2012

Speeding Up finite Field Inversion for Cryptographic Applications

Walid Mahmoud
University of Windsor

Speeding Up Finite Field Inversion for Cryptographic Applications

by

Walid Mustafa Mahmoud

A Dissertation
Submitted to the Faculty of Graduate Studies
through the Department of Electrical Engineering
in Partial Fulfillment of the Requirements for the
Degree of Doctor of Philosophy in Engineering

at the University of Windsor

Windsor, Ontario, Canada

2011

Speeding Up Finite Field Inversion for Cryptographic Applications

by

Walid Mustafa Mahmoud

Approved By:

November 8, 2011

Declaration

Abstract

In elliptic curves cryptography, the curves are always defined over a particular finite field to provide the required cryptographic services. Currently, such services are the engine of most network security applications in practice. Scalar multiplication is the core operation of most such cryptographic services. Scalar multiplication performs field inversion very frequently in the underlying finite field. Field inversion is the most time-consuming operation that requires a special attention. Therefore, by accelerating field inversion, in addition to their inherent high level of security, such cryptographic services are executed fast.

In finite extension fields $GF(p^m)$ with the extension degree m, accelerating field inversion by following Fermat's approach is reduced to the problem of finding a clever way to compute an exponentiation, which is a function of the field's extension degree m. By applying the concept of short addition chains combined with the idea of decomposing $(m-1)$ into several factors plus a remainder, with some restrictions applied, field inversion in such fields is computed very fast.

Two field inversion algorithms are proposed based on the suggested methods above. They are mainly proposed for extension fields of characteristic p *two* and *three* using normal basis representation. Fast Frobenius map operation proposed and extended to higher characteristic extension fields. Both algorithms, relative to existing inversion algorithms, require the minimal number of field multiplications, the second costly operations, those necessary to perform the exponentiation for field inversion. The obtained results confirmed the validity of the proposed ideas herein.

After three days without reading, talk becomes flavorless.

–A proverb

Dedicated to my beloved parents, family and to all who cares.

2011

Fearing ALLAH is the
beginning of knowledge.

Acknowledgments

Many thanks go to my supervisor Dr. Huapeng Wu, who has supported me extensively in this type of research work, and for his continued encouragement to achieve my final goal. I give him all my respect.[1]

Many thanks go to my brother Dr. Mufeed Mahmoud, who has always supported me to reach this level of eduction, and for his continued encouragement to achieve my final goal. I give him all my respect.[2]

Special thanks go to the committee members for the time, effort and for their continuous followup during the progress of this research work.

Many thanks go to all who have helped me directly or indirectly !

[1]A professor in electrical engineering department at the University of Windsor, Ontario, Canada.
[2]A professor in electrical engineering department at the University of UML, Massachusetts, USA.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Elliptic curve cryptography (ECC) was independently proposed by Miller [1] and Koblitz [2] in 1985. After the emergence of ECC as the alternative public-key system, instead of the legacy system that previously proposed by Rivest, Shamir and Adleman (RSA: [3]), such systems gained much more interest by many society sectors than its predecessors.

The strength of elliptic curves in practical cryptographic-applications due to the strength of their associated group structures, qualified them to be recognized and standardized internationally by ISO and IETF, and in the USA by NIST and ANSI organizations. As a result of this, ECC was accepted in commercial sector in 1990 [4], and since then, many commercial cryptographic-systems were designed and dispensed in the markets.

In academic sector, many researchers have directed their focus on the interesting field of elliptic curves, either by paying more attention on the mathematical aspects of such curves, or by somehow optimizing their associated arithmetic operations for use in cryptographic applications (such as in ECC), as in this dissertation.

1.1 Finite Fields and ECC

An elliptic curve is simply a mathematical curve of cubic-degree, which is a non-singular complete algebraic curve. Therefore, such a curve is defined by a characteristic equation of two unknown variables, with some coefficients known a priori. The characteristic equation used to describe the elliptic curve is defined over (its variables and coefficients are taken from) some mathematical field, such as the field of complex numbers, field of real numbers, field of rational numbers, etc [4].

In using elliptic curves for cryptographic-applications, such as in ECC, they must be defined over mathematical fields with finite number of elements, which have a strong number-theoretic foundations, to satisfy the security requirements imposed by the concerned application. Such fields are exist, well-defined mathematically and referred to as finite fields [5].

Earlier in this chapter, it was mentioned that an elliptic curve in cryptography is associated with a strong group structure. The obtained abelian group of finite curve-points, resulted by defining an elliptic curve over a specific finite field, indeed has a strong algebraic properties, since the finite field has an inherent strong number-theoretic foundation [6].

In ECC, curve-operations such as scalar multiplication (SM), point addition (PA) and point doubling (PD) are highly dependent on the arithmetic operations in the field over which the elliptic curve is defined (referred to as, the underlying field). Thus, to accelerate cryptographic-applications (or -algorithms) such as the elliptic curve digital signature algorithm (ECDSA) for authentication, elliptic curve diffie-hellman (ECDH) algorithm for key-exchange, or elliptic curve el-gamal (ECEl-Gamal) algorithm for encryption, we need to optimize the arithmetic operations in the underlying field.

Scalar multiplication is the core algorithm in most cryptographic-applications based on elliptic curves, including the previously mentioned ones. In fact, it dominates their execution-times to a great extent, since it heavily relies on (performs very frequently) field inversion in the underlying field [7].

Since finite fields are simply mathematical fields, they have all arithmetic operations defined such as addition, squaring, multiplication, inversion, etc. Field inversion is the most time-consuming operation, the costlier operation, in finite fields [8]. It relies heavily on field multiplication, the second costly operation [9], to calculate the inverse of the concerned field element. Therefore, to accelerate cryptographic-algorithms based on elliptic curves, it is imperative to accelerate field inversion in the underlying field by reducing the required number of multiplications, which is the main interest in this dissertation.

1.2 Existing Work and Contribution

In finite extension fields $GF(p^m)$, when the characteristic p is equal to two and the extension degree m is greater than one, the fields are referred to as binary extension fields $GF(2^m)$, with field elements represented as binary vectors of dimension m. When p is equal to three and m is greater than one, the fields are referred to as ternary (characteristic three) extension fields $GF(3^m)$, with field elements represented as ternary vectors of dimension m. In $GF(3^m)$, the coefficients of each vector belong to the set $\{0, 1, 2\}$, or more generally, the set $\{0, 1, \cdots, p-1\}$, which is exactly the subfield $GF(p)$. In both types of finite extension fields, the vectors are interpreted in different way depending on the used representation basis [10].

1.2.1 Binary Extension Fields $GF(2^m)$

In $GF(2^m)$ with field elements represented using normal basis representation (NB), arithmetic (or field) operations such as additions and squarings are simply (mod 2)-additions and cyclic-shifts, respectively. As a result of this, such fields are attractive from software and hardware perspectives (*no rounding-errors and carry-propagation chains exist*). For this reason, they have predominant usage in many applications including ECC [11, 12, 13]. In such applications, the associated finite field arithmetic operations must be executed fast, in particular, accelerating the execution time of finite field inversion (or the inverse) algorithm is of paramount importance, as it was mentioned earlier herein.

A group of interested academic researchers previously proposed algorithms for finding the inverse fast in $GF(2^m)$ [14, 15, 16, 17]. Such algorithms are based on Fermat's Little Theorem (FLT) and using NB for the field elements. In using Fermat's approach, inverse calculation is basically an exponentiation that requires repeated squarings and multiplications. For example, given a nonzero element $\delta \in GF(2^m)$, its inverse using FLT method is given by

$$\delta^{-1} = \delta^{2^m-2} = \delta^{2^1} \times \delta^{2^2} \times \cdots \times \delta^{2^{m-2}} \times \delta^{2^{m-1}}. \tag{1.1}$$

Inverse calculation without any attempt to manipulate Eq. (1.1), with the aim to reduce the required number of multiplications, requires $(m-1)$ field squarings and $(m-2)$ extension field multiplications. Wang et al. [18] basically proposed a VLSI implementation of FLT method, given in Eq. (1.1) above, for computing the inverse in $GF(2^m)$.

In using NB, field squaring of a nonzero element $\delta \in GF(2^m)$ is simply a right cyclic-shift that is computed very fast compared to field multiplication. Therefore, it is imperative to reduce the number of the required multiplications in Eq. (1.1) above for fast inverse computation. Itoh and Tsujii [14] computed the inverse fast using their proposed inversion algorithm (referred to as, ITA) by computing Eq. (1.1) above in an ingenious way. The number of the required extension field multiplications necessary for inversion (inversion cost) is given by

$$\left[\ell(m-1) + w(m-1) - 2\right], \tag{1.2}$$

whereby $\ell(.)$ and $w(.)$ are the binary-length and Hamming-weight (the number of $1s$ in the binary-representation) of the argument $(m-1)$ that is given by a positive integer, respectively. The worst case in ITA algorithm happens when $(m-1)$ is a full-weight value, whereby $\ell(m-1) = w(m-1)$, which has encouraged academic researchers to devise other mathematical ways to reduce the inversion cost for such a case in computing the inverse.

Chang et al. [15] proposed inversion algorithm (referred to as, CEA) which further reduces the inversion cost in some ms, or in some binary extension fields $GF(2^m)$ since each m represents a unique extension field, by factorizing $(m-1)$ into two factors. Given that $(m-1) = s \times t$, i.e., factorized into two factors, then the inversion cost using CEA algorithm is given by

$$[(\ell(s) + w(s) - 2) + (\ell(t) + w(t) - 2)]\,. \tag{1.3}$$

Unfortunately enough, CEA algorithm is not applicable in the cases when $(m-1)$ is a prime value.

Takagi et al. [16] proposed inversion algorithm (referred to as, TYT) which improves on CEA algorithm by decomposing $(m-1)$ into several factors and a small remainder h. Further reductions in the inversion cost is achieved in some ms, or in some $GF(2^m)$, and the remainder h handled the prime case of $(m-1)$ in which CEA algorithm is not applicable. Given that $(m-1) = \prod_{j=1}^{k} r_j + h$, i.e., decomposed into several factors and a remainder, then the inversion cost using TYT algorithm is given by

$$\left(\sum_{j=1}^{k} \ell(r_j) + w(r_j) - 2 \right) + h. \tag{1.4}$$

Notice how Eq. (1.4) above is partly a linear function of the remainder h itself, with h restricted to the value of 1 as reflected in their paper. The restriction of h to a single value minimizes the set of $GF(2^m)$ those associated with the minimal number of the required multiplications for inversion, or in other words, minimizes the set of those $GF(2^m)$ that are associated with the minimal inversion cost.

Li et al. [17] proposed inversion algorithm (referred to as, LCA) which improves on TYT algorithm by re-using some intermediate results, after decomposing $(m-1)$ in the same manner, without restricting the remainder h to the value of 1, but to a specific set of values. Therefore, further reductions in the inversion cost in some ms, or in some $GF(2^m)$, is obtained. Given that $(m-1) = \prod_{j=1}^{k} r_j + h$, i.e., decomposed into several factors and a remainder, then the inversion cost using LCA algorithm is given by

$$\left(\sum_{j=1}^{k} \ell(r_j) + w(r_j) - 2 \right) + w(h). \tag{1.5}$$

Notice how Eq. (1.5) above is partly a function of the Hamming weight of the remainder h, rather than a linear function of h itself as in TYT algorithm.

Our main contribution relevant to inversion in characteristic two extension fields (see Chapter 4.1) is represented by the following:

1. Propose a decomposition algorithm for the extension degree m of $GF(2^m)$ that results in further reductions in the inversion cost
2. Propose a field inversion algorithm to implement our decomposition algorithm in $GF(2^m)$ using NB for the field elements

By appropriately decomposing $(m-1)$ into several factors and a remainder h, i.e., $(m-1) = \prod_{j=1}^{k} r_j + h$, the remainder h if properly selected to belong to the short addition chain (SAC[1]) of the first factor r_1 (denoted as C_{r_1}), or any other factor in $(m-1)$ as long as h is less than or equal to that specific factor, all relevant multiplications to h are saved, thus, the inversion cost of the proposed algorithm is given by

[1]The minimum possible addition chain for r_1 in which h is one of the elements (see Chapter 2.4).

$$\left(\sum_{j=1}^{k} \ell(r_j) + w(r_j) - 2\right) + 1. \tag{1.6}$$

Notice how the required number of multiplications is now independent of the remainder h, as it is clear from Eq. (1.6) above.

In subsequent chapters we are going to prove Eq. (1.6) and to show that our proposed inversion algorithm in $GF(2^m)$ is expected to have as low as or even lower inversion cost than other existing inversion algorithms. This finding is applicable to many extension degrees ms, or equivalently is applicable to many $GF(2^m)$, that are suitable for use in ECC-based cryptographic applications, including the ones recommended for governmental cryptographic applications.

1.2.2 Ternary Extension Fields $GF(3^m)$

Ternary extension fields $GF(3^m)$ are very important in ECC for use in cryptographic systems based on bilinear-mapping, such as the Weil pairing or Tate pairing, and exhibits more bandwidth efficiency relative to other extension fields as declared by Galbraith in [19]. In keeping up with our interest to accelerate field inversion, we are going to present what has been done so far, in the academic literature, in such type of extension fields, and then show our contribution in such fields.

In particular, in $GF(3^m)$ many authors in the literature including [20, 21, 22], depend on the inversion scheme that previously proposed by Itoh and Tsujii in [23], which is based on Fermat's inversion approach. The scheme simplifies inversion in such extension fields, to inversion in the subfield $GF(3)$, in addition to the need for performing a logarithmic number of extension field multiplications in such fields, i.e., $GF(3^m)$-multiplications.

The aforementioned inversion scheme is also applicable to any odd-characteristic extension field $GF(p^m)$, for prime characteristic $p > 2$ and $m > 1$, where in such a case, the subfield is $GF(p)$, and the required multiplications are performed in the extension field $GF(p^m)$.

For example, given $p = 3$ and a nonzero element $\delta \in GF(3^m)$, its inverse based on the inversion scheme in [23] is computed as in the following expression (referred to as, ITI expression):

$$\delta^{-1} = (\delta^r)^{-1} \times (\delta)^{r-1}, \tag{1.7}$$

which has inversion cost equal to $[\ell(m-1) + w(m-1)]$ $GF(3^m)$-multiplications, in addition to the need for a subfield inversion for finding the inverse of δ.

ITI expression, given in Eq. (1.7) above, is not written as a complete field inversion algorithm in $GF(p^m)$, for any odd-prime characteristic p and $m > 1$. The authors just mentioned the use of addition chains to calculate $(\delta)^{r-1}$ term, without writing an algorithm to perform this task or even the task of computing the field inverse itself. A group of authors rely on ITI expression to compute and accelerate the inverse, with attempts to improve it from software and hardware perspectives, by defining their own domain parameters and representation basis, while imposing the specific constraints and conditions [24, 25] to achieve their final goals.

Our main contribution relevant to inversion in characteristic three extension fields (see Chapter 4.2) is represented by the following:

1. Propose a cohesive and complete field inversion algorithm in $GF(3^m)$ using NB somehow based on ITI expression

2. Employ our decomposition algorithm and propose the relevant inversion algorithm to implement it in $GF(3^m)$ using NB

3. Propose fast Frobenius map in $GF(3^m)$ and use it as the basis for extending the idea to higher characteristic extension fields

We propose a cohesive and complete field inversion algorithm in $GF(3^m)$ using NB and somehow based on ITI expression. Given a nonzero element $\delta \in GF(3^m)$ in which is inverse is required, the inverse is computed using our first algorithm proposed in such fields with inversion cost that is given by

$$[\ell(m-1)+w(m-1)-1]\, GF(3^m)\text{-Multiplications.} \tag{1.8}$$

By comparing the inversion cost of our algorithm given in Eq. (1.8) against the inversion cost of other existing algorithms given in Eq. (1.7) above, in addition to avoiding the required subfield inversion, our algorithm also requires less number of $GF(3^m)$-multiplications.

In addition, we propose a second field inversion algorithm in $GF(3^m)$ using NB, which implements the decomposition algorithm that proposed earlier herein for $GF(2^m)$. As a result of this, some algorithms are modified in order to be suitable for use in $GF(3^m)$. As mentioned earlier herein, our final goal is to reduce the inversion cost, especially when $(m-1) = \prod_{i=1}^{k} r_i + h$, i.e., is appropriately decomposed into several factors and a remainder h. Based on previous discussion, given a nonzero element $\delta \in GF(3^m)$ in which its inverse is required, the inversion cost, using our second algorithm proposed in such field is given by

$$\left(\sum_{j=1}^{k} \ell(r_j)+w(r_j)-2\right)+1. \tag{1.9}$$

The inversion cost given in Eq. (1.9) above, is exactly similar to that result we previously obtained in $GF(2^m)$ [see Eq. (1.6) above]. This in fact is true since our decomposition algorithm depends on the extension degree m, but not on the characteristic p of the concerned field.

Our proposed field inversion algorithms in $GF(3^m)$ are expected to have as low as or even lower inversion cost than other existing inversion algorithms. This finding is applicable to many extension degrees ms, or equivalently is applicable to many $GF(3^m)$ that are of particular interest for ECC, whereby our algorithms in some of such fields require lower inversion cost [26].

Furthermore, we propose an accelerated Frobenius map operation in $GF(p^m)$ for odd-prime p and $m > 1$ using polynomial basis representation (PB). We start by considering characteristic $p = 3$ and extend the idea to higher characteristic extension fields. In $GF(3^m)$ using PB, the Frobenius map is equivalent to cubing operation. The proposed fast Frobenius map operation is expected to have almost free runtime and space complexity, especially when $GF(p^m)$ follows the definition of Type-II optimal extension fields (OEFs), as it will be clarified later on.

1.3 Notes and Dissertation Organization

The astute reader, notice that in this dissertation most of our research work is the application of engineering principles through mathematics. This is represented by providing some mathematical facts (or ideas) that finally translated to a set of newly derived algorithms. Such algorithms are intended to solve a specific problem with low complexity or cost from the engineering perspective.

Our final goal herein is to speed up field inversion for cryptographic applications, especially those applications based on ECC. To achieve this, we will start by considering the aforementioned FLT method, in attempt to reduce the required number of multiplications for inversion (inversion cost), which in turn, accelerates the execution time of any proposed field inversion algorithm more apparently by using NB. This reduction can be achieved through the mathematical manipulation for the extension degree m of the concerned $GF(p^m)$. Based on this, our proposed field inversion algorithms in binary and ternary extension fields are the outcomes of such manipulation.

Furthermore, we will consider the Frobenius map operation, by proposing a fast operation with almost free runtime and space complexity in ternary and higher characteristic extension fields, given that such fields follow the definition of Type-II OEFs. The proposed fast Frobenius map operation is necessary for accelerating inversion in finite fields using PB for the field elements.

The astute reader will notice the sequential way for solving the inverse problem with the lowest inversion cost possible in finite extension fields. More specifically, in the literature review chapter (see Chapter 3), we will move from the previous work achieved in accelerating field inversion in binary and ternary extension fields to the current work in a sequential manner. We will start our presentation by explaining the ideas, that previously proposed by academic researchers by considering their published work, from the oldest to the newest, which coincides with their attempts to improve the inversion cost of inversion algorithms in finite extension fields. Our proposed work in this dissertation is one of such attempts to further improve the inversion cost in such fields.

This dissertation consist of six chapters arranged in logical sequence, in such a way, the reader can grasp the ideas and the mathematical concepts presented herein. The mathematical background chapter mainly focus on the mathematical aspects of finite fields those necessary for understanding the different types of finite fields to be used herein, the representation bases for the elements in such fields, and the effect of such representation on the arithmetic operations in the concerned field.

The literature review chapter is included mainly to present the state-of-the-art mathematical solutions to the problem of accelerating field inversion, as previously and currently suggested by the academic researchers in the literature. Also, it allows the reader to observe the diverse mathematical flavors for the solution, until the reader reaches the solution proposed by the current authors herein.

The proposed algorithms chapter is included mainly to introduce the reader to our newly proposed ideas and concepts, dedicated to further improve the solution for accelerating inversion problem in finite fields. In that chapter, our proposed field inversion algorithms for binary and ternary extension fields are presented and elaborated. In addition, all relevant helper algorithms are provided, which assist main algorithms in computing the inverse.

The results and analyses chapter aims mainly to provide comparison tables to compare the inversion cost, in terms of the required number of extension field multiplications, using our proposed inversion algorithms in comparison with other existing inversion algorithms. In addition, it presents a set of useful propositions relevant to the proposed algorithms and to the associated helper algorithms. Furthermore, the results will be analyzed to give the reader a better idea about the competitive advantage in using our proposed inversion algorithms over other existing algorithms.

Finally, this dissertation is concluded with closing remarks that summarizes the main things achieved from our proposed research work. In addition, it provides the future recommendations to pinpoint the potential tracks to follow in such type of research field, which may attract the attention of academic researchers.

Chapter 2

Mathematical Background

This chapter is mainly concerned with the mathematical aspects and concepts relevant to finite fields. First, we will present the available different types of finite fields, their commonly used representation bases, and their associated arithmetic operations. Then, we are going to define and show how the inverse of a nonzero field element is calculated using the inversion approach based on Fermat's little theorem. Finally, we are going to give an idea about the short addition chains, first, by defining such chains, then, by showing how they are useful for inverse calculation in finite fields[1].

2.1 Finite Fields

Finite fields are mathematical fields, identical to the well-known regular fields such as the field of complex numbers, field of real numbers, or field of rational numbers, etc, except that they contain a finite number of elements. Such a number is referred to as the field order.

Since finite fields are simply mathematical fields, they have all the well-known arithmetic operations defined such as addition, multiplication, division (or inversion), etc. Mathematically, a finite field is defined as a finite set of elements denoted as $\mathbb{F}$, that is associated with two binary operations (addition: $+$ and multiplication: $\times$),

[1]The reader is referred to [27] to better understand most mathematical material presented herein.

which satisfies the following set of axioms [4]:

1. $(\mathbb{F}, +)$ forms abelian group, with '0' the additive identity
2. $(\mathbb{F}^*, \times)$ forms multiplicative group, with '1' the identity element
3. Closure law: $(a + b) = c$, or $(a \times b) = d$, for all $a, b, c, d \in \mathbb{F}$ holds
4. Distributive law: $(a + b) \times c = (a \times c) + (b \times c)$, for all $a, b, c, d \in \mathbb{F}$ holds

The closure law guarantees that the result of an arithmetic operation is again belongs to the concerned field. Notice how the field arithmetic operation, namely field division (or inversion), is implied from the items 2 and 3 in the above list, since every nonzero element in the multiplicative group $\mathbb{F}^*$ has a multiplicative inverse, which is another element that belongs to the group itself. Also, this is because the finite field is a commutative division ring. The French mathematician *Evariste Galois*, who laid the foundations for Galois theory [28], declared that any field with a prime-power order (i.e., field order is a power of a prime number) is a finite field, whereby the prime is the field's characteristic p and the power is the extension degree m of the field. Based on his declaration, there are three major classes of finite fields (or Galois fields), as it will be shown in what follows.

2.1.1 Prime Fields

Given that p is a prime number, the set of integers Z modulo p is referred to as the prime field, denoted as $GF(p)$. The prime field is exactly the set $\{0, 1, 2, \cdots, p-1\}$ of prime-power order, whereby the prime is p and the power is simply equal to 1. Mathematically, the prime field is defined as follows.

Definition 1. *The integer ring R_Z is equivalent to the set of all integers Z together with two binary operations defined, namely the addition and multiplication.*

Definition 2. *The principal ideal (p) is the smallest ideal, that is, a subset of the ring R_Z that is generated by the unique prime integer $p \in R_Z$.*

Definition 3. *The quotient ring* $Q = \frac{R_Z}{(p)}$ ***is isomorphic to*** $GF(p)$*, which is the set of cosets of* (p)*, or equivalently, the set* $\{0, 1, 2, \cdots, p-1\}$ *of coprime numbers with* p*.*

In prime fields, all arithmetic operations are modular, which means, the result is reduced to modulo p. As a symbolic example, we will just show the addition, multiplication and division operations. Given the nonzero elements $a, b \in GF(p)$, for any prime integer p, then we have

$$a + b \longrightarrow z = a + b \pmod{p} : \; z \in GF(p).$$

$$a \times b \longrightarrow z = a \times b \pmod{p} : \; z \in GF(p).$$

$$\frac{a}{b} \longrightarrow z = a \times c \pmod{p} : \; c = b^{-1}, \; z \in GF(p),$$

whereby

$$b \times c \equiv 1 \pmod{p}.$$

Basically, the inverse c of b is another element in $GF(p)$ that gives the result of 1 when multiplied by b over the modulus p. It can be solved either by using the extended Euclidean algorithm (or one of its variants), or by using the inversion approach based on Fermat's little theorem (more on this later on). In general, since all arithmetic operations in prime fields are modular, then they follow the well-known number-theoretic rules.

2.1.2 Binary Extension Fields

A binary extension field is a Galois field with a prime-power order, whereby the prime is the integer $p = 2$ and the power is any positive integer $m > 1$. Thus, simply its order, or its number of elements, is a power of 2 value. In the academic literature, such a field is denoted as $GF(2^m)$, whereby $p = 2$ is the field's characteristic and m is the field's extension degree relative to the subfield $GF(2)$, which simply equals the set $\{0, 1\}$. Mathematically, the binary extension field is defined as follows.

Definition 4. *The binary polynomial $f(x)$, is an irreducible field polynomial of degree $m > 1$ defined over, or its coefficients are taken from the subfield $GF(2)$, which has its roots in the extension field $GF(2^m)$.*

Definition 5. *The polynomial ring $R_p = GF(2)[x]$, is the set of all polynomials of any degree defined over, or with coefficients taken from the subfield $GF(2)$.*

Definition 6. *The principal ideal $P = (f(x))$ is the smallest ideal, that is, a subset of the ring R_p that is generated by the irreducible field polynomial $f(x) \in R_p$.*

Definition 7. *The quotient ring $Q = \frac{R_p}{P}$* ***is isomorphic to*** *$GF(2^m)$, which is the set of all polynomials of degrees $< m$ defined over $GF(2)$, that is exactly the binary extension field.*

Based on the above definitions, the elements of a binary extension field $GF(2^m)$ can be thought of as polynomials of degrees never exceed m whose coefficients equal to either 0 or 1. Also, they can be considered as binary vectors of dimension (or size) m, the extension degree.

For every positive integer m, or extension degree, there exist at least one binary irreducible polynomial $f(x)$ of degree m, necessary for generating the associated extension field $GF(2^m)$, whereby the field elements are represented either by using normal or polynomial basis representation (see next section).

As a symbolic example, we will just show the addition, multiplication and division operations. Given the nonzero elements $\alpha, \beta \in GF(2^m)$, then we have

$$\alpha + \beta \longrightarrow \gamma = \alpha + \beta \pmod 2 : \ \gamma \in GF(2^m).$$

$$\alpha \times \beta \longrightarrow \gamma = [\alpha \times \beta \pmod 2] \pmod{f(x)} : \ \gamma \in GF(2^m).$$

$$\frac{\alpha}{\beta} \longrightarrow \gamma = [\alpha \times \delta \pmod 2] \pmod{f(x)} : \ \delta = \beta^{-1}, \ \gamma \in GF(2^m),$$

whereby

$$\beta \times \delta \equiv 1 \pmod{f(x)}.$$

Basically, the inverse δ of β is another element in $GF(2^m)$ that gives the result of 1 when multiplied by β over the modulus $f(x)$. It can be solved either by using the extended Euclidean algorithm (or one of its variants), or by using the inversion approach based on Fermat's little theorem (more on this later on). Notice that when multiplying two extension field elements together, two modular reductions are performed. The first is referred to as the *subfield modular reduction* that ensures the coefficients belong to $GF(2)$, represented by (mod 2) operation above. The other is referred to as the *extension field modular reduction* that ensures the degrees are less than m, represented by (mod $f(x)$) operation above. In general, all arithmetic operations in binary extension fields require two modular reductions to produce the final result, except for the addition operation in the concerned extension field.

