$. 

134
Let the finite binary extension field $GF(2^k)$, be represented as the set of polynomials modulo an irreducible polynomial $f(x)$, $\deg(f(x)) = k$, i.e.,

$$GF(2)[x]/<f(x)> = \{a_0 + \ldots + a_{k-1}x^{k-1} \mid a_i \in GF(2)\}, \quad (6.7)$$

and where $f(x)$ is primitive with $f(\alpha) = 0$, so that

$$GF(2^k) = \{0, 1, \alpha, \alpha^2, \ldots, \alpha^{2^k-2}\}. \quad (6.8)$$

The input polynomials $g_i(x) \in GF(2)[x]/<f(x)>$ are evaluated at $k$ distinct elements from the set

$$T = \{\alpha_j \in GF(2^k) \mid j \in \{1, \ldots, k\}\}. \quad (6.9)$$

Then, there exists a mapping $\phi$

$$\phi : GF(2)[x]/<f(x)> \mapsto GF(2^k)[x]/<x - \alpha_1 \times \ldots \times x - \alpha_k>,$$

such that each input polynomial $g_i(x) \in GF(2)[x]/<f(x)>$ is evaluated at $k$ distinct elements from the set (6.9), i.e.,

$$g_i(x) \leftrightarrow (g_i(\alpha_1), g_i(\alpha_2), \ldots, g_i(\alpha_k)), \quad (6.10)$$

where $g_i(\alpha_j) \in GF(2^k)$ are evaluations of the input polynomials $g_i(x) \in GF(2)[x]/<f(x)>$ at distinct elements from the set (6.9).

**Lemma 6.2.1.** LR modular multiplication Algorithm 14 can be applied to the field $GF(2^k)$ if and only if the finite field is represented as in (6.7) and (6.8).

**Proof.** Since the inputs to the computation are elements in the LR, and since we require at least $2k$ polynomial evaluations for LR modular multiplication Algorithm 14, by considering elements of the $GF(2^k)$ as in (6.7) and (6.8), each element of $GF(2^k)$ in polynomial representation can be evaluated at least at the $2k$ field elements represented as in (6.8), since $2^k > 2k$ for $k \geq 3$. \quad \square

Firstly, the computation is performed in parallel in the field $GF(2^k)$ as in step 1, 2 and 4 of Algorithm 14. As a Montgomery factor we chose polynomial

$$p(x) = \prod_{i=1}^{k} (x - \alpha_i), \quad \alpha_i \in T, \quad (6.11)$$
such that \( \gcd(p(x), f(x)) = 1 \). Since, \( r(x) = t(x) + q(x)f'(x), \ f'(x) \equiv f^{-1}(x)(\text{mod } p(x)) \), is a multiple of \( p(x) \), its LR representation at points from set \( T \) are composed of 0s. Therefore, to avoid this we choose new set of distinct elements

\[
T' = \{ \alpha_j \in GF(2^k) \mid j \in \{ k+1, \ldots, 2k \} \},
\]

(6.12)
such that \( \alpha_i \neq \alpha_j, i \neq j, i, j = 1, \ldots, 2k \). Also we define

\[
p'(x) = \prod_{j=k+1}^{2k} (x - \alpha_j), \ \alpha_j \in T'
\]

with the constraint that \( \gcd(p(x), p'(x)) = 1 \).

Therefore, the computation happens in the new direct product ring

\[
R' = GF(2^k)[x]/ < x - \alpha_{k+1} > \times \ldots \times GF(2^k)[x]/ < x - \alpha_{2k} >,
\]

where \( \alpha_j \in T' \). Let

\[
q(x) = \sum_{i=1}^{k} q_i \prod_{j=1}^{k} \frac{x - \alpha_j}{\alpha_i - \alpha_j}, \ \alpha_i, \alpha_j \in T,
\]

such that

\[
w_{m,i} = \prod_{\substack{j=1 \atop j \neq i}}^{k} \frac{\alpha'_m - \alpha_j}{\alpha_i - \alpha_j}, \ \alpha'_m \in T', \ \alpha_i, \alpha_j \in T,
\]

and the change of LR in step 3 is achieved by:

\[
\begin{pmatrix}
q_{k+1} \\
\vdots \\
q_{2k-1} \\
q_{2k}
\end{pmatrix} = \begin{pmatrix}
w_{1,1} & \ldots & w_{1,k} \\
\vdots & \vdots & \vdots \\
w_{k-1,1} & \ldots & w_{k-1,k} \\
w_{k,1} & \ldots & w_{k,k}
\end{pmatrix} \begin{pmatrix}
q_1 \\
\vdots \\
q_{k-1} \\
q_k
\end{pmatrix}.
\]

(6.13)

In step 4 of Algorithm 14, \( r_i = (t_i + q_if_i)\xi_i, \ i \in \{ k+1, \ldots, 2k \} \), is computed in parallel where

\[
\xi_i = \left( \prod_{j=1}^{k} (\alpha_i - \alpha_j) \right)^{-1} \text{mod } f(\alpha), \ \alpha_i \in T'.
\]
To get the vector $r$ such that its components are polynomial evaluations at distinct elements of the set $T$, we perform the change of LR, i.e., let

$$w'_{m,i} = \prod_{\substack{j=1\atop j \neq i}}^{k} \frac{\alpha_m - \alpha'_j}{\alpha'_i - \alpha'_j}, \quad \alpha_m \in T, \quad \alpha'_i, \alpha'_j \in T',$$

then

$$\begin{pmatrix}
    r_1 \\
    \vdots \\
    r_{k-1} \\
    r_k
\end{pmatrix} = \begin{pmatrix}
    w'_{1,1} & \cdots & w'_{1,k} \\
    \vdots & \vdots & \vdots \\
    w'_{k-1,1} & \cdots & w'_{k-1,k} \\
    w'_{k,1} & \cdots & w'_{k,k}
\end{pmatrix} \begin{pmatrix}
    r_{k+1} \\
    \vdots \\
    r_{2k-1} \\
    r_{2k}
\end{pmatrix}. \quad (6.14)$$

By Lagrange interpolation (LI), the $k$ output components at distinct elements form set $T$ will determine a unique polynomial of degree $\leq k - 1$ with coefficients $a_i \in GF(2^k)$. In the next subsection we extend use of this algorithm for fault tolerant computation in the field $GF(2^k)$ by use of redundancy.

