ELLIPTIC CURVE CRYPTOSYSTEM OVER OPTIMAL EXTENSION FIELDS FOR COMPUTATIONALLY CONSTRAINED DEVICES

by

Adnan Mohammed I. Abu Mahfouz

Submitted in partial fulfillment of the requirements for the degree

Master of Engineering (Computer Engineering)

in the

Faculty of Engineering, Built Environment and Information Technology

UNIVERSITY OF PRETORIA

PRETORIA

September 2004

ELLIPTIC CURVE CRYPTOSYSTEM OVER

OPTIMAL EXTENSION FIELDS FOR

COMPUTATIONALLY CONSTRAINED DEVICES

by

Adnan Mohammed I. Abu Mahfouz

Leader: Prof. G. P. Hancke

Department: Electrical, Electronic and Computer Engineering

Degree: Master of Engineering (Computer Engineering)

SUMMARY

Data security will play a central role in the design of future IT systems. The PC has been a major driver of the digital economy. Recently, there has been a shift towards IT applications realized as embedded systems, because they have proved to be good solutions for many applications, especially those which require data processing in real time. Examples include security for wireless phones, wireless computing, pay-TV, and copy protection schemes for audio/video consumer products and digital cinemas. Most of these embedded applications will be wireless, which makes the communication channel vulnerable.

The implementation of cryptographic systems presents several requirements and challenges. For example, the performance of algorithms is often crucial, and guaranteeing security is a formidable challenge. One needs encryption algorithms to run at the transmission rates of the communication links at speeds that are achieved through custom hardware devices.

Public-key cryptosystems such as RSA, DSA and DSS have traditionally been used to accomplish secure communication via insecure channels. Elliptic curves are the basis for a relatively new class of public-key schemes. It is predicted that elliptic curve cryptosystems (ECCs) will replace many existing schemes in the near future. The main reason for the attractiveness of ECC is the fact that significantly smaller parameters can be used in ECC than in other competitive system, but with equivalent levels of security. The benefits of having smaller key size include faster computations, and reduction in processing power, storage space and bandwidth. This makes ECC ideal for constrained environments where resources such as power, processing time and memory are limited.

The implementation of ECC requires several choices, such as the type of the underlying finite field, algorithms for implementing the finite field arithmetic, the type of the elliptic curve, algorithms for implementing the elliptic curve group operation, and elliptic curve protocols. Many of these selections may have a major impact on overall performance.

In this dissertation a finite field from a special class called the Optimal Extension Field (OEF) is chosen as the underlying finite field of implementing ECC. OEFs utilize the fast integer arithmetic available on modern microcontrollers to produce very efficient results without resorting to multiprecision operations or arithmetic using polynomials of large degree. This dissertation discusses the theoretical and implementation issues associated with the development of this finite field in a low end embedded system. It also presents various improvement techniques for OEF arithmetic.

The main objectives of this dissertation are to

- Implement the functions required to perform the finite field arithmetic operations.
- Implement the functions required to generate an elliptic curve and to embed data on that elliptic curve.
- Implement the functions required to perform the elliptic curve group operation.

All of these functions constitute a library that could be used to implement any elliptic curve cryptosystem. In this dissertation this library is implemented in an 8-bit AVR Atmel microcontroller.

KEYWORDS

Elliptic Curve, Cryptography, ECC, Public-key, Finite field, Optimal Extension Field, OEF, Diffie-Hellman, ElGamal, ECDH, Discrete Logarithm Problem, ECDLP, Itoh Tsujii Inversion, Frobenius, Karatsuba algorithm, Extended Euclidean algorithm, Schoolbook method, Addition Chain algorithm, Non-Adjacent Form, Quadratic Residue, Legendre Symbol, Embedded System.

‘N KRIPTOGRAFIESE STELSEL GEBASEER OP ELLIPTIESE KROMMES OOR OPTIMALE UITGEBREIDE VELDE VIR VERWERKERS MET BEPERKTE REKENKUNDIGE VERMOËNS

deur

Adnan Mohammed I. Abu Mahfouz

Leier: Prof. G. P. Hancke

Departement: Elektriese, Elektroniese en Rekenaar-ingenieurswese

Graad: Meester in Ingenieurswese (Rekenaar-ingenieurswese)

OPSOMMING

Datasekuriteit sal ‘n kernrol in die ontwerp van toekomstige inligtingstegnologie-stelsels speel. Die persoonlike rekenaar was ‘n belangrike dryfkrag agter die digitale ekonomie. Daar was die afgelope tyd ‘n verskuiwing na IT-toepassings wat as ingebedde stelsels gerealiseer word, omdat daar bewys is dat hulle goeie oplossings bied vir vele toepassings, veral die wat intydse dataverwerking verg. Voorbeelde hiervan sluit sekuriteit vir draadlose telefone, draadlose berekening, betaal-TV en kopieerbeskermingskemas vir oudio-/videoverbruikerprodukte en syferkameras in. Die meeste van hierdie ingebedde toepassings sal draadloos wees, wat die kommunikasiekanaal nog kwesbaarder maak.

Die implementering van kriptografiese stelsels bied verskeie vereistes en uitdagings. Die werkverrigting van algoritmes is dikwels kritiek en om sekuriteit te waarborg is ‘n enorme uitdaging. ‘n Mens het enkripsie-algoritmes nodig wat teen die transmissietempo’s van die kommunikasieskakels fuksioneer teen snelhede wat deur doelgemaakte hardewaretoestelle bereik word.

Publiekesleutel-kriptostelsels soos RSA, DSA en DSS is voorheen gebruik om kommunikasie oor onveilige kanale te beveilig. Elliptiese krommes is die grondslag vir ‘n betreklik nuwe klas publiekesleutel-skemas. Daar word voorspel dat elliptiesekromme-kriptostelsels (ECC) baie bestaande skemas in die nabye toekoms sal vervang. Die belangrikste rede vir die aantreklikheid van ECC is die feit dat aansienlik kleiner parameters in ECC as in ander mededingende stelsels gebruik kan word, maar met gelyke sekuriteitsvlakke. Die voordele om ‘n kleiner sleutelgrootte te hê sluit vinniger berekeninge en vermindering van verwerkingkrag, bergruimte en bandwydte in. Dit maak ECC ideaal vir omgewings waar hulpbronne soos krag, verwerkingtyd en geheue beperk is.

Die implementering van ECC benodig verskeie keuses soos die tipe onderliggende eindige veld, algoritmes vir die implementering van die eindigeveld-rekenkunde, die tipe elliptiese kromme, algoritmes vir die implementering van die elliptiesekromme-groepwerking en elliptiesekromme-protokolle. Baie van hierdie keuses kan ‘n belangrike impak op die algehele werkverrigting hê.

In hierdie verhandeling word ‘n eindige veld uit ‘n spesiale klas bekend as Optimale Uitbreidingveld (OEF) gekies as die onderliggende eindige veld om ECC te implementeer. OEFs benut vinnige integraalrekenkunde wat op moderne mikrobeheerders beskikbaar is om baie doeltreffende resultate te lewer sonder om in ‘n groot mate staat te maak op multipresisiewerkinge of rekenkunde wat polinomiale gebruik. Hierdie verhandeling bespreek die teoretiese en implementeringvraagstukke wat met die ontwikkeling van hierdie eindige veld in ‘n lae-end-ingebedde stelsel verband hou. Dit bied ook verskeie verbeteringtegnieke vir OEF-rekenkunde.

Die hoofoogmerke van hierdie verhandeling is om

- Die funksies te implementeer wat nodig is om die eindigeveld-rekenkundige werkinge uit te voer.

- Die funksies te implementeer wat nodig is om ‘n elliptiese kromme te genereer en om data op daardie elliptiese kromme in te bed.

- Die funksies in werking te stel wat nodig is om die elliptiesekromme-groepwerking uit te voer.

Al hierdie funksies maak ‘n biblioteek uit wat gebruik kan word om enige elliptiesekromme-kriptostelsel te implementeer. In hierdie verhandeling word hierdie biblioteek in ‘n 8-bit AVR Atmel-mikrobeheerder geïmplementeer.

SLEUTELWOORDE

Elliptiese Kromme, Kriptografie, ECC, Publieke Sleutel, Eindige Veld, Optimale Uitbreidingveld, OEF, Diffie-Hellman, ElGamal, ECDH, Diskrete Algoritmeprobleem, ECDLP, Itoh Tsujii Inversie, Frobenius, Karatsuba-algoritme, Uitgebreide Euklidiese algoritme, Skoolboekmetode, Byvoegingkettingalgoritme, Nieaangrensende Vorm, Kwadratiese Res, Legendre-simbool, Ingebedde Stelsel.

ACKNOWLEDGEMENT

Firstly, I would like to thank God for help and support, He always grants me, which has made me work hard to complete this dissertation.

I would like to express my heartfelt thanks to Prof. G. P. Hancke, even though they are not enough; because he was not just a supervisor that guided and advised me throughout the progress of this dissertation but he was always ready to help and support me in everything that I needed in my work in this dissertation. I really appreciate his help.

I would like to thank Prof WT. Penzhorn for his helpful suggestions at the beginning of this dissertation.

I am grateful to Dr. Mike Rosing from University of Wisconsin in Madison, who has always replied to my emails and answered my questions about elliptic curves.

I would like to thank Mr. Patrick Lenahan for his time that he spent in editing this dissertation.

I would like to thank Mrs. Mari Ferreira and my colleagues Noel Roberts and Alan Huang for their friendship and encouragement.

I would like to thank all the friends and people who have helped me in the research, the implementation and writing of this dissertation, whether it be direct assistance or in the form of moral support.

Lastly, and not least, I would like to dedicate this dissertation to my family. Although they were so far from me they were always close to my heart. I would like to thank them so much because they have done more than I care to mention here, and because without their unconditional love, help and support, this work would not have been possible.

To My Family… … …

LIST OF ABBREVIATIONS

ACC	Accumulated value
ADC	Analog-to-Digital Converter
AES	Advanced Encryption Standard
ALU	Arithmetic Logic Unit
BBE	Balanced Binary Expansion
CISC	Complex Instruction Set Computer
CMOS	Complementary Metal Oxide Semiconductor
CPU	Central Processing Unit
DES	Data Encryption Standard
DLP	Discrete Logarithm Problem
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
EC	Elliptic Curve
ECC	Elliptic Curve Cryptosystem
ECDH	Elliptic Curve Diffie-Hellman
ECDLP	Elliptic Curve Discrete Logarithm Problem
ECDSA	Elliptic Curve Digital Signature Algorithm
EEPROM	Electrically Erasable Programmable Read-Only Memory
GCC	GNU C Compiler
GNU	Gnu's Not Unix
IDEA	International Data Encryption Algorithm
ITI	ITOH - TSUJII INVERSION
JTAG	Joint Test Action Group
KA	Karatsuba Algorithm
MVL	Multi-Value Logic
NAF	Non-Adjacent Form
OEF	Optimal Extension Field

PK	Public Key
PWM	Pulse Width Modulation
RISC	Reduced Instruction Set Computer
RSA	Rivest, Shamir and Adleman
RTC	Real Time Counter
SPI	In-System Programming
SRAM	Static Random Access Memory
USART	Universal Synchronous/Asynchronous Receiver/Transmitter

CONTENTS

Chapter 1

INTRODUCTION

1.1 CRYPTOGRAPHY

The fundamental aim of cryptography has always been to provide secure communications over an insecure channel, so that only a set of intended recipients can understand the message. Cryptography draws from a broad range of sciences, including mathematics, computer science, information theory and human psychology. With the growing rate of electronic commerce and information exchange, cryptography is gaining in importance and getting attention from corporations, governments and individuals alike.

Cryptography involves the study of mathematical techniques that allow the practitioner to provide the following security services [1]:

Confidentiality

A service used to keep the content of information accessible to only those authorized to have it.

Integrity

A service that requires the computer system assets and transmitted information be capable of modification only by authorized users.

Authentication

A service that is concerned with assuring that the origin of a message is correctly identified.

Non-repudiation

A service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions.

Availability

A measure of the ability of the system to function efficiently in providing security.

These security services are provided by using cryptographic algorithms. There are two major classes of algorithms in cryptography: Private-key or Symmetric-key algorithms and Public-key algorithms.

1.1.1 Symmetric-key Algorithms

Private-key or Symmetric-key algorithms are algorithms where the encryption and decryption key are the same. The main function of these algorithms is encryption of data.

There are two types of symmetric-key algorithms, which are commonly distinguished: block ciphers and stream ciphers. Block ciphers are encryption schemes in which the message is broken into strings (called blocks) of fixed length and encrypted one block at a time. Examples include the Data Encryption Standard (DES) [2], the International Data Encryption Algorithm (IDEA) [3], and the Advanced Encryption Standard (AES) [4]. Stream ciphers operate on a single bit of plaintext at a time. They are useful because the encryption transformation can change for each symbol of the message being encrypted.

1.1.2 Public-key Algorithms

Public-key (PK) cryptography is based on the idea of separating the key used to encrypt a message from the one used to decrypt it. In general, one can divide practical public-key algorithms into three families:

Algorithms based on the integer factorization problem

Given a positive integer n, find its prime factorization, e.g. RSA [5].

Algorithms based on the discrete logarithm problem

Given y and q find x such that $y = q^x \bmod p$, the Diffie-Hellman [6] key exchange protocol is based on this problem as well as many other protocols, including the Digital Signature Algorithm (DSA) [1, 7].

Algorithms based on Elliptic Curves

Elliptic curve cryptosystems are the most recent family of practical public-key algorithms, but are rapidly gaining acceptance. Due to their reduced processing needs, elliptic curves are especially attractive for embedded applications.

Despite the differences between these mathematical problems, all three algorithm families have something in common: they all perform complex operations on very large numbers, typically 1024-2048 bits in length for the RSA and discrete logarithm systems or 160-256 bits in length for the elliptic curve systems. Since elliptic curves are somewhat less computationally intensive than the other two algorithm families, they seem especially attractive for embedded applications.

Public-key cryptosystems solve in a very elegant way the key distribution problem of symmetric-key schemes. However, PK systems have a major disadvantage: public-key algorithms are very arithmetic intensive. Hence, in practice, cryptographic systems are a mixture of symmetric-key and public-key cryptosystems. Usually, a public-key algorithm is chosen for key establishment and authentication through digital signatures, and then a symmetric-key algorithm is chosen to encrypt the communications and the data transfer, achieving in this way high throughput rates. Additional information on cryptography can be found in [1, 7].

1.2 EMBEDDED SYSTEM[1]

An embedded system is a specialized computer system that is part of a larger system or machine. Typically, an embedded device is housed on a single microprocessor board with the programs stored in ROM. Virtually all appliances that have a digital interface (watches, microwaves, VCRs, cars) utilize embedded systems. Some embedded systems include an operating system, but many are so specialized that the entire logic can be implemented as a single program [8].

Technological improvements have made possible the development of powerful, low cost and low power microprocessors. Embedded systems are quickly becoming ubiquitous in our daily life for keeping track of our addresses, appointments, messages, and so on. The next revolution after the Internet is expected to be driven by Embedded System applications. Embedded processors are already an integral part of many communications devices and their importance will continue to increase.

Embedded devices are very different from PCs from the computational resources and memory availability point of view. In addition, embedded systems are usually designed to consume small amounts of energy. Despite these constraints, one should be able to run the same types of applications that run today in a fast computer. These devices become increasingly connected; the communication channels between them remain highly insecure. Thus the existing situation demands the use of cryptography, but does not possess quite enough power to run the traditional algorithms, which are computationally intensive by design, in a reasonable amount of time. Constrained environments present a difficult challenge, where every possible optimization must be used simply to enable the very use of security, and to obtain acceptable levels of performance.

8-bit microcontrollers are one example of embedded device used for extremely low-power and low-cost applications. The contribution of this dissertation deals with the implementation of Elliptic Curve Cryptosystems (ECC) on such 8-bit microcontrollers, and tries to achieve the

[1] In this dissertation "embedded system" will be used to refer to a low end embedded system with limited resources.

following objectives for the implementation of ECC: low code size, acceptable level of security, and acceptable speed of execution.

1.3 WHY ELLIPTIC CURVE CRYPTOSYSTEMS?

The use of elliptic curves in public key cryptography was first proposed independently by Koblitz [9] and Miller [10] in 1980s. Since then, elliptic curve cryptography has been the focus of a lot of attention and gained great popularity due to the identical level of security they provide but with much smaller key sizes than conventional public key cryptosystems have.

Elliptic curve cryptosystems are based on the group of points on an elliptic curve over a finite field. They rely on the complexity of finding the value of a scalar, given a point and the scalar multiple of that point. This corresponds to solving the Discrete Logarithm Problem (DLP). However, it is more difficult to solve the Elliptic Curve Discrete Logarithm Problem (ECDLP) than its original counterpart. Thus, elliptic curve cryptosystems provide equivalent security compared to traditional public key cryptosystems such as RSA and discrete logarithm based systems like ElGamal and Diffie-Hellman algorithms, but with much smaller key sizes and computationally more efficient algorithms. Therefore they have smaller bandwidth and memory requirements, which makes them extremely desirable for embedded systems, where resources such as power, processing time and memory are limited for example smart cards, as well as use on personal computers and workstations.

An elliptic curve cryptosystem relies on the assumed complexity of the Elliptic Curve Discrete Logarithm Problem (ECDLP) for its security. An instance of the ECDLP is posed for an elliptic curve defined over a finite field. The rule to perform the elliptic curve group operation can be expressed in terms of arithmetic operations in the finite field; thus the speed of the field arithmetic determines the speed of the cryptosystem.

Traditionally, ECCs have been developed over finite fields which have either prime order (greater than 3), or characteristic 2. Prime fields have the advantages of using integer operations, whereas binary extension fields can use the exclusive-or (XOR) and shift operations instead of addition and multiplication respectively, which lead to significant improvements in speed. These fields have the following advantages:

- All subfield arithmetic is integer arithmetic and does not require any multi precision calculations.

- The polynomial degree m is very small for all secure fields, so that the number of subfield calculations is minimized.

Efficient algorithms for finite field operations have been closely investigated. Besides the standard basis, alternative representations such as the normal bases and dual bases have been studied [11]. Optimal Extension Fields (OEFs) have been found to be especially successful in embedded software implementations of elliptic curve schemes. The arithmetic operations in OEFs are much more efficient than in characteristic two extensions or prime fields, due to the use of a large characteristic ground field and the selection of a binomial as the field polynomial.

In the elliptic curve scalar point multiplication, a large number of field multiplications and inversions are computed. Inversion is inherently more complex and at least several times more costly than multiplication. The adaptation of Itoh-Tsujii method [12] for standard basis, particularly for Optimal Extension Fields, has been effective in achieving fast inversion.

In this dissertation, OEFs over a finite field with prime power order, that is $GF(p^m)$ where p is an odd prime and m is a positive integer, are implemented to achieve the above advantages

1.4 CONTRIBUTION OF THE DISSERTATION

An elliptic curve cryptosystem relies on the assumed complexity of the Elliptic Curve Discrete Logarithm Problem (ECDLP) for its security. An instance of ECDLP is posed for an elliptic curve defined over an extension finite field $GF(p^m)$ for p a prime and m a positive integer. The rule to perform the elliptic curve group operation can be expressed in terms of arithmetic operations in the finite field; thus the speed of the field arithmetic determines the speed of the cryptosystem.

An elliptic curve cryptosystem can be represented by three layers as depicted in figure (1.1). The first layer represents the underlying finite field, which consists of arithmetic operations required to perform elliptic curve group operation and to implement any elliptic curve cryptosystem. This layer can be considered as a core of the entire system. The second layer represents the elliptic curve, which consists of the elliptic curve group operation. The external layer is the cryptography layer, in which any elliptic curve cryptographic algorithm can be implemented.

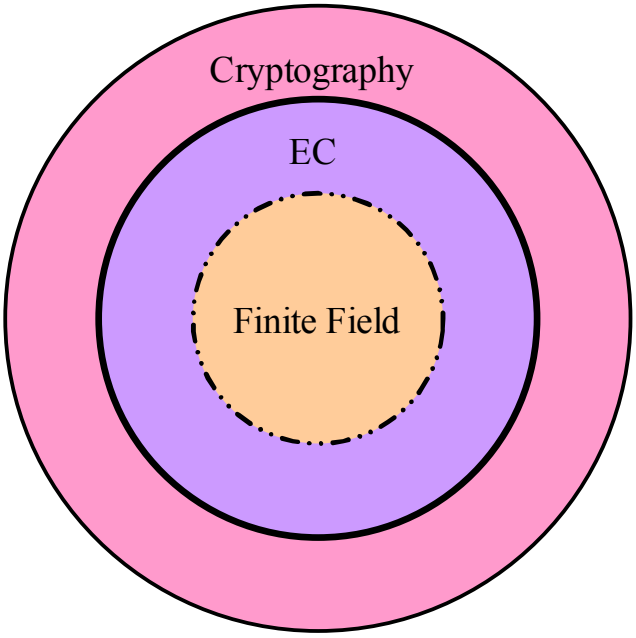


Figure 1.1 Elliptic Curve Cryptosystem Structure

The contribution of this dissertation is to discuss the theoretical issues of ECC and to implement the first two layers in a low end embedded system, which can be considered as a library that contains all functions and definitions required to implement any elliptic curve cryptosystem such as ECDH, ECDSA, and other cryptosystems in a low end embedded system.