2.1.3 Odd Characteristic Extension Fields

An odd-characteristic extension field is a Galois field with odd prime-power order, whereby the prime is the integer $p > 2$ and the power is any positive integer $m > 1$. Thus, simply its order, or its number of elements, is a power of any odd prime. In the academic literature, such a field is denoted as $GF(p^m)$, whereby p is the field's characteristic, and m is the field's extension degree relative to the subfield $GF(p)$ that simply equals the set $\{0, 1, 2, \cdots, p-1\}$. Mathematically, the odd-characteristic extension field is defined as follows.

Definition 8. *The field polynomial $f(x)$, is an irreducible polynomial of degree $m > 1$ defined over, or its coefficients are taken from the subfield $GF(p)$, which has its roots in the extension field $GF(p^m)$.*

Definition 9. *The polynomial ring $R_p = GF(p)[x]$, is the set of all polynomials of any degree defined over, or with coefficients taken from the subfield $GF(p)$.*

Definition 10. *The principal ideal $P = (f(x))$ is the smallest ideal, that is, a subset of the ring R_p generated by the irreducible field polynomial $f(x) \in R_p$.*

Definition 11. *The quotient ring $Q = \frac{R_p}{P}$* ***is isomorphic to*** *$GF(p^m)$, which is the set of all polynomials of degrees $< m$ defined over $GF(p)$, that is exactly the odd-characteristic extension field.*

Based on the above definitions, the elements of odd-characteristic extension field $GF(p^m)$ can be thought of as polynomials of degrees never exceed m whose coefficients never exceed p. Also, they can be considered as $GF(p)$ vectors of dimension (or size) m, the extension degree.

For every positive integer m, or extension degree, there exist at least one field irreducible polynomial $f(x)$ of degree m, necessary for generating the associated extension field $GF(p^m)$, whereby the field elements are represented either by using normal or polynomial basis representation (see next section).

As a symbolic example, we will just show the addition, multiplication and division operations. Given the nonzero elements $\alpha, \beta \in GF(p^m)$, then we have

$$\alpha + \beta \longrightarrow \gamma = \alpha + \beta \pmod{p} : \ \gamma \in GF(p^m).$$

$$\alpha \times \beta \longrightarrow \gamma = [\alpha \times \beta \pmod{p}] \pmod{f(x)} : \ \gamma \in GF(p^m).$$

$$\frac{\alpha}{\beta} \longrightarrow \gamma = [\alpha \times \delta \pmod{p}] \pmod{f(x)} : \ \delta = \beta^{-1}, \ \gamma \in GF(p^m),$$

whereby

$$\beta \times \delta \equiv 1 \pmod{f(x)}.$$

Basically, the inverse δ of β is another element in $GF(p^m)$ that gives the result of 1 when multiplied by β over the modulus $f(x)$. It can be solved either by using the extended Euclidean algorithm (or one of its variants), or by using the inversion approach based on Fermat's little theorem (more on this later on). Again, when multiplying two extension field elements together, two modular reductions are performed. The first is referred to as the *subfield modular reduction* that ensures the coefficients belong to $GF(p)$, represented by (mod p) operation above. The other is referred to as the *extension field modular reduction* that ensures the degrees are less than m, and represented by (mod $f(x)$) operation above. In general, all arithmetic operations in odd-characteristic extension fields require two modular reductions to produce the final result, except for the addition operation in the concerned extension field.

When the field characteristic $p = 3$, then we have the ternary (or characteristic three) extension field. In fact such a field follows all the rules and specifications of odd-characteristic extension fields.

2.1.4 Optimal Extension Fields

In general, when odd-characteristic extension fields are subject to a specific restrictions in order to accelerate their associated arithmetic operations, when such operations are implemented in software or hardware, such fields are referred to as optimal extension fields (OEFs).

On the one hand, in attempt to accelerate subfield modular reduction operation, with respect to the field's characteristic p as the modulus, then the optimized field is referred to as Type-I OEF. On the other hand, in attempt to accelerate extension field modular reduction operation, with respect to the field's irreducible polynomial $f(x)$ as the modulus, then the optimized field is referred to as Type-II OEF, as it will be clear from the following presentation.

Definition 12. *Given $c \in \mathbb{N}$ (a positive integer), a pseudo-Mersenne prime is a prime number $N = 2^k - c$ with $\lfloor \frac{k}{2} \rfloor \geq \log_2(c)$. If $c = 1$, then N is a Mersenne prime. If $c = -1$, then N is a Fermat prime.*

Definition 13. *The odd-characteristic extension field $GF(p^m)$ is referred to as optimal extension field (OEF), if the following is satisfied:*

1. *p is a pseudo-Mersenne prime (if $c = \pm 1$, then Type-I OEF).*

2. *$f(x) = x^m - w$ is the field irreducible binomial (if $w = 2$, then Type-II OEF).*

When a pseudo-Mersenne prime p is used as the characteristic of the concerned field, then it simplifies the subfield modular reduction operation, whereas when the irreducible binomial $f(x)$ is used as the field polynomial of the concerned field, then it simplifies the extension field modular reduction operation [29]. The existence of an irreducible binomial for a particular $GF(p^m)$ is asserted by the following theorem [30].

Theorem 1. *Given $m \geq 2 \in \mathbb{N}$ (a positive integer) and $w \in GF^*(p)$ (a nonzero subfield element), then $f(x) = x^m - w$ is an irreducible binomial in the polynomial ring $GF(p)[x]$ if and only if the following is satisfied:*

1. *each prime factor of m, including m itself, divides $z = Order(w)$ over $GF(p)$, but not $\frac{p-1}{z}$ and,*

2. *$p \equiv 1 \pmod 4$ if $m \equiv 0 \pmod 4$.*

The following corollary specify the condition through which a polynomial is defined as an irreducible binomial based on the above theorem.

Corollary 1. *If $w \in GF(p)$ is a primitive element and m is a divisor of $(p-1)$, then $f(x) = x^m - 2$ is an irreducible binomial for the field $GF(p^m)$.*

From the above corollary, given a primitive element in the subfield along with a divisor extension degree m for $(p-1)$, whereby p is the characteristic of the concerned field, then it is very easy to find a field irreducible binomial. A primitive element is a field element which has order less by 1 than the order of the field to which it belongs. In mathematical way, if $w \in GF(p)$ is a primitive element, then its order is $z = Order(w) = (p-1)$. The primitive element is also known as the generator element in finite fields.

2.2 Representation Bases

Unlike the elements in the prime field, which has a unique representation for its elements as integers, the elements in extension fields have more than a representation basis. Apart from the other available representation bases, normal and polynomial basis representations are the most commonly used ones in the academic literature. In what follows such representation bases are introduced. In this section, the presentation relevant to finite fields arithmetic operations is made general and is applicable to any extension field, regardless of the characteristic p of the concerned field.

2.2.1 Normal Basis

The normal basis representation (NB) is being used extensively in the academic literature, as is the case in this dissertation, to represent the elements of the concerned extension field $GF(p^m)$. It has the advantage of performing p^e-th powers (for characteristic p and a positive integer $e \in \mathbb{N}$) with free computation-time. Such powers are needed in the computation of field inversion.

In the contrary, NB multiplication is known to have an expensive computation-time from software and hardware perspective. Given the fact that a minor reduction in the required number of NB multiplications, in the computation of field inversion for example, has a great effect on improving the overall performance. Accelerating NB multiplication is an active topic that attracts the interest of many academic researchers in the literature [31, 32].

Definition 14. *Given a basis element $\delta \in GF(p^m)$ that is a root of the irreducible normal polynomial $f(x)$, the set of basis elements $N = (\delta^{p^0}, \delta^{p^1}, \cdots, \delta^{p^{m-2}}, \delta^{p^{m-1}})$ defines a normal basis for $GF(p^m)$, with the property that none of its subsets adds to zero. In linear algebra terms, the elements in N are linearly independent.*

In using NB, any field element $\alpha \in GF(p^m)$ can be written as

$$\alpha = \sum_{i=0}^{m-1} a_i \delta^{p^i} \qquad \forall \;\; a_i \;\in\; GF(p), \tag{2.1}$$

with vector representation as $(a_0 a_1 \cdots a_{m-2} a_{m-1})_p$. Notice how a_{m-1} is on the right-hand side, whereas a_0 is on the left-hand side in such a representation.

Addition

Adding two extension field elements is a very straightforward operation in using NB. Given the field elements α and $\beta \in GF(p^m)$, with vector representations as

$$\alpha = (a_0 a_1 \cdots a_{m-2} a_{m-1})_p,$$

and

$$\beta = (b_0 b_1 \cdots b_{m-2} b_{m-1})_p,$$

respectively, then their sum is given by

$$\alpha + \beta = \sum_{i=0}^{m-1} (a_i + b_i \; mod \; p) \delta^{p^i}. \tag{2.2}$$

From Eq. (2.2) above, the addition of field elements is simply equivalent to coefficients-wise (mod p)-additions of the corresponding two field elements, whereby the coefficients are taken from the concerned subfield $GF(p)$. The addition of field elements is considered a fast execution-time field operation.

p^e-th Powers

The p^e-th powers (for characteristic p and a positive integer $e \in \mathbb{N}$) are necessary for the computation of field inversion in finite extension fields $GF(p^m)$. This subsection is dedicated mainly to describe how such powers are obtained in using NB for the field elements.

Theorem 2. *Given any nonzero element $\alpha \in GF(p^m)$ and an irreducible polynomial $f(x)$ of degree m, if the greatest common divisor $gcd(\alpha, p^m) = 1$, then α is invertible in $GF(p^m)$ and $\alpha^{p^m-1} \equiv 1 \mod f(x)$ holds (Fermat's little theroem).*

Corollary 2. *Fermat's theorem above asserts that $\alpha^{p^m} \equiv \alpha \mod f(x)$, which means that $GF(p^m)$ is a cyclic finite field.*

In using NB, since the characteristic p of the concerned field is included in exponent part of the basis elements, raising an extension field element to the *p-th* power is a linear operation. Therefore, given any nonzero element $\alpha \in GF(p^m)$, since

$$\alpha = \sum_{i=0}^{m-1} a_i \delta^{p^i} = (a_0 a_1 \cdots a_{m-2} a_{m-1})_p, \tag{2.3}$$

then

$$\alpha^p = \sum_{i=0}^{m-1} a_i \delta^{p^{i+1}} = (a_{m-1} a_0 \cdots a_{m-3} a_{m-2})_p. \tag{2.4}$$

By comparing Eqs. (2.3) and (2.4) given above, in general, the p^e-*th* powers are simply *e-th* right cyclic-shifts in using NB at software-level, or wiring re-arrangements at hardware-level (no special hardware is required to perform such an operation). In other words, raising an extension field element to a prime-power, whereby the characteristic p of the concerned field is the prime, is a linear operation with free execution-time in using NB.

Multiplication

Finite field multiplication using NB is the field operation that is being studied extensively in the academic literature, at both software and hardware levels. As it was mentioned earlier herein, in particular, extension field multiplication is the second costly operation in terms of the execution time in finite fields. Such a multiplication requires both subfield and extension field modular reductions, whereby in the worst case scenario, the number of times in which each of the aforementioned modular reductions is performed is given as a linear function of the extension degree m of the concerned extension field $GF(p^m)$.

Given $f(x)$ as the irreducible polynomial of the concerned field, to multiply two nonzero extension field elements α and $\beta \in GF(p^m)$, with vector representations as

$$\alpha = (a_0 a_1 \cdots a_{m-2} a_{m-1})_p,$$

and

$$\beta = (b_0 b_1 \cdots b_{m-2} b_{m-1})_p,$$

respectively, and

$$\gamma = (c_0 c_1 \cdots c_{m-2} c_{m-1})_p,$$

which represents the product vector, then, the extension field multiplication using NB is given by

$$\begin{aligned}\gamma = \alpha \times \beta &= \sum_{i=0}^{m-1} a_i \delta^{p^i} \times \sum_{j=0}^{m-1} b_j \delta^{p^j} \\ &= \sum_{i=0,j=0}^{m-1} a_i b_j \delta^{p^i} \delta^{p^j} = \sum_{i=0,j=0}^{m-1} a_i b_j \sum_{s=0}^{m-1} \lambda_{ij}^{(s)} \delta^{p^s} \\ &= \sum_{s=0}^{m-1} \left(\sum_{i=0,j=0}^{m-1} a_i b_j \lambda_{ij}^{(s)} \right) \delta^{p^s} = \sum_{s=0}^{m-1} \left(\sum_{i=0,j=0}^{m-1} a_{i+s} b_{j+s} \lambda_{ij}^{(0)} \right) \delta^{p^s}.\end{aligned}$$

Therefore, the coefficients of γ, i.e., c_s for $(0 \leq s \leq m-1)$, are computed as

$$c_s = \sum_{i=0}^{m-1} \sum_{j=0}^{m-1} a_{i+s} b_{j+s} \lambda_{ij}^{0},$$

where $[\lambda_{ij}^{0}]$ is the multiplication matrix M. It is an $m \times m$ matrix with entries belong to the subfield $GF(p)$. Its complexity, denoted as C_M, is defined as the number of nonzero entries. Such a matrix is well-defined and it is well-known that $C_M \geq 2m-1$ [33] in using NB.

Although it is not shown in the expressions above, both extension and subfield field modular reductions are required, especially, when the subfield multiplications for the coefficients of the concerned field elements produce values $> p$ (field's characteristic), and the exponents of the basis elements having values $> m$ (field's extension degree) of the concerned field.

Fast software implementation of NB multiplication is currently an active research topic in the academic literature [34, 35]. In part, this is due to the fact that such multiplication type has costlier execution-time than the multiplication using PB. Add to this, combined with the fact that the p^e-th powers are free using NB, this is the other reason that has pushed the academic researchers to currently put much of their effort to accelerate NB multiplication, to make it the basis of choice for field inversion in the near future.

2.2.2 Polynomial Basis

In the academic literature, polynomial basis representation (PB) is also referred to as the standard or canonical basis. This subsection is mainly dedicated to define such a basis along with its associated arithmetic operations. In other meaning, we are going to define the way in which field arithmetic operations are performed using such a representation basis for the field elements.

Definition 15. *Given a basis element $\delta \in GF(p^m)$ that is a root of the irreducible polynomial $f(x)$, the set of basis elements $P = (\delta^{m-1}, \delta^{m-2}, \cdots, \delta^1, \delta^0)$ defines a polynomial basis for $GF(p^m)$, with the property that none of its subsets adds to zero. In linear algebra terms, the elements in P are linearly independent.*

In using PB, any field element $\alpha \in GF(p^m)$ can be written as

$$\alpha = \sum_{i=0}^{m-1} a_i \delta^i \qquad \forall \;\; a_i \;\in\; GF(p), \tag{2.5}$$

with vector representation as $(a_{m-1}a_{m-2}\cdots a_1a_0)_p$. Notice how a_{m-1} is on the left-hand side, whereas a_0 is on the right-hand side in such a representation.

Addition

Adding two extension field elements is a very straightforward operation in using PB. Given the nonzero elements α and $\beta \in GF(p^m)$, with vector representations as

$$\alpha = (a_{m-1}a_{m-2}\cdots a_1a_0)_p,$$

and

$$\beta = (b_{m-1}b_{m-2}\cdots b_1b_0)_p,$$

respectively, then their sum is given by

$$\alpha + \beta = \sum_{i=0}^{m-1}(a_i + b_i \; mod \; p)\delta^i. \tag{2.6}$$

From Eq. (2.6) above, notice that the addition of field elements is simply equivalent to coefficients-wise (mod p)-additions of the corresponding two field elements, whereby the coefficients are taken from the concerned subfield $GF(p)$. The addition of field elements is considered a fast execution-time field operation.

p^e-*th* Powers

Raising an extension field element to a specific power, or performing an exponentiation operation using PB, whereby the exponent is a power of the characteristic p of the concerned field, works in a different way than that followed in using NB.

For example, in using PB, raising a field element to a p^e-*th* power is linear, but not a free-time operation, and must be performed through a specific linear map operation, referred to as the *e-th* iterate of Frobenius map, as it is evident from what follows.

Theorem 3. *Given any nonzero element $\alpha \in GF(p^m)$, the e-th iterate of Frobenius Map is defined as*

$$\mathbb{F}^e : \alpha \longmapsto \alpha^{p^e} \qquad \forall \text{ characteristic } p, \text{and a positive integer } e \in \mathbb{N}.$$

Notice that both elements α and $\alpha^{p^e} \in GF(p^m)$, which means that the *e-th* iterate of Frobenius map $\mathbb{F}^e$ is an automorphism, or a commutative endomorphism as it is also known in the academic literature. That is, it is a linear mapping from a finite field to itself, whereby the mapping inputs and outputs are belong to the same field.

Basically, $\mathbb{F}^e$ is a linear operation with respect to the coefficients $a_i \in GF(p)$, since it has no effect on them. But, it is not linear with respect to basis elements δ^i for $(1 \leq i \leq m-1)$, since it incurs significant increase in their exponents to values much greater than the field extension degree m, as evident from what follows: Given any nonzero element $\alpha \in GF(p^m)$ and the field irreducible polynomial $f(x)$, then we have $\alpha = \sum_{i=0}^{m-1} a_i \delta^i$, and its $\mathbb{F}^e$ is given by

$$\alpha^{p^e} = \sum_{i=0}^{m-1} a_i \delta^{ip^e} = a_0 + a_1 \delta^{p^e} + \cdots + a_{m-1} \delta^{(m-1)p^e} \pmod{f(x)} . \tag{2.7}$$

From Eq. (2.7) above, the effect of $\mathbb{F}^e$ on basis elements, i.e., δ^i for $(1 \leq i \leq m-1)$, falls within the scope of extension field modular reduction operation, which depends on the choice of field irreducible polynomial. The effect of choosing the appropriate field irreducible polynomial $f(x)$ on accelerating $\mathbb{F}^e$ (or the p^e-*th* power) will be clarified in subsequent sections, which is very useful in accelerating field inversion in using PB, as it was the case earlier in using NB for the field elements.

Multiplication

Finite field multiplication using PB is studied extensively in the academic literature, at both software and hardware levels. As it was mentioned earlier herein, PB multiplication requires less execution-time relative to NB multiplication. Nonetheless, extension field multiplication is the second costly operation in finite fields, after extension field inversion, since it still requires both subfield and extension field modular reductions, regardless of the used representation basis.

Given $f(x)$ as the irreducible field polynomial, to multiply two nonzero extension field elements α and $\beta \in GF(p^m)$, with vector representations as

$$\alpha = (a_{m-1} a_{m-2} \cdots a_1 a_0)_p,$$

and

$$\beta = (b_{m-1}b_{m-2}\cdots b_1b_0)_p,$$

respectively, and

$$\gamma = (c_{2m-2}c_{2m-3}\cdots c_1c_0)_p,$$

which represents the product vector, then, the extension field multiplication using PB is given by

$$\gamma = \alpha \times \beta = \sum_{i=0}^{m-1} a_i\delta^i \times \sum_{j=0}^{m-1} b_j\delta^j$$
$$= \sum_{i=0}^{m-1}\sum_{j=0}^{m-1}(a_ib_j \ mod \ p)\delta^{(i+j)} = \sum_{k=0}^{2m-2} c_k\delta^k,$$

whereby

$$c_k = \sum_{i+j=k}(a_ib_j \ mod \ p),$$

for $(0 \leq i, j \leq m-1)$ and characteristic p of the concerned field.

Several techniques to implement field multiplication using PB are exist in the academic literature. It is well-known that a single extension field multiplication is equivalent to exactly m^2 subfield multiplications, in addition to their associated subfield modular reductions, whereby m is the extension degree of the concerned extension field $GF(p^m)$.

Techniques such as the standard product, operand scanning, or product scanning can be combined with convolution-based algorithms such as Karatsuba's algorithm [36] to further accelerate the extension field multiplication. Such algorithms, in their functioning, rely on increasing the number of subfield additions in order to reduce the required number of subfield multiplications, those necessary to implement the required extension field multiplication in $GF(p^m)$.

2.3 Fermat's Inversion Approach

In general, there are two major approaches for calculating the inverse in finite fields. Regardless of the specific type of finite field under consideration and the used representation basis for the field elements, the first approach is based on Euclid's inversion method or one of its variants [37, 38]. The other inversion approach is based on Fermat's little theorem (FLT) [39, 40], which is the focus of attention in this dissertation.

2.3.1 Definition

The inversion approach based on FLT, or simply Fermat's inversion approach, since its discovery, has attracted the interest of many academic researchers, especially those who are interested in finding the inverse in finite extension fields using NB for the field elements. The fact that, in such approach, inverse calculation is reduced to calculating an exponentiation operation, that is calculated with frequent extension field multiplications and p^e-*th* powers, for characteristic p and any positive integer e, armed with the fact that the p^e-*th* powers are simply right cyclic-shifts using NB, such facts are the main reasons for the current interest in Fermat's inversion approach using NB. The following theorem defines such an approach in $GF(2^m)$.

Theorem 4. *Given any nonzero field element $\alpha \in GF(2^m)$ and an irreducible field polynomial $f(x)$ of degree m, if the greatest common divisor $gcd(\alpha, 2^m) = 1$, then α is invertible and $\alpha^{2^m-1} \equiv 1 \mod f(x)$ holds.*

Since α is invertible, dividing both sides in the expression $\alpha^{2^m-1} \equiv 1$ in the above theorem by α, gives us $\alpha^{2^m-2} = \alpha^{-1}$. Given the following series decomposition

$$2^m - 2 = 2^1 + 2^2 + \cdots + 2^{m-2} + 2^{m-1},$$

then we have,

$$\alpha^{-1} = \alpha^{2^m-2} = \alpha^{2^1+2^2+\cdots+2^{m-2}+2^{m-1}}.$$

Therefore, the inverse of α using FLT method is given by

$$\alpha^{-1} = \alpha^{2^1} \times \alpha^{2^2} \times \cdots \times \alpha^{2^{m-2}} \times \alpha^{2^{m-1}}. \tag{2.8}$$

From Eq. (2.8) above, field inversion using FLT method in its basic form requires $(m-2)$ $GF(2^m)$-multiplications and $(m-1)$ 2^e-th powers. When using NB for the field elements, from both software and hardware perspectives, field squaring is a right cyclic-shift and wiring rearrangement, respectively. Therefore, in using NB, field squarings are very fast operations compared to extension field multiplications.

Despite the use of the previous fact in the design of the fast Massey-Omura field multiplier [41], a NB multiplication still a very time-consuming operation relative to a field squaring in $GF(2^m)$. Therefore, in using NB, speeding up inverse calculation is achieved either by accelerating the execution time of NB multiplier, or by reducing the number of the required NB multiplications in FLT method, which is our main focus herein.

2.3.2 How it Works

The following example provides an idea on how FLT method works for finding the inverse in $GF(2^m)$. The example is represented by an expression showing the progressive computation for the inverse. The relevant figure is also included to illustrate the computation process graphically. Both of them, the expression and the figure, are included here to give the reader a better idea on how the method works. Given a nonzero extension field element $\alpha \in GF(2^6)$, then its inverse, i.e., α^{-1}, is computed using FLT method as follows

$$\alpha^{-1} = \alpha^{2^6-2} = \alpha^{62} = (\alpha \times (\alpha \times (\alpha \times (\alpha \times \alpha^2)^2)^2)^2)^2. \tag{2.9}$$

It is seen from the expression in Eq. (2.9) above, that the number of extension field multiplications are $(m-2) = (6-2) = 4$ $GF(2^6)$-multiplications. In addition, the number of squarings are $(m-1) = (6-1) = 5$ $GF(2^6)$-squarings. This is the straightforward (or the basic) way for computing the inverse. In subsequent chapters we will show the other available ways to compute the inverse fast by mathematically manipulating FLT method. Figure 2.1 illustrates inverse computation graphically based on FLT method.

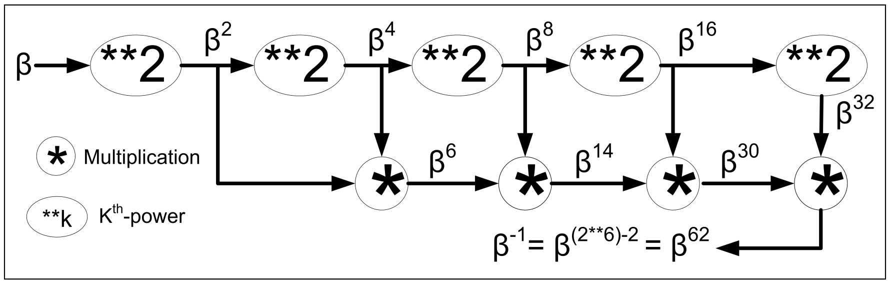


Figure 2.1: Graphical Inversion (FLT Method)

By referring to Figure 2.1 above, two computation lines are shown there representing the required number of extension field arithmetic operations necessary for inversion. The required number of such operations is exactly the same as that we previously obtained by using the expression in Eq. (2.9) above.

It can be seen from Figure 2.1 above that the required number of both extension field arithmetic operations, namely field squaring and multiplication operations, is a linear function with the extension degree m, which means, as m becomes very large the required number of such operations is also very large, thus, the execution time for field inversion using FLT method is very large too.

2.4 Short Addition Chains

The concept of short addition chains (SACs) is very useful and has many theoretical and practical applications. It provides a way for calculating terms like (α^{p^r-1}) in finite extension fields, whereby α is a nonzero field element, p is the field's characteristic and r is a positive integer. This subsection is dedicated mainly to provide an introduction on such a concept. More illustrative examples on such a concept exist in subsequent chapters, more specifically, in the places where it is required to calculate the inverse of a nonzero field element in binary and ternary extension fields.

2.4.1 Definition

Definition 16. *The Short Addition Chain (SAC) of a positive integer r, denoted as C_r, is a short chain (*sequence*) of elements (*integers*) of length l, defined with the property that r (*the last chain-element*) is obtained by the gradual addition of the previous elements within the chain (or the gradual addition of previous chain-elements) [42, 43].*

Mathematically, the SAC of a positive integer r is given by

$$C_r = (c_0, c_1, \cdots, c_{l-1}, c_l),$$

with $c_0 = 1$, $c_l = r$ and the i^{th} chain-element is given by

$$c_i = c_{i_1} + c_{i_2},$$

for $(0 \leq i, i_1, i_2 \leq l)$ and $(i > i_1, i_2)$. C_r is associated with another short sequence of integer pairs, whereby each pair is representing the i^{th} subsequent chain-element c_i in C_r, and is given by

$$A_r = ((c_{i_1}, c_{i_2}) \mid 0 \leq i_1, i_2 \leq l-1).$$

For example, given the positive integer $r = 18$, then

$$C_{18} = (1, 2, 4, 8, 16, 18), \tag{2.10}$$

which is associated with $A_{18} = ((1,1),(2,2),(4,4),(8,,8),(16,2))$ and following the rule $c_i = c_{i-1} + c_{i-1} = 2c_{i-1}$ for $i \in \{1,2,3,4\}$, except for $i = 5$, where $c_5 = c_4 + c_1$.