### 6.2.2 Montgomery Lagrange Representation FTC

To protect multiplication of the finite field we use redundancy by adding more parallel channels than the minimum required, i.e., $c > k$, see Fig. 6.2. Thus, inputs are evaluated at $c$ distinct points ($c > k$) from the set

$$T = \{ \alpha_i \in GF(2^k) \mid i \in \{1, \ldots, k, \ldots, c\} \}, \quad (6.15)$$

where $c$ depends on security required, i.e., inputs are evaluated at additional $c - k$ distinct elements $\alpha_i \in GF(2^k)$. By adding $c - k$ extra redundant polynomial evaluations, the computation now happens in the larger direct product ring

$$R = GF(2^k)[x]/<x - \alpha_1 \times \ldots \times GF(2^k)[x]/<x - \alpha_c>, \quad (6.16)$$

where $\alpha_i \in T$, and

$$R \cong R[x] = GF(2^k)[x]/<m(x)>, \quad (6.17)$$

137
where
\[
m(x) = \prod_{i=1}^{c} (x - \alpha_i), \quad c > k, \quad \alpha_i \in T. \tag{6.18}
\]
The operations used are componentwise multiplication \(\otimes\), and componentwise addition \(\oplus\).

Inputs \(g_i(x) \in GF(2^k)[x]/<f(x)>\) are evaluated at \(c\) distinct elements from the set (6.15), i.e.,
\[
g_i(x) \rightarrow (g_i(\alpha_1), \ldots, g_i(\alpha_k), \ldots, g_i(\alpha_c)),
\]
where \(g_i(\alpha_1), \ldots, g_i(\alpha_k)\) are \textit{non-redundant components}, and \(g_i(\alpha_{k+1}), \ldots, g_i(\alpha_c)\) are \textit{redundant components}.

In Algorithm 15, steps 1, 2 and 4 are performed in parallel in the field \(GF(2^k)\). As a \textit{Montgomery factor} we choose polynomial
\[
p(x) = \prod_{i=1}^{c} (x - \alpha_i), \quad \alpha_i \in T, \tag{6.19}
\]

\begin{algorithm}
\textbf{Algorithm 15} Fault tolerant LR Modular Multiplication.

\textbf{Inputs:} \(a(x), b(x) \in GF(2)[x]/<f(x)>\), irreducible polynomial \(f(x) \in GF(2)[x]\), \(p(x), p'(x) \in GF(2^k)[x]\), \(\deg(p) = \deg(p') = c\), \(\gcd(p, p') = \gcd(p, f) = 1\), \(T = \{\alpha_i \in GF(2^k) | i \in \{1, \ldots, c\}\}\), \(T' = \{\alpha_j \in GF(2^k) | j \in \{c+1, \ldots, 2c\}\}\), \(\alpha_i \neq \alpha_j\). Pre-computed: \(f'(x), \xi(x) \in GF(2^k)[x]\), \(c \times c\) matrices \(w, w'\).

\textbf{Output:} \(r(x) \in GF(2^k)[x]/\prod_{i=1}^{c} (x - \alpha_i) >, \alpha_i \in T\).

1. \((t_1, \ldots, t_k, \ldots, t_c) \leftarrow (a_1, \ldots, a_k, \ldots, a_c) \otimes (b_1, \ldots, b_k, \ldots, b_c)\)

2. \((q_1, \ldots, q_k, \ldots, q_c) \leftarrow (t_1, \ldots, t_k, \ldots, t_c) \otimes (f'_1, \ldots, f'_k, \ldots, f'_c)\)

3. Change of LR: \((q_1, \ldots, q_c) \rightarrow (q_{c+1}, \ldots, q_{2c})\)

4. \((r_{c+1}, \ldots, r_{2c}) \leftarrow [(t_{c+1}, \ldots, t_{2c}) \oplus (q_{c+1}, \ldots, q_{2c}) \otimes (f_{c+1}, \ldots, f_{2c})] \otimes (\xi_{c+1}, \ldots, \xi_{2c})\)

5. Change of LR: \((r_{c+1}, \ldots, r_{2c}) \rightarrow (r_1, \ldots, r_c)\)

6. Lagrange interpolation: \((r_1, \ldots, r_c)\).
\end{algorithm}
such that \( \gcd(p(x), f(x)) = 1 \). Since, \( r(x) = t(x) + q(x)f'(x), \ f'(x) \equiv f^{-1}(x) (\mod p(x)) \), is a multiple of (6.19), we choose new set of distinct elements

\[
T' = \{ \alpha_j \in GF(2^k) \mid j \in \{ c + 1, \ldots, 2c \} \},
\]

such that \( \alpha_i \neq \alpha_j, \ i \neq j, \ i, j = 1, \ldots, 2c \), and we define

\[
p'(x) = \prod_{j=c+1}^{2c} (x - \alpha_j), \quad \alpha_j \in T'
\]

with constraint that \( \gcd(p(x), p'(x)) = 1 \). Therefore, the computation happens in the new direct product ring

\[
R' = GF(2^k)[x] / (x - \alpha_{c+1}) \times \cdots \times GF(2^k)[x] / (x - \alpha_{2c}),
\]

where \( \alpha_j \in T' \). Let

\[
q(x) = \sum_{i=1}^{c} q_i \prod_{\substack{j=1
\]

such that

\[
w_{m,i} = \prod_{\substack{j=1
\]

and the change of LR is achieved by:

\[
\begin{pmatrix}
q_{c+1} \\
\vdots \\
q_{2c-1} \\
q_{2c}
\end{pmatrix}
= \begin{pmatrix}
w_{1,1} & \cdots & w_{1,k} & \cdots & w_{1,c} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
w_{c-1,1} & \cdots & w_{c-1,k} & \cdots & w_{c-1,c} \\
w_{c,1} & \cdots & w_{c,k} & \cdots & w_{c,c}
\end{pmatrix}
\begin{pmatrix}
q_1 \\
\vdots \\
q_{c-1} \\
q_c
\end{pmatrix}.
\]