In this dissertation, a special type of the extension finite field will be used; this field is named Optimal Extension Field (OEF). Efficient algorithms will be used for performing finite field operations in OEFs. In order to practically verify the theoretical algorithms discussed, a complete library of the first two layers was implemented on an AVR Atmel microcontroller for a medium sized field whose elements can be represented with 134 bits.

1.5 THE DISSERTATION OUTLINE

This dissertation consists of 10 chapters, which are organized in such a way that makes it easy to understand the operation of elliptic curve cryptosystem. The structure shown in figure (1.1) is not only useful to achieve efficient implementation of an elliptic curve cryptosystem, but it is also useful to describe this system, and so chapter 5 starts from the center, which is the underlying finite field, then the progress is carried on, in the next chapters, step by step until describe the complete structure.

This dissertation is organized as follows

Chapter 2 starts by giving a brief description of an abelian group and a finite field, then shows how an elliptic curve works over real numbers, and over the different types of finite fields, which are the prime finite field, binary finite field, binary composite finite field, extension finite field and optimal extension finite field. The last section compares these finite fields and describes the advantages and disadvantages of each field. From this section it can be concluded that implementing an elliptic curve over an optimal extension field is consider to be the best choice for embedded system.

Chapter 3 summarizes previous work on elliptic curve cryptosystem and previous implementations of elliptic curve cryptosystems over different finite fields found in the literature. It also summarizes the use of some efficient algorithms used to implement finite field's arithmetic operations, especially finite field inversion, and algorithms used to speed up the scalar multiplication of a point on an elliptic curve.

Chapter 4 describes some definitions and properties of an optimal extension field. Then it describes the primary efficient algorithms that will be used in the finite field arithmetic operations and the elliptic curve group operation and implementation. For example the Schoolbook method and Karatsuba algorithm could be used in finite field multiplication and the Itoh and Tsujii Inversion algorithm to implement finite field inversion.

Chapter 5 gives an elementary introduction to optimal extension finite field. Then it describes the arithmetic operations on an optimal extension field: addition, subtraction, multiplication, squaring, inversion and exponentiation. It shows an efficient algorithm to perform a subfield

reduction. It also describes a method to speed up the multiplication operation by performing a single reduction for each coefficient, instead of performing a subfield reduction after each subfield multiplication. Then it shows how the Itoh and Tsujii Inversion algorithm reduces the problem of extension field inversion to subfield inversion, which can get through table look-up. This chapter describes the first layer of the elliptic curve cryptosystems structure shown in figure (1.1).

Chapter 6 describes the basic method used to compute the multiplication of points on an elliptic curve. This method is called the Addition-Subtraction method, and it is faster than the Binary-Double-and-Add algorithm, which uses binary expansion. The Addition- Subtraction method used the Balanced Binary Expansion that can be found using the Non Adjacent Form method. At the end of this chapter there is a comparison of the complexity of these two methods.

Chapter 7 shows how to embed a random data and a normal message as a point on an elliptic curve. This embedding of data is necessary because an elliptic curve group operation works on points on a curve, and encryption a message using elliptic curve cryptosystems required to embed this message as a point on a curve before starting the encryption process. This chapter presents an efficient method to solve the quadratic equation, which is required to embed data on an elliptic curve. It also describes two types of data conversion: the first one is between an OEF element and a long integer that represented as a string of decimal digits. The second conversion is between an OEF element and an octet string message. This chapter and chapter 6 describe the second layer of the elliptic curve cryptosystem structure shown in figure (1.1).

Chapter 8 gives an introduction to elliptic curve cryptography and a brief description of the elliptic curve discrete logarithm problem. Then it describes two simple cryptosystems that could be implemented in the external layer of the elliptic curve cryptosystem design: the elliptic curve Diffie-Hellman, which is analog to the Diffie-Hellman key exchange protocol, as well as a system analog to the ElGamal cryptosystem. These two cryptosystems and other cryptosystems could be implemented using the library functions defined in the first two layers.

Chapter 9 gives a brief description of a microcontroller that is used in the implementation, and a brief description of the underlying finite field used. Then it presents the design architecture of the software implementation. At the end it summarizes the results and timing of the software implementation.

Finally, chapter 10 summarizes the conclusions gained in this dissertation, and presents a summary of the dissertation's achievements. At the end recommendations are made for future work, which could further enhance OEF and elliptic curve algorithms.

Chapter 2

ELLIPTIC CURVES

2.1 INTRODUCTION

The study of elliptic curves is an important branch of mathematics. Elliptic curves are simple functions that can be drawn as gently looping lines in the (x, y) plane. They have been well studied by mathematicians for many years, which in the latter half of the 20^{th} century has yielded some very significant results. Elliptic Curve Cryptography is just one application of elliptic curve theory.

Elliptic curves can provide versions of public-key methods that, in some cases, are faster and use smaller keys, while providing an equivalent level of security. Their advantage comes from using a different kind of mathematical group for public-key arithmetic. All practical public-key systems today exploit the properties of arithmetic using large finite groups.

An elliptic curve is defined by an equation in 2 variables, with coefficients. For cryptography, the variables and coefficients are restricted to elements in a finite field, which results in the definition of a finite abelian group; a group is a set of elements with custom-defined arithmetic operations on those elements. For elliptic curve groups, these specific operations are defined geometrically. Introducing more stringent properties to the elements of a group, such as limiting the number of points on such a curve, creates an underlying field for an elliptic curve group [13].

2.1.1 Abelian Groups

An abelian group G (or $\{G, \bullet\}$) is a set of elements with a binary operation, denoted by $\bullet$, satisfying the following axioms, [1]:

Closure: $\forall\, a, b \in G, \quad \exists (a \bullet b) \in G$.

Associative: $a \bullet (b \bullet c) = (a \bullet b) \bullet c, \quad \forall\, a, b \text{ and } c \in G$.

Identity element: $\forall\, a \in G, \quad \exists\, e \in G$ such that

$$a \bullet e = e \bullet a = a .$$

Inverse element: $\forall\, a \in G, \quad \exists\, a' \in G$ such that

$$a \bullet a' = a' \bullet a = e .$$

Commutative: $\forall\, a, b \in G, \quad a \bullet b = b \bullet a$.

2.1.2 Finite Field

A finite field F (or $\{F, +, \times\}$) is an algebraic system consisting of a finite set of elements with two binary operations called addition (+) and multiplication ($\times$), satisfying the following axioms:

- F is an abelian group with respect to (+).
- $F \setminus \{0\}$ is an abelian group with respect to ($\times$).
- Distributive: $\forall\, a, b, c \in F$

$$a \times (b + c) = (a \times b) + (a \times c)$$

$$(a+b)\times c=(a\times c)+(b\times c).$$

The order of a finite field is the number of elements in the field. There exists a finite field of order q if and only if q is a prime power (i.e. $q=p^m$, where p is a prime number and m is a positive integer), the finite field of order p^m is generally written[2] $GF(p^m)$ [14]. p is called the characteristic of the finite field $GF(p^m)$. Before looking at this, firstly elliptic curves in which the variables and coefficients are real numbers will be explained; this probably makes it easier to understand the operations over elliptic curves.

2.2 ELLIPTIC CURVES OVER REAL NUMBERS

An elliptic curve over real numbers is defined as the set of points (x, y), which satisfy an elliptic curve equation of the form that mathematicians call the "Weierstrass" form:

$$y^2+a_1xy+a_3y=x^3+a_2x^2+a_4x+a_6. \qquad (2.1)$$

For the purpose of this dissertation it is sufficient to deal with a special case of equation (2.1) which is:

$$y^2=x^3+a_4x+a_6. \qquad (2.2)$$

To plot such a curve, it is required to compute

$$y=\sqrt{x^3+a_4x+a_6}\ . \qquad (2.3)$$

For given values of a_4 and a_6, the plot consists of positive and negative values of y for each value of x. Also included in the definition of an elliptic curve is a single element denoted O and called the point at infinity. Figure (2.1) shows an example of the elliptic curve for $a_4=-7, a_6=5$.

[2] GF stands for Galois field, in honor of the mathematician who first studied finites field GF(p), this symbol will be used in this dissertation.

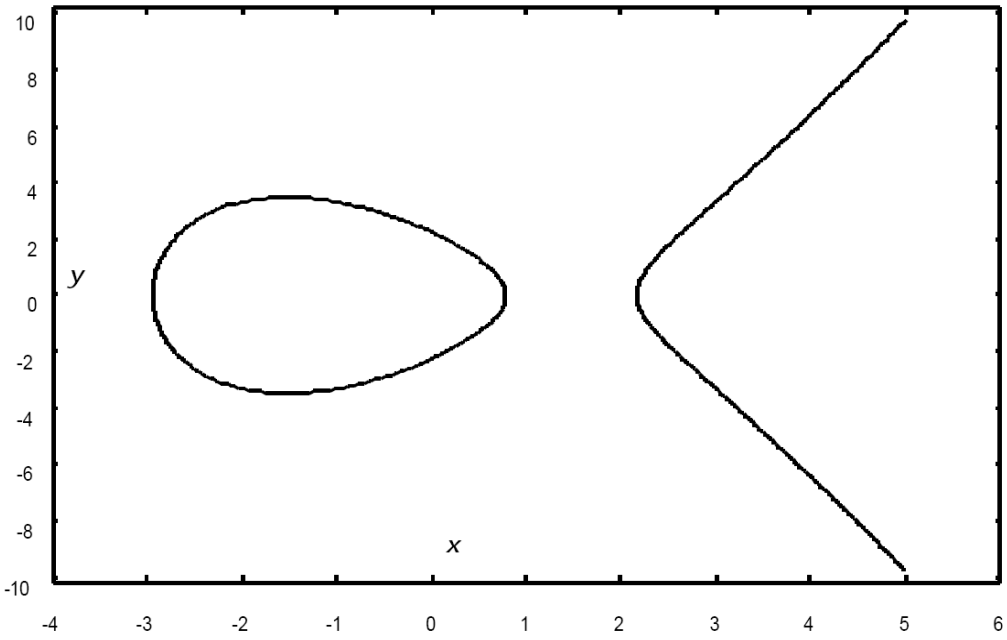


Figure 2.1 Plot of elliptic curve $y^2 = x^3 - 7x + 5$ [15]

2.2.1 Geometric Description of Addition

It can be shown that a group can be defined based on the set[3] $E(a_4, a_6)$ provided that $x^3 + a_4 x + a_6$, has no repeated factors. This is equivalent to the condition

$$4a_4^3 + 27a_6^2 \neq 0. \qquad (2.4)$$

The basic idea is to find a way to define "addition $(+)$" of two points (P, Q) that lie on the curve such that the "sum" is another point on the curve. the rules of addition over an elliptic curve can be defined as described in [16]. Assume that $P \neq O$ and $Q \neq O$:

1. O serves as the additive identity, thus $O = -O$ and $P + O = P$.

[3] The set of points $E(a_4, a_6)$ consist of all of the points (x, y) that satisfy Equation (2.2) together with the element *O*.

2. The negative of a point P is the point with the same x coordinate but the negative of the y coordinate, i.e. if $P=(x,y)$, then $-P=(x,-y)$.

3. If P and Q have different x coordinates, then the line $\overline{PQ}$ intersects the curve in exactly one more point R (unless the line is tangent to the curve at either P or Q, then $R=P$ or $R=Q$) then define $P+Q=-R$. Figure (2.2) illustrates this construction.

4. If $P=-Q$, then define $P+Q=O$.

5. If $P=Q$, draw the tangent line to the curve at point P, and find the intersection point R with the curve; then define $2P=-R$.

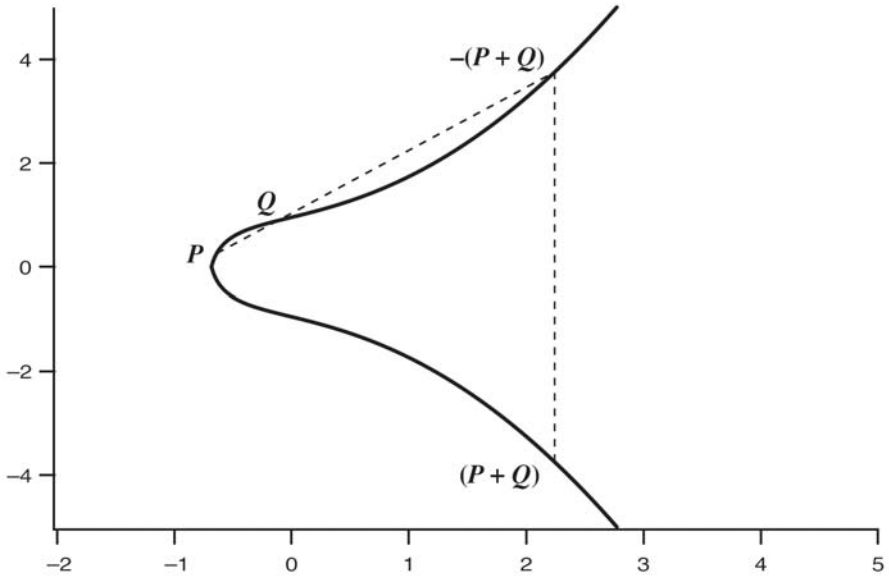


Figure 2.2 Addition of elliptic curve points over a real number [1]

2.2.2 Algebraic Description of Addition

For two distinct points on the curve $P=(x_P,y_P)$ and $Q=(x_Q,y_Q)$ that $P\neq -Q$, the slope of the line that joins them is:

$$\Delta=(y_Q-y_P)/(x_Q-x_P). \qquad (2.5)$$

For derivations of equation (2.5) see [16]. Then the sum $R = P + Q$ can be expressed as follows:

$$x_R = \Delta^2 - x_P - x_Q \qquad (2.6)$$

$$y_R = -y_P + \Delta(x_P - x_R). \qquad (2.7)$$

To double a point P, then $R = 2P$, where $y_P \neq 0$; the expressions are:

$$\Delta = (3x_P^2 + a_4)/2y_P \qquad (2.8)$$

$$x_R = \Delta^2 - 2x_P \qquad (2.9)$$

$$y_R = \Delta(x_P - x_R) - y_P. \qquad (2.10)$$

2.3 ELLIPTIC CURVES OVER FINITE FIELDS

Calculations over the real numbers are slow and inaccurate due to round-off error. Cryptographic applications require fast and precise arithmetic; so what does it have to do with cryptography? It turns out that similar formulas work if replace real numbers with finite fields. The formulas stated previously do not change. But instead of using floating-point arithmetic use a large number and do all calculations modulo a large prime. This method is mentioned in the IEEE 1363a standard [17].

The key to the implementation of ECCs is the selection of elliptic curve groups over the finite field of $GF(p)$ and $GF(p^m)$, where p is a prime and m is a positive integer. By definition, elliptic curve groups are additive groups. Any such field is isomorphic to $GF(p)[x]/P(x)$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x_i, p_i \in GF(p)$, is a monic irreducible polynomial of degree m over $GF(p)$.

Various finite fields admit the use of different algorithms for arithmetic. Unsurprisingly, the choices of p, m, and $P(x)$ can have a dramatic impact on the performance of the ECC. In particular, there are generic algorithms for arithmetic in an arbitrary finite field and there are specialized algorithms which provide better performance in finite fields of a particular form.

The following sections briefly describe field types proposed for ECC, where background reading can be found in [18].

2.3.1 Prime Finite Fields $GF(p)$

For the Galois finite field $GF(p^m)$, of p^m elements, for some prime p and positive integer $m=1$, there is a Galois finite field[4] $GF(p)$, which is called a prime finite field and consists of the set of integers modulo p, which are the all possible results of reduction modulo p:

$$\{0, 1, 2, ..., p-1\}.$$

The arithmetic operations on $GF(p)$ are the usual addition, subtraction and multiplication modulo p.

Prime fields are commonly used for software implementations of elliptic curve cryptosystems since they are based on integer arithmetic which is optimized in modern microprocessors.

For elliptic curve E over a finite field $GF(p)$, equation (2.2) can be used in which the variable and coefficients all take on values in the set of integers modulo p. For some prime number $p \neq 2,3$, equation (2.2) can be rewritten as:

$$y^2 \bmod p = (x^3 + a_4 x + a_6) \bmod p \qquad (2.11)$$

[4] Usually denoted this finite field by Z_p

where the values of a_4 and a_6 satisfy $\left(4a_4^3 + 27a_6^2\right) \bmod p \neq 0$, the rules for addition over $E_p(a_4, a_6)$ are the same for elliptic curves defined over real numbers except that the arithmetic operations are modulo p.

Example 2.1

Let $p = 23$ and consider the elliptic curve $y^2 = x^3 + x + 1$, in this case $a_4 = a_6 = 1$. Table (2.1) lists the points (other than O) that are part of $E_{23}(1,1)$. Figure (2.3) shows the points of $E_{23}(1,1)$; note that the points, with one exception, are symmetrical about $y = 11.5$.

Table 2.1 Points on the Elliptic Curve $E_{23}(1,1)$

(0, 1)	(6,4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

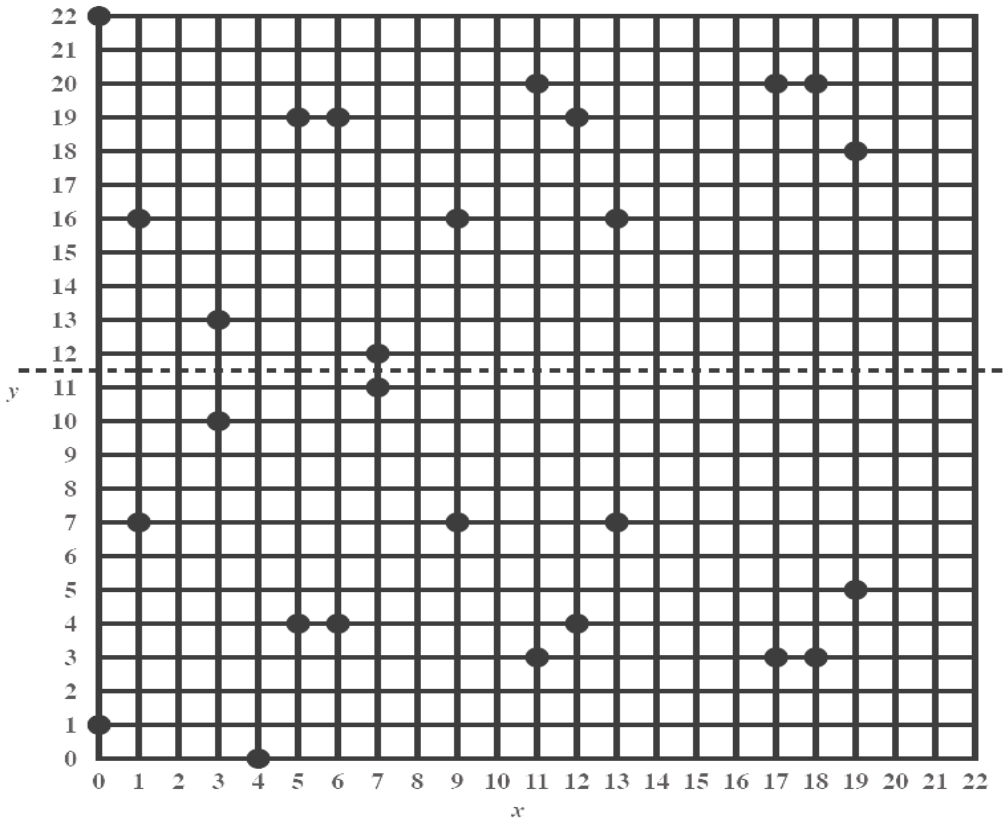


Figure 2.3 The Elliptic Curve $E_{23}(1,1)$ [1]

For example, let $P=(3,10)$ and $Q=(9,7)$ in $E_{23}(1,1)$, then to find $P+Q$ use equations (2.5 – 2.7)

$$\Delta=\left(\frac{7-10}{9-3}\right)\bmod 23=\left(\frac{-1}{2}\right)\bmod 23=11$$

$$x_R=\left(11^2-3-9\right)\bmod 23=109\bmod 23=17$$

$$y_R=\left(11(3-7)-10\right)\bmod 23=-164\bmod 23=20$$

so $P+Q=(17,20)$.

To find *2P* use equations (2.8 – 2.9)

$$\Delta = \left(\frac{3(3^2)+1}{2\times 10}\right) \bmod 23 = \left(\frac{1}{4}\right) \bmod 23 = 6$$

$$x_R = (6^2 - 3 - 3) \bmod 23 = 30 \bmod 23 = 7$$

$$y_R = (6(3-7) - 10) \bmod 23 = -34 \bmod 23 = 12$$

and $2P = (7, 12)$.

The preceding equation involves taking the multiplicative inverse in $GF(p)$[5].