Another *SAC* for $r = 18$ is given by $C_{18} = (1,2,3,6,12,18)$, which is associated with $A_{18} = ((1,1),(2,1),(3,3),(6,,6),(12,6))$ following the rule $c_i = c_{i-1} + c_{i-1} = 2c_{i-1}$ for $i \in \{1,3,4\}$, and $c_i = c_{i-1} + c_{i-2}$ for $i \in \{2,5\}$.

2.4.2 How it Works

Given a nonzero field element $\alpha \in GF(2^m)$, SACs are used to perform calculations of the form (α^{2^r-1}) using C_r. More specifically, (α^{2^r-1}) term is obtained gradually by using the previous elements of C_r. The length l of C_r represents the number of the required extension field multiplications necessary to produce (α^{2^r-1}) term. Such calculation type using SACs can be generalized for any field characteristic p. For example, given a nonzero field element $\alpha \in GF(p^m)$, then calculating (α^{p^r-1}) term using C_r is possible too, which requires the minimal number of $GF(p^m)$-multiplications. Let us consider the following general example on calculations based on short addition chains (SACs).

Let $\alpha \in GF(p^m)$, and r a positive integer that has the following general SAC given by

$$C_r = (c_0, c_1, c_2, c_3, c_l), \tag{2.11}$$

whereby $c_0 = 1$ and $c_l = r$.

Given that the chain-element $c_1 = (c_0 + c_0)$, chain-element $c_2 = (c_1 + c_0)$, chain-element $c_3 = (c_2 + c_2)$, and chain-element $c_l = (c_3 + c_2)$, then the sequence of integer pairs is given by

$$A_r = ((c_0, c_0), (c_1, c_0), (c_2, c_2), (c_3, c_2)) .$$

The general steps for calculating the term $(\alpha^{p^r-1}) = (\alpha^{p^{c_l}-1})$ using C_r are given as in the following:

$$\begin{aligned}
\alpha^{p^{c_1}-1} &= (\alpha^{p^{c_0}-1})^{p^{c_0}} \times \alpha^{p^{c_0}-1} \\
\alpha^{p^{c_2}-1} &= (\alpha^{p^{c_1}-1})^{p^{c_0}} \times \alpha^{p^{c_0}-1} \\
\alpha^{p^{c_3}-1} &= (\alpha^{p^{c_2}-1})^{p^{c_2}} \times \alpha^{p^{c_2}-1} \\
\alpha^{p^{c_l}-1} &= (\alpha^{p^{c_3}-1})^{p^{c_2}} \times \alpha^{p^{c_2}-1}
\end{aligned}$$

From above calculation steps, it is apparent that the required extension field multiplications to compute (α^{p^r-1}) term are 4 $GF(p^m)$-multiplications that is exactly equal to the length of C_r, which is represented by the number of commas separating its elements (see Eq. (2.11) above).

Notice also that the required number of extension field multiplications is equivalent to $[\ell(r)+w(r)-2]$. Assuming $\alpha \in GF(2^m)$ and we need to calculate (α^{2^r-1}) term using C_r, given that $r = 18$, then we have $[\ell(18) + w(18) - 2] = 5$ $GF(2^m)$-multiplications. Such multiplications number to calculate the term is equal to the number of commas separating the elements in C_{18} (see E.q (2.10) above).

In subsequent chapters there will be some examples on how terms like (α^{p^r-1}) can be calculated using the concept of short addition chains. Such terms are necessary for finding the inverse in finite fields and are necessary to understand some of the proposed ideas in this dissertation.

Chapter 3

Literature Review

This chapter is mainly dedicated to review the state-of-the-art inversion algorithms, those previously proposed by interested academic researchers to solve field inversion problem, more specifically, for the elements in binary and ternary extension fields. The focus here is on inversion algorithms using NB in such fields and based on Fermat's inversion approach. In using such approach, field inversion problem is reduced to solving exponentiation in the concerned field.

In the first section, we review field inversion algorithms those previously proposed in $GF(2^m)$ using NB and based on Fermat's approach. In such algorithms, the various attempts by academic researchers for solving field inversion problem might attract the reader attention. The presentation will proceed in a sequential manner, with respect to the publication date of each author, which coincides with each author's attempt to improve the solution for field inversion problem in $GF(2^m)$.

In the second section, we review field inversion algorithms those previously proposed in $GF(3^m)$ based on Fermat's approach. Despite the fact that many academic authors in their work have relied on the same inversion approach, proposed earlier for such fields, however, some authors put more effort on improving the approach at hardware-level by using different techniques with some restrictions imposed to achieve their final goals.

3.1 Field Inversion in $GF(2^m)$

Binary extension fields $GF(2^m)$ are of paramount importance for use in ECC-based cryptographic applications. On the one hand, this is due to their suitability for hardware implementation, since they are simply binary vectors. On the other hand, this is due to the absence of rounding-errors and carry-propagation chains. In such fields the outputs of their arithmetic operations, when operating on their elements which consist of coefficients belong to the subfield $GF(2)$, are always reduced modulo 2, i.e., modular reductions are performed on the final results.

A group of interested academic researchers have proposed algorithms for fast field inversion in $GF(2^m)$. Such algorithms are based on FLT with the elements of the concerned field represented by using NB. This section is mainly dedicated to preview such inversion algorithms.

3.1.1 Itoh and Tsujii Algorithm

Itoh and Tsujii proposed an algorithm that significantly reduces the number of the required multiplications for inversion [14]. Their inversion algorithm is referred to as ITA in the sequel and can be described as follows: Given that $2^m - 2 = 2(2^{m-1} - 1)$, by representing $m - 1$ as a q-bit binary number $(m_{q-1}m_{q-2}\dots m_1m_0)_2$ with the most significant bit (MSB) $m_{q-1} = 1$, then we have

$$2^{m-1} - 1 \;=\; 2^{(m_{q-1}m_{q-2}\dots m_1m_0)_2} - 1. \tag{3.1}$$

Given that

$$2^{(m_j\dots m_0)_2} - 1 = (2^{m_j 2^j} - 1)2^{(m_{j-1}\dots m_0)_2} + 2^{(m_{j-1}\dots m_0)_2} - 1, \tag{3.2}$$

by applying Eq. (3.2) to Eq. (3.1) repeatedly with $j = q-1, q-2, \dots, 1$, then it follows

$$\begin{aligned} 2^{m-1}-1 &= (2^{m_{q-1}2^{q-1}}-1)2^{(m_{q-2}\dots m_0)_2} + (2^{m_{q-2}2^{q-2}}-1)2^{(m_{q-3}\dots m_0)_2} + \cdots \\ &\qquad \cdots + (2^{m_1 2^1}-1)2^{m_0} + (2^{m_0}-1). \end{aligned} \tag{3.3}$$

In addition, given that

$$\begin{aligned} (2^{m_j 2^j}-1) &= m_j(2^{2^{j-1}}+1)(2^{2^{j-1}}-1) \\ &= m_j(2^{2^{j-1}}+1)(2^{2^{j-2}}+1)\cdots(2^{2^0}+1)(2^{2^0}-1) \\ &= m_j(2^{2^{j-1}}+1)(2^{2^{j-2}}+1)\cdots(2^{2^0}+1), \end{aligned} \tag{3.4}$$

by substituting Eq. (3.4) in Eq. (3.3) with $j = q-1, q-2, \dots, 1$, and noting that $(2^{m_0}-1) = m_0$ and $m_{q-1} = 1$, then we have

$$\begin{aligned} 2^{m-1}-1 &= (\cdots((1+2^{2^{q-2}})2^{m_{q-2}2^{q-2}} + m_{q-2})(1+2^{2^{q-3}})2^{m_{q-3}2^{q-3}} + \cdots \\ &\qquad \cdots + m_1)(1+2^{2^0})2^{m_0 2^0} + m_0. \end{aligned}$$

Therefore, the inverse of a nonzero element $\alpha \in GF(2^m)$ using ITA expression is given by

$$\begin{aligned} \alpha^{-1} &= (((\cdots(((\alpha^{1+2^{2^{q-2}}})^{2^{m_{q-2}2^{q-2}}} \times \alpha^{m_{q-2}})^{1+2^{2^{q-3}}})^{2^{m_{q-3}2^{q-3}}} \times \cdots \\ &\qquad \cdots \times \alpha^{m_1})^{1+2^{2^0}})^{2^{m_0 2^0}} \times \alpha^{m_0})^2. \end{aligned} \tag{3.5}$$

The number of multiplication operations involved in Eq. (3.5) above can be estimated to be the sum of the following two parts: i) the number of '+' sign in any exponent, and ii) the number of '×' sign. It can be seen from Eq. (3.5) above that the number of '+' sign is $(q-1)$ or $[\ell(m-1)-1]$. The number of '×' sign depends on whether or not m_j is equal to one for $j = 0, 1, \dots, q-2$, since $\alpha^{m_j} = 1$ if $m_j = 0$ and the sign '×' immediately preceding α^{m_j} $(= 1)$ can be saved. Note that $m_{q-1} = 1$, thus we have part ii) is $[w(m-1)-1]$. So the total number of multiplication operations required to compute Eq. (3.5) above, or the inversion cost of ITA inversion algorithm is given by

$$[\ell(m-1) + w(m-1) - 2]\,. \tag{3.6}$$

ITA inversion algorithm was derived based on previous discussion, which is a translation for the expression given in Eq. (3.5) above (see Algorithm 1).

Algorithm 1 ITA Inversion Algorithm in $GF(2^m)$ [14]

```
Input: $\alpha \in GF^*(2^m)$, and $m-1 = (1m_{q-2}...m_1m_0)_2$.
Output: $\delta = \alpha^{2^m-2} = \alpha^{-1} \in GF(2^m)$
Initialization: $\delta := \alpha$;
for $i := q-2$ to 0 do
    $\delta := \delta \times \delta^{2^{2^i}}$;
    if $m_i = 1$ then
        $\delta := \alpha \times \delta^{2^{2^i}}$;
    end if
end for
$\delta := \delta^2$;
return $\delta$
```

3.1.2 Chang et al. Algorithm

In order to handle the worst case in ITA inversion algorithm especially when $(m-1)$ is a full-weight value, whereby $\ell(m-1) = w(m-1)$, many authors attempted to find various mathematical expressions, represented by decomposing $(m-1)$ value into several factors and remainders, in order to reduce the number of the required extension field multiplications in inverse calculation, or in other words, in order to lower the inversion cost.

Chang et al. [15] proposed CEA inversion algorithm that achieves more reductions in inversion cost in comparison with ITA algorithm in some $GF(2^m)$ by factoring $(m-1)$ into two non-trivial factors. Their algorithm can be described as follows: Given that $(m-1) = x \times y$, then we have

$$2^m - 2 = 2(2^{m-1} - 1) = 2(2^{x \times y} - 1). \tag{3.7}$$

By further expanding the expression in Eq. (3.7) above, then we have

$$2^m - 2 = 2(2^{x \times y} - 1) = (2^{x+1} - 2)((2^x)^{y-1} + (2^x)^{y-2} + \cdots + (2^x)^1 + (2^x)^0). \tag{3.8}$$

The expression given in Eq. (3.8) above can be rewritten as follows

$$2^m - 2 = 2 \times \left((2^x - 1)((2^x)^{y-1} + (2^x)^{y-2} + \cdots + (2^x)^1 + (2^x)^0)\right). \tag{3.9}$$

Therefore, α^{-1} can be written as

$$\alpha^{-1} = \alpha^{2^m-2} = \left((\alpha^{2^x-1})^{((2^x)^{y-1}+(2^x)^{y-2}+\cdots+(2^x)^1+(2^x)^0)}\right)^2.$$

Assume that $\delta = (\alpha^{2^x-1})$ has been computed with an instance of ITA algorithm by setting the algorithm input as α and $(m-1) = x$, which apparently requires $[\ell(x)+w(x)-2]$ multiplications. Given that factor y is represented by the *r-bit* binary vector $(1y_{r-2}\cdots y_1 y_0)_2$, by following a similar procedure to that previously used in the derivation of ITA algorithm, the inverse of a nonzero element $\alpha \in GF(2^m)$ using CEA expression is given by

$$\begin{aligned}\alpha^{-1} \;&= \Big(((\cdots(((\delta^{1+2^{x2^{r-2}}})^{2^{y_{r-2}x2^{y-2}}} \times \delta^{y_{r-2}})^{1+2^{x2^{r-3}}})^{2^{y_{r-3}x2^{r-3}}} \times \cdots \\ &\qquad \cdots \times \delta^{y_1})^{1+2^{x2^0}})^{2^{y_0x2^0}} \times \delta^{y_0})\Big)^2.\end{aligned} \tag{3.10}$$

In general, CEA inversion algorithm by itself consists of two portions. The first, is exactly an instance of ITA algorithm, which is necessary to calculate $\delta = (\alpha^{2^x-1})$ term as mentioned above. The second, is a similar instance of the algorithm, except for, the inclusion of the factor x in exponent part in computation steps as evident from CEA expression given in Eq. (3.10) above. Apparently, the second portion requires $[\ell(y) + w(y) - 2]$ multiplications. Thus, the total inversion cost of CEA inversion algorithm in some $GF(2^m)$ is given by

$$[\ell(x) + w(x) - 2] + [\ell(y) + w(y) - 2]\,. \tag{3.11}$$

It is evident from the above expression given in Eq. (3.11) that such factors, in some cases, can reduce the inversion cost further in comparison with the case of not factoring $(m-1)$ value. One thing to note is that CEA algorithm is not applicable in the case when $(m-1)$ is a prime value. Such prime values reduce the set of $GF(2^m)$ those expected to have low inversion cost based on their factoring method.

3.1.3 Takagi et al. Algorithm

Takagi et al. [16] proposed TYT inversion algorithm that achieves more reductions in inversion cost in comparison with CEA algorithm in some $GF(2^m)$ given that $(m-1) = \prod_{j=1}^{k} r_j + h$. TYT algorithm is a further improvement on top of CEA and ITA in two aspects: Firstly, it allows for prime $(m-1)$ values by using the remainder h (*cases not applicable using CEA algorithm*). Secondly, it allows for more than two factors as long as the inversion cost is low. TYT algorithm can be described as follows: Assuming that $(m-1) = \prod_{j=1}^{k} r_j + h$, i.e., decomposed into several factors and a small remainder h. In [16], the inverse equivalent value in $GF(2^m)$, i.e., $2^m - 2$, is expressed by using the following expression which is given by

$$2^m - 2 = 2^{m-1} + 2^{m-2} + \cdots + 2^{m-h} + 2^{m-h} - 2, \tag{3.12}$$

whereby h, the remainder, is any positive integer that determines the required number of terms in the expression.

Based on the expression given in Eq. (3.12) above, the inverse of a nonzero element $\alpha \in GF(2^m)$ using TYT expression is given by

$$\alpha^{-1} = \alpha^{2^m-2} = \alpha^{2^{m-1}} \times \alpha^{2^{m-2}} \times \cdots \times \alpha^{2^{m-h}} \times \alpha^{2^{m-h}-2}, \tag{3.13}$$

which can be rewritten as

$$\alpha^{-1} = \alpha^{2^m-2} \;=\; \underbrace{\alpha^{2^{m-1}} \times \alpha^{2^{m-2}} \times \cdots \times \alpha^{2^{m-h}} \times}_{h \text{ multiplications}} (\alpha^{2^{m-h-1}-1})^2. \tag{3.14}$$

Firstly, consider the computation of the underbraced terms given in Eq. (3.14) above. It is clearly evident that they require h multiplications. Secondly, given that $(m-h-1) = \prod_{j=1}^{k} r_j$, i.e., the factors portion in $(m-1)$, then the last term $(\alpha^{2^{m-h-1}-1}) = (\alpha^{2^{r_1 \times r_2 \times \cdots r_k}-1})$ in Eq. (3.14) above can be computed by applying CEA algorithm recursively: Firstly, let $x = r_1 \times \cdots \times r_{k-1}$, $y = r_k$, and $\delta = (\alpha^{2^x-1})$, then based on the expression given in Eq. (3.10) above, computing δ requires $[\ell(r_k) + w(r_k) - 2]$ multiplications. Secondly, let $x = r_1 \times \cdots \times r_{k-2}$, $y = r_{k-1}$, and $\delta = (\alpha^{2^x-1})$, to compute δ, or equivalently $(\alpha^{2^{r_1 \times \cdots \times r_{k-1}}-1})$ term it requires $[\ell(r_{k-1}) + w(r_{k-1}) - 2]$ multiplications, etc. Finally, to compute $(\alpha^{2^{r_1}-1})$ term using an instance of ITA algorithm requires $[\ell(r_1) + w(r_1) - 2]$ multiplications. Thus, the total inversion cost of TYT inversion algorithm in some $GF(2^m)$ is given by

$$\left(\sum_{j=1}^{k} \ell(r_j) + w(r_j) - 2 \right) + h. \tag{3.15}$$

It is evident from the above expression given in Eq. (3.15) above that the inversion cost, in terms of the required number of extension field multiplications, is a logarithmic function with each individual factor in $(m-1)$, and a linear function with the remainder h. However, in [16] h was restricted to the value of 1. Although it handled the prime $(m-1)$ case, but this restriction reduces the set of $GF(2^m)$ those expected to have low inversion cost based on their decomposition method.

Algorithm 2 TYT Inversion Algorithm in $GF(2^m)$ [16]

```
Input: $\alpha \in GF^*(2^m)$, and $m$
Output: $\eta = \alpha^{-1} \in GF(2^m)$
Initial: $(m-1)$;
if $(m-1)$ is not decomposed then
    return $\eta := ITA(\alpha, m-1)$;
else
    $(m-1) = \prod_{j=1}^{k} r_j + h$;
    $\delta := ITA(\alpha, r_1)$;
    $r := 1$;
    for $j := 2$ to $k$ do
        $\eta := \delta$;
        $r := r \times r_{j-1}$;
        for $i := q_j - 2$ to $0$ do
            $\eta := \eta \times \eta^{2^{r2^i}}$;
            if $m_i^{(j)} = 1$ then
                $\eta := \delta \times \eta^{2^{r2^i}}$;
            end if
        end for
        $\delta := \eta$;
    end for
    for $i := 1$ to $h$ do
        $\eta := \eta \times \alpha^{2^{m-i}}$;
    end for
    return $\eta$
end if
```

TYT inversion algorithm was derived based on previous discussion, which is a translation for the expression given in Eq. (3.14) above (see Algorithm 2).

Given that the inversion cost depends mainly on the way of decomposing $(m-1)$ value, the authors in [16] have defined an optimal decomposition (OD) criterion. Therefore, the OD for $(m-1)$ value is the one that minimizes the number of the required extension field multiplications necessary for inversion, or the inversion cost, and consists of the fewest components (*factors and remainders*) based on TYT method. Hence, an exhaustive search with efficient pruning is necessary to find such optimal decomposition.

3.1.4 Li et al. Algorithm

Li et al. [17] proposed LCA algorithm that achieves more reductions inversion cost in comparison with TYT algorithm in some $GF(2^m)$ given that $(m-1) = \prod_{j=1}^{k} r_j + h$, i.e., decomposed into several factors plus a remainder h. LCA algorithm attains two aspects of improvement on top of TYT algorithm: Firstly, in LCA algorithm the remainder h is not restricted to a single value, but to a specific set of values. This in fact increase the set of $GF(2^m)$ those associated with low inversion cost based on their decomposition method. Secondly, LCA algorithm reuse some intermediate results to save $\ell(h)$-dependent computations. This in fact reduce the inversion cost further and render it a function of the Hamming weight of h, i.e., $w(h)$. This is why in [17], the remainder h is restricted to minimum $w(h)$ values. LCA algorithm can be described as follows: Given the following expression

$$2^m - 2 = 2^{m-1} + 2^{m-2} + \cdots + 2^{m-h} + 2^{m-h} - 2,$$

by rearranging the expression with $(m-1)$ in mind, then we have

$$2^m - 2 = 2^{m-h}\left(2^h - 1\right) + 2\ \left(2^{m-h-1} - 1\right). \tag{3.16}$$

Since $(m-1) = \prod_{j=1}^{k} r_j + h$, then $(m-h-1) = \prod_{j=1}^{k} r_j$, the available factors in $(m-1)$. Therefore, the inverse of a nonzero element $\alpha \in GF(2^m)$ using LCA expression is given by

$$\alpha^{-1} = \alpha^{2^m-2} = \left(\alpha^{2^h-1}\right)^{2^{m-h}} \times \left(\alpha^{2^{\prod_{j=1}^{k} r_j}-1}\right)^2 = \left(\alpha^{2^h-1}\right)^{2^{m-h}} \times \left((\alpha^{2^{r1}-1})^2\right)^e,$$

given that,

$$e = \left((2^{r_1})^{r_2-1} + \cdots + 1)\cdots((2^{r_1\times\cdots\times r_{k-1}})^{r_k-1} + \cdots + 1)\right). \tag{3.17}$$

The variable e in Eq. (3.17) above is computed is in a way similar to our previous computation for the factors in $(m-1)$, exactly when we considered TYT algorithm as given in Eq. (3.14) above, which requires $\sum_{j=2}^{k}(\ell(r_j)+w(r_j)-2)$ multiplications.

With the focus on both $(\alpha^{2^{r_1}-1})$ and (α^{2^h-1}) terms in Eq. (3.17) above, more reduction in inversion cost is obtained as follows: Given that

$$r_1 = \sum_{i=1}^{n} 2^{u_i},$$

with $u_1 > u_2 > \cdots > u_n$, and

$$h = \sum_{i=1}^{l} 2^{t_i},$$

with $t_1 > t_2 > \cdots > t_l$. Now if $r_1 \geq h$, then we have $u_1 \geq t_1$. In addition, given that

$$(\alpha^{2^{r_1}-1}) = (\alpha^{s_{u_n}})(\cdots(\alpha^{s_{u_3}})((\alpha^{s_{u_2}})(\alpha^{s_{u_1}})^{2^{2^{u_2}}})^{2^{2^{u_3}}}\cdots)^{2^{2^{u_n}}}, \tag{3.18}$$

and

$$(\alpha^{2^h-1}) = (\alpha^{s_{t_l}})(\cdots(\alpha^{s_{t_3}})((\alpha^{s_{t_2}})(\alpha^{s_{t_1}})^{2^{2^{t_2}}})^{2^{2^{t_3}}}\cdots)^{2^{2^{t_l}}} \tag{3.19}$$

as in [17], thus in computing $(\alpha^{s_{u_1}})$ term in Eq. (3.18) above (*which depends on $\ell(r_1)$, the binary-length of* r_1), all other terms such as $(\alpha^{s_{u_i}})$ and $(\alpha^{s_{t_i}})$ given in Eqs. (3.18) and (3.19) for $(1 \leq i \leq n, l)$ are available in intermediate results. Therefore, finding the inverse using LCA algorithm proceeds as follows: Assuming that $(\alpha^{2^{r_1}-1})$ term is computed using an instance of ITA algorithm, which requires $[\ell(r_1)+(r_1)-2]$ multiplications, add to this $\sum_{j=2}^{k}(\ell(r_j)+w(r_j)-2)$ multiplications of variable e, with an extra $w(h)$ multiplications of the remaining computations necessary to calculate (α^{2^h-1}) term, then the inverse of a nonzero element $\alpha \in GF(2^m)$ is computed using LCA algorithm with inversion cost that is given by

$$\left(\sum_{j=1}^{k} \ell(r_j) + w(r_j) - 2\right) + w(h). \tag{3.20}$$

It is evident from Eq. (3.20) above that the inversion cost of LCA algorithm, is a logarithmic function with each individual factor in $(m-1)$, and a function with the Hamming-weight of the remainder h, i.e., $w(h)$, rather than a function with h itself as in using TYT inversion algorithm.

In [17], OD criterion is again defined in a similar way as in [16]. Therefore, the OD for $(m-1)$ is the one that minimizes the number of the required extension field multiplications necessary for inversion, or the inversion cost, and consists of the fewest components (*factors and remainders*) based on LCA method. Hence, an exhaustive search with efficient pruning is required to find such optimal decomposition.

3.2 Field Inversion in $GF(3^m)$

In general, characteristic three (or ternary) extension fields $GF(3^m)$ are of paramount importance for use in ECC-based cryptographic applications, and in particular, for use in cryptographic applications based on bilinear mapping in ECC. On the one hand, this is due to the availability of field irreducible binomials those necessary to accelerate some arithmetic operations in such fields. On the other hand, this is due to their bandwidth efficiency relative to other extension fields in calculating the Weil and Tate pairings in ECC.

A group of interested academic researchers have proposed algorithms for fast field inversion in $GF(3^m)$. Such algorithms are mainly based on the approach that previously proposed by Itoh and Tsujii in [23]. In what follows, we review the relevant inversion algorithms in $GF(3^m)$ based on such approach.

3.2.1 ITI Algorithm

Presumably, the approach that previously proposed by Itoh and Tsujii [23] is considered as the standard method for finding the inverse in odd-characteristic extension fields $GF(p^m)$, for odd-prime characteristic p and $m > 1$. In fact, many academic researchers have relied on such approach with attempts to improve it from software and hardware perspectives. This was achieved by applying their own representation basis and domain parameters, while in the mean time, imposing the specific restrictions and conditions to achieve their final goals. This has encouraged the lack, to some extent, of other inversion approaches for odd-characteristic extension fields $GF(p^m)$ in the literature. In this subsection, we fill this gap by proposing our inversion approach in such fields, more specifically, in $GF(3^m)$.

The basic idea in [23] can be described as follows: Given the positive integer $r = \frac{p^m-1}{p-1}$, which is the quotient when dividing both orders for the multiplicative groups of the extension field, i.e., $GF^*(p^m)$, and its subfield, i.e., $GF^*(p)$, then we have

$$r = \frac{p^m - 1}{p - 1} = p^{m-1} + p^{m-2} + \cdots + p^1 + 1, \tag{3.21}$$

and

$$(r - 1) = p^{m-1} + p^{m-2} + \cdots + p^1, \tag{3.22}$$

which is equivalent to the *p-dic* representation of $(r - 1) = (11\ldots110)_p$. Therefore, given any nonzero element $\alpha \in GF(p^m)$, its inverse based on ITI expression is given by

$$\alpha^{-1} = (\alpha^r)^{-1} \times \alpha^{r-1} = (\alpha^r)^{-1} \times \overbrace{\alpha^{p^{m-1}+p^{m-2}+\cdots+p^2+p^1}}^{(r-1)} \tag{3.23}$$

The inverse of α using ITI expression given in Eq. (3.23) above is computed by using the following four steps:

Step 1. Exponentiation in $GF(p^m)$ to get the element α^{r-1}

Step 2. $GF(p^m)$-multiplication of $\alpha.\alpha^{r-1}$ to get the element α^r

Step 3. Subfield inversion for α^r to get the element $\beta = (\alpha^r)^{-1}$

Step 4. $GF(p^m)$-multiplication of $\beta.\alpha^{r-1}$ to get the inverse α^{-1}

Indeed, the output of the above four steps is the inverse of the nonzero element $\alpha \in GF(p^m)$. Notice that when performing the above four steps, we are reducing the computation of extension field inversion to a subfield inversion, in addition to the necessity to perform some logarithmic number of extension field multiplications. Also, notice that α^r is a subfield element, i.e., $\alpha^r \in GF(p)$, since the mathematical expression for the exponent r, is equivalent to finding the **norm** for the extension field element α. The norm is a map from an extension field element to a subfield element. In addition, notice that terms calculation in the overbraced expression relevant to $(r-1)$ given in Eq. (3.23) above require the computation of the *e-th* iterates of the Frobenius map, for all $e \in \{1, 2, \cdots, m-1\}$ (see Chapter 2.2.2).