In step 4 of Algorithm 15, \( r_i = (t_i + q_if_i)\xi_i, \ i \in \{ c + 1, \ldots, 2c \} \) is computed in parallel where

\[
\xi_i = \left( \prod_{j=1}^{c} (\alpha_i - \alpha_j) \right)^{-1} \mod f(\alpha), \quad \alpha_i \in T'.
\]

Now, to get a vector \( r \) such that its components are polynomial evaluations at distinct elements of the set \( T \), we do change of LR, i.e., let

\[
w'_{m,i} = \prod_{\substack{j=1
\]

139
then
\[
\begin{pmatrix}
r_1 \\
\vdots \\
r_{c-1} \\
r_c
\end{pmatrix} = \begin{pmatrix}
w'_{1,1} & \cdots & w'_{1,k} & \cdots & w'_{1,c} \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
w'_{c-1,1} & \cdots & w'_{c-1,k} & \cdots & w'_{c-1,c} \\
w'_{c,1} & \cdots & w'_{c,k} & \cdots & w'_{c,c}
\end{pmatrix} \begin{pmatrix}
r_{c+1} \\
\vdots \\
r_{2c-1} \\
r_{2c}
\end{pmatrix},
\]
(6.23)

**Theorem 6.2.2.** If there are no fault effects, Algorithm 15 will determine a unique polynomial of degree \( k - 1 \) with coefficients \( a_i \in \text{GF}(2^k) \), otherwise, it will be of degree \( \geq k \).

*Proof.* See proof of Theorem 6.1.2. \( \square \)

**Definition 6.2.3.** The set of correct results of computation is

\[
C = \left\{ r(x) \in R[x] \mid \deg(r(x)) < k, a_i \in \text{GF}(2^k) \right\}.
\]

**Computational Complexity**

Polynomial interpolation is only done at the end of the computation, and its complexity is \( O(c^2) \), while complexity of polynomial evaluation is \( O(ck) \), \( c > k \). Matrices \( w, w' \) are precomputed, as well as \( \xi \) and \( f'(x) \).

**Lemma 6.2.4.** Total computational complexity of the Algorithm 15 is \( O(c^2) \), \( c > k \).

*Proof.* In Algorithm 15, steps 1 and 2 require \( c \) operations each, while steps 3 and 4 require \( 2c^2 - c \) operations each. Also, step 4 requires \( c^2 + c \). By taking into consideration complexity of polynomial evaluation and interpolation we have that total complexity of the Algorithm 15 is \( O(c^2) \). \( \square \)

**Lemma 6.2.5.** The matrices (6.22) and (6.23) satisfy relation \( w_{i,j} = w'_{i+1,j} \) if and only if sets of \( c \) distinct points are chosen as \( T = \{ \alpha^{2m_i} \in \text{GF}(2^k) \mid i \in \{1, \ldots, c\} \} \), \( T' = \{ \alpha^{2m_i+1} \in \text{GF}(2^k) \mid i \in \{1, \ldots, c\} \} \).
Proof. Let
\[ w_{i,j} = \prod_{s=1}^{c} \frac{\alpha^{2m_{s+1}} - \alpha^{2m_{s}}}{\alpha^{2m_{j}} - \alpha^{2m_{s}}}, \] (6.24)
and multiply (6.24) by \( \alpha^{c-1}/\alpha^{c-1} \), then
\[ w_{i,j} = \prod_{s=1}^{c} \frac{\alpha^{2m_{s}+1} - \alpha^{2m_{j}+1}}{\alpha^{2m_{j}+1} - \alpha^{2m_{s}+1}} = \prod_{s=1}^{c} \frac{\alpha^{2(m_{s}+1)} - \alpha^{2m_{s}+1}}{\alpha^{2m_{j}+1} - \alpha^{2m_{s}+1}}. \]
Since \( m_{i+1} = m_{i} + 1 \) we have
\[ w_{i,j} = \prod_{s=1}^{c} \frac{\alpha^{2m_{s}+1} - \alpha^{2m_{s}}}{\alpha^{2m_{j}+1} - \alpha^{2m_{s}+1}} = w'_{i+1,j}. \]

\[ \square \]

### 6.2.3 Error Correction and Detection

There is one processor per independent channel as in Fig. 6.2. Let us assume that we have \( c \) processors, where each processor computes \( i \)-th polynomial evaluation and operations of the finite field \( GF(2^{k}) \). Also, we assume that all precomputed inputs are error free, as well as the Lagrange interpolation in step 6 of Algorithm 15. As before, we assume that a fault attack induces faults into processors by some physical means. As a reaction, the attacked processor malfunctions, and it does not compute the correct output given its input. We are concerned with the effect of a fault as it manifests itself in a modified data, or a modified program execution. Therefore, we consider the fault models presented in Chapter 3. Since computation is decomposed into parallel, mutually independent, identical channels, the adversary can use either RFM, or AFM, or SFM per channel.

Assume that at most \( c-k \) channels have faults. Let \( r' \in R \) be computed vector with \( c \) components, where \( e_{j} \in GF(2^{k}) \) is the error at \( j \)-th position; then the computed component at the \( j \)-th positions is
\[ r_{j} = r(\alpha_{j}) + e_{j}, \] (6.25)

Figure 6.2: Fault tolerant multiplication over the finite field $GF(2^k)$. 
and each processor will have as an output component

\[ r_j = \begin{cases} r(\alpha_j) + e_j, & j \in \{j_1, \ldots, j_t\}, \\ r(\alpha_j), & \text{else}. \end{cases} \]