2.3.2 Binary Finite Fields $GF(2^m)$

The finite field $GF(2^m)$, called a binary finite field, of 2^m elements, can be viewed as a vector space of dimension m over $GF(2)$. That is, there exists a set of m elements $\{\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_{m-1}\}$ in $GF(2^m)$ such that each $a \in GF(2^m)$ can be written uniquely in the form:

$$a = \sum_{i=0}^{m-1} a_i \alpha_i \qquad (2.12)$$

where $a_i \in \{0, 1\}$.

[5] The Binary Extended Euclidean Algorithm can be used to find the inverse

Example 2.2

For the finite field $GF(2^2)$, let $\alpha^3 = 1$ and $\alpha \neq 1$.

The elements of $GF(2^2)$ are

0, 1, α and $\alpha^2 = \alpha + 1$.

Using the standard trinomial basis, it is possible to write

$$GF(2^2) = \{(00) = 0, (01) = 1, (10) = \alpha, (11) = \alpha^2\}.$$

Simple operations on $GF(2^2)$ are:

$$\alpha + \alpha = \alpha^2 + \alpha^2 = 1 + 1 = 0$$

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

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

$$\alpha \bullet \alpha = \alpha^2$$

$$\alpha \bullet \alpha^2 = 1$$

$$\alpha^2 \bullet \alpha^2 = \alpha .$$

The elements of $GF(2^m)$ should be represented by bit strings of length m. There are several ways of performing arithmetic in $GF(2^m)$. The specific rules depend on how the bit strings are represented. There are two common structures for basis representations, which are discussed in the IEEE 1363a standard [17]: polynomial basis representations and normal basis representations; see [15, 19] for more details.

There are two types of elliptic curve over $GF(2^m)$: A supersingular elliptic curve is the set of solutions to the equation

$$y^2 + a_3 y = x^3 + a_4 x + a_6 \qquad (2.13)$$

where a_4 and a_6 satisfy equation (2.4) and $a_3^4 \neq 0$. This type of curve can be computed quickly, but it has some very special properties that make it unsuitable for cryptography.

A non-supersingular elliptic curve is the set of solutions to the equation

$$y^2 + xy = x^3 + a_2 x^2 + a_6 \qquad (2.14)$$

where $4a_2^3 + 27a_6^2 \neq 0$. Curves of this type are excellent for cryptography application, to check the rules of addition for this type of curve refer to [1].

2.3.3 Binary Composite Fields $GF((2^n)^m)$

There are various ways to represent the elements of $GF(2^k)$, depending on the choice of the basis or the particular construction method. If k is the product of two integers as $k = mn$, then it is possible to derive a different representation method by defining $GF(2^k)$ over the field $GF(2^n)$, which is called the ground field.

An extension field defined over a subfield of $GF(2^k)$ is known as a composite field, which is denoted by $GF((2^n)^m)$. Both the binary and the composite fields refer to the same field, and it is possible to obtain one representation from the other. Sunar et al. [20] describe the construction of the composite field.

The two pairs [21]

$$GF(2^n), \quad Q(y) = y^n + \sum_{i=0}^{n-1} q_i y^i \qquad (2.15)$$

and

$$GF((2^n)^m), \quad P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i \qquad (2.16)$$

are called a composite field if

- $GF(2^n)$ is constructed from $GF(2)$ by $Q(y)$.
- $GF((2^n)^m)$ is constructed form $GF(2^n)$ by $P(x)$.

Where $Q(y)$ is an irreducible polynomial over $GF(2)$ and $P(x)$ is also irreducible over $GF(2^n)$, these irreducible polynomials should have minimal weight, in order to provide superior performance as compared with binary fields.

The advantage of the composite field is that its operations are computed using arithmetic in the subfield $GF(2^n)$, and the operations in the subfield can be efficiently performed by index table look-up if n is not too large.

J. Guajardo and C. Paar [22] describe efficient implementations of elliptic curve cryptosystems using a composite finite field.

2.3.4 Prime Extension Fields $GF(p^m)$

For any positive integer m, it is possible to extend the prime field $GF(p)$ to a field of p^m elements which is called an extension field of $GF(p)$ and denoted as $GF(p^m)$, which combine the advantages of both prime and binary fields. In this field the multiprecision algorithms are not necessary because of the integer arithmetic using small polynomials.

2.3.5 Optimal Extension Fields (OEFs) $GF((2^n \pm c)^m)$

Optimal Extension Fields (OEFs) were introduced by C. Bailey and C. Paar in [23]. They are a class of extension fields $GF(p^m)$, which exploit the optimizations of integer arithmetic in modern processors to produce the fastest multiplication results over binary and prime fields.

The OEF is defined as $GF(p^m)$ which satisfies the following:

- p is a prime less than but close to the word size of the processor.
- p is a pseudo-Mersenne prime given in the form $p = 2^n \pm c$, where $\log_2 c \leq \frac{1}{2} n$.

The elements of $GF(p^m)$ should be represented by a sequence of m words. All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the modular reduction; the algorithmic and implementation details for OEFs will be discussed in chapter 5.

2.4 PERFORMANCE COMPARISON OF FIELD TYPES

Arithmetic in finite fields is an integral part of many public-key algorithms, including those based on the discrete logarithm problem in finite fields and elliptic curve based schemes. The performance of these schemes depends on the performance of the arithmetic in the underlying finite field. Figure (2.4) shows the different finite fields types that are proposed for use in public key cryptography; these types were briefly described in the previous section.

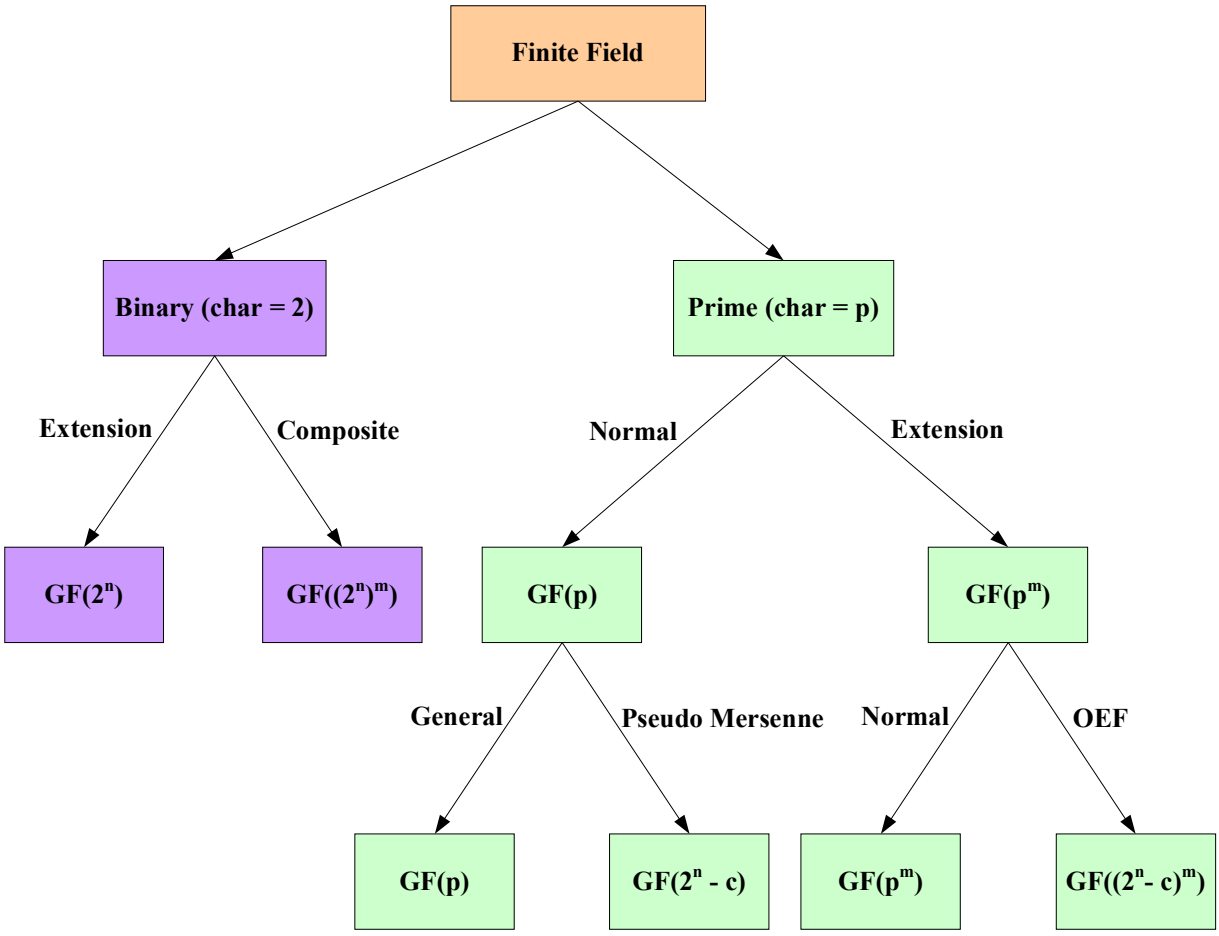


Figure 2.4 Finite Field Types

Most of the previous work on elliptic curve cryptography based over finite fields has focused on prime fields $GF(p)$ and binary extension fields $GF(2^m)$. Let us then look at the advantages and disadvantages of different finite fields types.

2.4.1 Binary Finite Fields

The binary extension field $GF(2^m)$ is a good choice for hardware circuit design of finite field multipliers, because it is possible to represent the elements of the subfield $GF(2)$ using logical values “0" and “1”, so each field operation requires *m* bit-wise operations which is faster in the hardware design; this makes field operation easier and more efficient. However, this type of field does not offer the same computational advantages in a software implementation, since most of modern microprocessors are deigned to be efficient for operations that deal with words rather than bits; this makes the operations very slow for large values of *m*.

2.4.2 Prime Finite Fields

The implementation of the prime field $GF(p)$ for large *p* on standard computers is also facing computational difficulties. Multiple machine words are required to represent elements of this field, since typical word sizes are simply not large enough. The problem with this representation is that during computation, the carries between words must be propagated, and the reduction modulo *p* must be performed over several machine words.

There has been a large amount of research dealing with methods for doing long-number multi-precision arithmetic efficiently; one of the most popular methods in this context is based on the Montgomery reduction [24]. On the other hand, using the processor's multi-precision arithmetic comes at a cost to time efficiency, especially with the reduction modulo *p* operations.

2.4.3 Binary Composite Fields

In spite of the fact that both the binary and the composite fields $GF((2^n)^m)$ refer to the same field, efficient hardware and software implementations can be obtained for composite fields, since this field provides efficient implementations for specific operations such as multiplication, inversion and exponentiation.

The advantage of composite field is that its operations are computed using arithmetic in the subfield $GF(2^n)$, and the operations in the subfield can be efficiently performed by index

table look-up if n is not too large. Thus, instead of performing the computations in the binary field, it is more efficient to implement the composite field to perform the computations.

Gaudry et al. [25] show a way of attacking the original elliptic curve cryptosystem over composite fields; this makes their use in practice questionable.

2.4.4 Optimal Extension Fields

Optimal Extension Fields $GF\left(\left(2^n \pm c\right)^m\right)$ offer considerable computational advantages by selecting p and m specifically to match the underlying hardware used to perform the arithmetic. All arithmetic operations are performed modulo the field polynomial, and thus there is no need for the multi-precision arithmetic. The key performance advantage of OEFs is due to fast modular reduction in the subfield.

The proper selection of p and m provide the following advantages:

- All subfield operations utilize the processor's fast integer arithmetic, since the size of p is less than the word size of the processor.
- p is a pseudo-Mersenne prime; this allows for efficient subfield modular reduction.
- An irreducible binomial $x^m - w$ exists for efficient extension field modular reduction.

Because of these advantages, as well as others that will be discovered later, the OEFs are considered to be the best choice for embedded system.

Chapter 3

PREVIOUS WORK

3.1 PREVIOUS WORK

Elliptic curves are algebraic curves that have been studied by many mathematicians since the seventeenth century. In 1985, Neal Koblitz [9] and Victor Miller [10] independently proposed public key cryptosystems using the group of points on an elliptic curve. This represents the creation of the elliptic curve cryptosystem, which has been shown to be a secure and computationally efficient method of performing public-key operation. Furthermore it is a good choice for a smaller and faster PK cryptosystems: a practical and secure technology, even for the most constrained environments.

Numerous researchers and developers have spent years researching the strength of ECC and improving techniques for its implementation. This chapter reviews some of the most relevant previous contributions.

De Win et al. [26] present a software implementation of arithmetic operations in a finite field $GF\left(\left(2^n\right)^m\right)$, with focus on $GF\left(\left(2^{16}\right)^{11}\right)$. This construction yields an extension field with 2^{176} elements; polynomials with coefficients in the subfield $GF\left(2^{16}\right)$ are used to represent these elements. Look up tables are used to carry out the calculations in the subfield; the small size of these tables allow them to fit in the computer memory. Optimizations for multiplication and inversion in such composite fields of characteristic two are described in [22].

Another paper by De Win et al. [27] presents a detailed implementation of elliptic curve arithmetic on a desktop PC, with a focus on its application to digital signature schemes using the fields $GF(p)$ with p a 192-bit prime and $GF\left(2^{191}\right)$. For ECCs over prime fields, their

construction uses projective coordinates to eliminate the need for inversion, along with a balanced ternary representation of the multiplicand.

Schroeppel et al. [28], optimize a software implementation of an elliptic curve analogue of the Diffie-Hellman key exchange [6] over the field $GF(2^{155})$; a polynomial bases is used to represent the field elements. They describe how to efficiently compute the field operations in this field.

A great deal of work has been done in studying aspects of inversion in a finite field, especially since inversion is the most costly of the four basic operations. There are two principle approaches to inversion in finite fields. One is based on the Extended Euclidean algorithm, the other one is based on Fermat's Little theorem. In the case of prime fields, Knuth [29] demonstrates that the Extended Euclidean algorithm requires $843 \log_2 s + 1.47$ divisions in the average case, for s the element to be inverted. A great number of variants on Euclid's algorithm have been developed for use in cryptographic applications. Schroeppel et al. [28] optimize a version of the Euclidean algorithm named "Almost Inverse Algorithm" for an EC implementation over $GF(2^{155})$; De Win et al. [26] also apply the same algorithm for EC over the composite field $GF((2^{16})^{11})$.

The authors in [30, 31] propose two types of fast algorithms for computing multiplicative inverses in $GF(2^m)$, one is of sequential type and the other is of recursive type. Itoh and Tsujii [32] present an algorithm for multiplicative inversion in $GF(2^m)$ based on the idea of reducing extension field inversion to the problem of subfield inversion. Their method is presented in the context of normal bases, where exponentiation to the q^{th} power is very efficient.

Takagi et al. [33] improve a fast algorithm for multiplicative inversion in $GF(2^m)$ that is proposed by Itoh and Tsujii [32] and reduce the number of multiplications by decomposing $m-1$ into several factors and a small remainder.

Guajardo and Paar [22] describe a version of Itoh and Tsujii's algorithm for inversion when applied to composite Galois fields $GF((2^n)^m)$ in a polynomial basis. In another paper [12] they

generalize this algorithm in extension fields $GF(p^m)$, and show that the Frobenius map can be explored to perform the exponentiations required for the inversion algorithm efficiently.

The authors in [34] present multiplier architectures for composite fields of the form $GF((3^n)^3)$ using Multi-Value Logic (MVL); the architecture is based on a modified version of the Karatsuba-ofman algorithm [35] for polynomial multiplication over $GF((3^n)^3)$. Elements of $GF((3^n)^3)$ are represented as polynomials of maximum degree 2 with coefficients in $GF(3^n)$. Multiplication in $GF(3^n)$ is achieved in the obvious way. Karatsuba multiplication is combined with modular reduction over $GF((3^n)^m)$ to reduce the complexity of their design. The authors estimate the complexity of their design for arithmetic over $GF((3^2)^3)$ as 56 "mod 3 adders" and 67 "mod 3 multipliers". The work in [36] describes $GF(3^n)$ architectures for applications of cryptographic significance, and treats the hardware implementation of fields of characteristic 3.

The work in [37] is one of the few that explicitly treated the general case of a $GF(p^m)$ multiplier; $GF(p^m)$ multiplication is computed in two stages. First the polynomial product is computed modulo a highly factorizable degree S polynomial $M(x)$, with $S \geq 2m-1$, thus the product is computed using a polynomial residue number system. In the second step an irreducible polynomial $P(x)$, which is defined over $GF(p^m)$, is used to reduce the result. The authors discuss a suitable choice for S, $M(x)$ and $P(x)$, and present an iterative method for the factorization. However, due to the constraints on the size of m, this method does not seem to be able to apply to field sizes acceptable in cryptographic applications.

Hasegawa et al. [38] describe an ECDSA implementation over $GF(p)$ on the M16C, a 16-bit 10 MHz microcomputer. They propose the use of a field of characteristic $p = e2^c \pm 1$, where e is an integer within the machine word size and c is a multiple of the machine word size. A multiplication in this field could be implemented in a small amount of memory. To reduce the number of operations needed for an EC point doubling, the authors use a randomly generated

curve with a coefficient of EC equal to $p-3$. They also modify the point addition algorithm in [17] to reduce the number of temporary variables from 4 to 2.

Different research has worked on speeding up the scalar multiplication $Q = kP$. The authors in [39] propose a new addition formula in projective coordinates for an elliptic curve over $GF(2^n)$, by rewriting the elliptic curve addition formula. This new formula speeds up the elliptic curve scalar multiplication by reducing the number of underlying field multiplications. The authors show that the addition formula in projective coordinates is improved by about 10 percent for general field elements and 12 percent for restricted coefficients.

The work in [23] introduces Optimal Extension Fields and their implementation on high-end RISC workstations. The authors employ well-known techniques for fast finite field arithmetic and how to perform them in an OEF. They provide performance statistics of OEF arithmetic on RISC workstations, and show that an OEF yields a considerable speed advantage over previous software implementations of Galois field arithmetic for elliptic curve cryptography. Some other papers extend the work on OEFs and report their performance on high-end RISC workstations.

Chapter 4

RELEVANT ALGORITHMS

The success of a cryptosystem depends on certain issues. One of the issues that leads to a successful and efficient implementation of a cryptosystem is the choice of algorithm to implement and optimize the arithmetic. In spite of the huge number of algorithms, it is important to carefully choose the best combination. This chapter discusses the primary algorithms that will be used in the finite field arithmetic and the elliptic curve implementation. Understanding these algorithms makes it easier to appreciate how the finite field arithmetic can be optimized and the elliptic curve cryptosystem can be implemented efficiently using these algorithms, as shown in the next chapters.

As mentioned previously, the underlying finite field that will be used in the implementation is the Optimal Extension Field. Therefore before starting with the algorithms relevant to this field, the first section explains some definitions and properties of OEF.

4.1 OPTIMAL EXTENSION FIELDS

The Optimal Extension Fields were introduced by Bailey and Paar in [23]. This section explains some definitions of the OEF, which are taken from [23].

Definition 4.1

Let c be a positive rational integer. A pseudo-Mersenne prime is a prime number of the form

$$2^n \pm c, \quad where \log_2 c \leq \tfrac{1}{2} n .$$

Definition 4.2

An Optimal Extension Field is a finite field $GF(p^m)$ such that:

1 p is a pseudo-Mersenne prime.

2 An irreducible binomial $P(x) = x^m - w$ exists over $GF(p)$.

The following theorem describes the cases when an irreducible binomial exists.

Theorem 4.1 [40]

Let $m \geq 2$ be an integer and $w \in GF(p)^*$, where $GF(p)^* = GF(p) \setminus \{0\}$ Then the binomial $x^m - w$ is irreducible in $GF(p)[x]$ if and only if the following two conditions are satisfied:

1. Each prime factor of m divides the order e of w over $GF(p)^*$ but not $(p-1)/e$.

2. $p \equiv 1 \bmod 4$, if $m \equiv 0 \bmod 4$.

Corollary 4.1 [23]

Let w be a primitive element for $GF(p)$ and let m be a divisor of $p-1$. Then $x^m - w$ is an irreducible polynomial of order $(p-1)^m$ over $GF(p)$.

There are two special cases of OEF, which yield additional arithmetic advantages; they are called Type I and Type II.

Definition 4.3

A Type I OEF has $p = 2^n \pm 1$.

A Type I OEF allows for subfield modular reduction with very low complexity.

Definition 4.4

A Type II OEF has an irreducible binomial $x^m - 2$.

A Type II OEF allows for speedups in extension field modular reduction since the multiplications by *w* can be implemented using shifts instead of explicit multiplications.

The range of possible *m* for a given *p* depends on the factorization of $p-1$ due to Theorem (4.1) and Corollary (4.1). For more details regarding the construction method for OEFs, refer to [23].