The straightforward approach to compute α^{r-1} term in Step 1. above requires exactly $(m-2)$ extension field multiplications (denoted as $GF(p^m)$-multiplications). Thus, extension field inversion is computed with two extra $GF(p^m)$-multiplications and a subfield inversion, which totals to m $GF(p^m)$-multiplications plus a subfield inversion. In using the straightforward approach for inverse computation, the required number of $GF(p^m)$-multiplications is a linear function with the extension degree m.

Inversion algorithm in [23] was proposed mainly for use with polynomial basis representation (PB) and, the extension field $GF(p^m)$ was generated using an irreducible polynomial of the form $f(x) = x^m + \sum_{i=0}^{m-1} f_i x^i$ defined over $GF(p)$. In using PB, the Frobenius map (see Chapter 2.2.2) is an important operation for inversion algorithms, which is necessary for generating the p^e-*th* powers, for characteristic p and any positive integer e.

In [23], the *e-th* iterate of Frobenius map is calculated using $M \times M$ matrix, with complexity of m^2 subfield multiplications. The algorithm was slightly modified to perform as many subsequent *p-th* powers as possible between $GF(p^m)$-multiplications, in order to reduce the required number of the *e-th* iterates of Frobenius map to a logarithmic value.

In the academic literature, it has been mentioned that applying the concept of addition chains (ACs) is the best alternative for computing α^{r-1} term given in Step 1. above, which requires a logarithmic performance, in terms of the required number of $GF(p^m)$-multiplications. The term is computed with $[\ell(m-1)+w(m-1)-\ 2]$ $GF(p^m)$-multiplications in [23]. With extra two $GF(p^m)$-multiplications and a subfield inversion, in using the ACs approach for inversion, the inversion cost of a nonzero element $\alpha \in GF(p^m)$ using ITI expression is given by

$$\begin{gathered}
[\ell(m-1)+w(m-1)]\ \ GF(p^m)\text{-Multiplications,} \\
\text{Subfield GF(p)-Inversion,} \\
[\ell(m-1)+w(m-1)-1]\ \ GF(p^m)\text{-Frobenius Maps.}
\end{gathered} \tag{3.24}$$

The authors in [20] have followed the ACs approach for inversion but, they accelerated the Frobenius map operation further in using PB for the field elements. Given that $\alpha = \sum_{j=0}^{m-1} a_j x^j$ using PB, after expressing the *i-th* iterate of the Frobenius map for a nonzero element $\alpha \in GF(p^m)$ as follows

$$\alpha^{p^i}(x) = a_{m-1}x^{(m-1)p^i} + \cdots + a_1 x^{p^i} + a_0 \mod f(x), \tag{3.25}$$

for any positive integer i representing the *i-th* iterate and $(0 \le j \le m-1)$, whereby $f(x) = x^m - w$ (*the field irreducible binomial*) is defined over $GF(p)$, $w \in GF(p)$, the authors in [20] have focused on the elements that are not kept fixed, like (x^{jp^i}) terms those given in Eq. (3.25) above for all $(1 \le j \le m-1)$.

By making use of the special properties of the field irreducible binomial $f(x)$, the authors in [20] expressed the terms affected by the *i-th* iterate of the Frobenius map as follows

$$x^e \equiv w^q x^s \mod f(x),$$

whereby $s \equiv e \pmod{m}$, $q = \frac{e-s}{m}$ and $e = jp^i$. Based on Corollary 2 in their paper, they further reduced the expression to

$$x^e \equiv w^q x^j \mod f(x).$$

From above discussion, the cost of the *i-th* iterate of Frobenius map is exactly $(m-1)$ subfield multiplications that still requires subfield modular reductions, in addition to the pre-computed values of $w^q x^j$ terms for all $(1 \leq j \leq m-1)$. The Frobenius map operation, as it was mentioned earlier herein, is needed for performing the p^i-*th* powers necessary for field inversion. The number of the required $GF(p^m)$-multiplications is not changed in comparison with [23]. Thus, the inversion cost in [20] is identical to that value given in Eq. (3.24) above, except that, the Frobenius map operation is computed fast based on their method.

The advantage of using normal basis representation (NB) for field inversion in odd-characteristic extension fields, more specifically, in $GF(3^m)$ is utilized in [44]. As it was mentioned earlier herein (see Chapter 2.2.1), in using NB the *e-th* iterates of Frobenius map are simply reduced to either *e-th* right cyclic-shifts in software, or *e-th* permutations in hardware. Therefore, the computation of the *e-th* iterates of Frobenius map is totally free in using NB for the field elements.

Again, the cost analysis in [44] is identical to that given in Eq. (3.24) above, except that, the *e-th* iterates of Frobenius map became free-time operations using NB for the field elements and, the PB multipliers are replaced with NB multipliers. Currently, accelerating NB multipliers is an active research topic by many interested researchers in the academic literature [34, 45].

As a final note, the above ITI expression is not translated to a cohesive and complete field inversion algorithm in $GF(p^m)$, or even in $GF(3^m)$. The academic researchers just mentioned the use of the addition chains to calculate α^{r-1} term, without writing an algorithm to perform such a task or even the task of finding the inverse by itself, which is clearly evident from the above presentation.

Chapter 4

Proposed Inversion Algorithms

This chapter is mainly dedicated to present the proposed ideas herein. In the first section, we are going to present our proposed decomposition algorithm for m of the concerned $GF(2^m)$. The algorithm employ a decomposition method that achieve further reduction in inversion cost in comparison with other methods in some $GF(2^m)$. Then, we are going to show our main field inversion algorithm that proposed to implement the decomposition algorithm using NB for the field elements, which relies on FLT inversion approach. Furthermore, we are going to provide all other algorithms required to assist in the functioning of the main inversion algorithm.

In the second section, we are going to show our field inversion algorithms those mainly proposed for use in $GF(3^m)$ using NB for the field elements, which rely on FLT inversion approach. Our first inversion algorithm in such fields is somehow based on ITI expression (mentioned earlier herein). It is the first algorithm that has been written in full for finding the inverse in $GF(3^m)$ in the literature. In addition, it has a relatively lower inversion cost in comparison with other algorithms that are exactly based on ITI expression.

Our second inversion algorithm in $GF(3^m)$ is basically based on the decomposition algorithm that proposed earlier by us in $GF(2^m)$ (see Chapter 4.1). The algorithm simply implements the decomposition method employed in that algorithm for computing the inverse in $GF(3^m)$. We have modified most of previous algorithms in $GF(2^m)$ to implement the decomposition algorithm in $GF(3^m)$. The algorithm is expected to have the lowest possible inversion cost in such fields. Finally, we are going to propose a fast Frobenius map operation in such fields and extend it to higher characteristic fields, which requires almost free runtime and space complexity, as it will be clear later on in this chapter.

4.1 Characteristic Two Algorithm

Given that $(m-1) = \prod_{j=1}^{k} r_j + h$, while the inversion cost of TYT algorithm depends on the value of the remainder h, which is restricted to the value of 1 (see Chapter 3.1.3), the inversion cost of LCA algorithm depends on the Hamming weight of h, and this is why h is restricted to minimum Hamming weight values (see Chapter 3.1.4).

In this section, we propose an improved field inversion algorithm in $GF(2^m)$ using NB for the field elements, which is based on FLT inversion approach. The inversion cost, in terms of the number of the required extension field multiplications necessary for inversion, of the algorithm is dependent on neither the value nor the Hamming weight of h and is given by

$$\left(\sum_{j=1}^{k}[\ell(r_j) + w(r_j) - 2] + 1\right).$$

Furthermore, in the proposed algorithm the value of h is not limited to small values or low Hamming weight values as in other algorithms. This in fact increase the set of $GF(2^m)$ those expected to have the lowest possible inversion cost based on the decomposition method of our field inversion algorithm.

Our proposed field inversion algorithm in $GF(2^m)$ can be described as follows: Given that $(m-1) = \prod_{j=1}^{k} r_j + h$, and since

$$2^m - 2 = 2\ \left(2^{m-1} - 1\right), \tag{4.1}$$

by substituting the value of $(m-1)$ in the exponent part in Eq. (4.1) above, then we have

$$2^m - 2 = 2\ \left(2^{\prod_{j=1}^{k} r_j + h} - 1\right),$$

which can be expressed as

$$2^m - 2 = 2\ \left(2^h \times (2^{\prod_{j=1}^{k} r_j} - 1) + (2^h - 1)\right). \tag{4.2}$$

By performing further mathematical manipulations on the expression in Eq. (4.2) above, then it can be expressed as follows

$$2^m - 2 = 2\ \left((2^{r_1} - 1) \times e \times 2^h + (2^h - 1)\right), \tag{4.3}$$

whereby the expression for variable e is given in the following equation for the inverse.

Therefore, based on Eq. (4.3) above, the inverse of a nonzero element $\alpha \in GF(2^m)$ using our proposed IVR expression is given by

$$\alpha^{-1} = \alpha^{2^m-2} = \left(\alpha^{2^{m-1}-1}\right)^2 = \left(\alpha^{2^{\prod_{j=1}^{k} r_j + h}-1}\right)^2 =$$
$$\left((\alpha^{2^{r_1 \times \cdots \times r_k}-1})^{2^h} \times (\alpha^{2^h-1})\right)^2 = \left((\alpha^{2^{r_1}-1})^{e2^h} \times (\alpha^{2^h-1})\right)^2,$$

given that

$$e = \left(((2^{r_1})^{r_2-1} + \cdots + 1) \cdots ((2^{r_1 \times \cdots \times r_{k-1}})^{r_k-1} + \cdots + 1)\right). \tag{4.4}$$

Theorem 5. *Given that* $(m-1) = \prod_{j=1}^{k} r_j + h$. *Assuming that* h *is appropriately selected to belong to the short addition chain of* r_1, *denoted as* C_{r_1}, *or any factor in* $(m-1)$, *then, the computation of* $(\alpha^{2^{r_1}-1})$ *term guarantees the availability of* (α^{2^h-1}) *term as an intermediate value, and the inversion cost is given by*

$$\left(\sum_{j=1}^{k} [\ell(r_j) + w(r_j) - 2] + 1\right)\ GF(2^m)\text{-Multiplications.} \tag{4.5}$$

Proof. From Eq. (4.4) above we have the inverse of α that is given by

$$\alpha^{-1} = \left[(\alpha^{2^{r_1}-1})^{e2^h} \times (\alpha^{2^h-1})\right]^2. \tag{4.6}$$

In using NB in $GF(2^m)$ we can safely assume that all power of two exponents are free. Such powers are simply cyclic-shifts. So the inverse expression is reduced to the following

$$\alpha^{-1} = \left[(\alpha^{2^{r_1}-1})^{e} \times (\alpha^{2^h-1})\right]. \tag{4.7}$$

If we forget about the computational cost of $(\alpha^{2^{r_1}-1})$ term for now (we return to it before ending the proof), the computational cost of $(\alpha^{2^{r_1}-1})^e$ term is given by

$$\sum_{j=2}^{k}[\ell(r_j) + w(r_j) - 2], \tag{4.8}$$

and its detailed steps are given as follows: Firstly, the computational cost of $(\alpha^{2^{r_1}-1})^{(2^{r_1})^{r_2-1}+\cdots+(2^{r_1})^0} = (\alpha^{2^{r_1\times r_2}-1})$ term is $[\ell(r_2) + w(r_2) - 2]$ multiplications. Secondly, the computational cost of $(\alpha^{2^{r_1\times r_2}-1})^{(2^{r_1\times r_2})^{r_3-1}+\cdots+(2^{r_1\times r_2})^0} = (\alpha^{2^{r_1\times r_2\times r_3}-1})$ is $[\ell(r_3) + w(r_3) - 2]$ multiplications. Finally, the computational cost of

$$(\alpha^{2^{r_1\times\cdots\times r_{k-1}}-1})^{(2^{r_1\times\cdots\times r_{k-1}})^{r_k-1}+\cdots+(2^{r_1\times\cdots\times r_{k-1}})^0} = (\alpha^{2^{r_1\times\cdots\times r_k}-1}) = (\alpha^{2^{r_1}-1})^e,$$

is $[\ell(r_k) + w(r_k) - 2]$ multiplications. Thus, the computational cost relevant to all factors in $(m-1)$, except for r_1, is exactly as given in Eq. (4.8) above.

By returning to $(\alpha^{2^{r_1}-1})$ term, its computational cost can be given as follows: Let the short addition chain (SAC) for r_1 given by

$$C_{r_1} = \{c_0, c_1, c_2, c_3\},$$

whereby $c_0 = 1$ and $c_3 = r_1$. In addition, let the sequence of integer pairs of C_{r_1} given by

$$A_{r_1} = \{(c_0, c_0), (c_1, c_1), (c_2, c_2)\},$$

using the addition rule $c_1 = c_0 + c_0$, $c_2 = c_1 + c_1$, and $c_3 = c_2 + c_2$. Then, it follows that $(\alpha^{2^{r_1}-1}) = (\alpha^{2^{c_3}-1})$ is computed as follows

$$\begin{aligned}
(\alpha^{2^{c_0}-1})^{2^{c_0}} \times (\alpha^{2^{c_0}-1}) &= (\alpha^{2^{c_0+c_0}-1}) = (\alpha^{2^{c_1}-1}) \\
(\alpha^{2^{c_1}-1})^{2^{c_1}} \times (\alpha^{2^{c_1}-1}) &= (\alpha^{2^{c_1+c_1}-1}) = (\alpha^{2^{c_2}-1}) \\
(\alpha^{2^{c_2}-1})^{2^{c_2}} \times (\alpha^{2^{c_2}-1}) &= (\alpha^{2^{c_2+c_2}-1}) = (\alpha^{2^{c_3}-1}) \qquad (4.9)
\end{aligned}$$

By assigning any integer values for the elements of the chain C_{r_1} which satisfy the addition rule of A_{r_1}; let's say for example $c_0 = 1, c_1 = 2, c_2 = 4, c_3 = r_1 = 8$ such that $C_{r_1} = C_8 = \{1, 2, 4, 8\}$, then $[\ell(r_1) + w(r_1) - 2] = [\ell(8) + w(8) - 2] = 3$. The obtained value, namely 3, is exactly equal to the number of multiplications necessary to compute $(\alpha^{2^{r_1}-1}) = (\alpha^{2^8-1})$ term as given in Eq. (4.9) above. Or in other words, the value 3 is equal to the length of $C_{r_1} = C_8$, which represents the number of commas that separate the chain elements. Therefore, the computational cost of $(\alpha^{2^{r_1}-1})$ term is given by

$$[\ell(r_1) + w(r_1) - 2], \qquad (4.10)$$

multiplications. Notice that, if $h = c_i$ for $i \in [0, 1, 2, 3]$, then $(\alpha^{2^h-1}) = (\alpha^{2^{c_i}-1})$ is available as one of the intermediate results as evident from Eq. (4.9) above. Thus, all computational cost relevant to (α^{2^h-1}) term are saved, and we need only one extra multiplication to join the terms in Eq. (4.7) above. Therefore, by adding the costs given in Eqs. (4.8) and (4.10) plus an additional multiplication necessary to join (α^{2^h-1}) term, then the inversion cost of our proposed inversion algorithm is exactly as given in Theorem 5 above.

□

From the proof of Theorem 5 shown above, it is clearly evident that the inversion cost of our proposed field inversion algorithm in $GF(2^m)$ only depend on the constant 1 (*independent of the remainder* h), rather than its dependency on h itself as in TYT inversion algorithm, or the dependency on $w(h)$ as in LCA inversion algorithm.

In addition, the above inversion cost is applicable to a wide range of binary extension fields, especially after decomposing each extension degree m of the concerned $GF(2^m)$ using the decomposition method employed in our proposed decomposition algorithm. Similar to the case in other existing inversion algorithms, which rely on different types of decomposition methods, however, the set of $GF(2^m)$ those associated with the minimal inversion cost is quite wider using our proposed inversion algorithm. This is mainly true because the remainder h is not restricted in its value.

In the following, we introduce our proposed algorithms necessary for inversion in $GF(2^m)$ through a running example. We are going to consider the field $GF(2^{163})$ in our example. The selection criterion for this field is that it is one of the recommended fields for use in ECC. Assuming that $m = 163$ is the input to our proposed decomposition algorithm, which is referred to as $WHDA(m)$ (see Algorithm 3), then the output is the 3-tuple $(5, 32, 2)$. The 3-tuple means that $(m-1) = (163-1) = 162 = 5 \times 32 + 2$.

Given that the short addition chain for factor $r_1 = 5$ is given by $C_{r_1} = C_5 = \{1, 2, 3, 5\}$, then the sequence of integer pairs of C_5 is given by $A_{r_1} = A_5 = \{(1,1), (2,1), (3,2)\}$. The sequence follows the rule $c_i = c_{i-1} + c_{i-1}$ for $i = 1$, and $c_i = c_{i-1} + c_{i-2}$ for $i \in \{2, 3\}$, given that $c_0 = 1$ and $c_3 = 5$.

In continuation with our example, we assume that the field element in which its inverse is required is $\alpha \in GF^*(2^{163})$. By using the 3-tuple $(5, 32, 2)$, the above sequences we just obtained for factor $r_1 = 5$ (i.e., C_5, A_5) and the element α as the inputs to our proposed $WHCA(e, C_v, A_v, \kappa)$ algorithm, somehow follows the approach presented in [46] with major modifications, such as the incorporation of all code relevant to our decomposition method (see Algorithm 4). For example, if $e = \alpha$, $C_v = \{1, 2, 3, 5\}$, $A_v = \{(1, 1), (2, 1), (3, 2)\}$ and $\kappa = 2$, then, the computational steps using the algorithm proceed as follows:

$$\begin{aligned}
(\alpha^{2^1-1})^{2^1} \times (\alpha^{2^1-1}) &= (\alpha^3) = (\alpha^{2^2-1}) \\
(\alpha^{2^2-1})^{2^1} \times (\alpha^{2^1-1}) &= (\alpha^7) = (\alpha^{2^3-1}) \\
(\alpha^{2^3-1})^{2^2} \times (\alpha^{2^2-1}) &= (\alpha^{31}) = (\alpha^{2^5-1})
\end{aligned} \tag{4.11}$$

Algorithm 3 $WHDA(m)$ Algorithm in $GF(p^m)$

```
Input: extension degree $m$
Output: $(m-1) = r_1 \times n + h$ given as $(r_1, n, h)$
Initial: $t := (m-1),\ l := \sqrt{t},\ j := 0$;
**if** $t$ is odd: $S \leftarrow S_o = \{1, 3, \cdots, l := \lceil l \rceil\}$. Use $l := \lfloor l \rfloor$ if $l$ odd;
**if** $t$ is even: $S \leftarrow S_e = \{2, 4, \cdots, l := \lfloor l \rfloor\}$. Use $l := \lceil l \rceil$ if $l$ odd;
**for all** $i$ in the selected $S$ **do**
  $R := t - i$;
  **while** ($R \neq 2^k$ **and** $2|R$ **and** $\lceil R/2 \rceil \geq i$) **do**
    $R := \left(\frac{R}{2}\right)$, $j := j + 1$;
  **end while**
  save the resulted 3-tuple $(R, j, i)$ in array (memory)
  $j := 0$;
  **next** $i$;
**end for**
find $(R, j, i)$ with $i \in C_R$, large $j$;
**return** $(r_1 := R$, $n := 2^j$, $h := i)$
```

So there are two outputs generated by the algorithm as given in Eq. (4.11) above. The first is $(\alpha^{2^{r_1}-1}) = (\alpha^{2^5-1})$ term. The second is $(\alpha^{2^h-1}) = (\alpha^{2^2-1})$ term. Given that $h = 2$ is an element in the chain $C_{r_1} = C_5$, then, the latter term is our savings because the term is available as intermediate result. As we can see from Eq. (4.11) above, the cost of both terms is 3 $GF(2^{163})$-multiplications.

In continuation with our example, the factors in $(m-1)$, other than r_1, are represented by the variable n that is given in the 3-tuple $(5, 32, 2)$, thus, $n = 32$ is the value of such factors. Let $\lambda = (\alpha^{2^{r_1}-1}) = (\alpha^{2^5-1}) = \alpha^{31}$, *the output of the* $WHCA$ *algorithm*, and $v = n = 32$. Assume that λ and v are the inputs to our proposed $WHFA(\lambda, v)$ algorithm (see Algorithm 5).

Algorithm 4 $WHCA$ Algorithm in $GF(2^m)$

```
Input: $e \in GF^*(2^m)$, $C_v$ and $A_v$ precomputed,
    $\kappa := 0$ if $(m-1)$ not-decomposed, otherwise $\kappa := h$
Output: $\delta_{c_l}^2 = e^{-1} \in GF(2^m), f = (\alpha^{2^\kappa-1})$
Given: $\delta_{c_i}(e) = e^{2^{c_i}-1}$, $\delta_{c_{i_1}+c_{i_2}}(e) = [\delta c_{i_1}(e)]^{2^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$
Initial: $l := length(C_v), \delta_{c_0} := e$;
for $i := 1$ to $l$ do
    $\delta_{c_i}(e) := [\delta c_{i_1}(e)]^{2^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$;
    if $\kappa = c_i$ then
        $f := \delta_{c_i}(e)$;
    end if
end for
if $\kappa \neq 0$ then
    return $\delta_{c_l}, f$
end if
return $\delta_{c_l}^2, f$
```

Algorithm 5 can be described as follows: Given that we have only one factor, namely $v = n$, then the outer **for** loop (depends on the number of factors in $(m-1)$) is executed only once. In addition, the loop counter i in the inner **for** loop (depends on the binary length of each factor) will loop from 4 to 0, which is basically 5 loops. This is because $v = 32 = (100000)_2$, thus, the maximum value of the loop counter is given by $i = (q-2) = (6-2) = 4$, whereby q is the binary length of v. Given that the **if** statement is not going to be satisfied, and variable $r = r \times r_{j-1} = r \times r_1 = 1 \times 5 = 5$, then the computational steps with $a = \lambda = (\alpha^{2^{31}})$ will proceed as follows:

$$
\begin{aligned}
(a) \times (a)^{2^{5\times 2^4}} &= a^{(2^{5\times 2^4}+1)} = b \\
(b) \times (b)^{2^{5\times 2^3}} &= b^{(2^{5\times 2^3}+1)} = c \\
(c) \times (c)^{2^{5\times 2^2}} &= c^{(2^{5\times 2^2}+1)} = d \\
(d) \times (d)^{2^{5\times 2^1}} &= d^{(2^{5\times 2^1}+1)} = e \\
(e) \times (e)^{2^{5\times 2^0}} &= e^{(2^{5\times 2^0}+1)} = f
\end{aligned}
\tag{4.12}
$$

Notice that $f = (\alpha^{2^5-1})^{(2^{5\times 2^4}+1)(2^{5\times 2^3}+1)(2^{5\times 2^2}+1)(2^{5\times 2^1}+1)(2^{5\times 2^0}+1)}$ in Eq. (4.12) above. In general, $f = (\alpha^{2^{r_1}-1})^e$ which is exactly the first term on the right-hand side of Eq. (4.7) above. Given the availability of (α^{2^5-1}) term as given in Eq. (4.11) above, the cost of computing f is 5 $GF(2^{163})$-multiplications, which is represented by the '+' signs in the expression.

To complete the process of finding the inverse in our example, we need to join the available terms, such as f and (α^{2^h-1}), and apply some powers of two after joining the terms. Such powers are assumed free using NB for the field elements. In other words, we need to continue the computation of the expression given in Eq. (4.6) above. In fact, this is the responsibility of our proposed main inversion algorithm, which is referred to as $Inverse(\alpha, m)$ (see Algorithm 6).

Algorithm 5 $WHFA$ Algorithm in $GF(2^m)$

```
Input: $\lambda \in GF(2^m), v = \prod_{j=2}^{k} r_j : r_j = (1m_{q_j-2}^{(j)} \cdots m_0^{(j)})_2$
Output: $\mu = \lambda^e = (\alpha^{2^{r_1}-1})^e$
Initial: $r := 1$;
for $j := 2$ to $k$ do
  $\mu := \lambda$;
  $r := r \times r_{j-1}$;
  for $i := q_j - 2$ to $0$ do
    $\mu := \mu \times \mu^{2^{r2^i}}$;
    if $m_i^{(j)} = 1$ then
      $\mu := \lambda \times \mu^{2^{r2^i}}$;
    end if
  end for
  $\lambda := \mu$;
end for
return $\mu$
```

Algorithm 6, the main field inversion algorithm, consists of all other previously mentioned helper algorithms such as $WHDA$, $WHCA$ and $WHFA$ as its main building blocks, whose are ready for call by the the main algorithm, when appropriate and depending on the current case under consideration.

In using our main field inversion algorithm with the element α and $m = 163$ as the inputs, i.e., $Inverse(\alpha, 163)$, we need one extra $GF(2^{163})$-multiplication to join f and (α^{2^h-1}) terms. Given that the cost of computing f term is 5 $GF(2^{163})$-multiplications as given in Eq. (4.12) above, and the cost of computing $(\alpha^{2^{r_1}-1})$ and (α^{2^h-1}) terms is 3 $GF(2^{163})$-multiplications as given in Eq. (4.11) above, then, finding the inverse of $\alpha \in GF^*(2^{163})$ using our main inversion algorithm costs in terms of the required $GF(2^{163})$-multiplications, or its inversion cost is given by 9 $GF(2^{163})$-multiplications.

Algorithm 6 *Inverse* Algorithm in $GF(2^m)$

```
Input: $\alpha \in GF^*(2^m)$, extension degree $m$
Output: $\delta = \alpha^{-1} \in GF(2^m)$
Initial: $(m-1)$;
**Case** when $(m-1)$ is not-decomposed:
  **if** $((m-1) = 2^k$ **or** $w(m-1) = 2)$ **then**
    **return** $\delta := WHCA(\alpha, C_{m-1}, A_{m-1}, 0)$
**Case** when $(m-1)$ is decomposed:
  **else**
    fetch $(r_1, n, h)$ from the array (memory);
    $[\eta, \rho] := WHCA(\alpha, C_{r_1}, A_{r_1}, h)$;
    $\gamma := WHFA(\eta, n)$;
    **if** $(h = 0)$ **then**
     **return** $\delta := \gamma$
    **else**
     **return** $\delta := ((\gamma)^{2^h} \times \rho)^2$
  **end if**
```

In general, the following block diagram shown in Figure 4.1 gives an idea about the working principle of the proposed main field inversion algorithm, namely $Inverse(\alpha, m)$ algorithm. The diagram shows the relation between the main inversion algorithm and other helper algorithms that described in our previous discussion. The diagram is somehow applicable to any field inversion algorithm which is based on our proposed decomposition algorithm with negligible differences, more specifically, such as the algorithm that we are going to propose in $GF(3^m)$ as in the following sections.