Here, we have assumed that the set of error positions are \{j_1, \ldots, j_t\}, i.e., \( e_j \) is the effect of the fault in the channel \( j_i \). By Lagrange interpolation, the computed vector \( r' \in R \) with corresponding set of \( c \) distinct elements from set \( T \) in (6.15), gives as output the unique polynomial \( r'(x) \in GF(2^k)[x]/ < m(x) > \),

\[
r'(x) = \sum_{1 \leq i \leq c} r_i \prod_{1 \leq j \leq c, i \neq j} \frac{x - \alpha_j}{\alpha_i - \alpha_j} = \sum_{1 \leq i \leq c} r_i(\alpha_i) \prod_{1 \leq j \leq c, i \neq j} \frac{x - \alpha_j}{\alpha_i - \alpha_j} + \sum_{1 \leq i \leq t} e_{ji} \prod_{1 \leq j \leq c, j \neq i} \frac{x - \alpha_j}{\alpha_{ji} - \alpha_i} = r(x) + e(x),
\]

(6.26)

where \( r(x) \) is correct polynomial of computation such that \( \deg(r(x)) \leq k - 1 \) with coefficients from \( GF(2^k) \); and \( e(x) \) is the error polynomial such that:

**Theorem 6.2.6.** Let effects of the fault \( e_{j_1} \neq 0, \ldots, e_{j_t} \neq 0 \) be any set of \( 1 \leq t \leq c - k \) elements of \( GF(2^k) \), \( c > k \), then \( \deg(e(x)) > k - 1 \) whose coefficients \( a_i \in GF(2^k) \).

**Proof.** We have that

\[
e(x) = \sum_{1 \leq i \leq c} e_{ji} \prod_{1 \leq j \leq c, j \neq i} \frac{x - \alpha_j}{\alpha_{ji} - \alpha_i} = \prod_{1 \leq i \leq c} (x - \alpha_i) \left( \frac{e_{j_i}}{(x - \alpha_{j_i}) \prod_{1 \leq j \leq c, j \neq i} (\alpha_{ji} - \alpha_i)} + \ldots + \right),
\]

Since,

\[
\deg \left( \frac{\prod_{1 \leq i \leq c} (x - \alpha_i)}{(x - \alpha_{j_i})} \right) = c - 1, \ldots, \deg \left( \frac{\prod_{1 \leq i \leq c} (x - \alpha_i)}{(x - \alpha_{j_i})} \right) = c - 1,
\]

\( c > k \), then \( \deg(e(x)) > k - 1 \) with coefficients

\[
\frac{e_{j_k}}{(x - \alpha_{j_k}) \prod_{1 \leq i \leq c, j_k \neq i} (\alpha_{j_k} - \alpha_i)} \in GF(2^k).
\]

\[
\Box
\]
Therefore, the faulty processors affect the result in an additive manner, see (6.26).

**Lemma 6.2.7.** The error is masked iff error polynomial $e(x)$ has coefficients from $GF(2^k)$, and if $\deg(e(x)) \leq k - 1$.

*Proof.* Let $r'(x)$ be the computed polynomial as in (6.26). Since, $\deg(r(x)) \leq k - 1$ with coefficients $a_i \in GF(2^k)$, then if $\deg(e(x)) \leq k - 1$ with coefficients $a_i \in GF(2^k)$ we have that $\deg(r'(x)) \leq k - 1$ with coefficients $a_i \in GF(2^k)$ in which case error is masked. □

**Lemma 6.2.8.** Let $k$ be the degree of the finite field, and let $c > k$ be number of parallel independent channels (or number of processors). If there are no faulty processors then $r'(x) \in C$.

*Proof.* If there are no faulty processors, then clearly no errors occurred, and $\deg(r'(x)) \leq k - 1$ with coefficients $a_i \in GF(2^k)$, i.e., $r'(x) \in C$. □

**Lemma 6.2.9.** Let $k$ be the degree of the finite field, and let $c > k$ be the number of parallel independent channels (or number of processors). Then if up to $c - k$ channels fail, the output polynomial $r'(x) \notin C$.

*Proof.* By referring to (6.26), since $\deg(e(x)) > k - 1$ with coefficients $a_i \in GF(2^k)$, and $\deg(r(x)) \leq k - 1$ with coefficients $a_i \in GF(2^k)$, then the output polynomial $r'(x)$ has to be such that $\deg(r'(x)) > k - 1$ with $a_i \in GF(2^k)$, i.e., $r'(x) \notin C$. □

It is straightforward to appeal to the standard coding theory result below, to state the error detection and correction capability of our set up:

**Theorem 6.2.10.** (i) If the number of parallel, mutually independent, modular, redundant channels is $d + t \leq c - k$ ($d \geq t$), then up to $t$ faulty processors can be corrected, and up to $d$ simultaneously detected. (ii) By adding $2t$ redundant independent channels at most $t$ faulty processors can be corrected.

While it is true that arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed here, it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost.
We also remark that the Welch-Berlekamp Algorithm 16 is suitable for correcting the faults induced by the attacks described in this paper. Note that to specify the algorithm we choose a set of \( k \) indices \( K = \{0, 1, \ldots, k-1\} \), and \( \overline{K} = \{0, \ldots, c-1\} \setminus K \). Discussion of this Algorithm is given in Chapter 5, Subsection 5.2.5.

**Algorithm 16** Welch-Berlekamp Decoding of the Output Vector.

**Inputs:** output vector of computation \( r' = (r_0, \ldots, r_{k-1}, r_k, \ldots, r_{c-1}) \), set of \( c \) distinct points \( T = \{\alpha_i \mid \alpha_i \in GF(2^k)\} \), set of indices \( K = \{0, 1, \ldots, k-1\} \), polynomial \( g(x) = \prod_{i \in K} (x - \alpha_i) \)

**Outputs:** polynomials \( d(x), h(x) \).

1. By Lagrange interpolation, interpolate output vector \( r' \) in order to get polynomial \( r'(x) \)

2. if \( \text{deg}(r'(x)) < k \) and \( a_i \in GF(2^k) \) then \( r'(x) \)

3. else

4. for \( i \in K \) do find \( r'(x) \), where \( \text{deg}(r') < k \)

5. evaluate \( r'(x) \), at \( \alpha_i, l \in \overline{K} \)

6. determine syndromes \( S_l = r_l - r'(x_l), l \in \overline{K} \)

7. determine \( y_l = \frac{s_l}{g(x_l)} \)

8. solve key equation \( d(x_l)y_l = h(x_l) \)

9. return \( d(x), h(x) \).