Before starting with the algorithms, let us distinguish between the following two expressions:

$$a \equiv b \bmod n \qquad (4.1)$$

means *a* is congruent to *b* modulo *n*, and this is if $n \mid (a-b)$ [41]. In contrast

$$b = a \bmod n\,. \qquad (4.2)$$

4.2 THE SCHOOLBOOK METHOD

The Schoolbook method is the usual way to multiply two polynomials. Consider two degree-*d* polynomials with *m* coefficients, where $m = d+1$

$$A(x) = \sum_{i=0}^{d} a_i x^i \qquad (4.3)$$

$$B(x) = \sum_{i=0}^{d} b_i x^i\,. \qquad (4.4)$$

Then the product $C(x) = A(x)\,B(x)$, can be calculated using this method as

$$C(x) = \sum_{i=0}^{d} x^i \cdot \left(\sum_{s+t=i;\, s,t\geq 0} a_s b_t \right) = \sum_{i=0}^{d} \sum_{j=0}^{d} a_i b_j x^{i+j} \,. \qquad (4.5)$$

Calculating the product using this method requires m^2 multiplications and $(m-1)^2$ additions.

4.3 THE KARATSUBA ALGORITHM

Multiplying two polynomials efficiently is an important issue in a variety of applications, including signal processing, cryptography and coding theory. In elliptic curve implementation the polynomial multiplication is required to implement the elliptic curve group operation and the polynomial inversion. A fast multiplication algorithm was developed by Karatsuba and Ofman [35]; the key to the efficiency of this algorithm is to reduce coefficient multiplications at the cost of extra additions compared to the schoolbook or ordinary multiplication method. In order to decide which method is more efficient, the cost ratio between one multiplication and one addition should be known. Although many algorithms are simpler than the Karatsuba algorithm (KA), it shows that it has a better performance for polynomials of small degree which are used in many applications.

Knuth [29] describes briefly how to multiply polynomials in a fast way, based on the Karatsuba algorithm. In order to simplify the problem assume that the maximum degree of the two polynomials, which are multiplied, is identical. In the next subsection a simple case of degree-2 polynomials is given to demonstrate how the Karatsuba algorithm works, and then a general case for polynomials of arbitrary degree is described in section (4.3.2). Both sections show a comparison between the performance of this algorithm and the Schoolbook method.

4.3.1 KA for Degree-2 Polynomials

Consider two degree-2 Polynomials

$$A(x) = a_2 x^2 + a_1 x + a_0$$

$$B(x) = b_2 x^2 + b_1 x + b_0 .$$

The product of $A(x)$ and $B(x)$ using the Schoolbook method is given by

$$C(x) = A(x)\ B(x) = \sum_{i=0}^{4} c_i x^i = \sum_{i=0}^{2}\sum_{j=0}^{2} a_i b_j x^{i+j}$$

$$C(x) = [a_2 b_2]x^4 + [a_2 b_1 + a_1 b_2]x^3 + [a_2 b_0 + a_1 b_1 + a_0 b_2]x^2 + [a_1 b_0 + a_0 b_1]x + [a_0 b_0].$$

Using this method for polynomial multiplication requires nine multiplications and four additions. To show how KA works define the following auxiliary variables:

$$D_0 = a_0 b_0$$

$$D_1 = a_1 b_1$$

$$D_2 = a_2 b_2$$

$$D_{0,1} = (a_0 + a_1)(b_0 + b_1)$$

$$D_{0,2} = (a_0 + a_2)(b_0 + b_2)$$

$$D_{1,2} = (a_1 + a_2)(b_1 + b_2).$$

Then construct the coefficients of $C(x)$ from these auxiliary variables using only additions and subtractions:

$$c_0 = D_0$$

$$c_1 = D_{0,1} - D_1 - D_0$$

$$c_2 = D_{0,2} - D_2 - D_0 + D_1$$

$$c_3 = D_{1,2} - D_1 - D_2$$

$$c_4 = D_2 .$$

So $C(x)$ is given by

$$C(x) = c_4 x^4 + c_3 x^3 + c_2 x^2 + c_1 x + c_0 .$$

This algorithm requires six multiplications and thirteen additions. Let *#MUL* and *#ADD* be the number of multiplications and additions, respectively. Table (4.1) shows the complexity of these two algorithms:

Table 4.1 Complexity of the Schoolbook method and the Karatsuba Algorithm for $m = 3$

	#MUL	*#ADD*
Schoolbook	9	4
Karatsuba	6	13

Let *r* be the ratio between the cost of one multiplication and one addition; $r = T_{mul} / T_{add}$, where T_{mul} and T_{add} are the time required for one multiplication and one addition, respectively. The time complexity of the Schoolbook method is given by

$$T_{SB} = 9T_{mul} + 4T_{add} = (9r + 4)T_{add} .$$

The time complexity of KA can be similarly calculated as

$$T_{KA} = 6T_{mul} + 13T_{add} = (6r + 13)T_{add} .$$

Now we need to find the value of r for which the time of KA is less than the time of the Schoolbook method, therefore

$$T_{SB} > T_{KA}$$

$$(9r + 4)T_{add} > (6r + 13)T_{add}$$

$$r > 3 .$$

So, if the ratio between the cost of one multiplication and one addition is greater than three, it is more efficient to use the KA.

4.3.2 KA for Polynomial of Arbitrary Degree

For polynomials whose number of coefficients *m* is a power of 2, the KA can be applied in a recursive way, as shown in [42]. However it is not straightforward to apply the KA to polynomials of arbitrary degree. This section describes a method to multiply two arbitrary polynomials with *m* coefficients using a one-iteration KA [42].

Consider two degree-d polynomials with $m = d + 1$ coefficients

$$A(x) = \sum_{i=0}^{d} a_i x^i \qquad (4.6)$$

$$B(x) = \sum_{i=0}^{d} b_i x^i . \qquad (4.7)$$

Compute for each $i = 0, 1, \cdots, m-1$

$$D_i = a_i b_i . \qquad (4.8)$$

Calculate for each $i = 1, 2, \cdots, 2m-3$ and for all s and t with $s + t = i$ and $m-1 \geq t > s \geq 0$

$$D_{s,t} = (a_s + a_t)(b_s + b_t). \qquad (4.9)$$

Then

$$C(x) = A(x)\ B(x) = \sum_{i=0}^{2m-2} c_i x^i \qquad (4.10)$$

can be computed as

$$c_0 = D_0 \qquad (4.11)$$

$$c_{2m-2} = D_{m-1} \qquad (4.12)$$

$$c_i = \begin{cases} \sum_{s+t=i,\, t>s\geq 0} D_{s,t} - \sum_{s+t=i,\, m>t>s\geq 0} (D_s + D_t) & \text{for odd } i,\ 0 < i < 2m-2 \\ \sum_{s+t=i,\, t>s\geq 0} D_{s,t} - \sum_{s+t=i,\, m>t>s\geq 0} (D_s + D_t) + D_{i/2} & \text{for even } i,\ 0 < i < 2m-2 \end{cases} . \qquad (4.13)$$

To analyze the complexity of KA for arbitrary polynomials, let us first analyze the number of auxiliary variables D_i and $D_{s,t}$, denoted by $\#D_i$ and $\#D_{s,t}$

$$\#D_i = m \qquad (4.14)$$

$$\#D_{s,t} = \frac{1}{2}m^2 - \frac{1}{2}m \qquad (4.15)$$

$$\#D = \#D_i + \#D_{s,t} = \frac{1}{2}m^2 + \frac{1}{2}m. \quad (4.16)$$

One multiplication is required to determine each variable D_i and $D_{s,t}$, which results in

$$\#MUL = \frac{1}{2}m^2 + \frac{1}{2}m. \quad (4.17)$$

The number of required additions is given by

$$\#ADD = \frac{5}{2}m^2 - \frac{7}{2}m + 1. \quad (4.18)$$

For large m, the one-iteration KA approaches $0.5m^2$ coefficient multiplications, and $2.5m^2$ additions. The KA is efficient if the ratio r is larger than 3, since

$$T_{SB} > T_{KA} \quad (4.19)$$

$$m^2 T_{mul} + (m-1)^2 T_{add} > \left(\frac{1}{2}m^2 + \frac{1}{2}m\right)T_{mul} + \left(\frac{5}{2}m^2 - \frac{7}{2}m + 1\right)T_{add} \quad (4.20)$$

$$r > 3. \quad (4.21)$$

The detailed proofs of the analyses are omitted for brevity, but can be found in [42].

Table (4.2) shows the complexity of the KA and Schoolbook method for a few polynomials with a prime number of coefficients (m), $r = 3$ for all of these polynomials:

Table 4.2 Comparison of the KA and the Schoolbook method for polynomials with *m* coefficients

m	KA		Schoolbook	
	#MUL	#ADD	#MUL	#ADD
2	3	4	4	1
3	6	13	9	4
5	15	46	25	16
7	28	99	49	36
11	66	265	121	100
13	91	378	169	144
17	153	664	289	256

4.4 THE EXTENDED EUCLID ALGORITHM

Euclid devised a scheme to discover if two numbers have any common factors. Euclid's algorithm finds the greatest common divisor between two numbers. Rosen [41] describes several methods to find the greatest common divisor. What is more useful than the original Euclid algorithm is what is known as the Extended Euclid algorithm. It is more useful, because it can be used to find an inverse.

Inversion of numbers in a prime field is similar to solving the linear congruence $bx = c \bmod n$, replacing c with 1 to get $bx = 1 \bmod n$, x must be the inverse of $b \bmod n$. Extended Euclid algorithm can be used to solve this linear congruence, as shown in algorithm (4.1).

The Itoth and Tsujii algorithm has been used to calculate the inversion; this algorithm reduces the problem of extension field inversion to subfield inversion. The Extended Euclid algorithm can be used to compute the subfield inversion.

Algorithm 4.1 Extended Euclid Algorithm

Require: n, b are positive integers.

Ensure: $x = b^{-1} \bmod n$.

1: $b_0 \leftarrow b$

2: $n_0 \leftarrow n$

3: $t \leftarrow 1$

4: $q \leftarrow n_0 / b_0$

5: $r \leftarrow n_0 - q \times b_0$ r

6: $t_0 \leftarrow 0$

7: If $r > 0$

8: $temp \leftarrow (t_0 - q \times t) \bmod n$

9: $t_0 \leftarrow t$

10: $t \leftarrow temp$

11: $n_0 \leftarrow b_0$

12: $b_0 \leftarrow r$

13: $q \leftarrow n_0 / b_0$

14: $r \leftarrow n_0 - q \times b_0$

15: GoTo 7

16: End if

17: If $b_0 \neq 1$

18: $x \leftarrow 0$

19: End if

20: $x \leftarrow t$

4.5 THE ITOH AND TSUJII INVERSION (ITI) ALGORITHM

One advantage of an extension field relies on a special mapping that is defined for all finite fields. The norm function maps elements of the extension field to the subfield by raising them to the $\left(p^m - 1\right)/(p-1)$ power, i.e. if

$$A(x) \in GF\left(p^m\right)$$

then

$$A^{\left(p^m-1\right)/(p-1)} \in GF(p). \qquad (4.22)$$

Itoh and Tsujii in [32] make use of this advantage to reduce the problem of extension field inversion to subfield inversion. They introduce the following theorem:

Theorem 4.2

Let

$$A(x) \in GF\left(p^m\right)$$

and

$$r = \frac{\left(p^m - 1\right)}{(p-1)}. \qquad (4.23)$$

Then, the multiplicative inverse of an element A can be computed as

$$A^{-1} = \left(A^r\right)^{-1} A^{r-1}. \qquad (4.24)$$

This theorem, which was originally developed in [32] for use with composite fields $GF\left(\left(2^n\right)^m\right)$ in a normal basis representation, can be applied to extension fields $GF\left(p^m\right)$ in polynomial

representation, as shown in [12]. Algorithm (4.2) describes the procedures for computing the inverse according to equation (4.24):

Algorithm 4.2	General ITI Algorithm in $GF(p^m)$	
Require:	$A(x) \in GF(p^m)^*$.	
Ensure:	$C \equiv A^{-1} \bmod P(x)$.	
1:	$B \leftarrow A^{r-1}$	Addition Chain Algorithm
2:	$b \leftarrow BA = A^{r-1}\ A = A^r$	$A^r \in GF(p)$
3:	$b \leftarrow b^{-1} = (A^r)^{-1}$	Extended Euclid Algorithm
4:	$C \leftarrow bB = (A^r)^{-1}\ A^{r-1} = A^{-1}$	

Step 2 is a normal extension field multiplication, while the multiplication in step 4 is trivial since $(A^r)^{-1}$ is in the subfield, the inversion in step 3 is a subfield inversion since $A^r \in GF(p)$ and it can be calculated using the Extended Euclid algorithm or through table look-up. The core of the algorithm is an exponentiation to the $(r-1)$-th power in the extension field $GF(p^m)$ in step 1; the exponent is expanded as

$$r-1 = \frac{p^m - 1}{p-1} - 1 = p^{m-1} + p^{m-2} + \cdots + p^2 + p\,. \qquad (4.25)$$

Exponentiation of A to the $(r-1)$-th power requires the computation of powers A^{p^i} for $1 \le i \le m-1$; thus, the p-adic representation $(r-1) = (1\,1\,1\,1\,1 \cdots 1\,0)_p$. Now an efficient method is required to evaluate $A^{-1}(x)$. $(r-1)$ will be fixed for a given field; thus the problem is to raise a general element to a fixed exponent. The use of straightforward methods such as the Binary Method of Exponentiation [29] is very costly, while the Addition Chain algorithm is a popular method for evaluating the norm function.

To see how the Addition Chain Algorithm works, let us take a simple finite field $GF(p^6)$. Algorithm (4.3) shows how to compute $A^{r-1}(x)$, where $(r-1)=(111110)_p$. All exponents are expressed in base p for clarity. This simple example requires three exponentiations to the p-th power, one exponentiation to the p^2-th power and three general multiplications. The general expression for the complexity of this algorithm can be found in [32].

Algorithm 4.3	Addition Chain for A^{r-1} in $GF(p^6)$
Require:	$A(x) \in GF(p^m)^*$.
Ensure:	$B \equiv A^{r-1} \bmod P(x)$.
1:	$B \leftarrow A^p = A^{(10)}$
2:	$B_0 \leftarrow BA = A^{(11)}$
3:	$B \leftarrow B_0^{p^2} = A^{(1100)}$
4:	$B \leftarrow BB_0 = A^{(1111)}$
5:	$B \leftarrow B^p = A^{(11110)}$
6:	$B \leftarrow BA = A^{(11111)}$
7:	$B \leftarrow B^p = A^{(111110)}$

These exponentiations can be implemented efficiently using the Frobenius map, which is defined as $\sigma(A)=A^p$. Applying an i-th iterate of the Frobenius map can be viewed as shifting the exponent to the left by i digits, e.g.

$$\sigma^4(A)=A^{p^4}=A^{(10000)_p}. \qquad (4.26)$$

4.6 THE FROBENIUS MAP

The previous section has shown how the inverse of the finite field elements could be found using the Itoth and Tsujii Inversion algorithm. This algorithm requires a norm function to implement the exponentiation; the Frobenius map can be used efficiently to evaluate the norm function and implement the required exponentiations.

Definition 4.5 [43]

Let $A(x) \in GF(p^m)$. Then the mapping $A \rightarrow A^p$ is an automorphism[6] known as the Frobenius map, which is defined as

$$\sigma(A) = A^p . \qquad (4.27)$$

If i is a nonnegative integer, then i-th iterate of the Frobenius map $A \rightarrow A^{p^i}$ is also an automorphism [43].

Now let us see how to use the Frobenius map to implement the exponentiations $GF(p^m)$ elements.

Consider the arbitrary element

$$A(x) = \sum_{j=0}^{m-1} a_j x^j \qquad (4.28)$$

$\in GF(p^m)$, and $a_j \in GF(p)$. The i-th iterate of the Frobenius map is

$$\sigma^i(A) = A^{p^i} = \left(\sum_{j=0}^{m-1} a_j x^j \right)^{p^i} = \sum_{j=0}^{m-1} \left(a_j x^j \right)^{p^i} = \sum_{j=0}^{m-1} a_j^{p^i} \; x^{jp^i} . \qquad (4.29)$$

[6] An automorphism is an isomorphism of a system of objects onto itself. An isomorphism is a kind of interesting mapping (function) between objects.

By Fermate's Little Theorem [41] $a_j^p \equiv a_j \bmod p$, one can rewrite equation (4.29) as

$$A^{p^i} = \sum_{j=0}^{m-1} a_j x^{jp^i} . \qquad (4.30)$$

In this summation attention will be given to the powers x^{jp^i} for $0 < i, j \le m-1$, which can be simplified as shown in the next theorem.

Theorem 4.3 [12]

Let $P(x)$ be an irreducible polynomial of the form $P(x) = x^m - w$, over $GF(p^m)$, e an integer, $P(\alpha) = 0$, and it is understood that $p \ge 3$, then:

$$\alpha^e \equiv w^t \, \alpha^s \qquad (4.31)$$

where $s \equiv e \bmod m$, and $t = \dfrac{e-s}{m}$.

Proof

First, notice that since $P(\alpha) = 0$, then $\alpha^m \equiv w$. Now

$$\alpha^e = \alpha^{tm+s} \qquad (4.32)$$

$$\alpha^e = \alpha^{tm} \, \alpha^s \equiv w^t \, \alpha^s . \qquad (4.33)$$

□

Let e be the exponent in the equation (4.30), i.e. $e = jp^i$, then

$$x^{jp^i} \equiv w^t \, x^s \bmod P(x) \qquad (4.34)$$

where $s \equiv jp^i \bmod m$, by theorem (4.1), $m \mid (p-1)$ which implies $p = (p-1)+1 \equiv 1 \bmod m$, thus $s \equiv jp^i \equiv j \bmod m$, then

$$x^{jp^i} \equiv w^t \; x^j \bmod P(x). \qquad (4.35)$$

If $P(x)$ is given, then it is possible to precompute all x^{jp^i}, for $0 < i, j \le m-1$ in equation (4.30), utilizing a table look-up with entries

$$c_j = w^{\frac{jp^i - j \bmod m}{m}} \bmod p \qquad (4.36)$$

where $c_j \in GF(p)$. Now the equation (4.30) can be rewritten as

$$A^{p^i} = \sum_{j=0}^{m-1} (a_j \; c_j) x^j \; . \qquad (4.37)$$

A single subfield multiplication is required to compute $(a_j \; c_j)$; thus only $m-1$ subfield multiplications are required to compute the exponentiation A^{p^i}.

Example 4.1

Consider $p = 239$, $P(x) = x^{17} - 2$. Use the equation (4.37) to find the exponentiation for an arbitrary element $A \in GF(239^{17})$. First computes the values of c_j, $0 \le j \le 16$, for a certain i-th iterate.

Let us start with $i = 1$, A^{239} using the equation (4.36)

$$c_0 = 2^{\frac{0 \times 239 - 0}{17}} \bmod 239 = 1$$

$$c_1 = 2^{\frac{1 \times 239 - 1}{17}} \bmod 239 = 132$$

$\vdots$

$c_{16} = 2^{\frac{16\times 239 - 16}{17}} \bmod 239 = 67$.

For $i = 2$, A^{239^2} then

$c_0 = 2^{\frac{0\times 239^2 - 0}{17}} \bmod 239 = 1$

$c_1 = 2^{\frac{1\times 239^2 - 1}{17}} \bmod 239 = 216$

$\vdots$

$c_{16} = 2^{\frac{16\times 239^2 - 16}{17}} \bmod 239 = 187$.

As can be seen, one can use these values as a table look-up, which can be used for any arbitrary element $A \in GF\left(239^{17}\right)$.

4.7 THE NON-ADJACENT FORM (NAF)

Scalar multiplication is the basic cryptographic operation of elliptic curve cryptography, which is denoted by

$$Q = kP \qquad (4.38)$$

where Q and P are points on an elliptic curve and k is an integer. What this really means is that add P to itself $k-1$ times:

$$Q = \underbrace{P + P + \cdots + P}_{k-times}. \qquad (4.39)$$

Koblitz in [44] uses a more efficient algorithm instead of the normal Binary-Double-and-Add algorithm to compute the scalar multiplication kP, for example $15P = P + 2(P + 2(P + 2P))$. It is more efficient to compute $2(2(2(2P))) - P$, which requires five operations instead of six.

To compute kP, k can be represented as a sum of a minimal number of powers of 2 with coefficients ± 1, by converting a string of set bits to a string of zero bits followed by -1 and the leading 0 is set. Koblitz calls the result the "Balanced Binary Expansion (BBE)". The following example makes this clear:

Example 4.2

Let $k = 10045$.

Binary Expansion $= 1\,0\,0\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1$

Balanced Binary Expansion $= 1\,0\,1\,0\,0\,-1\,0\,1\,0\,0\,0\,-1\,0\,1$.

The Balanced Binary Expansion has the following properties:

- Each positive integer has a unique Balanced Binary Expansion.
- There are no two consecutive coefficients which are nonzero.
- It has the fewest nonzero coefficients of any signed binary expansion.

Solinas [45] describes a simple way to create the Balanced Binary Expansion. It is called the "Non-Adjacent Form (NAF)". In this method, the integer k is represented as

$$k = \sum_{j=0}^{l-1} k_j 2^j \qquad (4.40)$$

where each $k_j \in \{-1, 0, 1\}$, $k_j \cdot k_{j+1} = 0, \quad \forall j$, and l is the bit length of k.