Notice that $WHDA(m)$ algorithm shown in Figure 4.1 above can mainly run off-line. It means that the algorithm add no extra runtime in the main inversion

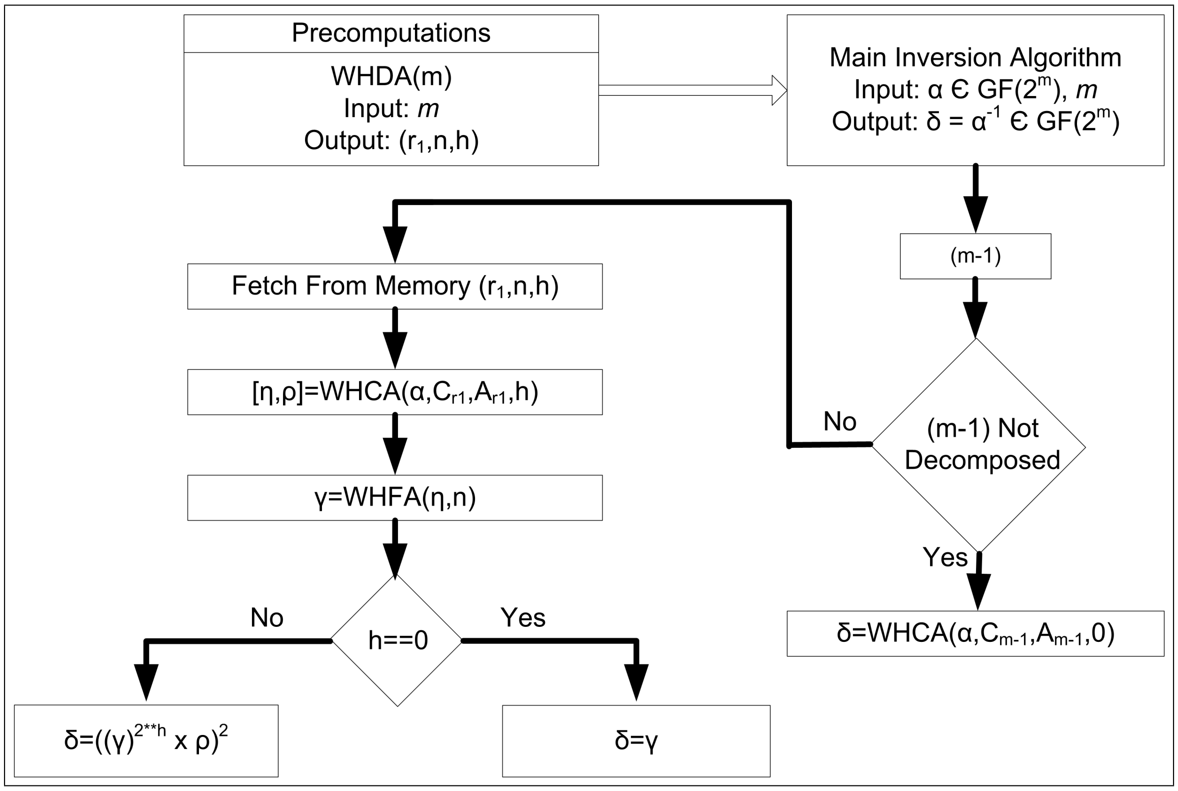


Figure 4.1: Main Inversion Algorithm Block Diagram

algorithm. In addition, given that the output of such algorithm in most cases is the 3-tuple (r_1, n, h), it means that the optimal decomposition criterion (OD), which is defined with other decomposition methods, is already applied in using our decomposition method. This even is applicable in the cases whereby the output of such algorithm is the prime factors of $(m-1)$, since the number of factors never exceeds three. Reminding the reader that the OD is associated with the lowest inversion cost, with the fewest factors and remainders that is suitable from hardware perspective.

4.2 Characteristic Three Algorithms

In this section, we introduce our proposed field inversion algorithms in $GF(3^m)$ using NB for the field elements, which are based on FLT inversion approach. Our first inversion algorithm is somehow based on ITI expression mentioned earlier herein. It is the first algorithm that is written in full for complete inversion in $GF(3^m)$. In addition, it is associate with lower inversion cost in comparison with other existing algorithms those exactly based on ITI expression. Our second inversion algorithm is basically based on the decomposition method we proposed earlier herein in $GF(2^m)$. In fact, it implements our proposed decomposition algorithm in $GF(3^m)$. The algorithm is expected to have the lowest inversion cost in such fields. Finally, we show the Fast Frobenius map operation in such fields which is extended to higher characteristic extension fields. Such operation is expected to have free runtime and space complexity.

4.2.1 ITI-Based Inversion Algorithm

Our first field inversion algorithm proposed in $GF(3^m)$ can be described as follows: Given the positive integer r as the quotient of dividing the order of the multiplicative group of the field $GF(p^m)$, namely $GF^*(p^m)$, on the order of its subfield's multiplicative group, namely $GF^*(p) = \{1, 2, \cdots, p-1\}$, then we have

$$r = \frac{p^m - 1}{p - 1} = p^{m-1} + p^{m-2} + \cdots + p^1 + 1, \tag{4.13}$$

thus

$$(r-1) = p^{m-1} + p^{m-2} + \cdots + p^2 + p^1, \tag{4.14}$$

which is the *p-adic* representation of $(r-1) = (11\dots110)_p$. Given that

$$p^m - 1 = (p-1) \times \left[p^{m-1} + p^{m-2} + \cdots + p^1 + p^0\right], \tag{4.15}$$

then it can be expressed as

$$p^m - 1 = (p-1) \times \left[p^{m-1} + p^{m-2} + \cdots + p^1\right] + (p-1). \tag{4.16}$$

For characteristic $p = 3$, we have

$$3^m - 1 = 2 \times \left[3^{m-1} + 3^{m-2} + \cdots + 3^1\right] + 2, \tag{4.17}$$

thus we have

$$3^m - 2 = 2 \times \left[3^{m-1} + 3^{m-2} + \cdots + 3^1\right] + 1, \tag{4.18}$$

which has the 3-*adic* representation of $(r-1)$, i.e., $(11\ldots110)_3$, as the main computational component.

Therefore, the inverse of $\alpha \in GF^*(3^m)$ using our first inversion algorithm proposed in such fields is given by the following expression (referred to as, IVI expression):

$$\alpha^{3^m-2} = \left(\alpha^2\right)^{3^{m-1}+\cdots+3^2+3^1} \times \alpha = \left(\alpha^2\right)^{\psi} \times \alpha, \tag{4.19}$$

whereby ψ is the 3-*adic* representation of $(r-1)$. Our first field inversion algorithm proposed in $GF(3^m)$, namely $WHTI(\alpha, m)$ algorithm, is basically a translation for the expression given in Eq. (4.19) above (see Algorithm 7).

Unlike other existing field inversion algorithms those exactly based on ITI expression. Such algorithms require an extra one extension field multiplication in addition to the subfield inversion for computing the inverse in the concerned extension field (see Chapter 3.2.1). Because subfield inversion is not required in our algorithm, the algorithm is modular thus suitable for VLSI implementation. This is mainly because it requires only two field operations, namely the multiplication operation and the cyclic-shifts for the coefficients, for computing the inverse in the concerned extension field as seen from the algorithm. The inversion cost of Algorithm 7 proposed for use in $GF(3^m)$, in terms of the required number of ternary extension field multiplications, is given by

$$[\ell(m-1) + w(m-1) - 1] \;\; GF(3^m)\text{-Multiplications},$$

Algorithm 7 WHTI Algorithm in $GF(3^m)$

```
Input: $\alpha \in GF^*(3^m)$, extension degree $m$
Output: $\gamma = \alpha^{-1} \in GF(3^m)$
Initial: $(m-1)$ as $(1m_{q-2}\cdots m_1m_0)_2$, $\gamma := \beta := \alpha^2$;
for $i := q-2$ to 0 do
    $\gamma := \gamma \times \gamma^{3^{2^i}}$;
    if $m_i = 1$ then
        $\gamma := \beta \times \gamma^{3^{2^i}}$;
    end if
end for
$\gamma := \gamma^3 \times \alpha$;
return $\gamma$
```

$$[\ell(m-1) + w(m-1) - 1] \quad GF(3^m)\text{-Frobenius Maps.} \tag{4.20}$$

From Eq. (4.20) above, in general, a Frobenius map represents a cubing operation in $GF(3^m)$, and in particular, such an operation is simply represented by a right cyclic-shift in such fields using NB for the field elements. For easy comparison with the result we have obtained earlier which is relevant to other algorithms that are based on ITI expression [see Chapter 3, Eq. (3.24)], $GF(3^m)$ given in Eq. (4.20) above can be changed to $GF(p^m)$ to indicate any odd-characteristic extension field, including $GF(3^m)$ by itself.

As we can see from Eq. (4.20) above, the inversion cost of Algorithm 7 mainly depends on two parameters. The first, depends on the binary length of $(m-1)$ value, and is represented by the unconditional statement within the main loop in the algorithm. The second, depends on the Hamming weight of the binary $(m-1)$ value, and is represented by the conditional statement within the main loop in the algorithm.

To show how Algorithm 7 works through an example, let us assume that the element in which its inverse is required is $\alpha \in GF(3^6)$. Inverse computation using the algorithm can be described as follows: Firstly, $m = 6$ and $(m-1) = 5 = (101)_2$, thus $q = 3$ and the loop counter i will loop from 1 down to 0. Secondly, given that $\gamma = \beta = \alpha^2$, then the computation proceed as follow:

For $i = 1$:

$$\gamma = \gamma \times \gamma^{3^{2^1}} = (\alpha^2) \times (\alpha^2)^{3^{2^1}} = \alpha^{20}$$

For $i = 0$:

$$\gamma = \gamma \times \gamma^{3^{2^0}} = (\alpha^{20}) \times (\alpha^{20})^{3^{2^0}} = \alpha^{80}$$

$$\gamma = \beta \times \gamma^{3^{2^0}} = (\alpha^2) \times (\alpha^{80})^{3^{2^0}} = \alpha^{242}$$

After **end for**:

$$\gamma = \alpha \times \gamma^3 = (\alpha) \times (\alpha^{242})^3 = \alpha^{727}. \tag{4.21}$$

Given that $\alpha^{-1} = \alpha^{3^6-2} = \alpha^{727}$ as given in Eq. (4.21) above, then the inversion cost of Algorithm 7, as evident from the above computational steps, is given by 4 $GF(3^6)$-multiplications in this specific example. This result is conforming to the one obtained using Eq. (4.20) above, whereby $[\ell(m-1)+w(m-1)-1] = [\ell(5)+w(5)-1] =$ 4 multiplications, given that the $GF(3^6)$-Frobenius maps are free operations in using NB for the field elements.

4.2.2 WHDA-Based Inversion Algorithm

The second field inversion algorithm we propose in $GF(3^m)$ is based on the decomposition algorithm that we have proposed earlier herein (see Chapter 4.1). The algorithm is based on FLT inversion approach using NB for the field elements, and can be described as follows: Given that $(m-1) = \prod_{j=1}^{k} r_j + h$ and the following expression

$$3^m - 2 = 3 \times \left(3^{m-1} - 1\right) + 1, \tag{4.22}$$

if we substitute the value of $(m-1)$ in Eq. (4.22) above, then we have

$$3^m - 2 = 3 \times \left(3^{\prod_{j=1}^{k} r_j + h} - 1\right) + 1. \tag{4.23}$$

Expanding the new exponent in Eq. (4.23) above, gives us

$$3^m - 2 = 3 \times \left[3^h \times (3^{r_1 \times \cdots \times r_k} - 1) + (3^h - 1)\right] + 1. \tag{4.24}$$

By doing further expansion and finally collecting terms in Eq. (4.24) above, then we have

$$3^m - 2 = 3 \times \left[e \times 3^h \times (3^{r_1} - 1) + (3^h - 1)\right] + 1, \tag{4.25}$$

whereby the expression for variable e in Eq. (4.25) above is given in what follows.

Therefore, based on Eq. (4.25) above, the inverse of a nonzero element $\alpha \in GF(3^m)$ using our proposed IVD expression is given by

$$\alpha^{-1} = \alpha^{3^m - 2} = \left(\alpha^{3^{m-1}-1}\right)^3 \times \alpha =$$
$$\left(\alpha^{3^{\prod_{j=1}^{k} r_j + h} - 1}\right)^3 \times \alpha = \left((\alpha^{3^{r_1 \times \ldots \times r_k} - 1})^{3^h} \times (\alpha^{3^h - 1})\right)^3 \times \alpha$$
$$= \left(((\alpha^{3^{r_1}-1})^e)^{3^h} \times (\alpha^{3^h-1})\right)^3 \times \alpha = \left((\alpha^{3^{r_1}-1})^e\right)^{3^{h+1}} \times \left(\overbrace{(\alpha^{3^h-1})^3 \times \alpha}^{f-\text{ term}}\right),$$

given that

$$e = \left((3^{r_1})^{r_2 - 1} + \cdots + 1) \cdots ((3^{r_1 \times \cdots \times r_{k-1}})^{r_k - 1} + \cdots + 1)\right). \tag{4.26}$$

Theorem 6. *Given that $(m-1) = \prod_{j=1}^{k} r_j + h$. Assuming that h is appropriately selected to belong to the short addition chain of r_1, denoted as C_{r_1}, or any factor in $(m-1)$, then, the computation of $(\alpha^{3^{r_1}-1})$ term guarantees the availability of (α^{3^h-1}) term as an intermediate value, and the inversion cost is given by*

$$\left(\sum_{j=1}^{k}[\ell(r_j) + w(r_j) - 2] + 1\right) \; GF(3^m)\text{-Multiplications.} \tag{4.27}$$

Proof. The proof here follows the one previously presented for the case of $GF(2^m)$ (see Chapter 4.1)

□

From Eq. (4.27) above, it is clearly evident that the inversion cost of our second proposed field inversion algorithm in $GF(3^m)$ only depend on the constant 1 (*independent of the remainder h*). Thus, we have obtained a similar result to that in $GF(2^m)$ as in Eq. (4.5) above. Notice that, $WHDA(m)$ algorithm [see Chapter 4.1, Algorithm 3], is the decomposition algorithm that we are going to use to decompose the extension degree m of the concerned $GF(3^m)$.

Furthermore, the above inversion cost is applicable to a wide range of ternary extension fields, especially after decomposing each extension degree m of the concerned $GF(3^m)$ using the decomposition method employed in our proposed decomposition algorithm. Notice that, decomposition algorithms in $GF(3^m)$ are rare, if not exist at all in the academic literature. In addition, such algorithms are mainly depend on the m, but not on the characteristic p of the concerned extension field. However, keep in mind that even when applying the same decomposition method in inversion algorithms for different extension fields, the inversion cost could be different.

In the following, we introduce our proposed algorithms necessary for inversion in $GF(3^m)$ through a running example. We are going to consider the field $GF(3^{167})$

in our example. The selection criterion for this field is that the field is of particular interest for ECC. Assuming that $m = 167$ is the input to our proposed decomposition algorithm, which is referred to as $WHDA(m)$ [see Chapter 4, Algorithm 3], then the output is the 3-tuple $(10, 16, 6)$. The 3-tuple means that $(m-1) = (167-1) = 166 = 10 \times 16 + 6$.

Given that the short addition chain for factor $r_1 = 10$ is given by $C_{r_1} = C_{10} = \{1, 2, 4, 6, 10\}$, then the sequence of integer pairs of C_{10} is given by $A_{r_1} = A_{10} = \{(1, 1), (2, 2), (4, 2), (6, 4)\}$. The sequence follows the rule $c_i = c_{i-1}+c_{i-1}$ for $i \in \{1, 2\}$, and $c_i = c_{i-1} + c_{i-2}$ for $i \in \{3, 4\}$, given that $c_0 = 1$ and $c_4 = 10$.

In continuation with our example, we assume that the field element in which its inverse is required is $\alpha \in GF^*(3^{167})$. By using the 3-tuple $(10, 16, 6)$, the above sequences we just obtained for factor $r_1 = 10$ (i.e., C_{10}, A_{10}) and the element α as the inputs to our proposed $WHTCA(e, C_v, A_v, \kappa)$ algorithm (see Algorithm 8), somehow follows our previous algorithm [see Chapter 4, Algorithm 4], except that all power 2s are replaced by power 3s and incorporation of f-term as given in Eq. (4.26) above. For example, if $e = \alpha$, $C_v = \{1, 2, 4, 6, 10\}$, $A_v = \{(1, 1), (2, 2), (4, 2), (6, 4)\}$ and $\kappa = 6$, then, the computational steps using the algorithm proceed as follows:

$$\begin{aligned}
(\alpha^{3^1-1})^{3^1} \times (\alpha^{3^1-1}) &= (\alpha^{8}) = (\alpha^{3^2-1}) \\
(\alpha^{3^2-1})^{3^2} \times (\alpha^{3^2-1}) &= (\alpha^{80}) = (\alpha^{3^4-1}) \\
(\alpha^{3^4-1})^{3^2} \times (\alpha^{3^2-1}) &= (\alpha^{728}) = (\alpha^{3^6-1}) \\
(\alpha^{3^6-1})^{3^4} \times (\alpha^{3^4-1}) &= (\alpha^{59048}) = (\alpha^{3^{10}-1})
\end{aligned} \tag{4.28}$$

So there are two outputs generated by the algorithm as given in Eq. (4.28) above. The first is $(\alpha^{3^{r_1}-1}) = (\alpha^{2^{10}-1})$ term. The second is $f = (\alpha^{3^h-1})^3 \times \alpha = (\alpha^{3^6-1})^3 \times \alpha$ term. Given that $h = 6$ is an element in the chain $C_{r_1} = C_{10}$, and the computation of f-term is concurrent with the computation of $(\alpha^{3^{r_1}-1})$ term, thus, as we can see from Eq. (4.28) above, the cost of both terms is 4 $GF(2^{167})$-multiplications.

Algorithm 8 $WHTCA$ Algorithm in $GF(3^m)$

```
Input: $e \in GF^*(3^m)$, $C_v$ *and* $A_v$ precomputed,
  $\kappa := 0$ if $(m-1)$ not-decomposed, otherwise $\kappa := h$
Output: $\delta^3_{c_l} = e^{-1} \in GF(3^m), f = (\alpha^{3^\kappa-1})$
Given: $\delta_{c_i}(e) = e^{3^{c_i}-1}$, $\delta_{c_{i_1}+c_{i_2}}(e) = [\delta c_{i_1}(e)]^{3^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$
Initial: $l := length(C_v), \delta_{c_0} := e$;
**for** $i := 1$ to $l$ **do**
  $\delta_{c_i}(e) := [\delta c_{i_1}(e)]^{3^{c_{i_2}}} \times \delta_{c_{i_2}}(e)$;
  **if** $\kappa = c_i$ **then**
    $f := [(\delta_{c_i}(e))^3 \times \alpha]$; $\Longleftarrow$ ($f$-term)
  **end if**
**end for**
**if** $\kappa \neq 0$ **then**
  **return** $\delta_{c_l}, f$
**end if**
**return** $\delta^3_{c_l}, f$
```

In continuation with our example, the factors in $(m-1)$, other than r_1, are represented by the variable n that is given in the 3-tuple $(10, 16, 6)$, thus, $n = 16$ is the value of such factors. Let $\lambda = (\alpha^{3^{r_1}-1}) = (\alpha^{3^{10}-1}) = \alpha^{59048}$, *the output of the $WHTCA$ algorithm*, and $v = n = 16$. Assume that λ and v are the inputs to our proposed $WHTFA(\lambda, v)$ algorithm (see Algorithm 9), somehow follows our previous algorithm [see Chapter 4, Algorithm 5], except that all power 2s are replaced by power 3s.

Algorithm 9 can be described as follows: Given that we have only one factor, namely $v = n$, then the outer **for** loop (depends on the number of factors in $(m-1)$) is executed only once. In addition, the loop counter i in the inner **for** loop (depends on the binary length of each factor) will loop from 3 to 0, which is basically 4 loops. This is because $v = 16 = (10000)_2$, thus, the maximum value of the loop counter is given by $i = (q-2) = (6-2) = 3$, whereby q is the binary length of v. Given that the **if** statement is not going to be satisfied, and variable $r = r \times r_{j-1} = r \times r_1 = 1 \times 10 = 10$, then the computational steps with $a = \lambda = (\alpha^{3^{59048}})$ will proceed as follows:

$$\begin{aligned}
(a) \times (a)^{3^{10\times 3^3}} &= a^{(3^{10\times 3^3}+1)} = b \\
(b) \times (b)^{3^{10\times 3^2}} &= b^{(3^{10\times 3^2}+1)} = c \\
(c) \times (c)^{3^{10\times 3^1}} &= c^{(3^{10\times 3^1}+1)} = d \\
(d) \times (d)^{3^{10\times 3^0}} &= d^{(3^{10\times 3^0}+1)} = e
\end{aligned} \tag{4.29}$$

Notice that $e = (\alpha^{3^{10}-1})^{(3^{10\times 3^3}+1)(3^{10\times 3^2}+1)(3^{10\times 3^1}+1)(3^{10\times 3^0}+1)}$ in Eq. (4.29) above. In general, $e = (\alpha^{3^{r_1}-1})^e$ which is exactly the first term on the right-hand side of Eq. (4.26) above. Given the availability of $(\alpha^{3^{10}-1})$ term as given in Eq. (4.28) above, the cost of computing e is 4 $GF(3^{167})$-multiplications, which is represented by the '+' signs in the expression.

Algorithm 9 $WHTFA$ Algorithm in $GF(3^m)$

```
Input: $\lambda \in GF(3^m), v = \prod_{j=2}^{k} r_j : r_j = (1m_{q_j-2}^{(j)} \cdots m_0^{(j)})_2$
Output: $\mu = \lambda^e = (\alpha^{3^{r_1}-1})^e$
Initial: $r := 1$;
for $j := 2$ to $k$ do
  $\mu := \lambda$;
  $r := r \times r_{j-1}$;
  for $i := q_j - 2$ to $0$ do
    $\mu := \mu \times \mu^{3^{r3^i}}$;
    if $m_i^{(j)} = 1$ then
      $\mu := \lambda \times \mu^{3^{r3^i}}$;
    end if
  end for
  $\lambda := \mu$;
end for
return $\mu$
```

To complete the process of finding the inverse in our example, we need to join the available terms, such as e and the f-term, and apply some powers of three after joining the terms. Such powers are assumed free using NB for the field elements. In other words, we need to continue the computation of the expression given in Eq. (4.26) above. In fact, this is the responsibility of our proposed main inversion algorithm, which is referred to as $ITnverse(\alpha, m)$ (see Algorithm 10).

Algorithm 10, the main field inversion algorithm, consists of all other previously mentioned helper algorithms such as $WHDA$, $WHTCA$ and $WHTFA$ as its main building blocks, whose are ready for call by the the main algorithm, when appropriate and depending on the current case under consideration.

Algorithm 10 *ITnverse* Algorithm in $GF(3^m)$

```
Input: $\alpha \in GF^*(3^m)$, extension degree $m$
Output: $\delta = \alpha^{-1} \in GF(3^m)$
Initial: $(m-1)$;
Case when $(m-1)$ is not-decomposed:
    if $((m-1) = 2^k$ or $w(m-1) = 2)$ then
        return $\delta := WHTCA(\alpha, C_{m-1}, A_{m-1}, 0)$
Case when $(m-1)$ is decomposed:
    else
        fetch $(r_1, n, h)$ from the array (memory);
        $[\eta, \rho] := WHTCA(\alpha, C_{r_1}, A_{r_1}, h)$;
        $\gamma := WHTFA(\eta, n)$;
        if $(h = 0)$ then
            return $\delta := (\gamma \times \alpha)$
        else
            return $\delta := \Big((\gamma)^{3^{h+1}} \times \rho\Big)$
    end if
```

In using our main field inversion algorithm with the element α and $m = 167$ as the inputs, i.e., $ITnverse(\alpha, 167)$, we need one extra $GF(3^{167})$-multiplication to join e and f-term. Given that the cost of computing e term is 4 $GF(3^{167})$-multiplications as given in Eq. (4.29) above, and the cost of computing $(\alpha^{3^{r_1}-1})$ and f-term is 4 $GF(3^{167})$-multiplications as given in Eq. (4.28) above, then, finding the inverse of $\alpha \in GF^*(3^{167})$ using our main inversion algorithm costs in terms of the required $GF(3^{167})$-multiplications, or its inversion cost is given by 9 $GF(3^{167})$-multiplications.

In general, the working principle of Algorithm 10 somehow follows the block diagram shown in Figure 4.1 above. The diagram gives an idea about the relation between the main inversion algorithm and other helper algorithms that described in our previous discussion.

4.2.3 Accelerating Frobenius Map Operation

In using Type-II optimal extension fields (OEFs) (see Chapter 2.1.4), it is possible to accelerate the *j-th* iterate of the Frobenius map, or the 3^j-*th* power computation in $GF(3^m)$ using PB for the field elements. Unless it is clearly mentioned that the field characteristic $p = 3$, the following presentation assumes the general extension field $GF(p^m)$, namely the odd-characteristic extension field with odd-prime p and $m > 1$.

Given a nonzero element $\alpha \in GF(p^m)$ represented as $\alpha = \sum_{i=0}^{m-1} a_i x^i$ using PB, its *j-th* iterate of the Frobenius map is given by

$$\alpha^{p^j} = \sum_{i=0}^{m-1} a_i x^{ip^j} = a_0 + a_1 x^{p^j} + \cdots + a_{m-1} x^{(m-1)p^j} \mod f(x). \tag{4.30}$$

Given that in using Type-II OEFs, the field polynomial generating the field $GF(p^m)$ is the irreducible binomial $f(x) = x^m - w$ of degree m, and given that the basis element x is a root of $f(x)$, then we have

$$w \equiv x^m \pmod{f(x)}, \tag{4.31}$$

or equivalently $w = x^m$. As given in [4], if m is square-free value, then we have

$$p \equiv 1 \pmod{m}. \tag{4.32}$$

Based on Eq. (4.32) above, we can express the terms affected by the action of the Frobenius map as follows

$$x^{ip^j} = x^{\lfloor \frac{ip^j}{m} \rfloor m} x^{ip^j \pmod{m}} = x^{\lfloor \frac{ip^j}{m} \rfloor m} x^i = w^{q_i^j} x^i, \tag{4.33}$$

for $q_i^j = \lfloor \frac{ip^j}{m} \rfloor$, any *j-th* iterate of Frobenius map and $(1 \leq i \leq m-1)$. This is because $x^m = w$ based on Eq. (4.31) above, and $ip^j \pmod{m} = i$ based on Eq. (4.32) above.

Therefore, based on Eq. (4.33) above, the *j-th* iterate of the Frobenius map is given by

$$\alpha^{p^j} = \sum_{i=0}^{m-1}(a_i w^{q_i^j})x^i \quad \forall \ (0 \le i \le m-1). \tag{4.34}$$

Given that $w = 2$ in using Type-II OEFs, with the fact that the exponent (q_i^j) in Eq. (4.34) above is simply a positive integer, then we have

$$\alpha^{p^j} = \sum_{i=0}^{m-1}(a_i 2^k)x^i \quad \forall \ (0 \le i \le m-1), k = q_i^j, w = 2. \tag{4.35}$$

The $(m-1)$ subfield multiplications of $(a_i 2^k)$ terms for $(1 \le i \le m-1)$ given in Eq. (4.35) above can be reduced to arbitrary shifts. This is the result of multiplying the subfield coefficients a_i for $(1 \le i \le m-1)$ by an arbitrary power of 2 value, i.e., 2^k for a particular positive integer k. However, such shifts still require subfield modular reductions as given in [4, 20, 49].