**Example 6.2.11.** Assume that we want to protect computation in the finite binary field \( GF(2^5) \cong GF(2)[x]/<x^5 + x^2 + 1> \), where \( \alpha \) is a primitive root of the primitive polynomial \( x^5 + x^2 + 1 \), i.e., \( \alpha^5 + \alpha^2 + 1 = 0 \).

Let the inputs to the computation be the following finite field elements: \( a(x) = x^3 + x^2 + 1 \), \( b(x) = x^3 + 1 \). We want to compute following expression \((a(x)b(x)) \mod f(x)\). Since, \( k = 5 \), the minimum number of polynomial evaluations is 5, but in order to correct a single error, we add \( c - k = 2 \) extra channels. Therefore, we choose following sets of distinct points

\[
T = \{\alpha^{18}, \alpha^{20}, \alpha^{22}, \alpha^{24}, \alpha^{26}, \alpha^{28}, \alpha^{30}\} \quad \text{and} \quad T' = \{\alpha^{19}, \alpha^{21}, \alpha^{23}, \alpha^{25}, \alpha^{27}, \alpha^{29}, \alpha^{31}\},
\]

such that \( p(x) = \prod_{i=1}^{7} (x - \alpha_i), \alpha_i \in T \), and \( p'(x) = \prod_{i=1}^{7} (x - \alpha_j), \alpha_j \in T' \), and
gcd\(p(x), p'(x)\) = 1.

At elements of the set \(T\) and \(T'\) we evaluate inputs, i.e.,

\[
a_T = (\alpha^{27}, \alpha^{30}, \alpha^8, \alpha^{18}, \alpha, \alpha^8, \alpha^{24}), \quad a_{T'} = (\alpha, \alpha^2, \alpha^6, \alpha^{16}, \alpha^3, \alpha^{17}, 1),
\]

\[
b_T = (\alpha^{12}, \alpha^3, \alpha^{10}, \alpha^4, \alpha^9, \alpha^7, \alpha^{26}), \quad b_{T'} = (\alpha^{28}, \alpha^{18}, \alpha^{22}, \alpha^{14}, \alpha^{11}, \alpha^{21}, 0).
\]

Also,

\[
f_{T'} = (\alpha^{10}, \alpha^3, \alpha^{27}, \alpha^5, \alpha^{29}, \alpha^{30}, 1), \quad f_T' = (\alpha, \alpha^8, \alpha^{19}, \alpha^6, \alpha^{14}, \alpha^{13}, \alpha^{16}),
\]

\[
\xi = (\alpha^{23}, \alpha^2, \alpha^7, 1, \alpha^{12}, \alpha^{19}, \alpha^{22}).
\]

The interpolation matrices are

\[
w = \begin{pmatrix}
\alpha^{23} & \alpha^{30} & \alpha^5 & \alpha^{19} & \alpha^2 & \alpha^{12} & \alpha^{22} \\
\alpha^2 & \alpha^{19} & \alpha^4 & \alpha^{11} & \alpha^{10} & \alpha^{25} & \alpha^{13} \\
\alpha^{24} & \alpha^3 & \alpha^{29} & \alpha^{15} & \alpha^7 & \alpha^7 & 1 \\
\alpha^{11} & \alpha^6 & \alpha^{25} & \alpha^{21} & \alpha^{23} & \alpha^{16} & \alpha^{25} \\
\alpha^5 & \alpha^5 & \alpha^{29} & \alpha^{10} & \alpha^{13} & \alpha^{15} \\
\alpha^{26} & \alpha^4 & \alpha^{13} & \alpha^{18} & \alpha^{23} & \alpha^5 & \alpha^{17} \\
\alpha^{28} & \alpha^{29} & \alpha^{16} & \alpha^{26} & \alpha^{16} & \alpha^{22} & \alpha^{13}
\end{pmatrix}, \quad w' = \begin{pmatrix}
\alpha^7 & \alpha^4 & \alpha^{17} & \alpha^{15} & \alpha^{24} & \alpha^{25} & \alpha^{12} \\
\alpha^{23} & \alpha^{30} & \alpha^5 & \alpha^{19} & \alpha^2 & \alpha^{12} & \alpha^{22} \\
\alpha^2 & \alpha^{19} & \alpha^4 & \alpha^{11} & \alpha^{10} & \alpha^{25} & \alpha^{13} \\
\alpha^{24} & \alpha^3 & \alpha^{29} & \alpha^{15} & \alpha^7 & \alpha^7 & 1 \\
\alpha^{11} & \alpha^6 & \alpha^{25} & \alpha^{21} & \alpha^{23} & \alpha^{16} & \alpha^{25} \\
\alpha^5 & \alpha^5 & \alpha^{29} & \alpha^{10} & \alpha^{13} & \alpha^{15} \\
\alpha^{26} & \alpha^4 & \alpha^{13} & \alpha^{18} & \alpha^{23} & \alpha^5 & \alpha^{17}
\end{pmatrix}.
\]

Therefore,

\[
t_T = (\alpha^8, \alpha^2, \alpha^{18}, \alpha^{22}, \alpha^{10}, \alpha^{15}, \alpha^{19}) \quad \text{and} \quad q_T = (\alpha^9, \alpha^{10}, \alpha^6, \alpha^{28}, \alpha^{24}, \alpha^{28}, \alpha^4).
\]

By change of LR from \(T \rightarrow T'\) we have

\[
q_{T'} = (\alpha^{24}, \alpha^{13}, \alpha^{11}, \alpha^{16}, \alpha^{22}, \alpha^{17}, \alpha^{30}), \quad \text{such that} \quad r_{T'} = (\alpha^{23}, \alpha^{28}, \alpha^8, \alpha^{19}, \alpha^3, \alpha^{22}).
\]

By change of LR from \(T' \rightarrow T\) we have

\[
r_T = (\alpha^{23}, \alpha^{12}, \alpha^4, \alpha^9, \alpha^{19}, \alpha^3, \alpha^{22}). \quad (6.27)
\]

By interpolating \((6.27)\) at distinct points of \(T\) we have

\[
r(x) = \alpha^{30}x^4 + \alpha^{26}x^3 + \alpha^{28}x^2 + \alpha^{24}x + \alpha^{22}.
\]
Assume that an adversary induces faults into point \( P \in E/GF(2^5) \) by inducing faults into one of 7 processors, by some physical set up, causing attacked processor to be faulty, such that the erroneous output of the computation is, i.e., \( r = (\alpha^{23}, \alpha^{12}, \alpha^2, \alpha^9, \alpha^{19}, \alpha^3, \alpha^{22}) \).