Algorithm (4.4), slightly modified from [45], shows how to find a Balanced Binary Expansion of an integer using the Non-Adjacent Form method:

Algorithm 4.4	Non-Adjacent Form
Require:	n integer number.
Ensure:	S Balanced Binary Expansion of n.
1:	$k \leftarrow n$
2:	$l \leftarrow 0$
3:	While $k > 0$
4:	If k odd
5:	$S[l] \leftarrow 2 - (k \ \& \ 3)$
6:	If $S[l] < 0$
7:	$k \leftarrow k + 1$
8:	End if
9:	Else
10:	$s[l] \leftarrow 0$
11:	End if
12:	$l \leftarrow l + 1$
13:	$k \leftarrow k/2$
14:	End while

Chapter 5

OPTIMAL EXTENSION FIELD ARITHMETIC

5.1 INTRODUCTION

An Extension Field $GF(p^m)$ is isomorphic to $GF(p)[x]/P(x)$, where $P(x)=x^m+\sum_{i=0}^{m-1}p_i x^i$, $p_i \in GF(p)$, is a monic irreducible polynomial of degree m over $GF(p)$. A residue class will be identified with the polynomial of least degree, and a standard basis representation will be used to represent a field element $A(x)\in GF(p^m)$

$$A(x)=\sum_{i=0}^{m-1}a_i x^i = a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_1 x+a_0 \qquad (5.1)$$

where $a_i \in GF(p)$.

The Optimal Extension Field is a class of extension field $GF(p^m)$ for p, a prime of special form and m, a positive integer. OEF fully exploits the optimizations of integer arithmetic in modern processors to produce the fastest multiplication results over binary extension and prime fields.

To optimize the arithmetic in $GF(p^m)$, the following properties are required for both p and m [23]:

1. Choose p to be less than but as close as possible to the word size of the processor so that all subfield operations utilize the processor's fast integer arithmetic.

2. Choose p to be a pseudo-Mersenne prime that is of the form $2^n \pm c$ for some $\log_2 c \leq \frac{1}{2}n$ to allow for efficient subfield modular reduction.

3. Choose m so that an irreducible binomial $x^m - w$ exists for efficient extension field modular reduction. The extension degree m can be small if the processor word size allows for large p.

Since p is less than the processor's word size $A(x)$ can be represented using an array of size m, each array element contains one a_i.

All arithmetic operations are performed modulo the field polynomial. The choice of field polynomial determines the complexity of the modular reduction. For example, on a microcontroller with 8-bit architecture, p would be chosen closest to 2^8, which would fully exploit the microcontroller ability to quickly perform 8-bit × 8-bit integer multiplications and thus perform subfield multiplication using a single multiply instruction followed by modular reduction. A suitable m is then chosen to provide a field of suitable order.

The elliptic curve operation requires one multiplication, one inversion, one squaring (or two squaring operations in the case of doubling) and a number of additions that are relatively fast compared with the first three; all of this arithmetic is in the OEFs. The operation of inversion is the most costly of the four basic operations, and its performance depends on the speed of multiplication. Therefore the speed of a single extension field multiplication determines the speed of the group operation in general.

This chapter describes the basic construction for arithmetic in the OEFs, and the required algorithms to implement these operations. The algorithms are described for use in general cases; however, to make it easier to visualize, in some sections the optimal extension field with $p = 2^8 - 17$, and $m = 17$ will be used, i.e. using the following extension field:

$$GF\left(\left(2^8 - 17\right)^{17}\right)$$

and an irreducible polynomial:

$$P(x) = x^{17} - 2 .$$

5.2 ADDITION AND SUBTRACTION

Addition and Subtraction of two field elements $A, B \in GF(p^m)$ is performed in an intuitive way, first by adding or subtracting the coefficients of the two polynomials in $GF(p)$ as follows:

$$A(x) \pm B(x) = \sum_{i=0}^{m-1} a_i x^i \pm \sum_{i=0}^{m-1} b_i x^i = \sum (a_i \pm b_i) x^i . \qquad (5.2)$$

Then it is required to perform the modular reduction; since $a_i, b_i \in GF(p)$, the maximum value is $2(p-1)$, in addition, and the minimum value is $-(p-1)$, in subtraction, the modular reduction can be performed simply by subtracting p once from the intermediate result if it is more than or equal p, in addition operation, and adding p once to the intermediate result if it is less than 0, in subtraction operation. This method of modular reduction is more efficient in addition and subtraction operations than using the subfield modular reduction method shown in algorithm (5.4).

Algorithm (5.1) and algorithm (5.2) describe the OEFs addition and subtraction respectively:

Algorithm 5.1	OEF Addition
Enquire:	$A(x)=a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_1x+a_0$
	$B(x)=b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\cdots+b_1x+b_0,\quad A, B\in GF(p^m).$
Ensure:	$A(x)+B(x)\equiv C(x)\in GF(p^m).$
1:	for $i\leftarrow 0$ upto $m-1$ do
2:	$c_i\leftarrow a_i+b_i$
3:	if $c_i\geq p$ then
4:	$c_i\leftarrow c_i-p$
5:	endif
6:	endfor

Algorithm 5.2	OEF Subtraction
Enquire:	$A(x)=a_{m-1}x^{m-1}+a_{m-2}x^{m-2}+\cdots+a_1x+a_0$
	$B(x)=b_{m-1}x^{m-1}+b_{m-2}x^{m-2}+\cdots+b_1x+b_0,\quad A, B\in GF(p^m).$
Ensure:	$A(x)-B(x)\equiv C(x)\in GF(p^m).$
1:	for $i\leftarrow 0$ upto $m-1$ do
2:	$c_i\leftarrow a_i-b_i$
3:	if $c_i<0$ then
4:	$c_i\leftarrow c_i+p$
5:	endif
6:	endfor

5.3 MULTIPLICATION

Field multiplication can be performed in two stages. First, multiply the two polynomials, $A(x)$ and $B(x)$, using ordinary polynomial multiplication to form an intermediate product $C'(x)$; and then perform modular reduction on $C'(x)$ to produce the result $C(x)$.

Consider $A(x)$ and $B(x)$ to be polynomials of degree $m-1$ in $GF(p^m)$

$$A(x)=\sum_{i=0}^{m-1} a_i x^i \qquad (5.3)$$

$$B(x)=\sum_{i=0}^{m-1} b_i x^i \qquad (5.4)$$

where $a_i, b_i \in GF(p)$.

The normal Schoolbook multiplication method can be used to calculate the intermediate product $C'(x)$ of degree less than or equal to $2m-2$

$$C'(x)=A(x)\times B(x)=\sum_{i=0}^{m-1}\sum_{j=0}^{m-1} a_i b_j x^{i+j} = c'_{2m-2}x^{2m-2}+\cdots+c'_1 x+c'_0 \qquad (5.5)$$

where the coefficients $c'_i \in GF(p)$.

Algorithm (5.3) describes this method, which requires m^2 multiplications and $(m-1)^2$ additions in the subfield $GF(p)$ to calculate the coefficients $c'_i, i=0,1,\cdots,2m-2$.

Algorithm 5.3	Schoolbook Method for Polynomial Multiplication
Require:	$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0$.
	$B(x) = b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots + b_1 x + b_0$, $A, B \in GF(p^m)$.
Ensure:	$C'(x) = A(x) \times B(x)$
1:	for $i \leftarrow 0$ upto $2m-2$ d0
2:	$c'_i \leftarrow 0$
3:	if $i < m$
4:	$s \leftarrow i$
5:	$e \leftarrow 0$
6:	else
7:	$s \leftarrow m-1$
8:	$e \leftarrow i - m + 1$
9:	endif
10:	for $j \leftarrow s$ downto e
11:	$c'_i = c'_i + a_{i-j} b_j$
12:	endfor
13:	endfor

In the next stage of OEF multiplication, the intermediate result $C'(x)$ has to be reduced modulo the irreducible polynomial $P(x) = x^m - w$. It could be observed that

$$x^m \equiv w \bmod P(x). \qquad (5.6)$$

This observation leads to the general expression for this reduction, given by

$$C(x) \equiv C'(x) \bmod P(x) \qquad (5.7)$$

$$C(x) \equiv \sum_{i=0}^{2m-2} c_i' x^i \bmod x^m - w. \qquad (5.8)$$

Section (5.3.2) describes an efficient method to calculate the residue $C(x)$. But let us first explain in the next section how to compute a fast subfield modular reduction, which is required by a subfield multiplication.

5.3.1 Subfield Modular Reduction

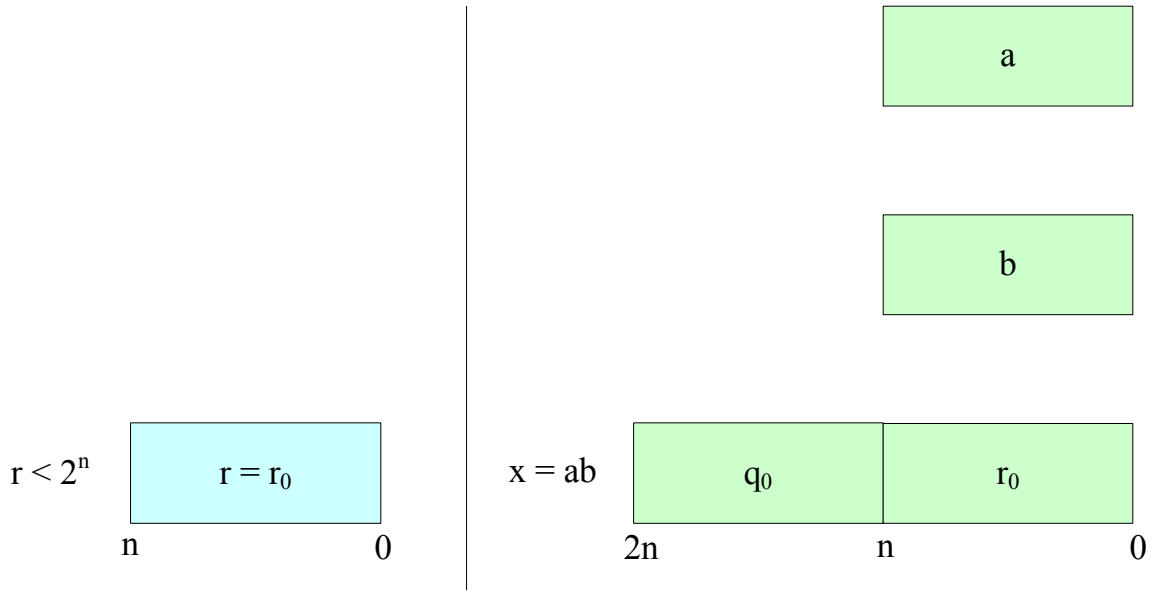
The over-all performance of OEFs greatly relies on the fact that subfield multiplication is fast. Subfield arithmetic in $GF(p)$ is implemented with standard modular integer techniques. The multiplication of two elements $a, b \in GF(p)$, yields the integer product $x = a \times b$. In general, case x has about twice the bit length of p, and it needs to be reduced to the same size of p; this can be performed by a modular reduction as $a \times b \equiv x \bmod p$.

For OEF, the characteristic p is a pseudo-Mersenne prime, which makes the subfield multiplication efficient and allows for an efficient reduction modulo p. Mohan and Adiga [46] show a technique for fast modular reduction for modulo of the form $2^n \pm c$, where c is a small integer, without an explicit integer division.

A form of such a modular reduction algorithm has been presented, which is adapted from [46]. This algorithm addresses a modulo of the form $2^n - c$. The original algorithm needs several shifts, which is modified in algorithm (5.4) in order to eliminate these shifts without any additional operations.

Algorithm 5.4	Subfield Modular Reduction
Require:	$x < p^2, \quad p = 2^n - c, \quad \log_2 c \leq \frac{1}{2}n$.
Ensure:	$r \equiv x \pmod{p}$.
1:	x[i] refers to i-th n-bit word of x
2:	$r \leftarrow x[0]$
3:	$q \leftarrow x[1]$
4:	while $q > 0$ do
5:	$q \leftarrow q \times c$
6:	$r \leftarrow r + q[0]$
7:	$q \leftarrow q[1]$
8:	endwhile
9:	while $r \geq p$ do
10:	$r \leftarrow r - p$
11:	endwhile

To understand the operation of this algorithm, consider the following diagram:



Let $a_i, b_i \in GF(p)$, which are less than 2^n. The size of a product $x = a \times b$ is less than 2^{2n}, but in general larger than 2^n. The above algorithm will be used to perform a modular reduction.

Let r_0 be the lower n bits of the product x, and the upper n bits q_0, so:

$$x = a \times b = 2^n q_0 + r_0 \qquad (5.9)$$

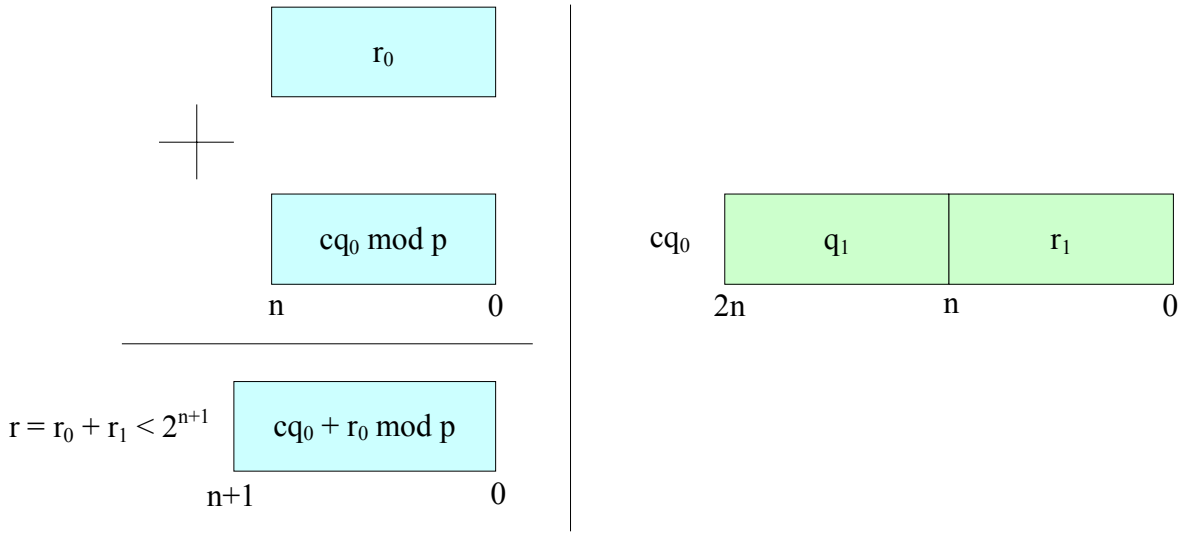
but

$$2^n \equiv c \pmod{p} \qquad (5.10)$$

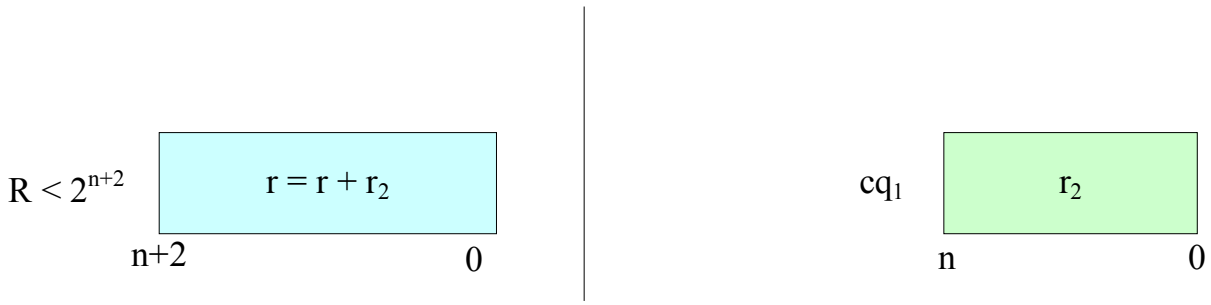
then

$$r = x \equiv cq_0 + r_0 \pmod{p}. \qquad (5.11)$$

This can be represented as:



This new expression for the residue class is still larger than 2^n, so the process is repeated: q_1 is the upper n bits of cq_0 and r_1 is the lower n bits of cq_0. Again rewrite the equations and replace 2^n by c:



Now the result residue of $a \times b$ that in general is less than 2^{n+2}, subtract p from the intermediate result r once or twice to complete the reduction. This algorithm requires only two multiplications by c and several additions and subtractions, so one OEF subfield multiplication is actually three integer multiplications.

5.3.2 Extension Field Modular Reduction

After finishing the first stage of multiplying two polynomials, the intermediate result $C'(x)$ is obtained, which in general has a degree greater than or equal to m. The next stage is to reduce $C'(x)$ so that it has a maximum degree of $m-1$. This can be achieved by taking the remainder after polynomial long division with the divisor being the irreducible field polynomial.

OEFs have a field polynomial of the form

$$P(x) = x^m - w. \qquad (5.12)$$

Using an irreducible binomial as a field polynomial allows for a more efficient computational of the modular reduction.

In general, the intermediate result $C'(x)$ has the form

$$C'(x) = c'_{2m-2}x^{2m-2} + c'_{2m-3}x^{2m-3} + \cdots + c'_1 x + c'_0. \qquad (5.13)$$

Then only the terms $c'_{m+i}x^{m+i}$, where $i \geq 0$ must be reduced modulo $P(x)$.

First observe that the following congruencies hold:

$$x^m \equiv w \bmod P(x) \tag{5.14}$$

$$x^{m+1} \equiv wx \bmod P(x) \tag{5.15}$$

$$\vdots$$

$$x^{2m-2} \equiv wx^{m-2} \bmod P(x). \tag{5.16}$$

Using these congruencies, $C(x) = A(x) \times B(x) \bmod P(x)$ can be computed as:

$$\begin{aligned} C(x) &= \left(\sum_{i=0}^{m-1} a_i x^i \cdot \sum_{j=0}^{m-1} b_j x^j \right) \bmod P(x) \\ &= \sum_{i,j \geq 0,\, i+j=k<m} a_i b_j x^k + w \sum_{i,j \geq 0,\, i+j=k \geq m} a_i b_j x^{k-m} \\ &= \sum_{k=0}^{m-1} \left(\sum_{i=0}^{k} a_i b_{k-i} + w \sum_{i=k+1}^{m-1} a_i b_{m+k-i} \right) x^k . \end{aligned} \tag{5.17}$$

So the coefficient c_i can be computed as:

$$c_i = \left(\sum_{j=0}^{i} a_j b_{i-j} + w \sum_{j=i-m+1}^{m-1} a_j b_{i-j} \right) \bmod p . \tag{5.18}$$

Extension field multiplication requires m^2 $GF(p)$ products $a_i b_j$, and $m-1$ multiplications by w when the Schoolbook method is used, the total of $m^2 + m - 1$ subfield multiplications that form the performance critical part of a field multiplication. For OEFs with $p = 2^n - c$, a subfield multiplication can be performed as single-precision integer multiplication, resulting in

a double-precision product with a subfield reduction modulo p, by using algorithm (5.4). Each subfield reduction requires two multiplications and several additions, so $2(m^2+m-1)$ additional multiplications are required to perform the extension field multiplication using this technique.

Using the accumulation-and-then-reduction technique shown in equation (5.18) reduces the number of reductions mod p to m; only one reduction per coefficient $c_i, i=0,1,\cdots,m-1$, is required. This can be done by using an integer larger than p to represent the residue class of the sum of integer products, and then reduce the result, which in general is more than two words.

Using Type II OEFs offers additional optimization; since it uses an irreducible binomial $P(x)=x^m-2$, the multiplication by w can be replaced by left shift one bit, which reduces the number of multiplication to m^2.

To see how this works let us take the following example:

$$A(x), B(x) \in GF\left((2^8-17)^{17}\right)$$

$$P(x)=x^{17}-2.$$

First, calculate the intermediate values for $c'_i, i=17,18,\cdots,32$

$$c'_{17}=a_1b_{16}+a_2b_{15}+\cdots+a_{14}b_3+a_{15}b_2+a_{16}b_1$$

$$c'_{18}=a_2b_{16}+a_3b_{15}+\cdots+a_{15}b_3+a_{16}b_2$$

$$\vdots$$

$$c'_{31}=a_{15}b_{16}+a_{16}b_{15}$$

$$c'_{32}=a_{16}b_{16}.$$

Now calculate $c_i, i = 0, 1, \cdots, 16$

$$c_0 = a_0 b_0 + w c'_{17} \bmod 239$$

$$c_1 = a_0 b_1 + a_1 b_0 + w c'_{18} \bmod 239$$

$$\vdots$$

$$c_{15} = a_0 b_{15} + a_1 b_{14} + \cdots + a_{14} b_1 + a_{15} b_0 + w c'_{32} \bmod 239$$

$$c_{16} = a_0 b_{16} + a_1 b_{15} + \cdots + a_{14} b_2 + a_{15} b_1 + a_{16} b_0 \bmod 239 .$$

The maximum value for the multi-word integer c_i before reduction can be calculated as:

1 A product multiplication $a_i b_j$ has a maximum value of $(p-1)^2$.

2 Accumulate 17 products, 16 of which are multiplied by $w = 2$.

3 $ACC_{\max} = 33(p-1)^2 = 1869252 = 1\text{C85C4h} < 2^{21}$.