To further improve the calculation of $(a_i 2^k \mod p)$ expression for $(1 \le i \le m-1)$, i.e., the subfield multiplication and modular reduction operations in $GF(3^m)$, whereby the characteristic $p = 3$, we have noticed the following. Given that in $GF(3^m)$ the elements have coefficients $a_i \in GF(3) = \{0, 1, 2\}$ for all $(0 \le i \le m-1)$, with the fact that k is an arbitrary positive integer, then the interesting result in such fields can be described as follows:

Given $a_i = 0$ for $(0 \le i \le m-1)$, *the coefficients whose value is equal to zero*, the result of calculating $(a_i 2^k \mod p)$ expression is always equal to zero, regardless of the value of the positive integer k, and even the characteristic p.

Given $a_i = 1$ for $(0 \le i \le m-1)$, *the coefficients whose value is equal to one*, the result of calculating $(a_i 2^k \mod p)$ expression is given as follows: For $k = 1, 2, 3, \cdots$, we have

$$(a_i 2^k \mod p) = p-1, p-2, p-1, p-2, \cdots = 2, 1, 2, 1, \cdots \ : p = 3. \tag{4.36}$$

Furthermore, given $a_i = 2$ for $(0 \leq i \leq m-1)$, *the coefficients whose value is equal to two*, the result of calculating $(a_i 2^k \mod p)$ expression is given as follows: For $k = 1, 2, 3, \cdots$, we have

$$(a_i 2^k \mod p) = p-2, p-1, p-2, p-1, \cdots = 1, 2, 1, 2, \cdots \ : p = 3. \tag{4.37}$$

Therefore, based on Eqs. (4.36) and (4.37) given above, performing subfield multiplication and modular reduction is a well-defined sequence of repeated values. It is simply a cyclic-toggling operation for the values of a_i coefficients for $(1 \leq i \leq m-1)$, to a new values again taken from $GF(p)$, the subfield to which a_i coefficients belong. Each specific new value in the sequence depend on the exponent k, more specifically, if its value is odd or even.

Table 4.1 presents the cyclic-toggling sequences obtained for $(a_i 2^k \mod 3)$ whereby the coefficients $a_i \in GF(3) = \{0, 1, 2\}$, the subfield of $GF(3^m)$, for all $(0 \leq i \leq m-1)$ and a positive integer $k = 0, 1, 2, \cdots$.

Table 4.1: Cyclic-Toggling for $(a_i 2^k \mod 3)$

	$\mathbf{a_i \in GF(3)}$		
$\mathbf{2^k}$	**0**	**1**	**2**
2^1	0	2	1
2^2	0	1	2
2^3	0	2	1
2^4	0	1	2
2^5	0	2	1
.	.	.	.

By referring to Table 4.1 above, if the positive integer k has any odd value and the *i-th* coefficient $a_i = 1$ for $(1 \leq i \leq m-1)$, then $(a_i 2^k \mod 3) = 2$ always. If the positive integer k has any odd value and the *i-th* coefficient $a_i = 2$ for $(0 \leq i \leq m-1)$, then $(a_i 2^k \mod 3) = 1$ always. In both cases, if the positive integer k has any even value, the coefficients a_i for $(1 \leq i \leq m-1)$ are kept unchanged. Thus, by just knowing if the positive integer k is either odd or even, we know ahead whether toggling the concerned coefficient is necessary or not, which instantaneously produces the result of the subfield multiplication and modular reduction operations.

Given that the positive integer $k = q_i^j = \lfloor \frac{ip^j}{m} \rfloor$ is data-independent, which means it does not depend on the field element under consideration, thus its value can be calculated off-line. Since each term affected by the action of the Frobenius map has its own k value, rather than storing k itself, a hash value can be stored to represent the value type of k, for example, either 0 to indicate that k is even value or 1 to indicate that k is odd value. Thus, by a priori knowing whether to toggle the concerned coefficient or not, the terms affected by the action of the Frobenius map are calculated very fast.

Based on our previous discussion above, the *j-th* iterate of the Frobenius map is almost a free operation, i.e., it is associated with almost free runtime and space complexity. Given that such operation can substitute the *j-th* right cyclic-shifts required for inversion in $GF(3^m)$ using NB for the field elements, or equivalently the 3^j*-th* powers, such operation can accelerates field inversion in such fields using PB for the field elements. Notice that, such operation can be easily extended to higher characteristic extension fields, which means, it can be extended to any Type-II OEF regardless of its characteristic p.

For example, consider the extension field $GF(5^m)$ with characteristic $p = 5$, whereby the elements in such field have coefficients $a_i \in GF(5) = \{0, 1, 2, 3, 4\}$ for all $(0 \leq i \leq m-1)$. We are going to formulate the cyclic-toggling sequence for the coefficients a_i having the value 1 for $(1 \leq i \leq m-1)$. Then use it, the sequence, to derive other cyclic-toggling sequences for the other coefficient values in $GF(5)$, i.e., for 2, 3 and 4, respectively. Finally, we will present the sequences we have obtained for $(a_i 2^k \mod 5)$ in a tabular form.

To calculate subfield multiplication and modular reduction operations fast in characteristic-five extension fields $GF(5^m)$, i.e., calculating $(a_i 2^k \mod 5)$ expression fast based on our previous discussion, the calculation process can be described as follows:

Given $a_i = 0$ for $(0 \leq i \leq m-1)$, *the coefficients whose value is equal to zero*, the result of calculating $(a_i 2^k \mod p)$ expression is always equal to zero, regardless of the value of the positive integer k, and even the characteristic p.

Given $a_i = 1$ for $(0 \leq i \leq m-1)$, *the coefficients whose value is equal to one*, the result of calculating $(a_i 2^k \mod p)$ expression is given as follows: For $k = 1, 2, 3, \cdots$, we have

$$\begin{aligned}(a_i 2^k \mod p) &= (p-3), (p-1), (p-2), (p-4), (p-3), (p-1), (p-2), (p-4), \cdots \\ &= 2, 4, 3, 1, 2, 4, 3, 1, \cdots \quad : p = 5. \qquad (4.38)\end{aligned}$$

Notice how the first element in the sequence given in Eq. (4.38) above has its value equal to 2. This is because its order is directly after the element which has the value 1 in the above sequence, whereby 1 is representing the coefficients value that is currently under consideration, i.e., $a_i = 1$.

Using the sequence we just obtained in Eq. (4.38) above, we can figure out the sequence for $a_i = 2$ given that $(0 \leq i \leq m-1)$, i.e., *the cyclic-toggling sequence for coefficients whose value is equal to two.* By considering the previous sequence, we start our new sequence for $a_i = 2$ with the first element having the value 4 and continue with the following elements in sequence. This is because its order is directly after the element which has the value 2 in the above sequence, whereby 2 is representing the coefficients value that is currently under consideration. Thus, our new sequence for $a_i = 2$ is given by

$$(a_i 2^k \mod 5) = 4, 3, 1, 2, 4, 3, 1, 2, \cdots \ : a_i = 2. \qquad (4.39)$$

By continuing in this manner, we can figure out the cyclic-toggling sequences for the remaining coefficient values, i.e., $a_i = \{3, 4\} \in GF(5)$ for $(0 \leq i \leq m-1)$. Table 4.2 presents the cyclic-toggling sequences obtained for $(a_i 2^k \mod 5)$ whereby the coefficients $a_i \in GF(5) = \{0, 1, 2, 3, 4\}$, the subfield of $GF(5^m)$, for all $(0 \leq i \leq m-1)$ and a positive integer $k = 0, 1, 2, \cdots$.

It can be seen from Table 4.2 above that any cyclic-toggling sequence for coefficients $a_i \in GF(p)$ for $(0 \leq i \leq m-1)$ can be obtained easily using the following expression

$$a_i \times 2^k \ (\text{mod} p) \quad : k = q_i^j = \lfloor \frac{ip^j}{m} \rfloor \ (\text{mod} p - 1). \tag{4.40}$$

Given that the expression in Eq. (4.40) above is independent of the field element in which its *j-th* iterate of Frobenius map is required in a given extension field $GF(p^m)$, such expression can be calculated off-line to accelerate such a map in the concerned extension field in using PB for the field elements.

Table 4.2: Cyclic-Toggling for $(a_i 2^k \mod 5)$

	$\mathbf{a_i \in GF(5)}$				
$\mathbf{2^k}$	**0**	**1**	**2**	**3**	**4**
2^1	0	2	4	1	3
2^2	0	4	3	2	1
2^3	0	3	1	4	2
2^4	0	1	2	3	4
2^5	0	2	4	1	3
2^6	0	4	3	2	1
2^7	0	3	1	4	2
2^8	0	1	2	3	4
.	.	.	.	.	.

The following example gives an idea about how the cyclic=toggling sequences can be use to accelerate the *j-th* iterate of Frobenius map in the concerned extension field. Assume $\alpha \in GF(5^2)$ and we want to calculate the 2-*th* iterate of Frobenius map in such field, or equivalently α^{5^2}. Given that the irreducible binomial is $f(x) = x^2 - 2$, $\alpha = x + 3 = [a_1\ a_0]_5 = [1\ 3]_5$, $k = 0$ for a_0 (or $k_{a_0} = 0$) and $k = 2$ for a_1 (or $k_{a_1} = 2$) by using Eq. (4.40) above, then the calculation steps can be described as follows:

Given that $\alpha = x+3 = [a_1\ a_0]_5 = [1\ 3]_5$ that is associate with $k = [k_{a_1}\ k_{a_0}] = [2\ 0]$, by referring to Table 4.2 above, when $k = 2$ and $a_1 = 1$, then we have the new value $a_1 = 4$. Again, when $k = 0$ and $a_0 = 3$ then we have the value of a_0 unchanged. Thus $\alpha^5 = 4x+3 = [a_1\ a_0]_5 = [4\ 3]_5$. Repeat previous steps with $k = [k_{a_1}\ k_{a_0}] = [2\ 0]$ again, then we have $\alpha^{5^2} = x+3 = [a_1\ a_0]_5 = [1\ 3]_5$. Thus, the 2-*th* iterate of Frobenius map $\alpha^{5^2} = \alpha$ using the cyclic-toggling concept that provided by means of previous steps.

To confirm the previous result we just obtained, we calculate α^{5^2} using the regular calculation method in $GF(5^2)$ given that $\alpha = x + 3$ as in the following:

$$\alpha^{5^2} = (x+3)^{5^2} = x^{5^2} + 3^{5^2}. \tag{4.41}$$

The result in Eq. (4.41) above follows from the fact that $(\gamma + \delta)^{p^i} = \gamma^{p^i} + \delta^{p^i}$ for any positive integer i. Given that $f(x) = x^2 - 2$, thus $x^2 = 2$. Therefore, we have

$$x^{5^2} = x^{25} = x^{24} \times x = (2)^{12} \times x = x. \tag{4.42}$$

The result in Eq. (4.42) above follows from the fact that $[(2)^{12} \pmod 5 = 1]$. In addition, we have

$$3^{5^2} = 3^{25 \pmod{2}} = 3^1 = 3. \tag{4.43}$$

Therefore, based on Eqs. (4.42) and (4.43) given above, we have $\alpha^{5^2} = x + 3 = \alpha$. This result conforms with the previously obtained result which is based on the cyclic-toggling concept. This means that such a concept can be used to further accelerate the Frobenius map operation in such extension fields.

As noted from the previous tables above, finding the cyclic-toggling sequence for any coefficient value in the concerned subfield, is a very straightforward task that is easy and quick. The cyclic-toggling sequences for the respective coefficients represent the calculation of multiplication and modular reduction operations in the concerned subfield $GF(p)$, in particular, in using Type-II OEFs.

In comparison with other existing methods for accelerating the Frobenius map operation [4, 20, 47], which require $(m-1)$ subfield multiplications that still require subfield modular reductions, our fast Frobenius map operation is associated with almost free runtime and space complexity. This in fact is achieved through the use of the cyclic-toggling sequences concept.

Chapter 5

Results and Analysis

This chapter is mainly dedicated to present the achieved results in tabular form. The results relevant to our proposed inversion algorithms has been obtained either through the direct application of inversion cost expressions, or using the output of a programmed algorithm. The first case is applicable to the proposed algorithms those based on ITI expression, including $WHTI(\alpha, m)$ algorithm [see Chapter 4.2, Algorithm 7]. The latter case is applicable to the proposed algorithms those based on our decomposition algorithm, namely $WHDA(m)$ algorithm [see Chapter 4.1, Algorithm 3]. The algorithm is programmed using C++ programming language to produce the decompositions for m of the concerned $GF(p^m)$, given that $p \in \{2, 3\}$.

In both cases, the results are simply the inversion cost values, expressed in terms of the required number of extension field multiplications necessary for inversion, using our proposed field inversion algorithms for binary and ternary extension fields. Such results are used for comparison purposes against other existing inversion algorithms available in the literature. The results will also be analyzed to give the reader a better idea about the competitive advantage of our proposed inversion algorithms.

In the following section, we first show the competitive advantage of our proposed inversion algorithm in $GF(2^m)$, in terms of the inversion cost, over other existing inversion algorithms in such fields. Then, we show the competitive advantage of the first inversion algorithm we have proposed in $GF(3^m)$ over other existing inversion algorithms in such fields. Furthermore, we show the competitive advantage of the second inversion algorithm we have proposed in $GF(3^m)$ over our first inversion algorithm we have proposed in such fields.

In the rest of this chapter, we are going to provide a summary on all chapter findings and highlight the main points relevant to the obtained results.

5.1 Comparison Tables

This section is mainly dedicated to present, in the form of tables, the obtained decompositions and inversion costs of various existing inversion algorithms available in the literature, including our proposed algorithms for comparison purposes. We recall that all field inversion algorithms, including our algorithms in binary $GF(2^m)$ and ternary $GF(3^m)$ extension fields, respectively, whether employing our proposed decomposition algorithm or not, their inversion cost is measured by the required number of extension field multiplications necessary for inversion. This is why the achieved results are listed in the form of tables, whereby each column represents a different inversion algorithm, while each row represents a different extension degree m of the concerned extension field.

For decomposition based inversion algorithms there are two types of entries within the tables. The first is the optimal decomposition (OD), which reflects the way in which each extension degree m is decomposed using the decomposition method employed in the concerned inversion algorithm. The second is the inversion cost (IC), which reflects the required number of extension field multiplications necessary for inversion. Notice that the smaller the inversion cost the better the performance of the concerned inversion algorithm, because the execution time is going to be faster.

The reader is going to observe the existence of six tables. The first, a table dedicated to compare our proposed $Inverse(\alpha, m)$ algorithm [see Chapter 4.1, Algorithm 6] with other existing inversion algorithms such as ITA, TYT and LCA algorithms [14, 16, 17]. The comparison is conducted over a set of applicable $GF(2^m)$ (or ms) for use in ECC including the ones recommended for government use. From that table, the reduction in inversion cost using our inversion algorithm is up to four $GF(2^m)$ multiplications in comparison with other existing algorithms.

The second and the third tables again dedicated to compare $Inverse(\alpha, m)$ algorithm with TYT and LCA algorithms, but now using some of their reported extension degrees ms, or $GF(2^m)$. Although the comparison is conducted over a small set of $GF(2^m)$, given the fact that our algorithm has inversion cost similar to that of LCA algorithm in that specific set, it has less inversion cost in comparison with TYT algorithm and up to one $GF(2^m)$ multiplication.

The forth, a table dedicated to compare our proposed $WHTI(\alpha, m)$ algorithm in $GF(3^m)$ [see Chapter 4.2, Algorithm 7], with other algorithms those exactly based ITI expression [20, 23, 44] over a set of ms, or $GF(3^m)$, of particular interest for use in ECC. From that table, the reduction in inversion cost is up to one $GF(3^m)$ multiplication in addition to avoiding the subfield inversion using our inversion algorithm in comparison with ITI-based inversion algorithms.

The fifth, a table dedicated to compare our proposed $GF(3^m)$ inversion algorithms with each other over a set of ms, or $GF(3^m)$, of particular interest for use in ECC. This comparison aim to show the advantage of $ITnverse(\alpha, m)$ algorithm [see Chapter 4.2, Algorithm 10] in comparison with $WHTI(\alpha, m)$ algorithm in such fields. From that table, the reduction in inversion cost is up to three $GF(3^m)$ multiplications using $ITnverse(\alpha, m)$ algorithm in comparison with $WHTI(\alpha, m)$ algorithm.

The sixth, a table dedicated to compare our proposed inversion algorithms, such as $Inverse(\alpha, m)$ and $ITnverse(\alpha, m)$ algorithms, against ITA, TYT and LCA algorithms, over a continuous set of extension degrees ms, or over a continuous set of extension fields. Since the inversion cost of $Inverse(\alpha, m)$ and $ITnverse(\alpha, m)$ algorithms is identical, the comparison here generally, reflect the effectiveness of our decomposition algorithm over other algorithms. From that table, using our inversion algorithms one third of inversion cost values are lower than that in other algorithms. Therefore, approximately a reduction of 32% in inversion cost is achieved in that continuous set, in comparison with other inversion algorithms.

Notice that the acronym IC that is shown in the following tables refers to inversion cost, while the acronym OD refers to the optimal decomposition for the extension degree m of the concerned extension field.

5.1.1 Characteristic Two

Given that our decomposition algorithm [$WHDA(m)$] was previously applied on a selected set of extension degrees ms of $GF(2^m)$, those selected from within the range ($100 \leq m < 571$), their optimal decompositions (OD), in addition to the corresponding number of the required extension field multiplications, or the inversion costs (IC) using our proposed field inversion algorithm [$Inverse(\alpha, m)$], are listed in Table 5.1 above. As a preliminary step to show the effectiveness of $Inverse(\alpha, m)$ algorithm and for comparison purposes, the OD and the IC of other existing inversion algorithms [ITA, TYT, LCA] are also listed in the same table.

It can be seen from Table 5.1 above that our proposed inversion algorithm achieves better reductions in inversion cost in comparison with other existing inversion algorithms. In some extension degrees ms, or in some binary extension fields $GF(2^m)$, such reductions in inversion cost are up to four extension field multiplications. Although the listed set of extension degrees ms of $GF(2^m)$ is not comprehensive, the results shown in the table reflect the applicability of our proposed decomposition method employed in our algorithm in accelerating field inversion in $GF(2^m)$, in comparison with other existing decomposition methods.

Table 5.1: Proposed *vs* Other Inversion Algorithms [$GF(2^m) : 100 \leq m < 571$]

$GF(2^m)$		**ITA** [14]	**TYT** [16]		**LCA** [17]		**Proposed** [Algo. 6]	
m	$(m-1)$	IC	OD	IC	OD	IC	OD	IC
100	99	9	11×9	9	11×9	9	3×32+3	8
108	107	10	2(13×4+1)+1	10	11×9+8	10	13×8+3	9
116	115	10	23×5	10	23×5	10	7×16+3	9
150	149	10	37×4+1	10	37×4+1	10	9×16+5	9
163	162	9	19×8	9	19×8	9	81×2	9
164	163	10	18×9+1	10	18×9+1	10	5×32+3	9
168	167	11	83×2+1	11	41×4+3	11	10×16+7	10
174	173	11	43×4+1	11	43×4+1	11	21×8+5	10
180	179	11	2(11×8+1)+1	11	11×16+3	11	11×16+3	10
184	183	12	14×13+1	11	14×13+1	11	11×16+7	10
208	207	12	23×9	11	23×9	11	24×8+15	10
215	214	11	107×2	11	53×4+2	11	13×16+6	10
216	215	12	43×5	11	43×5	11	13×16+7	10
228	227	11	113×2+1	11	25×9+2	11	7×32+3	10
231	230	11	23×10	11	23×10	11	7×32+6	10
233	232	10	29×8	10	29×8	10	29×8	10
280	279	12	31×9	12	31×9	12	17×16+7	11
283	282	11	141×2	11	20×14+2	11	17×16+10	11
294	293	11	73×4+1	11	73×4+1	11	9×32+5	10
299	298	11	2(37×4+1)	11	37×8+2	11	18×16+10	10
312	311	13	31×10+1	13	18×17+5	12	19×16+7	11
320	319	14	29×11	12	29×11	12	19×16+15	11
324	323	11	19×17	11	19×17	11	5×64+3	10
350	349	13	29×12+1	12	29×12+1	12	21×16+13	11
360	359	13	179×2+1	13	97×3+68	12	11×32+7	11
392	391	12	23×17	12	23×17	12	12×32+7	11
404	403	12	67×6+1	12	67×6+1	12	25×16+3	11
409	408	11	24×17	10	24×17	10	51×8	10
424	423	13	47×9	13	47×9	13	13×32+7	11
436	435	13	29×5×3	12	29×5×3	12	27×16+3	11
448	447	15	149×3	12	149×3	12	27×16+15	11
571	570	13	19×6×5	12	19×6×5	12	35×16+10	12

A selected set of extension degrees ms of $GF(2^m)$, selected from [17] is shown in Table 5.2 above. Such a set of binary extension fields is associated with the lowest possible inversion cost using the decomposition method employed in LCA algorithm. The aim of the following comparison is to show the effectiveness of our decomposition method, employed in our $Inverse(\alpha, m)$ algorithm, relative to the decomposition method of LCA algorithm over the provided set of $GF(2^m)$. Therefore, the OD and the IC using our algorithm for such ms of $GF(2^m)$, are also listed in the same table.

Table 5.2: Proposed *vs* LCA Algorithm

$\boldsymbol{GF(2^m)}$		**LCA** [17]		**Proposed** [Algo. 6]	
m	$(m-1)$	OD	IC	OD	IC
123	122	40×3+2	9	14×8+10	9
187	186	34×5+16	10	11×16+10	10
189	188	36×5+8	10	22×8+12	10
238	237	68×3+33	11	14×16+13	11
384	383	25×5×3+8	12	23×16+15	12
428	427	25×17+2	12	13×32+11	12

It can be seen from Table 5.2 above that the inversion cost of both algorithms is identical in that specific set of $GF(2^m)$. Therefore, our proposed inversion algorithm, namely $Inverse(\alpha, m)$ algorithm, can be a substitute for LCA algorithm to calculate the inverse in such binary extension fields. Knowing that our inversion algorithm is associated with the lowest possible inversion costs over a broader set of $GF(2^m)$ as clarified in Table 5.1 above in comparison with LCA algorithm.

A selected set of extension degrees ms of $GF(2^m)$, selected from [16] is shown in Table 5.3 above. Such a set of binary extension fields is associated with the lowest possible inversion cost using the decomposition method employed in TYT algorithm. To show that our decomposition method employed in $Inverse(\alpha, m)$ algorithm can achieve lower inversion costs in that specific set and in comparison with TYT decomposition method, the OD and the IC using our inversion algorithm for such ms, or in such $GF(2^m)$, are also listed in the table for comparison purposes with TYT inversion algorithm.

Table 5.3: Proposed *vs* TYT Algorithm

$\boldsymbol{GF(2^m)}$		**TYT** [16]		**Proposed** [Algo. 6]	
m	$(m-1)$	OD	IC	OD	IC
128	127	18×7+1	10	15×8+7	10
192	191	38×5+1	11	23×8+7	11
256	255	17×5×3	10	17×15	10
320	319	29×11	12	19×16+15	11
384	383	2(38×5+1)+1	13	23×16+15	12
416	415	83×5	12	25×16+15	11

It can be seen from Table 5.3 above that the inversion cost of both algorithms is the same in some $GF(2^m)$, except for the last three entries in the table, whereby they indicate that our algorithm achieves lower inversion costs in comparison with TYT inversion algorithm. Therefore, our proposed inversion algorithm, namely $Inverse(\alpha, m)$ algorithm, can be a substitute for TYT algorithm to calculate the inverse in such $GF(2^m)$. Knowing that our inversion algorithm is associated with the lowest possible inversion costs over a broader set of $GF(2^m)$ as clarified in Table 5.1 above in comparison with TYT algorithm.

The achieved results above reflect the applicability of our field inversion algorithm proposed mainly in $GF(2^m)$ [see Chapter 4.1, Algorithm 6]. It is clear that our inversion algorithm is associated with the minimal possible inversion cost in most $GF(2^m)$, including the ones recommended for use in ECC. Therefore, when our algorithm is employed as the main component for inverse computation in other algorithms, such as in the scalar multiplication algorithm in ECC that relies heavily on field inversion, the scalar multiplication algorithm is executed very fast.

The scalar multiplication is the core algorithm of most modern cryptographic applications (or algorithms) that must be executed fast. Some cryptographic algorithms based on ECC that have the scalar multiplication as the core algorithm are: elliptic curve digital signature algorithm (ECDSA), elliptic curve diffie-hellman (ECDH) key-agreement algorithm and elliptic curve elgamal (EC-ElGamal) encryption algorithm, etc.

We conclude this subsection by presenting a set of useful propositions necessary to understand the way in which our proposed main inversion algorithm works, namely $Inverse(\alpha, m)$ algorithm, and in the mean time, to clarify some other aspects relevant to the optimal decomposition(s) for m of $GF(2^m)$ using our decomposition algorithm, namely $WHDA(m)$ algorithm. Notice that in the following propositions we referred to our $WHCA$ algorithm [see Chapter 4.1, Algorithm 4].

Subsequently, other set of propositions is included to assist in searching for the short addition chains for a particular integer and to confirm the existence of such addition chains.

We will start by introducing some helpful notations for the propositions: The number of required extension field multiplications in calculating (α^{2^x-1}) by using the integer-value x ($NRMs(x)$), binary length of x ($l(x)$), Hamming weight of x ($w(x)$), x is a full-weight integer ($l(x) = w(x)$), the short addition chain of x (C_x), the length of C_x is (l_{C_x}). Notice that if $x = (m-1)$, then $NRMs((m-1))$ is equal to that number required for finding the inverse. Also, if $x = h$, then $NRMs(h) = 1$ (*based on our proposed decomposition method*). Furthermore, given $WHDA(m) = r_1 \times n + h$, then $NRMs(WHDA(m)) = NRMs(r_1 \times n + h) = NRMs(r_1) + NRMs(n) + 1$.