Now, we select set of \( k = 5 \) indices \( K = \{0, 1, 2, 3, 4\} \) such that by interpolating

\[
r = (\alpha^{23}, \alpha^{12}, \alpha^2, \alpha^9, \alpha^{19}) \quad \text{at} \quad (\alpha^{18}, \alpha^{20}, \alpha^{22}, \alpha^{24}, \alpha^{26})
\]

we get

\[
r(x) = \alpha^7 x^4 + \alpha^{18} x^3 + x + \alpha^{21} x^2 + \alpha^{30} x + \alpha^3.
\]

Given index selection \( K \) we evaluate \( r(x) \) at \( \alpha_5 = \alpha^{28}, \alpha_6 = \alpha^{30}, \) i.e., \( r(\alpha^{28}) = \alpha^{14}, r(\alpha^{30}) = \alpha^{12}, \) such that \( S = (0, 0, 0, 0, \alpha^{22}, \alpha^{16}) \). Therefore, syndromes are \( S_5 = \alpha^{22}, S_6 = \alpha^{16} \). Now define the polynomial

\[
g(x) = (x - \alpha^{18}) (x - \alpha^{20}) (x - \alpha^{22}) (x - \alpha^{24}) (x - \alpha^{26}).
\]

New interpolated data is given by \( \alpha_5 = \alpha^{28}, \alpha_6 = \alpha^{30}, \) and

\[
y_5 = \frac{S_5}{g(\alpha_5)} = \alpha, \quad \text{and} \quad y_6 = \frac{S_6}{g(\alpha_6)} = \alpha^8.
\]

The problem is to determine polynomials \( d(x), h(x) \) from \( d(\alpha_5) y_5 = h(x_5), d(\alpha_6) y_6 = h(x_6) \). By rational interpolation at points \( (\alpha^{28}, \alpha), (\alpha^{30}, \alpha^8) \) we obtain that \( d(x) = \alpha^{26} + x \alpha^4 \) and \( h(x) = \alpha^{23} \). Therefore, the error locations are the roots of the polynomial \( d(x) \), i.e., \( \alpha_2 = \alpha^{22}, \) while the error values are obtained by

\[
S(x) = \frac{h(x) g(x)}{d(x)} = \alpha^{19} x^4 + \alpha^{7} x^3 + \alpha^{25} x^2 + \alpha^{20} x + \alpha^{14},
\]

such that \( e_2 = S_2 - S(\alpha^{22}) = \alpha^7, \) so that \( r_2 - e_2 = \alpha^2 - \alpha^7 = \alpha^4 \). Therefore, the correct output vector of computation is

\[
r = (\alpha^{23}, \alpha^{12}, \alpha^4, \alpha^9, \alpha^{19}, \alpha^3, \alpha^{22}),
\]

which interpolated gives \( r(x) = \alpha^{30} x^4 + \alpha^{26} x^3 + \alpha^{28} x^2 + \alpha^{24} x + \alpha^{22}. \)

\[\square\]

6.2.4 LR Inversion in GF(2^k)

Here we adapt the inversion algorithm based on the extended Euclidean algorithm for polynomials defined over optimal extension field GF(p^k) in LR proposed by Bajard et
al. in [9] to the binary extension field $GF(2^k)$. This algorithm is based on the Lehmer’s Euclidean GCD algorithm ([57], [5], [93]) which computes $q = l(U(x))/l(V(x))$, $R = U(x) - qxV(x)$ if $\deg(U(x)) > \deg(V(x))$ and $t = \deg(U(x)) - \deg(V(x))$, till zero remainder is encountered.

**Lemma 6.2.12.** The LR inversion algorithm proposed over optimal extension field in [9] can be used in the $GF(2^k)$ if and only if finite field is represented as in (6.7) and (6.8).

**Proof.** See proof of Lemma 6.2.1.

#### Algorithm 17 Leading term $LT(U,m)$

**Input:** A polynomial $U(x) \in GF(2)[x]/ < f(x) >$ given in LR: $(u(\alpha_1), \ldots, u(\alpha_k))$.

**Precomputed:** $\xi_{i,t} = \left( \prod_{j=1}^{t} (\alpha_i - \alpha_j) \right)^{-1} (mod 2)$

**Output:** $(d, c)$, $d = \deg(a(x))$, $c = l(a(x))$, where $a(x) = cx^d + \ldots$

1. if $m = 0$ then
2. $c \leftarrow u(\alpha_1)$
3. $d \leftarrow 0$
4. else
5. $t \leftarrow m + 1$
6. $c = 0$
7. while $c = 0$ do
8. for $i \leftarrow 1$ to $t$ do
9. $c \leftarrow c + u(\alpha_i)\xi_{i,t}(mod 2)$
10. if $c = 0$ then
11. $t \leftarrow t - 1$
12. return $(t - 1, c)$

Therefore, we consider the representation of a finite field $GF(2^k)$ as in (6.7) and (6.8). Given input $a(x) \in GF(2)[x]/ < f(x) >$ and irreducible polynomial $f(x) \in GF(2)[x]$ in LR representation, we compute $a^{-1}(x) mod f(x)$ in LR representation. More precisely, to represent inputs in LR representation by using $k$ values, is used as an input $f(x) mod p(x)$,
so that the computed output is \((a^{-1}(x) \mod f(x)) \mod p(x)\), where \(p(x)\) is as in (6.11).

Bajard et al. [9] have proposed an Algorithm 17 which computes the degree and leading coefficient of a polynomial given in the LR representation, to avoid ignorance of the degree and coefficients of the polynomials while performing polynomial division in LR.