Now, expand the basic OEF reduction shown in algorithm (5.4) for multiple words.

As $\log_2 ACC_{\max} = 21$ bits, the number can be represented in the radix 2^8 with three digits as

$$x = x_2 2^{2n} + x_1 2^n + x_0 \qquad (5.19)$$

where $x_0, x_1, x_2 < 2^n$. Using the two congruencies

$$2^n \equiv c \left(\bmod 2^n - c\right) \qquad (5.20)$$

$$2^{2n} \equiv c^2 \pmod{2^n - c}. \qquad (5.21)$$

The first reduction is performed as

$$x' \equiv x_2 c^2 + x_1 c + x_0 \pmod{2^n - c} \qquad (5.22)$$

noting that $c^2 = 289 \equiv 50 \bmod 239$. The maximum value of $x' = 1734h$, can be obtained when $x = 1BFFFF$. Since $\log_2 x' = 13$ bits, which is less than 2^{2n}, algorithm (5.4) can be applied to reduce x' to an element $GF(239)$.

To perform the three-word reduction, two multiplications are required to get a 13-bit reduction, and then another two multiplications are required by algorithm (5.4), so each reduction requires four multiplications. The total number of multiplications that are required to perform extension field multiplication using this technique is $m^2 + 4m$.

$3m^2$ multiplications are required to perform extension field multiplication using subfield reduction to reduce each inner product individually.

Algorithm (5.5) shows the entire multiplication and reduction. Line 13 supposes that the maximum value of accumulation is less than 2^{3n}, otherwise additional reduction is required using the same principle.

Algorithm 5.5	OEF Multiplication with Reduction	
Require:	$A(x)=\sum_{i=0}^{m-1} a_i x^i$, $B(x)=\sum_{i=0}^{m-1} b_i x^i \in GF\left(\left(2^n-c\right)^m\right)/P(x)$, where	
	$P(x)=x^m-2$, $a_i, b_i \in GF\left(2^n-c\right)$.	
Ensure:	$C(x)\equiv A(x)B(x)\equiv\sum_{i=0}^{m-1} c_i x^i \pmod{P(x)}$, where $c_i \in GF\left(2^n-c\right)$.	
1:	$x[i]$ refer to i-th n-bit word of x, << means left shift	
2:	for $i \leftarrow 0$ upto $m-1$	
3:	$c_i \leftarrow 0$	
4:	if $i \neq m-1$	
5:	for $j \leftarrow m-1$ downto $i+1$	
6:	$c_i \leftarrow c_i + a_{i+m-j}\, b_j$	
7:	end for	
8:	$c_i \leftarrow c_i << 1$	Multiply by $w=2$
9:	end if	
10:	for $j \leftarrow i$ downto 0	
11:	$c_i \leftarrow c_i + a_{i-j}\, b_j$	
12:	end for	
13:	$c_i \leftarrow c_i[2]\times c^2 + c_i[1]\times c + c_i[0]$	Equation (5.22)
14:	$c_i \leftarrow$ Subfield_Reduction(c_i)	Algorithm (5.4)
15:	end for	

5.4 SQUARING

Extension field squaring is similar to multiplication and it can be implemented using the general multiplication algorithm shown above. Since the two inputs are equal, there are identical inner product terms. By taking advantage of this, the multiplication algorithm can be modified to get some additional computational efficiency.

For example, to compute the squaring of

$$A(x) \in GF\left(\left(2^8 - 17\right)^{17}\right)$$

$$P(x) = x^{17} - 2$$

first calculate the intermediate values for $c'_i, i = 17, 18, \cdots, 32$

$$c'_{17} = 2a_1a_{16} + 2a_2a_{15} + \cdots + 2a_6a_{11} + 2a_7a_{10} + 2a_8a_9$$

$$c'_{18} = 2a_2a_{16} + 2a_3a_{15} + \cdots + 2a_7a_{11} + 2a_8a_{10} + a_9^2$$

$$\vdots$$

$$c'_{31} = 2a_{15}a_{16}$$

$$c'_{32} = a_{16}^2 .$$

Now calculate $c_i, i = 0, 1, \cdots, 16$

$$c_0 = a_0^2 + wc'_{17} \bmod 239$$

$$c_1 = 2a_0a_1 + wc'_{18} \bmod 239$$

$$c_2 = 2a_0a_2 + a_1^2 + wc'_{19} \bmod 239$$

$\vdots$

$$c_{15} = 2a_0a_{15} + 2a_1a_{14} + \cdots + 2a_6a_9 + 2a_7a_8 + wc'_{32} \bmod 239$$

$$c_{16} = 2a_0a_{16} + 2a_1a_{15} + \cdots + 2a_6a_{10} + 2a_7a_9 + a_8^2 \bmod 239 .$$

Multiplication by two can be implemented as a left shift operation by one bit, which is faster than multiplication. The number of coefficient multiplications can be reduced to $m(m+1)/2$ instead of m^2 that is required by the general multiplication algorithm.

The following technique may be used to achieve more efficiency. Double one coefficient by shifting it one bit lift, then perform modular reduction by simply subtracting p from the doubled result if it is more than or equal to p, and store the new values. These can be used later when they are needed again without recalculating the doubled coefficient. Algorithm (5.6) shows an efficient method to square OEFs element.

Algorithm 5.6 OEF Squaring with Reduction

Require: $A(x)=\sum_{i=0}^{m-1} a_i x^i \in GF\left(\left(2^n-c\right)^m\right)/P(x)$, where $P(x)=x^m-2$,

$a_i \in GF\left(2^n-c\right)$.

Ensure: $C(x)\equiv A^2(x)\equiv \sum_{i=0}^{m-1} c_i x^i \pmod{P(x)}$, where $c_i \in GF\left(2^n-c\right)$.

1: $x[i]$ refer to i-th n-bit word of x, << means left shift

2: $B(x)\leftarrow 2A(x)$

3: for $i\leftarrow 0$ upto $m-1$

4: $c_i \leftarrow 0$

5: if $i\neq m-1$

6: for $j\leftarrow m-1$ downto $m-1-\left\lfloor\frac{m-1-i}{2}\right\rfloor$

7: $c_i \leftarrow c_i + a_{i+m-j}\, b_j$

8: end for

9: if i is odd

10: $c_i \leftarrow c_i + a_j^2$ end if

11: $c_i \leftarrow c_i << 1$ Multiply by $w=2$

12: end if

13: if $i\neq 0$

14: for $j\leftarrow i$ downto $1+\lfloor i/2\rfloor$

15: $c_i \leftarrow c_i + a_{i-j}\, b_j$

16: end for

17: end if

18: if i is even

19: $c_i \leftarrow c_i + a_{j/2}^2$ end if

20: $c_i \leftarrow c_i[2]\times c^2 + c_i[1]\times c + c_i[0]$ Equation (5.22)

21: $c_i \leftarrow$ Subfield_Reduction(c_i) Algorithm (5.4)

22: end for

5.5 INVERSION

The extension field inversion is the most costly arithmetic operation in elliptic curve system. Most of the finite field inversions are either based on the Extended Euclidean algorithm or on Fermat's Little theorem. The Itoth and Tsujii Inversion algorithm offers an alternative inversion method which is particularly suited to finite fields in polynomial basis that have a binomial as the field polynomial.

The Itoh and Tsujii Inversion takes advantage of the nature of OEFs to reduce the problem of extension field inversion to subfield inversion. This inversion reduction can be done by using a special mapping, which is defined for all finite fields; the norm function maps elements of the extension field to the subfield by raising them to a certain power.

The Itoh and Tsujii Inversion algorithm computes an inverse in $GF(p^m)$ as:

$$A^{-1} = \left(A^r\right)^{-1} A^{r-1} \bmod P(x). \qquad (5.23)$$

Note that $A^r \in GF(p)$, where $r = \dfrac{p^m - 1}{p - 1}$.

Algorithm (4.3) describes the procedure for computing the inverse according to equation (5.23); more details on this algorithm can be found in section (4.5). The Extended Euclid algorithm or a table look-up can be used to compute the subfield inversion $\left(A^r\right)^{-1}$.

The most costly operation is the computation of the exponentiation A^{r-1}. The Addition Chain algorithm is a popular method that can be derived to perform this exponentiation. This method requires four multiplications and five exponentiations to a p^i-th power to compute A^{r-1} in the case of $m = 17$.

The exponentiations occurring in the Addition Chain can be implemented efficiently using Frobenius maps; the exponentiation in an OEF is evaluated by multiplying each coefficient of

the element's polynomial representation (a_j) by the Frobenius constant (c_j), which is determined by the field and its irreducible binomial as shown in equation (5.24).

$$A^{p^i} = \sum_{j=0}^{m-1} \left(a_j \, c_j\right) x^j \qquad (5.24)$$

c_j can be computed as

$$c_j = w^{\frac{jp^i - j \bmod m}{m}} \bmod p\,. \qquad (5.25)$$

Computing the p-th iteration of the Frobenius map for an OEF with a binomial field polynomial requires at most $m-1$ multiplications in $GF(p)$. The Frobenius constants c_j are fixed as shown in equation (5.25), so these constants can be precomputed for a table look-up.

Frobenius maps can be defined as $\sigma(A) = A^p$. Applying an i-th iterate of the Frobenius map can be viewed as shifting the exponent to the left by i digits, e.g.

$$\sigma^4(A) = A^{p^4} = A^{(10000)_p}.$$

Algorithm (5.7) shows the details of finding the inversion of element in the finite field $GF\left(\left(2^8 - 17\right)^{17}\right)$, and an irreducible binomial $P(x) = x^{17} - 2$. In this example five extension field multiplications, five exponentiations to a p^i-th power and one subfield inversion are required. The Frobenius constants shown in table (5.1) may be used to compute the required exponentiations using equation (5.24).

Algorithm 5.7	OEF Inversion in $GF\left(\left(2^8-17\right)^{17}\right)$	
Require:	$A(x)=\sum_{i=0}^{16} a_i x^i \in GF\left(\left(2^8-17\right)^{17}\right)/P(x)$, where $P(x)=x^{17}-2$, $a_i \in GF\left(2^8-17\right)$.	
Ensure:	$C(x)\equiv A^{-1}(x)\,(\mathrm{mod}\,P(x))$, where $C(x)\in GF\left(\left(2^8-17\right)^{17}\right)$.	
1:	$C_0 \leftarrow A^p = A^{(10)_p}$	$\sigma(A)$
2:	$C_1 \leftarrow C_0\ A = A^{(11)_p}$	Multiplication
3:	$C_2 \leftarrow (C_1)^{p^2} = A^{(1100)_p}$	$\sigma^2(C_1)$
4:	$C_3 \leftarrow C_2\ C_1 = A^{(1111)_p}$	Multiplication
5:	$C_4 \leftarrow (C_3)^{p^4} = A^{(11110000)_p}$	$\sigma^4(C_3)$
6:	$C_5 \leftarrow C_4\ C_3 = A^{(11111111)_p}$	Multiplication
7:	$C_6 \leftarrow (C_5)^{p^8} = A^{(1111111100000000)_p}$	$\sigma^8(C_5)$
8:	$C_7 \leftarrow C_6\ C_5 = A^{(1111111111111111)_p}$	Multiplication
9:	$C_8 \leftarrow (C_7)^p = A^{(1111111111111110)_p} = A^{r-1}$	$\sigma(C_7)$
10:	$c \leftarrow C_8\ A = A^{r-1}\ A = A^r$	Multiplication
11:	$c \leftarrow c^{-1} = \left(A^r\right)^{-1}$	Subfield Inversion
12:	$C \leftarrow c\ C_8 = \left(A^r\right)^{-1}\ A^{r-1} = A^{-1}$	

Table 5.1 Frobenius constants $\sigma^i(A) = A^{p^i}$

Coefficient	Exponent			
	p	p^2	p^4	p^8
a_0	1	1	1	1
a_1	132	216	51	211
a_2	216	51	211	67
a_3	71	22	6	36
a_4	51	211	67	187
a_5	40	166	71	22
a_6	22	6	36	101
a_7	36	101	163	40
a_8	211	67	187	75
a_9	128	132	216	51
a_{10}	166	71	22	6
a_{11}	163	40	166	71
a_{12}	6	36	101	163
a_{13}	75	128	132	216
a_{13}	101	163	40	166
a_{15}	187	75	128	132
a_{16}	67	187	75	128

5.6 EXPONENTIATION

Raising an extension field element $A(x) \in GF(p^m)$ to the n-th power can be computed by multiplying $A(x)$ by $A(x)$ n times. Unfortunately this method is inefficient for large n. The binary method for exponentiation can be modified to compute the extension field exponentiation; this algorithm is simple and powerful.

The idea of this algorithm is to square and multiply by $A(x)$ until $A^n(x)$ has been reached. For example, to compute A^{13}, it can be expanded as:

$$A^{13} = \left(\left(A^2 \cdot A\right)^2\right)^2 \cdot A .$$

In fact this method is called left-to-right binary exponentiation, which can be described as follows. First write the exponent 13 in binary: 1101, starting chain with 1, the most significant bit is set so multiply by A, and then square the result. The next bit is set, so multiply again by A and square. For a clear bit simply square the result, and carry on until all bits are done.

Another method is called right-to-left binary exponentiation is shown in algorithm (5.8). This right-to-left method is easier to program, takes the same number of multiplications as the left-to-right method. After initializing C to 1 and B to A, enter a loop, in which always square B, while multiplying C by B only if n is odd.

For large value of n $(n > p)$; it will be better to use algorithm (5.8) with the Addition Chain algorithm, which describes in section (4.5). The Addition Chain algorithm is a popular method for evaluating the norm function. The Frobenius map can be used efficiently to evaluate the norm function and implement the required exponentiations.

Algorithm 5.8	OEF Exponentiation	
Require:	$A(x)=\sum_{i=0}^{m-1} a_i x^i$, $\in GF(p^m)$, where $a_i \in GF(p)$, n is an integer.	
Ensure:	$C(x) \equiv A^n(x) \equiv \sum_{i=0}^{m-1} c_i x^i$, where $c_i \in GF(p)$.	
1:	$C(x) \leftarrow 1$	$c_0 = 1$,other $c_i = 0$
2:	if $n < 0$	
3:	$n \leftarrow -n$	
4:	$B(x) \leftarrow A^{-1}(x)$	Algorithm (5.7)
5:	else	
6:	$B(x) \leftarrow A(x)$	
7:	end if	
8:	While $n \neq 0$ do	
9:	if n is odd	
10:	$C(x) \leftarrow C(x) \times B(x)$	
11:	end if	
12:	$B(x) \leftarrow B^2(x)$	
13:	$n \leftarrow n/2$	
14:	end while	

5.7 FAST MULTIPLICATION

As is shown above, extension field multiplication is a very important issue, since it is required to implement both the elliptic curve group operation and the algorithms for inversion and exponentiation. The improvement of implementing extension field multiplication enhances the performance and the efficiency of the entire elliptic curve system.

Section (5.3) has explained the normal multiplication method and has described a special technique, accumulation-and-then-reduction, to perform the extension field modular reduction, which reduces the complexity of the multiplication process.

In this section, a method to reduce the complexity of extension field multiplication will be used. This fast multiplication algorithm was developed by Karatsuba and Ofman [35], and using it reduces the number of subfield multiplications in exchange for an increased number of additions, compared to the Schoolbook multiplication method. Section (4.3) has provided a generalization and detailed analysis of this algorithm.

Algorithm (5.9) shows the details of implementing the Karatsuba algorithm.

Algorithm 5.9	OEF Fast Multiplication using Karatsuba Algorithm
Require:	$A(x)=\sum_{i=0}^{m-1} a_i x^i$, $B(x)=\sum_{i=0}^{m-1} b_i x^i \in GF\left(\left(2^n-c\right)^m\right)\Big/P(x)$, where
	$P(x)=x^m-2$, $a_i, b_i \in GF\left(2^n-c\right)$.
Ensure:	$C(x)\equiv A(x)B(x)\equiv\sum_{i=0}^{m-1} c_i x^i \pmod{P(x)}$, where $c_i \in GF\left(2^n-c\right)$.
1:	x[i] refer to i-th 8-bit word of x, << means left shift
2:	for $i \leftarrow 0$ upto $m-1$
3:	for $j \leftarrow i$ upto $m-1$
4:	if $i=j$
5:	$D_i \leftarrow a_i\, b_i$
6:	else
7:	$D_{i,j} \leftarrow \left(a_i+a_j\right)\left(b_i+b_j\right)$

8: end if
9: end for
10: end for
11: for $i \leftarrow 0$ upto $m-1$
12: $c_i \leftarrow 0$
13: if $i \neq m-1$
14: $k \leftarrow i+m$
15: for $j \leftarrow m-1$ downto $k/2+1$
16: $c_i \leftarrow c_i + D_{k-j,j} - D_{k-j} - D_j$
17: end for
18: if k is even
19: $c_i \leftarrow c_i + D_{k/2}$
20: end if
21: $c_i \leftarrow c_i << 1$ Multiply by $w=2$
22: end if
23: for $j \leftarrow i$ downto $i/2+1$
24: $c_i \leftarrow c_i + D_{i-j,j} - D_{i-j} - D_j$
25: end for
26: if i is even
27: $c_i \leftarrow c_i + D_{i/2}$
28: end if
29: $c_i \leftarrow c_i[2] \times c^2 + c_i[1] \times c + c_i[0]$ Equation (5.22)
30: $c_i \leftarrow$ Subfield_Reduction(c_i) Algorithm (5.4)
31: end for

The extension field multiplication using the Karatsuba algorithm requires

$$\#MUL = \frac{1}{2}m^2 + \frac{1}{2}m \qquad (5.26)$$

multiplications, and

$$\#ADD = \frac{5}{2}m^2 - \frac{7}{2}m + 1 \qquad (5.27)$$

additions, while the Schoolbook method requires m^2 multiplications and $(m-1)^2$ additions.

The choice between using the Schoolbook method or the Karatsuba algorithm to implement extension field multiplication depends on the cost ratio between one multiplication and one addition. If this ratio is high then it is more efficient to use the Karatsuba algorithm; refer to section (4.3.2) for more details on how to compute this ratio. On the other hand the Karatsuba algorithm requires more memory space than the schoolbook method, and so the memory access time is an important issue in the implementation.

Chapter 6

SCALAR MULTIPLICATION

6.1 ELLIPTIC CURVE POINTS SCALAR MULTIPLICATION

Scalar multiplication is the operation of computing kP for a given point P on an elliptic curve and an integer k. The primary operation in an elliptic curve cryptosystem is point multiplication, which is denoted by

$$Q = kP \qquad (6.1)$$

where Q and P are points on an elliptic curve and k is an integer; the cost of executing such cryptosystem depends mostly on the complexity of the scalar multiplication operation. Thus, using efficient algorithms for point multiplication has a strong influence on the performance and the execution time of elliptic curve cryptosystems.

Since scalar multiplication means the addition of P to itself $k-1$ times, the additive notation it could be used to represent kP as:

$$Q = \underbrace{P + P + \cdots + P}_{k-times}. \qquad (6.2)$$

For large k, computing kP is a costly endeavor, and it is inefficient to use a straightforward summation technique that requires $(k-1)$ elliptic additions, so other techniques should be used to efficiently compute this multiplication.

Scalar multiplication on an elliptic curve is analogous to exponentiation in the multiplicative group of integers modulo a fixed integer, which is based on square and multiply algorithm as

shown in section (5.6). The same technique can be used in scalar multiplication by replacing the squaring with doubling and the multiplication with addition over an elliptic curve; this technique is called The Binary-Double-and-Add algorithm. For instance, $25P$ can be computed using this algorithm as:

$$25P = P + 2(2(2(P + (2P)))). \quad (6.3)$$

The binary expansion of 25 is 11001_2, starting the chain with 0. The most significant bit is set, so add P, and then double the result. The next bit is set, so add P and double the result. Now the next bit is clear, so just double the result, and so on until all bits are done. This expansion requires four doubling operations, and two sums, instead of the 24 elliptic addition operations that are required by using the normal summation.

In general, using this algorithm to compute the scalar multiplication by k with l bits requires $(l-1)$ doubling operation and an average $(l-1)/2$ addition; thus it requires $3(l-1)/2$ operations.

The complexity of scalar multiplication can be reduced by using the Addition-Subtraction method which reduces the required number of addition operations. This method uses Balanced Binary Expansion (or signed binary expansion), in which no two consecutive coefficients are nonzero, instead of using the normal binary expansion that is used by the binary algorithm. For instance, the binary expansion of 10045 is $1\,0\,0\,1\,1\,1\,0\,0\,1\,1\,1\,1\,0\,1$, while the balanced binary expansion is $1\,0\,1\,0\,0\,-1\,0\,1\,0\,0\,0\,-1\,0\,1$. Algorithm (4.4) shows how to use the Non Adjacent Form (NAF) method to find a balance binary expansion.