Prop. **1.** *Given* $m = (2^k + 1)$ *for any positive integer* k*, no decomposition is required for* m*. In calling* $WHCA$ *algorithm, the inverse is obtained with* $NRMs(2^k) = k$ *multiplications.*

Prop. **2.** *Given* $m = (2^k + 2^j + 1)$ *for any positive integers* $k > j$ *(i.e.,* $w(m-1) = 2$*), no decomposition is required for* m*. In calling* $WHCA$ *algorithm, the inverse is obtained with* $NRMs(2^k + 2^j) = k + 1$ *multiplications.*

Prop. **3.** *Given $m = (p+1)$, for prime p with $w(p) = 3$, then $WHDA(p+1)$ is expected to produce more reductions, and the inverse is obtained with $NRMs(r_1 \times n + h)$ multiplications.*

Prop. **4.** *Given $m = (p \times q + 1)$, for prime p and q with $w(p \times q) = 3$, then $WHDA(p \times q + 1)$ is expected to produce more reductions, and the inverse is obtained with $NRMs(r_1 \times n + h)$ multiplications.*

Prop. **5.** *Given $m = (2^k \times p + 1)$ for any positive integer k and prime p. If $w(p) < 3$, no decomposition is required for m. In calling $WHCA$ algorithm, the inverse is obtained with $NRMs(2^k \times p) = k + NRMs(p)$ multiplications. If $w(p) \geq 3$, let $r_1 = 2^k$ and $r_2 = WHDA(p+1)$, thus the inverse is obtained with $NRMs(r_1 \times r_2) = k + NRMs(r_2)$ multiplications.*

Prop. **6.** *For the set of extension degrees ms those decomposed as $r_1 \times n + h$, if $h \in C_{r_1}$ is a full-weight remainder, $w(r_1) = 2$ and $n = 2^k$ for any positive integer k, when increasing k while fixing r_1 and h values, those ms are expected to have the minimum $NRMs(r_1 \times n + h)$ in inverse calculation relative to other ms.*

Prop. **7.** *For the set of extension degrees ms those decomposed as $r_1 \times n + h$ with $h = 1$ (*this h always $\in C_{r_1}$*), those ms in inverse calculation have $NRMs(r_1 \times n + h)$ same as that required by using TYT algorithm. With $h = 2^k$ for any positive integer k (*this h often $\in C_{r_1}$*), those ms in inverse calculation have $NRMs(r_1 \times n + h)$ same as that required by using LCA algorithm. Thus, such ms are also associated with lowest possible inversion cost using our proposed inversion algorithm.*

Prop. **8.** *For the set of extension degrees ms those decomposed as $r_1 \times n + h$ with $h = r_1$ (*this h always $\in C_{r_1}$*), those ms can be factorized into prime factors. In such a case, $WHDA(m)$ algorithm may or may not produce the minimum $NRMs(r_1 \times n + h)$ in inverse calculation compared to prime factors decomposition. Select first decomposition if associated with lower inversion cost, otherwise, select the latter using* *the prime factors r_1 and r_2, since $r_1 \times n + r_1 = r_1 \times (n+1) = r_1 \times r_2$. Factorize r_2 if possible, to achieve more reductions.*

The following propositions are helpful in finding the short addition chains (SACs) for an integer, which may be applicable to some cases. In searching for the SAC for the positive integer x (i.e., C_x):

Prop. **1.** *If x has a large value, then there are many possible SACs for x, thus, the set of values that can be taken by the remainder h is maximized, given that $h \in C_x$.*

Prop. **2.** *If $x = r_1 \times r_2 \times \cdots \times r_k$ (multiplication of several factors), given that h is equal to any factor r_i for $1 \leq i \leq k$, then h must belong to the SAC of x, i.e., $h \in C_x$.*

Prop. **3.** *If $x = r_1 \times r_2 \times \cdots \times r_k$ (multiplication of several factors), given that $r_1 > r_i$ for $2 \leq i \leq k$ and $x \neq 2^k$ for any positive integer k, then there must be a SAC of x of length $l_{C_x} = l_{C_{r_1}} + \lceil \log_2 r_i \rceil$ for $2 \leq i \leq k$.*

Prop. **4.** *If $x = 2^k$ for any positive integer k, then x has only one SAC of length $l_{C_x} = k$, which is the Power of Two SAC, i.e., $(1, 2, 4, \cdots, 2^k)$. Notice that the elements of the General SAC can have any value, not only power of two values.*

Prop. **5.** *If $x = 2^k + 2^j$ for any positive integers $k > j$ (i.e., $w(x) = 2$), then x has at least two SACs of length $l_{C_x} = k + 1$, where the Power of Two SAC is one of them.*

Prop. **6.** *If $x = k + n$, given that k is the largest 2^i value in x for $i = 1, 2, 3, \cdots$, if $w(n) < 3$, then C_x is obtained by using either the General or the Power of Two SAC. Otherwise, if $w(n) \geq 3$, then the General SAC is expected to produce the short addition chain of x, i.e., C_x.*

5.1.2 Characteristic Three

Recalling that the performance of a field inversion algorithm is measured by its required number of extension field multiplications, those necessary for finding the inverse of a nonzero element in the concerned ternary extension field $GF(3^m)$, thus, an optimal performance inversion algorithm requires less number of such multiplications, or less inversion cost, relative to its counterparts.

The performance of our first field inversion algorithm proposed mainly in $GF(3^m)$ [see Chapter 4.2, Algorithm 7] is better than its counterparts those based on ITI expression (see Chapter 3.2.1), in the sense that, in addition to avoiding the required subfield inversion, it requires less $GF(3^m)$ multiplications as evident from Eqs. (3.24) and (4.20) given above. Table 5.4 is dedicated to compare our proposed $WHTI(\alpha, m)$ algorithm with ITI-based inversion algorithms [20, 23, 44] over a set of ms, or $GF(3^m)$, taken from within the range ($79 \leq m < 239$) of particular interest for use in ECC. Notice that, in the table I_{sub} refers to a subfield inversion.

Table 5.4: Proposed *vs* Other Algorithms [$GF(3^m) : 79 \leq m < 239$]

$\boldsymbol{GF(3^m)}$		**ITI-Based Algorithms** [20,23,44]	**Proposed** Algo. 7
m	$(m-1)$	$\boldsymbol{IC = (Mults + I_{sub})}$	$\boldsymbol{IC = Mults}$
79	78	$(11+1)$	10
97	96	$(9+1)$	8
108	107	$(12+1)$	11
116	115	$(12+1)$	11
150	149	$(12+1)$	11
163	162	$(11+1)$	10
167	166	$(12+1)$	11
173	172	$(12+1)$	11
193	192	$(10+1)$	9
208	207	$(14+1)$	13
215	214	$(13+1)$	12
239	238	$(14+1)$	13

It can be seen from Table 5.4 above that our proposed inversion algorithm achieves better reductions in inversion cost in comparison with ITI-based inversion algorithms. For all ternary extension fields $GF(2^m)$, such reductions in inversion cost are up to one extension field multiplication. In addition, given that our algorithm does not the require subfield inversion as in other algorithms, this renders our algorithm modular and suitable for VLSI implementation. The modularity of our algorithm stems from its reliance on only two field operations for inversion, such as the multiplication operation and the cyclic-shifts operation.

We recall that our second field inversion algorithm proposed mainly in $GF(3^m)$ [see Chapter 4.2, Algorithm 10], works based on (or employs) our proposed decomposition algorithm for the extension degree m of the concerned extension field. Given that our decomposition algorithm [$WHDA(m)$] was previously applied on a selected set of extension degrees ms of $GF(3^m)$, those selected from within the range ($79 \leq m < 239$) of particular interest for use in ECC, their optimal decompositions (OD), in addition to the inversion costs (IC) using $ITnverse(\alpha, m)$ algorithm, are listed in Table 5.5 above for comparison purposes with $WHTI(\alpha, m)$ algorithm.

Table 5.5: Proposed Inversion Algorithms [$GF(3^m) : 79 \leq m < 239$]

$\boldsymbol{GF(3^m)}$		**Proposed [Algorithm 7]**	**Proposed [Algorithm 10]**	
m	$(m-1)$	IC	OD	IC
79	78	10	9×8+6	8
97	96	8	$\diamond\ C_{96}$	7
108	107	11	13×8+3	9
116	115	11	7×16+3	9
150	149	11	9×16+5	9
163	162	10	81×2	9
167	166	11	10×16+6	9
173	172	11	20×8+12	9
193	192	9	$\diamond\ C_{192}$	8
208	207	13	24×8+15	10
215	214	12	13×16+6	10
239	238	13	17×14	10

In table 5.5 above, notice that the entries preceded by the symbol $\diamond$ indicate that the short addition chain for m has been used for inversion, but that specific m of

$GF(3^m)$ has not decomposed. For example, when considering inversion in $GF(3^{96})$, then the short addition chain $C_{96} = \{1, 2, 4, 8, 16, 32, 64, 96\}$ is used, which requires 7 $GF(3^{96})$ multiplication for inversion. Also, when considering inversion in $GF(3^{192})$, then the short addition chain $C_{192} = \{1, 2, 4, 8, 16, 32, 64, 128, 192\}$ is used, which requires 8 $GF(3^{192})$ multiplication for inversion, and so on. In addition, notice that the reduction in the required number of $GF(3^m)$ multiplications necessary for field inversion (or the reduction in inversion cost) in using $ITnverse(\alpha, m)$ algorithm is up to three extension field multiplications relative to $WHTI(\alpha, m)$ algorithm.

In fact, the obtained results in Table 5.5 above confirm the applicability and the validity of our proposed decomposition method for accelerating inversion in ternary extension fields $GF(3^m)$, as it is exactly the case in binary extension fields $GF(2^m)$ based on our previous discussion herein. In addition, the obtained results confirm that $ITnverse(\alpha, m)$ algorithm has better performance, or less inversion cost, not only in comparison with our proposed $WHTI(\alpha, m)$ inversion algorithm, but also in comparison with ITI-based inversion algorithms in $GF(3^m)$. Such algorithms showed inferior performance relative to our $WHTI(\alpha, m)$ inversion algorithm, as it was clarified in Table 5.4 above.

We recall that this type of research work, the decomposition methods for the extension degree m, is rare in ternary extension fields $GF(3^m)$, if not exist at all. In addition, we recall that our inversion algorithms that employ our decomposition algorithm in binary $GF(2^m)$ and ternary $GF(3^m)$ extension fields, respectively, are associated with identical inversion cost. Therefore, the optimal decompositions (OD) and the inversion costs (IC) comparisons of such algorithms against ITA, TYT and LCA inversion algorithms [14, 16, 17], is a valid and applicable work. Keeping in mind that ITA, TYT and LCA algorithms employ different types of decomposition methods. Table 5.6 above provides such a comparison, over a continuous set of extension degrees ms, taken from within the range $(200 - 231)$. This set has been used to further emphasize the validity and applicability of our inversion algorithms.

Table 5.6: $WHDA(m)$-Based *vs* Other Algorithms [$GF(p^m): m = 200, ..., 231$]

$GF(p^m)$		ITA [14]	TYT [16]		LCA [17]		$WHDA(m)$-Based Algs.	
m	$(m-1)$	IC	OD	IC	OD	IC	OD	IC
200	199	11	18×11+1	11	18×11+1	11	12×16+7	10
201	200	9	20×10	9	20×10	9	25×8	9
202	201	10	20×10+1	10	20×10+1	10	12×16+9	10
203	202	10	2(25×4+1)	10	20×10+2	10	12×16+10	10
204	203	11	29×7	11	29×7	11	25×8+3	10
205	204	10	17×12	9	17×12	9	17×12	9
206	205	11	17×12+1	10	17×12+1	10	25×8+5	10
207	206	11	41×5+1	11	17×12+2	10	25×8+6	10
208	207	12	23×9	11	23×9	11	24×8+15	10
209	208	9	13×16	9	13×16	9	13×16	9
210	209	10	13×16+1	10	13×16+1	10	13×16+1	10
211	210	10	14×5×3	10	13×16+2	10	13×16+2	10
212	211	11	14×5×3+1	11	13×16+3	11	13×16+3	10
213	212	10	53×4	10	53×4	10	25×8+12	10
214	213	11	53×4+1	11	53×4+1	11	13×16+5	10
215	214	11	107×2	11	53×4+2	11	13×16+6	10
216	215	12	43×5	11	43×5	11	13×16+7	10
217	216	10	18×12	9	18×12	9	27×8	9
218	217	11	18×12+1	10	18×12+1	10	13×16+9	10
219	218	11	109×2	11	18×12+2	10	13×16+10	10
220	219	12	73×3	10	73×3	10	27×8+3	10
221	220	11	20×11	10	20×11	10	13×16+12	10
222	221	12	17×13	10	17×13	10	17×13	10
223	222	12	37×6	10	37×6	10	26×8+14	10
224	223	13	37×6+1	11	37×6+1	11	26×8+15	11
225	224	9	14×16	9	14×16	9	7×32	9
226	225	10	14×16+1	10	14×16+1	10	7×32+1	10
227	226	10	2(7×16+1)	10	14×16+2	10	7×32+2	10
228	227	11	113×2+1	11	25×9+2	11	7×32+3	10
229	228	10	19×12	10	19×12	10	57×4	10
230	229	11	19×12+1	11	19×12+1	11	7×32+5	10
231	230	11	23×10	11	23×10	11	7×32+6	10

It can be seen from Table 5.6 above that more than one third of inversion cost values of our proposed inversion algorithms are less than the values that associated with the other existing inversion algorithms. This is an indicator on the superior performance of our algorithms in such extension fields for inversion in comparison with other algorithms. The other two third of inversion cost values of our algorithms are exactly the same as the minimum values that attained by using the other existing inversion algorithms. Therefore, our proposed field inversion algorithms can be used as a substitute for other existing inversion algorithms. This is because they cover a very large set of extension degrees ms, or in other words, our algorithms are suitable for field inversion in almost all binary and ternary extension fields.

We recall that a minor reduction in the number of extension field multiplications, or inversion cost, has a remarkable positive effect on the overall performance of the concerned field inversion algorithm, especially in using normal basis representation for the elements of the concerned field. Thus, the achieved results in Table 5.6 above confirm the validity and the applicability of our proposed decomposition method in accelerating field inversion, not only in ternary extension fields $GF(3^m)$, but also in binary extension fields $GF(2^m)$, in comparison with different types of decomposition methods that employed in other existing inversion algorithms available in the literature.

5.2 Summary

At the beginning of this chapter, we showed how $Inverse(\alpha, m)$ algorithm in $GF(2^m)$ is associated with the lowest possible inversion cost in comparison with ITA, TYT and LCA inversion algorithms. The algorithm implements the decomposition method employed in our proposed decomposition algorithm, namely $WHDA(m)$ algorithm. Given the fact that other existing inversion algorithms rely on different decomposition methods for the extension degree m of the concerned $GF(2^m)$.

Furthermore, we showed how $WHTI(\alpha, m)$ inversion algorithm in $GF(3^m)$ is associated with lower inversion cost, in comparison with ITI-based inversion algorithms. In addition, we indicated the modularity of the algorithm because of its reliance on only two field operations for performing field inversion in such fields. The algorithm does not require the subfield inversion operation that is indispensable in using ITI-based inversion algorithms.

Finally, we showed how $ITnverse(\alpha, m)$ inversion algorithm in $GF(3^m)$ is associated with the lowest possible inversion cost in comparison with, not only $WHTI(\alpha, m)$ algorithm, but also in comparison with ITI-based inversion algorithms. The algorithm implements the decomposition method employed in our proposed decomposition algorithm, namely $WHDA(m)$ algorithm. Knowing that such decomposition methods for the extension degree m of the concerned $GF(3^m)$ are rare, if not exist at all in such fields.

Chapter 6

Discussion

The current chapter is mainly dedicated for wrapping up this dissertation. In the first section, we are going to provide a summary for the achieved work and highlight the main points mentioned in the previous chapters, and then, provide the closing remarks that are relevant to our research work presented herein. In the second section, we will present some future recommendations that will be provided in the form of suggestions, which are considered either as a continuation to our current research work presented herein or as a newly suggested ideas and notions that may attract the interest of academic researchers for future consideration.

6.1 Summary and Conclusions

In the first chapter of this dissertation we have provided an introduction that relates elliptic curves to finite fields, and we have showed how they are related to each other. In such introductory chapter we have clarified the paramount importance of finite fields, in general, for use in elliptic curve cryptography (ECC). In particular, when we have directed the spotlights on the arithmetic operations of the concerned elliptic curve (or simply, on curve-operations), and when we have assumed that such operations are located at the highest level of hierarchy, we have showed that all such curve-operations, in its functioning, are heavily related to the arithmetic operations in the underlying finite field over which the elliptic curve is defined, as in ECC. We have assumed that such field-operations are located at the lowest level of hierarchy.

Based on our discussion above, we have showed that by optimizing the underlying field-operations, curve-operations located at the highest level of hierarchy will also be optimized. Since finite field inversion (or simply, the inverse) is the most time-consuming operation, in terms of the execution time, and since the core curve-operation, namely the scalar multiplication operation, which is the heart of most cryptographic applications based on ECC, heavily relies on the inverse (performs field inversion very frequently in its functioning), this is why in the first chapter of this dissertation, we have focused mainly in our introduction on inverse acceleration in two different types of extension fields, namely the binary and ternary extension fields. In addition, we have reflected that our final goal is to accelerate the scalar multiplication operation, which is responsible for accelerating most cryptographic applications based on ECC, as it was mentioned earlier herein.

In the second chapter of this dissertation we have provided a mathematical background to clarify all the mathematical concepts either relevant to finite fields, such as their definitions, representation bases and their associated arithmetic operations, or relevant to other concepts that are useful for understanding how inverse calculation is performed in finite extension fields. In such a chapter, we have included only the necessary mathematical concepts for better understanding of most materials presented herein.

We started our journey by defining the classes of finite fields, which are classified into three main classes, the representation bases for their elements, like normal and polynomial bases, and their associated arithmetic operations. Then, we have moved forward and defined a specific type of finite fields, namely the optimal extension fields (OEFs), which in turn are classified into two types, namely Type-I and Type-II OEFs. We have mentioned earlier that such fields are very useful for accelerating field-operations at both the software and hardware levels, more specifically, for accelerating the most frequently used extension-field and subfield modular reduction operations.

After that, we have moved further forward and showed how Fermat's little theorem is very useful in calculating the inverse in finite fields. As we indicated earlier herein, such a theorem has converted the problem of field inversion to solving exponentiation operation in such fields. Such an exponentiation operation in finite fields is simply calculated by performing repeated multiplications and p^e-th powers. Therefore, we have moved to define the concept of the short addition chains (SACs), since such a concept is necessary for taking care of the first calculation part of exponentiation which represents the required multiplication operations. The SACs concept provides a systematic and fast way to calculate terms like (α^{p^r-1}), which has solved the computationally intensive part in field inversion, especially in using NB for the field elements. The other calculation part of the exponentiation which represents the required p^e-th powers has been solved in a straightforward manner. Such powers are simply represented by cyclic-shift operations in using NB for the field elements. This computational part in field inversion is associated with an extremely low execution time, if not free, since the cyclic-shift operations are very fast relative to multiplication operations.

In the third chapter of this dissertation we have provided a literature survey on all existing and relevant field inversion algorithms. We have started by considering inversion in binary extension fields $GF(2^m)$, by previewing the available algorithms based on Fermat's inversion approach and using NB for the field elements. We have explained how the ITA inversion algorithm works, which can be considered as the standard inversion algorithm in such fields. As we have mentioned earlier herein, in using ITA algorithm, the number of the required extension field multiplications necessary for inversion, or the inversion cost, is expressed as a logarithmic function of the extension degree m, which is a function with the binary-length $\ell(m-1)$ and the Hamming-weight $w(m-1)$. We recall that for some extension degrees ms, or in some $GF(2^m)$, ITA algorithm has a worst case scenario, which is represented by case when

$\ell(m-1) = w(m-1)$. We have also indicated earlier that the academic researchers put more attention on a such case, in attempt to improve it in their proposed work.

After that, we have moved to explain CEA inversion algorithm, which introduced the concept of factorization (or decomposition) for the extension degree m into two factors as the starting point to improve the worst case in ITA algorithm. Because of its limitations, we have moved further to explain TYT inversion algorithm, which has solved the case when $(m-1)$ is a prime value that is not applicable using CEA algorithm. TYT algorithm is also associated with lower inversion cost, whereby the number of multiplications necessary for inversion is given as a function with each factor and the remainder h in $(m-1)$, given that $(m-1)$ is decomposed into several factors and a remainder. Our final move in binary extension fields was toward explaining LCA inversion algorithm, which has a lower inversion cost, since the number of multiplications necessary for inversion is given as a function with each factor and the Hamming-weight of the remainder $w(h)$ in $(m-1)$, rather than a function with h itself as in using TYT inversion algorithm.

Finally in chapter three, we have started by considering field inversion in ternary extension fields $GF(3^m)$, and have provided a literature survey on all existing and relevant inversion algorithms in such fields. As we have mentioned earlier herein, most inversion algorithms that previously proposed in $GF(3^m)$, are mainly relied on the so-called ITI expression to calculate the inverse. The expression is not written in full as a cohesive inversion algorithm in such fields. Inverse calculation using ITI expression is only provided in the form of different steps, with different types of field operations. As it was mentioned earlier, many researchers have focused on software and hardware aspects in attempts to improve inverse calculation by using ITI expression, with some constraints and conditions applied to achieve their final goals.

In the fourth chapter of this dissertation we have introduced the decomposition method employed in our decomposition algorithm and the proposed field inversion algorithms. We started by considering binary extension fields $GF(2^m)$ by proposing an inversion algorithm, which is an improved version of the aforementioned LCA algorithm. As clarified earlier, our proposed decomposition method, when applied in such fields, it reduces the number of multiplications necessary for inversion to a function with each factor in $(m-1)$ plus a constant term of value equal to 1, which means that our proposed inversion algorithm is independent of the remainder h, contrary to other existing inversion algorithms, such as TYT algorithm, which depends on the remainder h itself, and LCA algorithm, which depends on the Hamming-weight of the remainder h. Our proposed inversion algorithm in $GF(2^m)$ showed the best performance in comparison with other inversion algorithms.

After that, we have started by considering field inversion in ternary extension fields $GF(3^m)$, by proposing our first inversion algorithm, which has better performance relative to ITI-based inversion algorithms in such fields. Then, we have moved to our second field inversion algorithm proposed in $GF(3^m)$, which has employed our proposed decomposition method. As it was pointed earlier herein, the performance of such inversion algorithm is better than our first inversion algorithm proposed in such fields, and also better than other existing inversion algorithms available in the literature. It is associated with the best performance since it requires the minimal possible number of extension field multiplications those necessary for inversion, as it was indicated earlier herein. Finally, we have moved to present our proposed fast Frobenius map operation that is applicable in Type-II OEFs. Such operation was extended to higher characteristic fields, which is associated with almost free runtime and space complexity.

In the fifth chapter of this dissertation we have provided all the obtained results and the associated analyses that are relevant to our research work herein. We have started by showing the obtained results relevant to inversion in extension fields in the form of tables, whereby we first considered binary extension fields $GF(2^m)$. Each column in the provided tables represented a different inversion algorithm and each row represented a different extension degree m of $GF(2^m)$, with the entries in each table representing inversion cost values that used for comparison purposes with other available algorithms in such fields.

After that, we have moved to show the obtained results in ternary extension fields $GF(3^m)$, whereby the results are presented again in the form of tables, exactly the same way as in the case of binary extension fields. We have also included a table of continuous range of extension degrees ms to show the behavior of our proposed decomposition method. From that table, it was revealed that one third of inversion cost values are lower in using our inversion algorithms, or in other words, approximately a reduction of 32% in inversion cost is achieved in using our algorithms in comparison with other inversion algorithms. The rest of inversion cost values of our algorithms were exactly the same as the minimum values that attained by using other existing inversion algorithm.

In the sixth chapter of this dissertation, which is exactly the current chapter, we have provided a summary for all what has been done in the previous chapters, and highlighted the main points that achieved out of our proposed work. In addition, we are going to conclude our dissertation as in what follows.

Based on what has been mentioned in our discussion above, it is clear that our research work is somewhat valid and applicable for use in practical cryptographic applications based on ECC. In fact, the set of selected extension degrees ms, or the set of chosen binary and ternary extension fields, those associated with fast inverse calculation, as a result of using our proposed field inversion algorithms, fall within the recommended ranges, or within the recommended finite fields, as recommended by NIST and IETF organizations for use in ECC.

Since our main focus here is on accelerating field inversion in binary and ternary extension fields, the most studied fields by academic researchers in the literature, this can be considered as the other competitive advantage which reinforces our research work herein. We recall that the binary extension fields $GF(2^m)$ are suitable for use in cryptographic applications from the software and hardware perspective (no carry-propagation chains or rounding errors exist), and ternary extension fields $GF(3^m)$ are suitable for use in cryptographic applications that are based on bilinear mappings, such as either Weil or Tate pairings, as it was mentioned earlier herein.

The above mentioned facts allow us to safely recommend and encourage the use of our proposed field inversion algorithms in elliptic curves cryptography, more specifically, for use in practical cryptographic applications (or algorithms) based on elliptic curves, since the core curve-operation in such applications, namely the scalar multiplication operation, which can use any of our proposed field inversion algorithms to find the inverse of a field element is going to be a very fast operation, in terms of the execution time, which in turn accelerates the execution time of the concerned cryptographic application.

6.2 Future Recommendations

In our research work herein we have mainly focused on accelerating field inversion in binary and ternary extension fields. We proposed a decomposition method for the extension degree m of the concerned field. Such a method is employed in our proposed field inversion algorithms. The method require the minimal number of extension field multiplications, or the lowest possible inversion cost, relative to other decomposition methods that employed in other existing inversion algorithms available in the literature.

In this section, we recommend some relevant research work, which may be useful for future consideration or may attract the interest of other academic researchers, those interested in such a type of research work. The recommendations are presented in the form of suggestions either for implementing a specific idea or a new algorithm similar to those previously mentioned herein, or for paving the road for continuing on our research work and the proposed ideas that presented herein.

Our first suggestion is to consider and think about hardware implementation for any of the field inversion algorithms proposed above, either in binary or ternary extension fields. By doing so, we can convert the proven mathematical facts into a reality by designing practical inversion systems with low space and time complexity. Such systems are expected to be suitable for implementing most of cryptographic applications that are based on elliptic curve cryptography.

Another suggestion is to extend the current research work by considering other representation bases, other than the normal and polynomial bases. In addition, by considering other extension fields, other than the binary or ternary extension fields. For example, by applying the our decomposition method in extension fields such as the “medium Galois fields” with new representation basis [48]. In such fields, both the extension degree m and the characteristic p that is a pseudo-Mersenne prime, are medium-valued integers.

The final suggestion is to consider accelerating field inversion for cryptographic applications, at both hierarchical levels simultaneously. By this we mean, to consider speeding up inversion by trying to optimize elliptic curve arithmetic operations (at the highest level), while at the same time, trying to optimize the underlying finite field arithmetic operations (at the lowest level), in order to accelerate field inversion at both hierarchical levels.

As a final note, it is known in the academic literature that the expressions (or algorithms) for finding the inverse are cumulative in its nature (a limitation). By itself, this can be a valid future research work, in the sense that, one can devise other mathematical expressions for inverse calculation to overcome this limitation. The reader can consider Appendix B included herein for a possible solution to this limitation, which significantly reduces the latency in inverse calculation.

Appendix A: Elliptic Curve Cryptography

Appendix A is mainly included herein to show how field inversion is very important in elliptic curve cryptography (ECC). It presents a short and quick introduction, in a concise way, to show this importance. First, it defines the ECC. Then, it gives a brief idea about some of the curve-operations, such as the scalar multiplication operation (also known as point multiplication in the academic literature), to show how the inverse is extremely required in performing such an operation. Scalar multiplication is the heart of most practical cryptographic-applications based on ECC, whereby it dominates their execution times.

Cryptography, in general, is the artistic science of writing secret codes, and in particular, it is the main tool that currently used to protect either stored information (or data), such as in computer systems, or to protect the exchanged information, such as in telecommunication systems. Cryptography provides important services to the field of network security such as encryption, authentication , key transportation, etc. Cryptography is mainly divided into two major categories, namely the private-key cryptography, used only for encryption, and the public-key cryptography, used for performing most cryptographic services such as those mentioned above. By itself, public-key cryptographic-systems are divided into two main classes, namely the RSA-based and the ECC-based systems.

ECC is standardized internationally by ISO and IETF, and in the USA by ANSI and NIST organizations. The ECC-based systems (or cryptographic applications) in their function rely heavily on the strong number-theoretic foundations of finite fields (or Galois fields), and because of this reliance, they require shorter keys in comparison with the RSA-based systems. Therefore, ECC-based systems are characterized by their hardware efficiency and the requirement for less computation times. The following highlights the main points relevant to ECC:

- Given an elliptic curve E that is defined over a finite field F_q, then the group of curve-points denoted as $G = E(F_q)$, has a number of points that defines the curve E, which is approximately equal to the field's order q.

- The strength of the group G operations stems from the mathematical hardness of the elliptic curve discrete-logarithm problem (ECDLP) that indirectly depends on the finite field over which the curve E is defined.