**Remark 6.2.13.** Let \(U(x) \in GF(2)[x]/<f(x)>\) be given in Lagrange representation \((u(\alpha_1), \ldots, u(\alpha_k)), \alpha_i \in T, T = \{\alpha_j \in GF(2^k) \mid j \in \{1, \ldots, k\}\}\), then by Proposition 5.2.1 we have

\[
U(x) = \sum_{i=1}^{k} u(\alpha_i) \prod_{\substack{j=1 \atop j \neq i}}^{k} (x - \alpha_j)(\alpha_i - \alpha_j)^{-1}
\]

\[
= \sum_{i=1}^{k} u(\alpha_i) \left( \prod_{\substack{j=1 \atop j \neq i}}^{k} (\alpha_i - \alpha_j) \right)^{-1} \prod_{\substack{j=1 \atop j \neq i}}^{k} (x - \alpha_j)
\]

\[
= \sum_{i=1}^{k} u(\alpha_i) \left( \prod_{\substack{j=1 \atop j \neq i}}^{k} (\alpha_i - \alpha_j) \right)^{-1} x^{k-1} + \ldots
\]

where leading coefficient of degree \(k - 1\) of \(U(x)\) is:

\[
l(U(x)) = \sum_{i=1}^{k} u(\alpha_i) \left( \prod_{\substack{j=1 \atop j \neq i}}^{k} (\alpha_i - \alpha_j) \right)^{-1} \pmod{2}.
\]

In Algorithm 17, \(m\) is the largest possible degree of \(U(x)\), i.e., \(m \leq k-1\), which is decre-mented by 1 each time coefficient is equal to 0. Values \(\xi_{i,t} = \left( \prod_{\substack{j=1 \atop j \neq i}}^{k} (\alpha_i - \alpha_j) \right)^{-1} \pmod{2}\), \(1 \leq i \leq t \leq k\) are precomputed and require storage of \(k(k+1)/2\) field elements from \(GF(2^k)\). The total number of operations of Algorithm 17 is \(mA + (m + 1)M\), where \(A\) stands for addition and \(M\) stands for multiplication. Complete complexity analysis of Algorithm 18 is given in [9], but for the sake of completeness we will mention it here.

In Step 11, each iteration requires one multiplication and one inversion while computing \(q\). By polynomial Lehmer’s Euclidean GCD algorithm, number of divisions is at most \(\deg(f(x) \mod p(x)) + \deg(a(x)) = 2k - 2\), while in Steps 12 and 13 for computation of \(U_1\) and \(U_3\) each iteration requires \(3k\) multiplications and \(2k\) additions. There are two calls of
Algorithm 18 Inversion over $GF(2^k)$ in LR

**Input:** polynomial $a(x)$ in LR, $a = (a(\alpha_1), \ldots, a(\alpha_k))$, polynomial $f(x)(mod\ p(x))$ in LR, $f' = (f'(\alpha_1), \ldots, f'(\alpha_k))$, $gcd(a(x), f(x)) = 1$

**Precomputed:** polynomial $X_t(x) = x^t$, $0 \leq t \leq k$ in LR, $X_t = (X_t(\alpha_1), \ldots, X_t(\alpha_k))$

**Output:** polynomial $(a^{-1}(x) mod\ f(x)) mod\ p(x)$ in LR

1. $(U_1, U_3) \leftarrow ((1, \ldots, 1), (a_1, \ldots, a_k))$
2. $(V_1, V_3) \leftarrow ((0, \ldots, 0), (f'(\alpha_1), \ldots, f'(\alpha_k))$
3. $(d(V_3), l(V_3)) \leftarrow (k, 1)$
4. $(d(U_3), l(U_3)) \leftarrow LT(U_3, k – 1)$
5. **while** $U_3 \neq 0$ **do**
6. $t \leftarrow d(U_3) - d(V_3)$
7. **if** $t < 0$ **then**
8. $(U_1, U_3) \leftarrow (V_1, V_3)$
9. $(d(U_3), l(U_3)) \leftarrow (d(V_3), l(V_3))$
10. $t \leftarrow -t$
11. $q \leftarrow l(U_3)l(V_3)^{-1}mod\ 2$
12. $U_1 \leftarrow U_1 + qX_tV_1$
13. $U_3 \leftarrow U_3 + qX_tV_3$
14. $(d(U_3), l(U_3)) \leftarrow LT(U_3, d(U_3) – 1)$
15. **return** $U_1$

$LT(U_3, i)$, $1 \leq i \leq k$. Total complexity is $(2k-2)$ inversions plus:

$$(2k-2)M + (2k-2)(3kM + 2kA) + 2 \sum_{i=1}^{k-1} LT(U_3, i) = (6k^2 - 4k - 2)M +$$

$$(4k^2 - 4k)A + 2 \sum_{i=1}^{k-1} (i+1)M + 2 \sum_{i=1}^{k-2} iA = (7k^2 - 3k - 4)M + (5k^2 - 5k)A,$$

which can be simplified to $2k-2$ inversions, plus $12k^2 - 8k - 4$ operations in $GF(2^k)$.

**Example 6.2.14.** Assume that we want to compute the inverse of an element in the finite binary field $GF(2^3) \cong GF(2)[x]/ < x^3 + x + 1 >$, where $\alpha$ is a primitive root of
the primitive polynomial $x^3 + x + 1$, i.e., $\alpha^3 + \alpha + 1 = 0$. Let $a(x) = x^2 + x + 1$ be an element whose inverse modulo $f(x)$, $f(x) = x^3 + x + 1$, we want to compute in LR, and let $T = \{1, \alpha, \alpha^2\}$ be the set of distinct points at which inputs are evaluated. Also, let $p(x) = (x - 1)(x - \alpha)(x - \alpha^2)$, and $f(x) \mod p(x) = \alpha^5 x^2 + \alpha^2 x + \alpha$. Inputs are given in LR representation, i.e.,

$$a(x) \leftrightarrow a = (1, \alpha^5, \alpha^3),$$

$$f(x) \mod p(x) \leftrightarrow f' = (1, 0, 0).$$

We provide Table 6.1 with iterations of Algorithm 18. Inverse of $a(x) \mod f(x)$ is given by $U_1 = (1, \alpha^2, \alpha^4)$, which by Lagrange interpolation corresponds to polynomial $x^2$. 