The operation of the Addition-Subtraction method is the same as the previous algorithm except that subtract P when the bit is -1. Algorithm (6.1) illustrates how point multiplication is done using the Addition-subtraction method. The inputs are the point P, and the signed binary expansion of k (NAF(k)), which is represented as

$$k = \sum_{j=0}^{l-1} k_j 2^j \quad (6.4)$$

where l is the bit length of k

During the loop there are two operations. The value of k_i determines which operation will be done; if k_i is 1 then the operation is addition, while if k_i is -1 then the operation is subtraction, and always doubles the result.

Algorithm 6.1	Addition-Subtraction method
Require:	A point $P \in E(GF(p^m))$, $NAF(k) = \sum_{i=0}^{l-1} k_i \, 2^i$.
Ensure:	$Q = kP$.
1:	$Q \leftarrow P$
2:	for $i \leftarrow l-2$ downto 1
3:	$Q \leftarrow 2P$
4:	if $k_i = 1$
5:	$Q \leftarrow Q + P$
6:	end if
7:	if $k_i = -1$
8:	$Q \leftarrow Q - P$
9:	end if
10:	end for

The number of additions is approximately the number of nonzero coefficients in NAF(k); the average number of nonzero coefficients in NAF(k) is $l/3$ [44]. Since this method requires at most l doublings, the average complexity of the Addition-Subtraction method is approximately l doublings and $l/3$ additions, for a total of $4l/3$ elliptic operations, as compared with the binary method. This is about one-eighth faster, since it reduces the number of elliptic additions from $(l-1)/2$ to $l/3$. Table (6.1), shows the complexity of the three methods that are used to compute the scalar multiplication.

Table 6.1 Complexity of Scalar Multiplication kP, where $\log_2 k = l$

Method	#ADDITION	#DOUBLING
Normal Summation	$k-1$	-
Binary Algorithm	$(l-1)/2$	$(l-1)$
Addition-Subtraction Method	$l/3$	l

Chapter 7

EMBEDDING DATA ON AN ELLIPTIC CURVE

The previous chapters have shown the arithmetic operations over the finite field, and point multiplications that are required in elliptic curve cryptography implementation. Before we go further into the cryptography techniques, we explain how to map data strings onto a curve, and how we can ensure that values picked at random satisfy the elliptic curve equation. This chapter describes how to embed random data and a normal message on elliptic curve and how to solve the quadratic equation.

7.1 EMBEDDING A POINT ON A CURVE

Embedding data on a curve does not mean encrypting that data; it simply encodes data as points on a given elliptic curve E defined over a finite field $GF(p^m)$. Consider the following elliptic curve equation:

$$y^2 = x^3 + a_4 x + a_6 . \qquad (7.1)$$

By converting the right side to a simple form $f(x)$

$$y^2 = f(x). \qquad (7.2)$$

Then the new equation is a simple quadratic equation. Later this chapter shows how to solve such a quadratic equation.

Even though the elliptic curve E is defined over the finite field $GF(p^m)$, there are many elements in this field that do not satisfy equation (7.2). Thus more bits are required in the finite field than the actual data, otherwise not all of the data will be on the curve and the cryptosystem is not useful.

There are several approaches to embed data on a curve. One of the relatively straightforward techniques, is the method proposed by Koblitz [16], which makes use of the fact that the density of points for any curve over a finite field is almost uniformly distributed, which means any subsection of the bits can be seen as an integer and increment it using simple arithmetic. For instance, for arbitrary x data, first check if it is on the curve, i.e. x satisfies equation (7.2); if not, increment x and perform the test again until the equation (7.2) can be solved. Then two values of y are found, and the points $P(x, y)$ and $P(x, -y)$ are on the curve.

Figure (7.1) shows the required procedures to embed a random point on elliptic curve; all of these steps will be explained in the next sections.

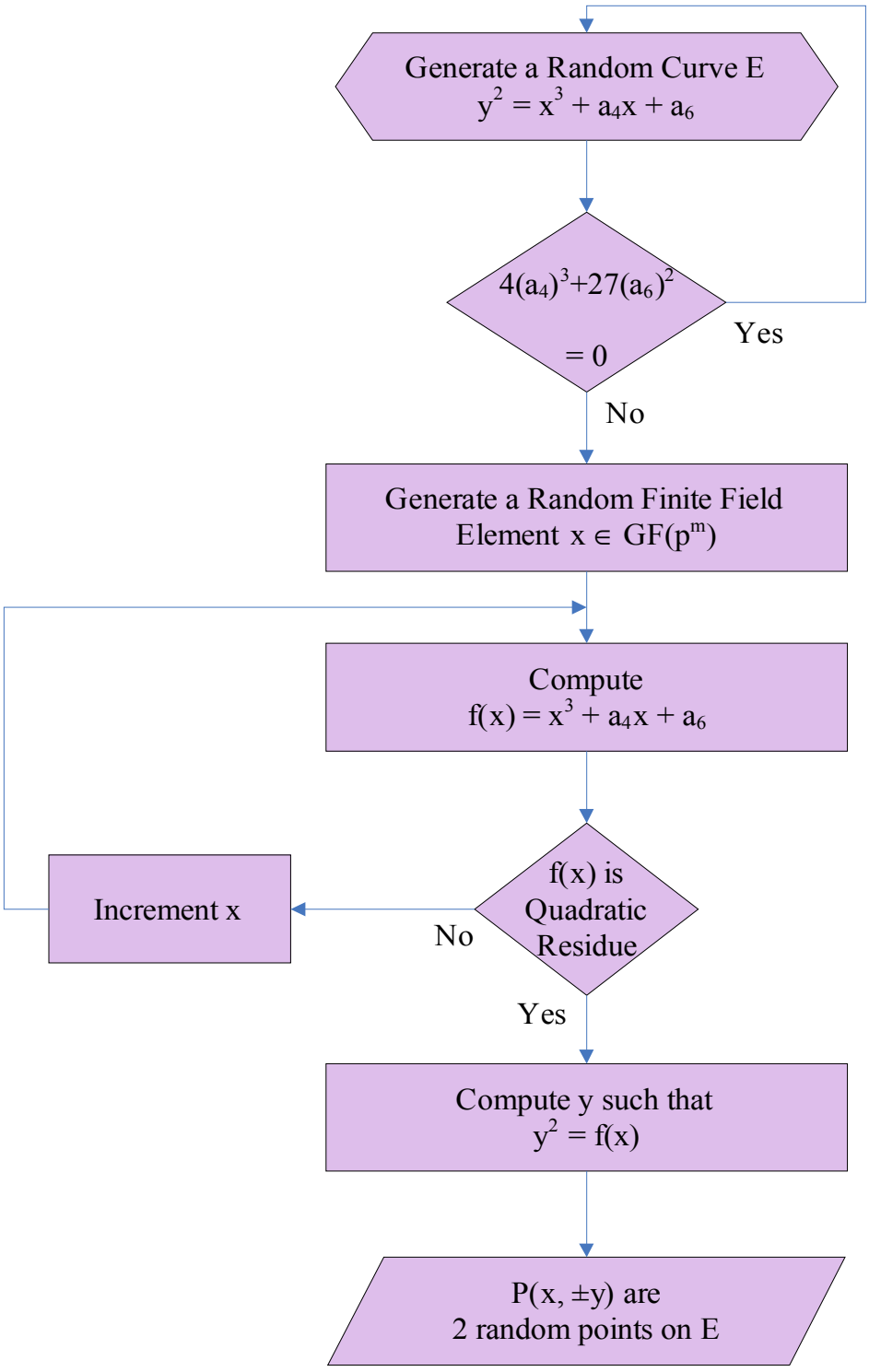


Figure 7.1 Procedures of Embedding Random Point on Elliptic Curve *E*

7.2 SOLVING THE SQUARE ROOT EQUATION

To find a point on an elliptic curve $E\left(GF\left(p^m\right)\right)$ for a given x-coordinate, the corresponding y-coordinate must be calculated by substituting the x-coordinate into the elliptic curve equation (7.2), which returns the value of y^2, so we need to find the square root of y^2.

Cohen reports in [47] that if p is an odd prime number, and a is a quadratic residue[7], then there exists an x such that $x^2 \equiv a \pmod{p}$. In a finite field $GF\left(p^m\right)$ where the prime p satisfies that $p \equiv 3 \pmod{4}$ and m is odd, the solution is given by

$$x = a^{\left(p^m+1\right)/4} \pmod{p}. \qquad (7.3)$$

But in order to find the value of x, first notice that if $m = 2k+1$, for some k [48]

$$\frac{p^m+1}{4} = \frac{p+1}{4}\left[p(p-1)\sum_{i=0}^{k-1}\left(p^2\right)^i + 1\right] \qquad (7.4)$$

so that

$$a^{\left(p^m+1\right)/4} = \left(\left(a^{\sum_{i=0}^{k-1}\left(p^2\right)^i}\right)^{p(p-1)} \cdot a\right)^{(p+1)/4}. \qquad (7.5)$$

These relations can be verified by straightforward induction. The quantity

$$a^{\sum_{i=0}^{k-1}u^i}$$

where $u = p^2$ can be efficiently computed in a technique analogous to the Itoh-Tsujii Inversion, based on the Frobenius map in characteristic p as

[7] The next section describes how to check if a certain value is a quadratic residue

$$a^{1+u+\cdots+u^{k-1}} = \begin{cases} \left(a^{1+u+\cdots+u^{\lfloor k/2\rfloor-1}}\right)\cdot\left(a^{1+u+\cdots+u^{\lfloor k/2\rfloor-1}}\right)^{u^{\lfloor k/2\rfloor}}, & k \text{ even} \\ \left(\left(a^{1+u+\cdots+u^{\lfloor k/2\rfloor-1}}\right)\cdot\left(a^{1+u+\cdots+u^{\lfloor k/2\rfloor-1}}\right)^{u^{\lfloor k/2\rfloor}}\right)^{u}\cdot a, & k \text{ odd} \end{cases} . \qquad (7.6)$$

Exponentiation to a power of p is a linear operation in characteristic p. After computing this quantity, a multiplication by a and exponentiation to $p-1$ and $(p+1)/4$ are required to complete the square root evaluation. The conventional exponentiation algorithm may be used to compute these exponentiations; however, if p is large, it may be an advantage to compute z^{p-1} as $z^{p}\cdot z^{-1}$.

Consider

$$A(x)\in GF(p^m)$$

where $m = 2\times 8+1$, so $k = 8$, assuming that A is a quadratic residue.

Using equation (7.6) to compute $A^{\sum_{i=0}^{k-1}u^i}$ as

$$A^{1+u+\cdots+u^7} = \left(A^{1+u+u^2+u^3}\right)\cdot\left(A^{1+u+u^2+u^3}\right)^{u^4}, \qquad k_0 = 8$$

$$A^{1+u+u^2+u^3} = \left(A^{1+u}\right)\cdot\left(A^{1+u}\right)^{u^2}, \qquad k_1 = 4$$

$$A^{1+u} = A\,.\,(A)^{u}, \qquad k_2 = 2\,.$$

Algorithm (7.1) shows how to compute $A^{\sum_{i=0}^{k-1}u^i}$ using the Frobenius map.

Algorithm 7.1	Computing $A^{\sum_{i=0}^{k-1} u^i}$ in $GF(p^{17})$	
Require:	$A(x) \in GF(p^m)$, where $m = 2k+1$, $k = 8$.	
Ensure:	$C(x) \equiv A^{\sum_{i=0}^{k-1} u^i}$, where $C(x) \in GF(p^m)$, $u = p^2$.	
1:	$C_0 \leftarrow A^{p^2} = A^u$	$\sigma^2(A)$
2:	$C_1 \leftarrow C_0\ A = A^{1+u}$	Multiplication
3:	$C_2 \leftarrow (C_1)^{p^4} = (A^{1+u})^{u^2}$	$\sigma^4(C_1)$
4:	$C_3 \leftarrow C_2\ C_1 = A^{1+u+u^2+u^3}$	Multiplication
5:	$C_4 \leftarrow (C_3)^{p^8} = (A^{1+u+u^2+u^3})^{u^4}$	$\sigma^8(C_3)$
6:	$C \leftarrow C_4\ C_3 = A^{1+u+u^2+\cdots+u^7} = A^{\sum_{i=0}^{k-1} u^i}$	Multiplication

Then use equation (7.5) to complete the evaluation of the square root of A. Algorithm (7.2) shows the details of computing the square root of an arbitrary element $A(x) \in GF(p^m)$, using equation (7.5).

Algorithm 7.2	OEF Square Root	
Require:	$A(x) \in GF(p^m)$, where $m = 2k+1$, k is integer.	
Ensure:	Find $C(x)$, such that $C^2(x) \equiv A(x) \pmod p$, where $C(x) \in GF(p^m)$.	
1:	$C_0 \leftarrow A^{\sum_{i=0}^{k-1} u^i}$	Equation (7.6)
2:	$C_1 \leftarrow C_0^p$	$\sigma(C_0)$
3:	$C_2 \leftarrow C_1^p$	$\sigma(C_1)$
4:	$C_3 \leftarrow (C_1)^{-1}$	Inversion
5:	$C_4 \leftarrow C_2\ C_3$	Multiplication
6:	$C_5 \leftarrow C_4\ A$	Multiplication
7:	$C \leftarrow (C_5)^{(p+1)/4}$	Exponentiation

This section gives an efficient method for finding a square root in Optimal Extension Fields $GF(p^m)$ where the prime p satisfies $p \equiv 3 \pmod 4$ and m is odd. A similar technique can be used to compute the square root when $p \equiv 5 \pmod 8$. The remaining case is $p \equiv 1 \pmod 8$, which is the most complex case. In the last case Shanks's algorithm can be applied to find the square root without benefit from this technique.

7.3 QUADRATIC RESIDUE

Let p be an odd prime, then the congruence

$$x^2 \equiv a \pmod p \qquad (7.7)$$

for a given a has three cases. Firstly there is no solution in this case, a is a quadratic non-residue modulo p. Secondly there is one solution if $a \equiv 0 \pmod p$. Finally there are two solutions in this case, a is a quadratic residue modulo p. A simple way of identifying whether or not an integer is a quadratic residue modulo p is the Legendre symbol.

The Legendre symbol [16]

Let a be an integer and $p > 2$ a prime. We define the Legendre symbol (a/p) as

$$\left(\frac{a}{p}\right) = \begin{cases} 0, & if\ p \mid a \\ 1, & if\ a\ is\ a\ quadratic\ residue\ \mathrm{modulo}\ p \\ -1, & if\ a\ is\ a\ quadratic\ nonresidue\ \mathrm{modulo}\ p \end{cases}. \qquad (7.8)$$

Then the number of solutions modulo p of the above congruence is $1 + (a/p)$.

By Fermat's Little theorem, in $GF(p)$ the square of $a^{(p-1)/2}$ is 1, so $a^{(p-1)/2}$ itself is ± 1. Hence the following congruence can be proved:

$$\left(\frac{a}{p}\right) \equiv a^{(p-1)/2} \pmod{p}. \qquad (7.9)$$

In $GF(p^m)$, if $m = 2k+1$, for some k

$$\frac{p^m - 1}{2} = \frac{p-1}{2}\left[p(p+1)\sum_{i=0}^{k-1}(p^2)^i + 1\right] \qquad (7.10)$$

so that

$$a^{(p^m-1)/2} = \left(\left(a^{\sum_{i=0}^{k-1}(p^2)^i}\right)^{p(p+1)} \cdot a\right)^{(p-1)/2}. \qquad (7.11)$$

Again the quantity $a^{\sum_{i=0}^{k-1}u^i}$ can be computed using equation (7.6) as shown in the previous section. Algorithm (7.3) shows the details of computing the value of the Legendre symbol:

Algorithm 7.3	Legendre Symbol	
Require:	$A(x) \in GF(p^m)$, where $m = 2k+1$, k is integer.	
Ensure:	Find $c = (a/p) \equiv A^{(p-1)/2} \pmod{p}$.	
1:	$C_0 \leftarrow A^{\sum_{i=0}^{k-1}u^i}$	Equation (7.6)
2:	$C_1 \leftarrow C_0^p$	$\sigma(C_0)$
3:	$C_2 \leftarrow C_1^p$	$\sigma(C_1)$
4:	$C_3 \leftarrow C_1\, C_2$	Multiplication
6:	$C_4 \leftarrow C_3\, A$	Multiplication
7:	$c \leftarrow (C_4)^{(p-1)/2}$	Exponentiation

7.4 RANDOM NUMBER GENERATOR

Many aspects of elliptic curve cryptosystems require large random numbers: OEF generation, elliptic curve generation, base element generation, encryption process, and so on.

The concept of randomness has little meaning for a digital computer, since it is designed to reproduce the same output for any given input and because of its simplicity it does not produce very random numbers. What is useful for cryptographic purposes is that an attacker cannot easily guess the next bit knowing all the previous data.

The random number generator used here was developed by Dr. Marsagila [49]; he reports that the repeat period is approximately 2^{250}. He calls his code "the mother of all random bit generators".

The internal state of the generator determines the output. An initial seed of 32-bit is used to create the initial internal state, so this should be as random as possible to begin with. Two arrays store carry values in their first element, and random 16-bit numbers in elements 1 to 8. These random numbers are moved to elements 2 to 9 and a new carry and number are generated and placed in elements 0 and 1, the two arrays are filled with random 16-bit values on first call of the random generator. A 32-bit random number is obtained by combining the output of the two generators.

To generate a finite field element $A(x)=\sum_{i=0}^{m-1} a_i x^i$, generate a random value for each a_i which is in $GF(p)$, so a reduction algorithm[8] should be used at the end of the random generator to reduce a 32-bit to a $GF(p)$ element.

7.5 EMBEDDING PLAINTEXT

The previous sections have shown how to embed a random point on an elliptic curve, but what about embedding a plaintext on a curve? The situation is different, because during the encryption process a plaintext must be embedded in a point on the curve in such a way that it

[8] Based on the technique used in section (5.2.2)

can be retrieved successfully from that point when decrypted, and so each message must correspond to a unique point and vice versa. However, the previous technique may be used here with some modification.

The first step is to divide the plaintext T into blocks t with fixed length l, and then represent each block as a finite field element by converting the string t to the OEF element x_t. The length of each block l, should be the maximum value that can be chosen, so that when we convert t to finite field element it could be represented as

$$t_{m-2}x^{m-2} + t_{m-3}x^{m-3} + \cdots + t_1 x + t_0 . \qquad (7.12)$$

Let $j = t_{m-1}$, then

$$x_t = j\, x^{m-1} + \sum_{i=0}^{m-2} t_i x^i \ . \qquad (7.13)$$

Compute the right side $f(x)$ of equation (7.2) for x_t, then check if $f(x)$ is a quadratic residue using algorithm (7.3); if so find y_t which is the square root of $f(x)$ using algorithm (7.2), then the point $P_t(x_t, y_t)$ represents the embedded message block t. If $f(x)$ is a quadratic non-residue, then increment j by 1 and try again with the new x_t. Koblitz proves in [16] that an x_t can be found for which $f(x)$ is a square before j gets to the maximum value of t_{m-1}.

It is easy to recover t from the point $P_t(x_t, y_t)$. All that one needs to do is simply clear the last term of x_t, which is t_{m-1}, and then convert x_t back to a string t.

7.6 DATA CONVERSION

All the arithmetic operations and the internal routines were used to implement an elliptic curve cryptosystem in this dissertation deal with variables, which are elements of the Optimal Extension Field. In practice it is required to convert these field elements to integer or octet strings, for example, the ECC may be used to generate a key that could be used for a

symmetric cipher, so that this key should be represented as an integer. In another example, to encrypt a message, then it should be converted from the octet string to a finite field element.

In the OEF, $p = 2^n - c$, where n is the size of the microcontroller word, so each element of the field can be represented in m microcontroller words. However, for each word, there will be only $2^n - c$ possible values instead of 2^n; thus the number of possible octet strings formed from the concatenation of m words is reduced by cm due to the representation.

In general, to convert finite field element to integer or octet string representation, perform radix conversion arithmetic. Thus the field element

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 \qquad (7.14)$$

may be represented by the integer I as

$$I = a_{m-1}p^{m-1} + a_{m-2}p^{m-2} + \cdots + a_1 p + a_0 . \qquad (7.15)$$

7.6.1 Converting Between OEF Elements and String of Decimal Digits

To convert the OEF element to an integer I as shown in equation (7.15) I will have a bit length of $\lceil m \log_2 p \rceil$, which is larger than the microcontroller word, and thus it can not be represented directly in a single variable. It is possible to represent this integer as ASCII string of decimal digits.

To convert an element A to an ASCII string S, repeatedly divide A by 10: the remainder at each step will be the i-th element of S. After each step, set the new value of A to the quotient and increment i, and at the end reverse the string S.

To find the element A, for each element of S, multiply A by 10 and add the i-th element to it. For example, the following ASCII string of decimal digits:

S = 26295515083977814746041579119561726815 04

can be represented as the following OEF element in $GF\left((239)^{17}\right)$

$$A(x) = 23x^{16} + 48x^{15} + 35x^{14} + 107x^{13} + 231x^{12} + 109x^{11} + 111x^{10} + 52x^{9} + 65x^{8} + 166x^{7} + 232x^{6} + 89x^{5} + 110x^{4} + 128x^{3} + 84x^{2} + 116x + 228.$$

7.6.2 Converting between OEF elements and octet string

Encrypting a message using ECC requires at first to represent this message as a finite field element, while the opposite conversion is required in the case of a decryption process to retrieve the original message.