- ECDLP: Given the points P and $Q \in G$, with neither of them equal to the point at infinity, find an integer k such that

  $$Q = k \times P. \tag{1}$$

  In this case, finding k is mathematically hard, whereby k is called the discrete-logarithm of Q to the base P. In the context of ECC, k is a large private integer (or secret key) and the point P is usually a base point on the curve E.

- ECC-based cryptographic systems provide comparable security level with much shorter keys in comparison with RSA-based cryptographic systems because of ECDLP, thus, they are more hardware-efficient with less computation times.

To show how such elliptic curve-operations work, for example the scalar multiplication operation, the following example is very useful for showing how field inversion is indispensable for such an operation. Given the elliptic curve E that is defined over F_{2^m} [or equivalently $GF(2^m)$], which is described by the following characteristic equation

$$y^2 + x.y = x^3 + ax^2 + b,$$

where $y, x, a, b \in F_{2^m}$. Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ two points on E with neither of them is the point at infinity and $P \neq -Q$ (not on the same vertical line), then $R = (x_3, y_3) = (P + Q)$ is given by

$$x_3 = \lambda^2 + \lambda + x_1 + x_2 + a,$$

and

$$y_3 = (x_1 + x_3)\lambda + x_3 + y_1,$$

whereby

$$\lambda = \frac{y_1 + y_2}{x_1 + x_2} \quad if \quad P \neq Q \quad \text{(PA: point Addition operation)}, \tag{2}$$

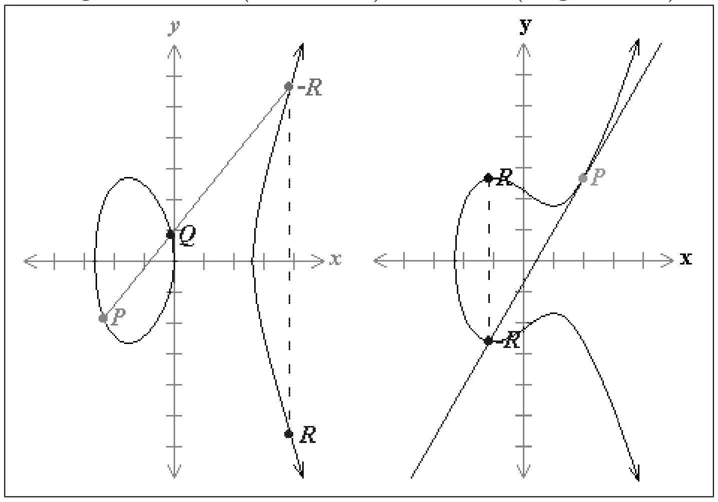
and

$$\lambda = \frac{y_1}{x_1} + x_1 \quad if \quad P = Q \quad \text{(PD: point doubling operation)}. \tag{3}$$

Point addition (PA) is the operation of adding two different curve-points to each other, whereby point doubling (PD) is the operation of adding a curve-point to itself. The geometrical meaning of such curve-points, i.e., PA and PD, is shown in the following figure (see Figure 6.2.1).

From Figure 6.2.1 above, to add two curve-points P and Q (i.e., point addition) graphically, we extend the line connecting them to find a third co-linear curve-point, namely $-R$, then, reflecting its y-coordinate across the x-axis to get the curve-point R which represents the final result of point addition. To add a curve-point P to itself (i.e., point doubling) graphically, we extend the tangent line of such a point to find the other co-linear curve-point, namely $-R$, then, reflecting its y-coordinate across the x-axis to get the curve-point R which represents the final result of point doubling.

Figure 1: PA (Left Side) and PD (Right Side)



Algorithm 11 Left-to-right Scalar Multiplication Algorithm [4]

```
Input: $k = (k_{r-1} \cdots k_1 k_0)_2, P \in G = E(F_{2^m})$
Output: $k \times P$
Initial: $Q := \infty$ (point at infinity)
for $i := r - 1$ to 0 do
    $Q := 2Q$; (PD)
    if $k_i = 1$ then
        $Q := Q + P$; (PA)
    end if
end for
return $Q$
```

Scalar multiplication (SM) (i.e., point multiplication) is the curve-operation that is responsible for computing the right-hand side in ECDLP expression as given in Eq. $(A.1)$ above, for large and private integer k and curve-point P on E. This computation is required in almost all the cryptographic applications based on ECC such as in elliptic curve digital signature algorithm (ECDSA) for authentication, elliptic curve diffie-hellman algorithm (ECDH) for key-exchange, and elliptic curve el-gamal algorithm (ECEl-Gamal) for encryption.

At algorithmic level, the scalar multiplication algorithm calls very frequently for performing both PA and PD operations for its final output to be ready, as evident from Algorithm 11 shown above.

The above shown algorithm reflects the reliance of the scalar multiplication on both point addition and doubling operations, as it was mentioned earlier. Since both curve-operations, namely PA and PD, as evident from their mathematical expressions given in Eqs. $(A.2)$ and $(A.3)$ above, must iteratively calculate λ and depending on r value, which is the binary length of the extremely large integer k. Given that in each computation for λ expression one field inversion operation must be performed, this is exactly how the scalar multiplication and field inversion operations are connected with each other in ECC-based cryptographic applications.

Appendix B: Reduced-Latency Inversion

Appendix B is mainly included herein to present our proposed reduced-latency field inversion algorithm along with its associated architecture (functional block diagram). Our algorithm (or architecture) is basically based on ITA algorithm, except that it is designed to reduce the latency in computing field inversion. Significant latency reduction is achieved using our algorithm, which is more apparent in the worst cases in ITA algorithm.

It is known in the literature, and previously mentioned by the academic researchers [49, 50] that it is difficult to parallelize ITA expression [see Chapter 3.1.1, Eq. (3.5)]. This is the result of its cumulative nature, whereby previous calculation steps must be ready and available in order to proceed with the following calculations. Our reduced-latency field inversion algorithm (see Algorithm 12) proposed mainly in binary extension fields $GF(2^m)$, is a parallel version of ITA expression. It is included herein to refute the previous claims issued by other authors.

From Algorithm 12 above, assuming that both **for** loops are executed simultaneously, the inverse of a nonzero element $\beta \in GF(2^m)$ is obtained at the q^{th} iterate, which is the time for $\ell(m-1)$ extension field multiplications, whereby $\ell(.)$ is the binary-length of the argument. This is achieved by running the two processes in tandem (*or running a dual-core processor in hardware-mapping*), whereby the first

Algorithm 12 Reduced-Latency Inversion Algorithm in $GF(2^m)$

```
Input: $\beta \in GF^*(2^m)$, extension degree $m$, $(m-1)$ as $(1m_{q-2}\cdots m_1 m_0)_2$
Output: $b^2 = \beta^{-1} \in GF(2^m)$
Initial: $t = [t_0 t_1 \cdots t_{q-2}]$, if $m_0 = 0 : b = 1, else : b = \beta$;
$\delta := \beta$;
**for** $i := 0$ to $q-2$ **do**
  $\delta := \delta \times \delta^{2^{2^i}}$;
  $t_i := \delta$;
**end for**
**for** $i := 1$ to $q-1$ **do**
  **if** $t_{i-1} \neq 0$ **then**
    **if** $m_i = 1$ **then**
      $a := (t_{i-1})^{2^{\sum_{j=0}^{i-1} m_j 2^j}}$;
    **else**
      $a := 1$;
    **end if**
  **end if**
  $b := b \times a$;
**end for**
**return** $b^2$
```

process computes the $\ell(m-1)$-dependent extension field multiplications, and the second process computes the $w(m-1)$-dependent extension field multiplications, whereby $w(.)$ is the Hamming-weight of the argument. Our previous discussion can be clarified by means of the following functional block diagram (see Figure 2).

Since both processes are running simultaneously, our approach renders inverse calculation independent of the time required for computing $w(m-1)$-dependent multiplications for any extension degree m. Therefore, the latency is significantly reduced to a value equal to the time of $\ell(m-1)$ extension field multiplications.

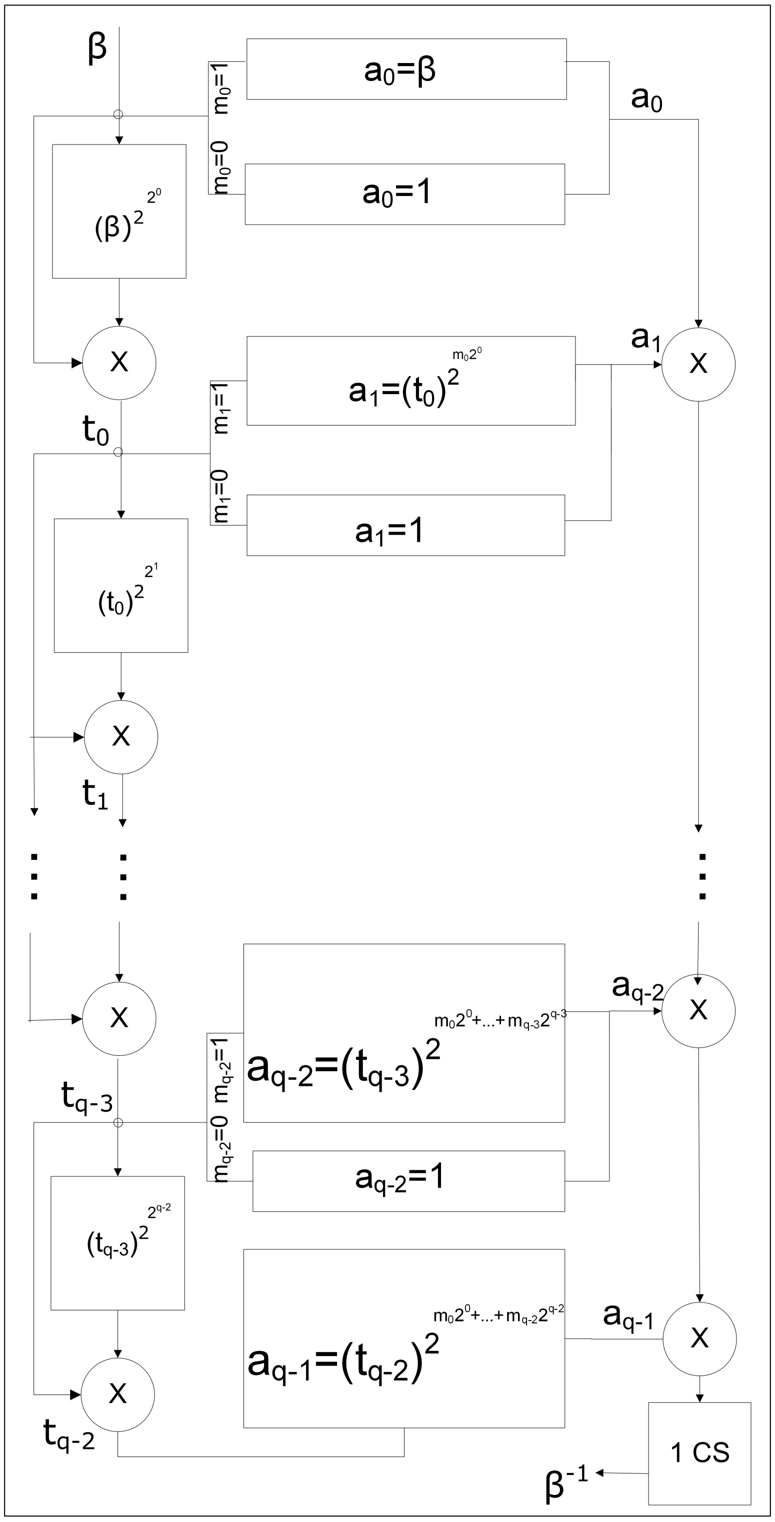


Figure 2: Reduced-Latency Architecture

However, the total multiplications required for inversion is same that required by ITA algorithm.

To show how field inversion is computed using our proposed algorithm, it is useful to consider the following example. Given that the extension degree $m = 16$ and a nonzero element $\beta \in GF(2^{16})$ in which its inverse is required, the computation will proceed as in the following figure (see Figure 3). Despite the use of 6 extension field multipliers, the latency of our proposed architecture is equivalent to the time of 4 extension field multiplications. This is because the multipliers pointed to by the curved-lines (2,3) in Figure 3 are running in parallel.

For this particular example, the latency of ITA inversion algorithm is equivalent to the time of 6 extension field multiplications, with the remark that, the latency of of ITA inversion algorithm grows logarithmically with consideration for the extra execution time incurred by $w(m-1)$ term, while it grows logarithmically in our proposed reduced-latency inversion algorithm without such a consideration.

Our proposed field inversion algorithm achieve its utmost competitive advantage, in terms of latency reduction, in comparison with ITA inversion algorithm for the extension degrees ms for which their binary representation is all ones, i.e., when $(m-1) = 2^e - 1$ for a positive integer e (or equivalently, when $m = 2^e$). Such extension degrees ms are known to be the worst cases in using ITA inversion algorithm, this is because the required extension field multiplications necessary for inversion and the associated latency are maximal, and are given by $(2e-2)$ value.

In summary, our proposed field inversion algorithm is associated with latency that is significantly reduced to the time of $\ell(m-1)$ multiplications, which is the number of extension field multiplications necessary for inversion expressed in terms of the binary-length of $(m-1)$. Unlike ITA inversion algorithm, which has latency that requires in addition to $\ell(m-1)$ value, an extra execution time of $w(m-1)$ multiplications, which is the number of the extra extension field multiplications necessary for inversion expressed in terms of the Hamming-weight of $(m-1)$. Our previous result of latency reduction is applicable to any binary extension field $GF(2^m)$. The maximum value for latency reduction is achieved in using our proposed algorithm in the worst cases in ITA algorithm, which reach a reduction of up to 50% in such cases.

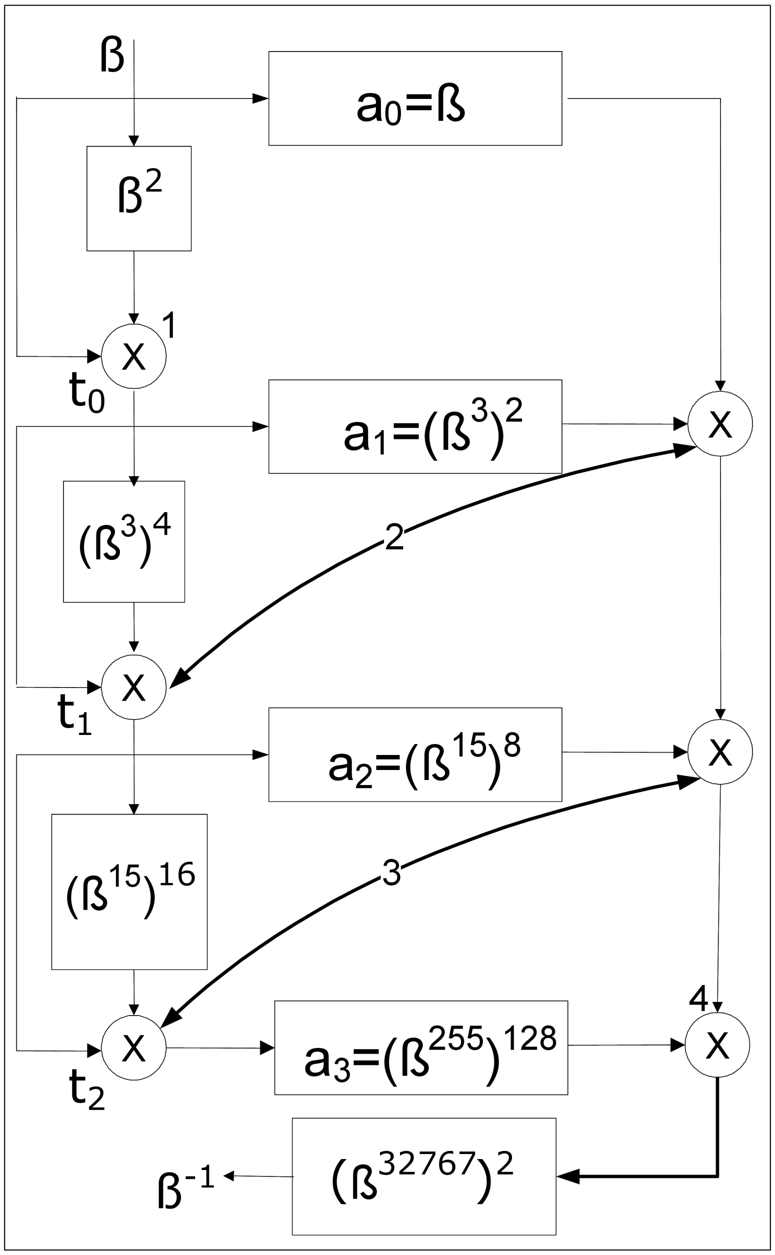


Figure 3: Reduced-Latency Example

Bibliography

[1] V. Miller, "Use of Elliptic Curves in Cryptography," *Advances in Cryptology*: Crypto. 85, Springer-Verlag Inc., London, UK, pp. 417-426, 1985.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, issue 48, pp. 203-209, 1987.

[3] R. Rivest, A. Shamir and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," *Communications of the ACM Journal*, volume 21, pp. 120-126, 1978.

[4] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag Inc., New York, 2004.

[5] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and Their Applications*, University Press, Cambridge 1994.

[6] E. Savas and C. Koç, "Finite Field Arithmetic for Cryptography," *IEEE Journals, Circuits and Systems Magazine*, vol. 10, issue 2, pp. 40-56, Feb. 2010.

[7] B. Ansari and M. Hasan, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," *IEEE Transactions on Computers*, vol. 57, issue 11, pp. 1443-1453, Nov. 2011.

[8] S. Fenn, M. Benaissa and D. Taylor, "Finite field inversion over the dual basis," *IEEE Transactions on VLSI Systems*, vol. 4, issue 1, pp. 134-137, Jan. 1996.

[9] A. Reyhani-Masoleh, "Efficient algorithms and architectures for field multiplication using Gaussian normal bases," *IEEE Transctions on Computers*, vol. 55, issue 1, pp. 34-47, January 2006.

[10] Robert J. Bond and William J. Keane, *An Introduction to Abstract Mathematics*, Gray W. Ostedt, 1999.

[11] M. Prabu, and R. Shanmugalakshmi, "A Comparative and Overview Analysis of Elliptic Curve Cryptography over Finite Fields," International Conference on Information and Multimedia Technology, pp. 495-499, 2009.

[12] S. Kartalopoulos, "A primer on cryptography in communications," *IEEE Communications Magazine*, vol. 44, issue 4, pp. 146-151, Apr. 2006.

[13] O. Pretzel, *Error Correcting Codes and Finite Fields*, Oxford, England, UK: Oxford University Press, 1996.

[14] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis," *Information and Computing Journal*, vol. 78, issue 7, pp. 171-177, Jul. 1988.

[15] T. Chang, E. Lu, Y. Lee, Y. Leu and H. Shyu, "Two Algorithms for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Basis," *accepted by Information Processing Letters Journal*, 1998.

[16] N. Takagi, J. Yoshiki and K. Takagi, "A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis" *IEEE Transactions on Computers*, vol. 50, issue 5, May 2001.

[17] Y. Li, G. Chen, Y. Chen and J. Li, "An Improvement of TYT Algorithm for $GF(2^m)$ Based on Reusing Intermediate Computation Results," *Communications in Mathematical Sciences Journal*, International Press of Boston. vol. 9, issue 1, pp. 277-287, Jan. 2011.

[18] C. Wang, T. Truong, H. Shao, L. Deutsch, J. Omura and I. Reed, "VLSI Architecture for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. 34, issue 8, pp. 709-716, Aug. 1985.

[19] S. Galbraith, "Supersingular curves in cryptography," 7th International Conference on the Theory and Application of Cryptology and Information Security: Advances in Cryptology, pp. 495-513, Springer-Verlag 2001.

[20] C. Paar, S. Kumar and G. Johann, "Architectural Support for Arithmetic in Optimal Extension Fields," 15th IEEE Intl. Conference on Application-Specific Systems, Architectures and Processors, Sept. 2004.

[21] Baktir and B. Sunar, "Optimal Tower Fields," *IEEE Transactions On Computers*, vol. 53, issue 10, pp. 1231-1243, October 2004.

[22] D. Bailey and C. Paar, "Efficient Arithmetic in Finite Field Extensions with Application in Elliptic Curve Cryptography," *presented at Journal of Cryptology*, pp. 153-176, 2003.

[23] J. Guajardo and C. Paar, "Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes," *Designs, Codes and Cryptography*, volume 25, Number 2, Springer-Verlag, pp. 1573-7586, 2002.

[24] D. Page and N. Smart, "Hardware Implementation of Finite Fields of Characteristic Three," Cryptographic Hardware and Embedded Systems (CHES), ISSN 0302-9743, pp. 529-539, February 2003.

[25] S. Oh, C. Kim, Y. Kim and Y. Park,"Implementation issues for arithmetic over extension fields of characteristic odd," *(English) Korean Communications and Mathematics Society*, vol. 18, No. 1, pp. 159-168, 2003.

[26] K. Harrison, D. Page, and N.P. Smart,"Software Implementation of Finite Fields of Characteristic Three, for Use in Pairing Based Cryptosystems," *LMS Journal of Computation and Mathematics*, vol. 5, No. 1, pp. 181-193, 2002.

[27] I. P1363/D1, Standard specifications for public-key cryptography, draft version 1st ed. "http://grouper.ieee.org/groups/1363/": IEEE standards documents, November 2009.

[28] Ian Stewart, *Galois Theory*, Chapman and Hall/CRC, Ohio, USA, 2ed, 1990.

[29] V. Bailey and C. Paar,"Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms," CRYPTO'98, LNCS 1462, Berlin Heidelberg: Springer-Verlag, pp. 472-485, 1998.

[30] R. Lidl and H. Niederreiter. *Finite Fields*, volume 20 of Encyclopedia of Mathematics and its Applications. Addison-Wesley, Reading, Massachusetts, 1983.

[31] Minglong Qi, Luo Zhong, and Qingping Guo,"On the Field Multiplication Algorithms over Binary Extension Field Using Normal Basis," International Conference on Research Challenges in Computer Science, pp. 157-159, 2009.

[32] H. Fan and Y. Dai, "Normal basis multiplication algorithm for $GF(2^n)$," *IET Journals (Electronics Letters)*, vol. 40, issue 18, pp. 1112-1113, Sep. 2004.

[33] R. Mullin, I. Onyszchuk, S. Vanstone and R. Wilson,"Optimal normal bases in $GF(p^n)$." *Discrete Applied Mathematics*, vol. 22, pp. 149-161, 1989.

[34] R. Dahab, D. Hankerson, F. Hu, M. Long, J. Lpez and A. Menezes,"Software Multiplication Using Normal Bases," *IEEE Transactions on Computers*, vol. 55, issue 8, pp. 974-984, August 2006.

[35] P. Ning and Y. Yin,"Efficient software implementation for finite field multiplication in normal basis," Information and Communications Security (LNCS 2229), pp. 177189, 2001.

[36] A. Karatsuba and Y. Ofman, Multiplication of Multidigit Numbers on Automata. Doklady Akademii Nauk SSSR, 145(2):293-294, 1962. English translation in Soviet Physics-Doklady, 7(7):595-596, 1963.

[37] K. Kobayashi and N. Takagi, "A Combined Circuit for Multiplication and Inversion in $GF(2^m)$," *IEEE Transactions on Circuits and Systems*, volume 55, issue 11, pp. 1144-1148, 2008.

[38] K. Kobayashi, N. Takagi and K. Takagi, "An Algorithm for Inversion in $GF(2^m)$ Suitable for Implementation Using a Polynomial Multiply Instruction on $GF(2)$," 18th IEEE Symposium on Computer Arithmetic, 2007. ARITH '07', pp. 105-112, 2007.

[39] Q. Deng, X. Bai, L. Guo and Y. Wang, "A fast hardware implementation of multiplicative inversion in $GF(2^m)$," Asia Pacific Conference on Postgraduate Research in Microelectronics & Electronics, pp. 472-475, Jan. 2009.

[40] M. Jing, J. Chen, Z. Chen and Y. Chen; , "Low Complexity Architecture for Multiplicative Inversion in $GF(2^m)$," IEEE Asia Pacific Conference on Circuits and Systems (APCCAS 2006), pp. 1492-1495, 2006.

[41] J. Massey and J. Omura, "Computational Method and Apparatus for Finite Field Arithmetic," US4587627 Patent Application, submitted 1981.

[42] F. Dong and Y. Li, "A Novel Shortest Addition Chains Algorithm Based on Euclid Algorithm," 4^{th} International Conference on Wireless Communications, Networking and Mobile Computing, pp. 1-4, Oct. 2008.

[43] N. Cruz-Cortes, F. Rodriuez-Henriquez and C. Coello, "An Artificial Immune System Heuristic for Generating Short Addition Chains," *IEEE Transactions on Evolutionary Computation*, vol. 12, issue 1, pp. 1-24, Jan. 2008.

[44] R. Granger, D. Page and M. Stam, "Hardware and software normal basis arithmetic for pairing-based cryptography in characteristic three," *IEEE Transactions on Computers*, vol. 54, issue 7, pp. 852-860, July 2005.

[45] A. Reyhani-Masoleh and M. Hasan, "Fast normal basis multiplication using general purpose processors," *IEEE Transactions on Computers*, vol. 52, issue 11, pp. 1379-1390 , Nov. 2003.

[46] F. Rodrıguez, N. Saqib and N. Cruz, "A fast implementation of multiplicative inversion over $GF(2^m)$," in International Symposium on Information Technology, Las Vegas, Nevada, U.S.A., vol. 1, pp. 574-579, Apr. 2005.

[47] T. Kobayashi, H. Morita, K. Kobayashi and F. Hoshino,"Fast Elliptic Curve Algorithm Combining Frobenius Map and Table Reference to Adapt to Higher Characteristic," EUROCRYPT Proceedings, pp. 176-189, 1999.

[48] P. Mihailescu,"Medium Galois Fields, their Bases and Arithmetic," Submissions to IEEE P1363a: Additional number-theoretic algorithms, February 2000.

[49] S. Morioka and Y. Katayama, "$O(\log_2 m)$ Iterative algorithm for multiplicative inversion in $GF(2^m)$," IEEE International Symposium on Information Theory, 2000. Proceedings, Sorrento , Italy. pp. 449, June 2000.

[50] A. Dinh, R. Bolton and R. Mason, "A low latency architecture for computing multiplicative inverses and divisions in $GF(2^m)$," *IEEE Trans. on Cts. and Syts. II*: Analog and Digital Signal Procs., vol. 48, issue 8, pp. 789-793, Aug. 2001.

Vita Auctoris

Walid Mahmoud has lived in Ontario, Canada since late 2000, who is currently a Canadian citizen. In 2004, he graduated from the University of Western Ontario in London, Ontario, Canada, where he obtained a Bachelor of Engineering Science (BESc) degree in Computer Engineering with Honors from the department of Electrical and Computer Engineering.

In 2007, he graduated from the University of Windsor in Windsor, Ontario, Canada, where he obtained a Master of Science (MASc) degree in Wireless Communications Engineering with High Honors from the department of Electrical and Computer Engineering.

In 2011, he graduated from the University of Windsor in Windsor, Ontario, Canada, where he obtained a Doctor of Philosophy (PhD) degree in Cryptography and Network Security with High Honors from the department of Electrical and Computer Engineering.

Currently, he is looking for an academic job in a well-recognized university, in any part of the world, as long as the offer is interesting, and the required skills somehow are relevant to his educational and academic credentials.