<table>
<thead>
<tr>
<th>$U_1$</th>
<th>$U_3$</th>
<th>$V_1$</th>
<th>$V_3$</th>
<th>$d(U_3)$</th>
<th>$l(U_3)$</th>
<th>$t$</th>
<th>$q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(1,1,1)$</td>
<td>$(1, \alpha^5, \alpha^3)$</td>
<td>$(0,0,0)$</td>
<td>$(1,0,0)$</td>
<td>2</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>$(0,0,0)$</td>
<td>$(1,0,0)$</td>
<td>$(1,1,1)$</td>
<td>$(1, \alpha^5, \alpha^3)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1,\alpha, \alpha^2)$</td>
<td>$(0, \alpha^6, \alpha^5)$</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(0, \alpha^3, \alpha^6)$</td>
<td>$(1, \alpha, \alpha^2)$</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1,1,1)$</td>
<td>$(1, \alpha^5, \alpha^3)$</td>
<td>$(0, \alpha^3, \alpha^6)$</td>
<td>$(1, \alpha, \alpha^2)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1, \alpha^5, \alpha^3)$</td>
<td>$(0, \alpha^3, \alpha^6)$</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1, \alpha^2, \alpha^4)$</td>
<td>$(1,1,1)$</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(0, \alpha^3, \alpha^6)$</td>
<td>$(1, \alpha, \alpha^2)$</td>
<td>$(1, \alpha^2, \alpha^4)$</td>
<td>$(1,1,1)$</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$(1,0,0)$</td>
<td>$(0,0,0)$</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.1: Iterations of LR Inversion Algorithm
6.3 Conclusion

In this chapter we have presented techniques for the protection of elliptic curve computation in a “tamper-proof” device by protection of the finite field computation against active side channel attacks, i.e., fault attacks. Our algorithms where field elements are represented by the redundant residue representation/redundant lagrange representation enables us to overcome the problem if one, or both coordinates $x, y \in GF(2^k)$ of the point $P \in E/GF(2^k)$ are corrupted. Computation of the field elements is decomposed into parallel, mutually independent, modular/identical channels, so that in case of fault at one channel, errors will not distribute to others. Therefore, during each run of the algorithm an adversary can use either RFM, or AFM, or SFM per channel. By assuming these fault models our proposed algorithms provide protection against error propagation. Arbitrarily powerful adversaries can create faults in enough channels and overwhelm the system proposed here, but it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost. Note that fault tolerant LR modular multiplication algorithm is more efficient than fault tolerant RR modular multiplication since computation is done over identical channels, i.e., arithmetic performed is modulo binomial of degree one. The downside of our proposed algorithms is that they can have masked errors, and will not be immune against attacks which can create those kind of errors. However, it is a difficult problem to counter masked errors since any anti-fault attack scheme will have some masked errors.
Chapter 7

Conclusion

In this thesis we have developed new algorithmic countermeasures that protect elliptic curve computation by protecting computation of the finite binary extension field, against fault attacks.

Firstly, we have proposed schemes, i.e., a Chinese Remainder Theorem based fault tolerant computation in finite field for use in ECCs, as well as Lagrange Interpolation based fault tolerant computation. Our approach is based on error correcting codes, i.e., redundant residue polynomial codes and the use of the first original approach of Reed-Solomon codes. Computation of the field elements is decomposed into parallel, mutually independent, modular/identical channels, so that in case of faults at one channel, errors will not distribute to other channels.

Based on these schemes we have developed new algorithms, namely fault tolerant residue representation modular multiplication algorithm and fault tolerant Lagrange representation modular multiplication algorithm, which are immune against error propagation under the fault models that we propose: Random Fault Model, Arbitrary Fault Model, and Single Bit Fault Model. These algorithms provide fault tolerant computation in $GF(2^k)$ for use in ECCs. Our new developed algorithms where inputs, i.e., field elements, are represented by the redundant residue representation/redundant Lagrange representation enables us to overcome the problem if during computation one, or both coordinates
\(x, y \in GF(2^k)\) of the point \(P \in E/GF(2^k)\) are corrupted. We assume that during each run of an attacked algorithm, in one single attack, an adversary can apply any of the proposed fault models, i.e., either Random Fault Model, or Arbitrary Fault Model, or Single Bit Fault Model. In this way more channels can be targeted, i.e., different fault models can be used on different channels. This is new, since no previous work has implied that different fault models can be used during each run of an attacked algorithm, in one single attack. Our fault models have been motivated by real devices and real physical world, where power of an adversary is compared with power of the countermeasures present on the smartcard. The single bit Fault Model is the strongest fault model, but for today’s smartcards this fault model may sound unrealistic by looking at the countermeasures. The more realistic fault models are Random Fault Model, and especially Arbitrary Fault Model, since countermeasures of today’s smartcards ensure that they are immune against really strong fault models such as Single bit Fault Model.

These fault models prove that our algorithmic countermeasures work, and that the attacker can not break a system and recover secret key. Arbitrarily powerful adversaries can simply create faults in enough channels and overwhelm the system proposed, but it is part of the design process to decide on how much security is enough, since all security (i.e. extra channels) has a cost. If fault effects are detected, they can be corrected by applying error correction algorithms, i.e., Euclidean decoding algorithm/Welch-Berlekamp decoding algorithm at the output vector of the corresponding computation. Moreover, if the number of parallel, mutually independent, modular/identical, redundant channels is \(d + t\ (d \geq t)\), then up to \(t\) faulty processors can be corrected, and up to \(d\) simultaneously detected. By adding \(2t\) redundant independent channels at most \(t\) faulty processors can be corrected.

Also, our proposed algorithms can have masked errors and will not be immune against attacks which can create those kind of errors, but it is a difficult problem to counter masked errors, since any anti-fault attack scheme will have some masked errors. Moreover, we have derived conditions that inflicted error needs to have in order to yield undetectable faulty point on non-supersingular elliptic curve over \(GF(2^k)\).
Our algorithmic countermeasures can be applied to any public key cryptosystem which performs computation over the finite field $GF(2^k)$. 
Bibliography