If a message M is large, then the first step is to divide M into blocks m_i of fixed size l

$$l = \left\lfloor \frac{\left(m \log_2\left(2^n - c\right)\right) - n}{8} \right\rfloor \qquad (7.16)$$

where $\lfloor a \rfloor$ is the biggest integer $\leq a$.

The next step is to convert each message block m_i to a finite field element. The same technique that was used in the previous section will be used, except that the division is by 256, and the conversion back to octet string uses multiplication by 256. For example, the following message

M = "hello world"

can be represented as the following OEF element in $GF\left((239)^{17}\right)$

$$A(x) = 207x^{10} + 129x^{9} + 108x^{8} + 203x^{7} + 89x^{6} + 145x^{5} + 128x^{4} + 37x^{3} + 40x^{2} + 94x + 72.$$

Chapter 8

ELLIPTIC CURVE CRYPTOGRAPHY

8.1 INTRODUCTION

After having covered all the math of finite fields, elliptic curves, and the basics of getting real data onto curve in the previous chapters, now it is the time to show how to use elliptic curve as cryptosystem and to actually hide data.

Although the use of elliptic curve cryptosystems is relatively new, they are a popular research area, and more secure than previous cryptosystems because the analogue of the discrete logarithm problem on elliptic curves is more complex than the classical discrete logarithm problem.

The elliptic curve cryptosystems do not use new cryptographic algorithms with elliptic curves over finite fields, but they implement existing algorithms that are similar to Diffie-Hellman and ElGamal algorithms. This chapter describes two simple elliptic curve cryptography protocols that analogue to Diffie-Hellman and ElGamal.

The elliptic curve cryptosystems have domain parameters, which determine the arithmetic operations on elliptic curves over finite fields; these domain parameters can be summarized as

- q: prime power used in the finite field $GF(q)$, that is $q = p^m$, where p is a prime.
- Field representation of the method used for representing field elements in $GF(p^m)$.

- a_4, a_6: field elements, they specify the equation $y^2 = x^3 + a_4 x + a_6$, of the elliptic curve E over $GF(p^m)$.

- G: a base point represented by (x_g, y_g) on $E(GF(p^m))$.

- n: order of the point G, that is n is the smallest positive integer such that $nG = O$.

Similar to the other public key cryptosystems, elliptic curve cryptography requires two entities, one at the encryption side and the other at the decryption side. Thus each user should have a public key and private key. In general, one can generate these two keys using elliptic curve cryptography as follows

- Choose a suitable finite field $GF(p^m)$.

- Generate a random elliptic curve E over the finite field $GF(p^m)$.

- Select a random finite field element k, where $k \in GF(p^m)$.

- Generate a base point G in the elliptic curve E.

- Compute $P = k\,G$, where P is a point on E, and $P \neq O$.

- k is the private key, while P is the public key.

8.2 ADVANTAGES OF ELLIPTIC CURVE CRYPTOGRAPHY

Elliptic curve cryptography offers several advantages, especially for those systems requiring strong security in constraint devices to give the most security per bit. A white paper by Certicom [50] reports some of these advantages:

- ECC offers considerably greater security for a given key size.

- The smaller key size also makes possible much more compact implementations for a given level of security, which means faster cryptographic operations, running on smaller chips or more compact software. This means less heat production and less power consumption, all of which is of particular advantage in constrained devices, but of some advantage anywhere.

- There are extremely efficient, compact hardware implementations available for ECC exponentiation operations, offering potential reductions in implementation footprint even beyond those due to the smaller key length alone.

- ECC provides longer running battery operated devices that produce less heat.

- ECC provides software applications that run faster and take up less memory.

- ECC provides scalable cryptography for the future.

On the other hand, the main advantage of using the underlying finite field is that different elliptic curves can be selected or can be changed at any time for security reasons, without changing or modifying the underlying finite field.

8.3 THE ELLIPTIC CURVE DISCRETE LOGARITHM PROBLEM

Each cryptosystem is based on a complex problem that makes the cryptosystem hard to attack. For example, the security of RSA depends on the hardness of extracting modular *e*-th roots, while the ElGamal cryptosystem bases its security on the intractability of the Discrete Logarithm Problem (DLP).

The Discrete Logarithm Problem can be defined as follows: given $a^i = b$, find the particular value for i that satisfies the equation, namely $i = \log_a b$. When solving for the Discrete Logarithm Problem over a prime field, this means finding the integer i for which $a^i \equiv b \pmod{p}$, where p is a prime.

The Discrete Logarithm Problem over a finite field group can be defined as follows:

Definition 8.1 [16]

If G is a finite field group, b is an element of G, and y is an element of G, which is a power of b, then the Discrete Logarithm of y to the base b is any integer x such that $b^x = y$.

ECC uses a group logarithm problem too, but it differs in the method by which the group is defined, and how the fundamental operations on the group are defined. The following is an analogue definition for the group of points on an elliptic curve:

Definition 8.2 [16]

If E is an elliptic curve over $GF(q)$ and G is a point of E, then the Discrete Logarithm Problem on E (to the base G) is the problem, given a point $P \in E$, of finding an integer $k \in Z$ such that $P = k\,G$, if such an integer k exists.

Point G is called the base point in ECDLP, while $k = \log_G P$ is the Elliptic Curve Discrete Logarithm of P to the base point G.

The strength of security of the Elliptic Curve Cryptography lies in the Elliptic Curve Discrete Logarithm Problem. ECDLP is believed to be unsolvable in sub-exponential time, while there are already algorithms to solve the DLP in sub-exponential time. On the other hand the ECDLP is more complex than the integer factorization problem.

Now, the superficially obvious, certain way of finding k would be to perform repeated addition operations stepping through G, $2G$, $3G$, and so on, until $k\,G$ is found. One would start by doubling G, then adding G to $2G$ finding $3G$, then $3G$ to G finding $4G$ and so on. This is the brute force method. The problem with this is, if one uses a large enough prime field, the

number of possible values for k becomes inconveniently large, so inconveniently large that it is quite practical to create a sufficiently large prime field that searching through the possible values of k would take all the processor time currently available on the planet thousands of years [50].

There are many possible algorithms to use for encryption with elliptic curves, and many discrete logarithm protocols can be converted to use elliptic curves, such as Diffe-Hellman and ElGamal.

8.4 THE DIFFIE-HELLMAN PROTOCOL

The Diffie-Hellman protocol is the original public key cryptosystem proposed for secret sharing, and it was the first public key algorithm ever invented. This algorithm was invented in 1976 by Whitfield Diffie and Martin E. Hellman [6]. It gets its security from calculating discrete logarithms in a finite field. The idea behind the Diffie-Hellman algorithm is to generate private information over an insecure communication channel, and share it in a secure fashion. This information can then be used as a private key in a symmetric key cryptosystem such as DES.

8.4.1 Diffie-Hellman Key Exchange

To see how the Diffie-Hellman key exchange works, assume that there are two users, A and B, who want to use this algorithm to generate and distribute a secret key:

- There are two publicly known numbers: a prime number q and an integer α that is a primitive root of q. A and B agree on these public elements.
- User A selects a random integer $X_A < q$, and computes $Y_A = \alpha^{X_A} \bmod q$.
- User B selects a random integer $X_B < q$, and computes $Y_B = \alpha^{X_B} \bmod q$.
- Each side keeps X value private and makes the Y value publicly available to the other side.

- User A computes $k_A = (Y_B)^{X_A} \bmod q$.

- User A computes $k_B = (Y_A)^{X_B} \bmod q$.

The keys k_A and k_B are identical

$$k_A = (Y_B)^{X_A} \bmod q = (\alpha)^{X_A X_B} \bmod q = (Y_A)^{X_B} \bmod q = k_B \qquad (8.1)$$

so both side share the same key

$$k = (\alpha)^{X_A X_B} \bmod q . \qquad (8.2)$$

The security of the Diffie-Hellman key exchange lies in the fact that, while it is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible. The Diffie-Hellman key exchange protocol can be easily extended to three or more people. More information can be found in [7].

8.4.2 Elliptic Curve Diffie-Hellman (ECDH)

The Diffie-Hellman key exchange algorithm can easily be implemented using elliptic curve. Suppose there are two users, A and B; then to distribute a key between them using Elliptic Curve Diffie-Hellman can be done in the following manner, figure (8.1):

- They first publicly choose a finite field $GF(p^m)$.

- They agree on an elliptic curve E over a finite field $GF(p^m)$.

- They pick a base point G in the elliptic curve E, which is also a public element.

- User A chooses a random finite field element $k_A \in GF(p^m)$, which is A's private key.

- User B chooses a random finite field element $k_B \in GF(p^m)$, which is B's private key.

- A generates a public key $Q_A = k_A \times G$, where Q_A is a point on the elliptic curve E, then sends it to B.

- B generates a public key $Q_B = k_B \times G$, where Q_B is a point on the elliptic curve E, then sends it to A.

- Now A generates the secret key $P_A = k_A \times Q_B$.

- Similarly, B generates the secret key $P_B = k_B \times Q_A$.

It is easy to show that the secret keys are the same, so they share the same key

$$P_s = (k_A\ k_B) \times G \qquad (8.3)$$

P_s is a point in elliptic curve E that could be represented as (x_s, y_s). For reasonable security, it is important to use only the x_s value, since the value of y_s could be recovered from the elliptic curve equation, as shown in the previous chapter. However, 1 bit of y_s is necessary in order to determine which root will be used: y_s or $-y_s$.

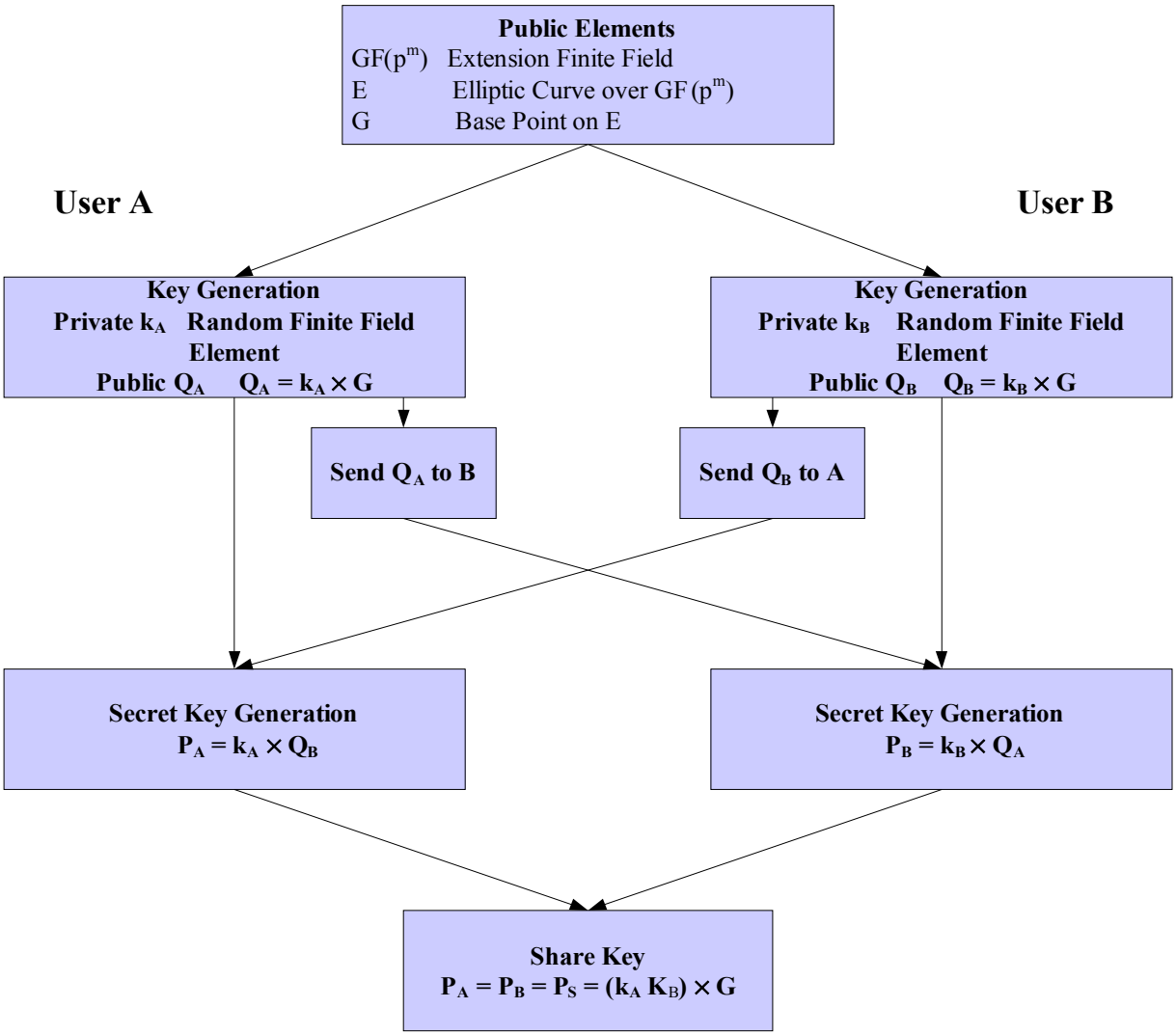


Figure 8.1 Elliptic Curve Diffie-Hellman (ECDH)

The security of ECDH is gained by the intractability of point P_s without solving the Elliptic Curve Diffie-Hellman Problem (ECDHP), which is defined as follows:

Definition 8.3

Given G, $Q \in E$, such that: $Q = k \times G$, where k is a finite field element. The ECDHP is to find k.

Thus it seems infeasible to compute P_s, k_A and k_B knowing only G, $k_A G$ and $k_B G$. Functions such as ECDH are known collectively as one-way function [6].

The advantage of this algorithm is that, after generating the secret key x_s, it could be used in a symmetric key cryptosystem for a period of time before regenerate a new key. Since this key can be seen as NAF representation, then the Addition-Subtraction method can be used to compute the scalar multiplication, as shown in chapter 6, and so the computations are speeded up.

8.5 THE ELGAMAL PROTOCOL

The ElGamal protocol is another public-key cryptosystem using the discrete logarithm problem as its core. To generate a key pair in the ElGamal cryptosystem, choose a large prime p, and another two numbers, x and g that are smaller than p. x is used as a private key. Now compute the public key y as:

$$y = g^x \bmod p . \qquad (8.4)$$

It is easy to compute y as shown in the equation above, but it is infeasible to find x given y, g and p by the property of discrete logarithm problem. Encryption using the public key and decryption using private key is an exponentiation operation. However decryption using the public key would require performing the difficult inverse operation especially for a large enough p prime value.

8.5.1 Elliptic Curve Analogue to ElGamal Protocol

The ElGamal cryptosystem is another popular system that can be implemented with elliptic curve. It is a very useful protocol for randomly generated curves and points, because it does not require knowledge of the order of the curve, the factors of that number, or the order of the base point. Another advantage is that it is not patented.

To generate a key pair in the ElGamal cryptosystem the same procedures used in Diffie-Hellman protocol would be followed. An elliptic curve E defined over a finite field $GF(p^m)$ is chosen, a base point, G which is public, must be chosen, each user choosing a random finite field element k_A and k_B, which are the private keys for user A and B respectively. Then each user computes his public key and sends it to the other user:

$$P_A = k_A \times G \qquad (8.5)$$

$$P_B = k_B \times G \qquad (8.6)$$

which are the public key of A and B respectively. Each user can encrypt and send messages using these public key points in such a way that no one can discover the data without solving the discrete elliptic curve logarithm problem.

Encryption Process

- User A embeds the message information onto the curve, E, as a message point P_m
- User A chooses a random finite field element r.
- User A computes the two points

$$P_r = r \times G \qquad (8.7)$$

and

$$P_h = P_m + r \times P_B . \qquad (8.8)$$

- Then user A sends both points P_r and P_h to B.

Decryption Process

- B computes the point.

$$P_s = k_B \times P_r . \qquad (8.9)$$

- B subtracts P_s from P_h to get the message point P_m.

$$P_m = P_h - P_s . \qquad (8.10)$$

- B recovers the original message from the message point P_m.

To see how this works: in equation (8.8), rP_B can be expanded as $r(k_B\ G)$, then the term P_h in equation (8.8) is rewritten as:

$$P_h = P_m + r(k_B \times G). \qquad (8.11)$$

Substitute equation (8.7) into equation (8.9), and combine this with equation (8.11) into equation (8.10), which gives:

$$P_m = P_m + r(k_B \times G) - k_B(r \times G) = P_m . \qquad (8.12)$$

Figure (8.2) shows the complete procedure of generating a pair key and encryption and decryption processes in the elliptic curve cryptography analogue to the ElGamal protocol.

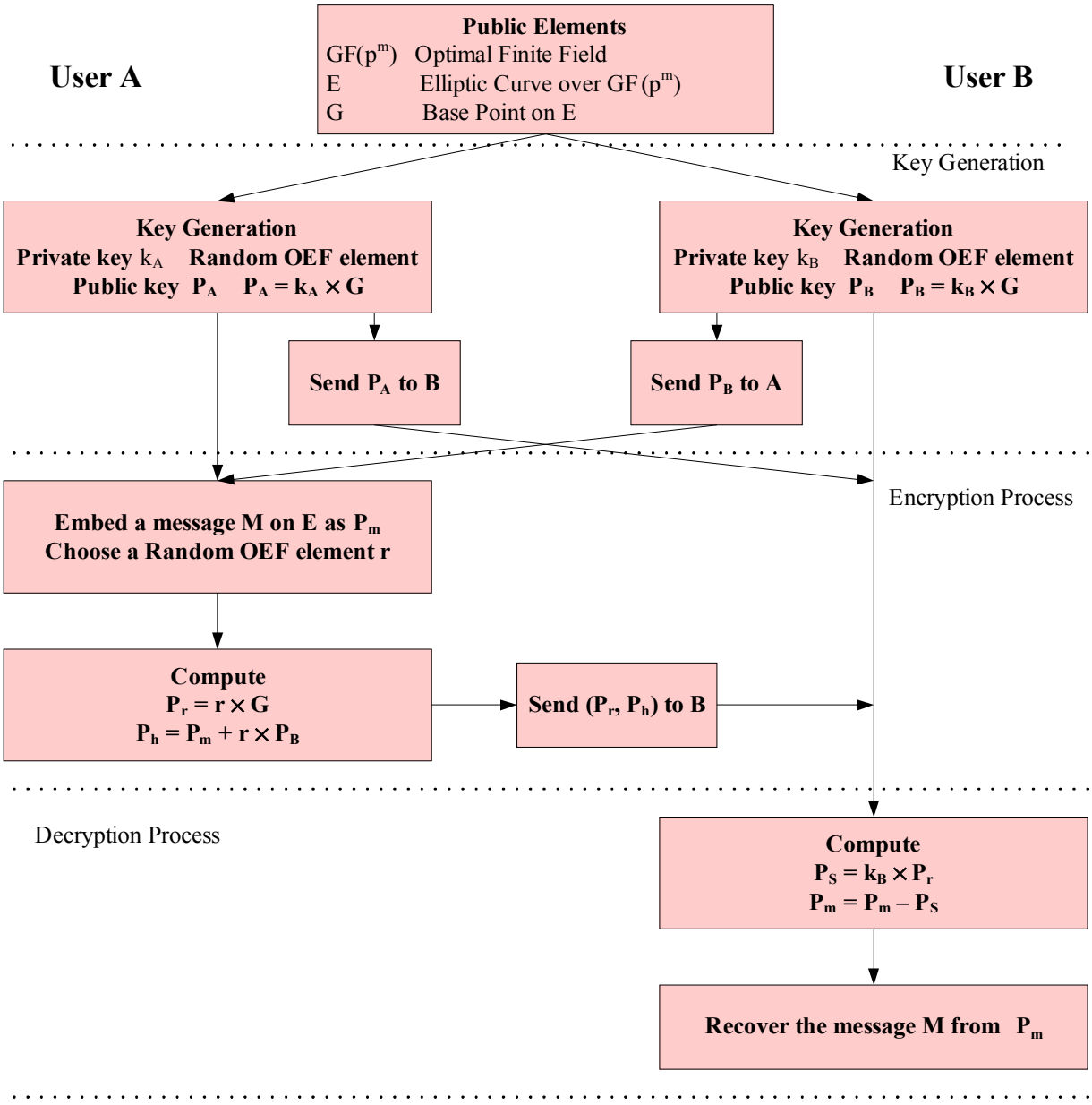


Figure 8.2 Elliptic Curve Analogue to ElGamal Protocol

One advantage of this protocol is that the public keys can stay public: there is no need to change them. Every time data are exchanged, a new random value r is chosen. Neither of the sides needs to remember r, and, if the field size is large enough, it will be very difficult to discover the secret finite field elements k_A or k_B. This provides both secret sharing and authentication.

There are some drawbacks to this protocol. This system requires a message expansion of 2, since a point message P_m is encrypted as (P_r, P_h). Another drawback, is that in this protocol a man-in-the-middle attack is possible. One possible solution to eliminate this attack is by verifying the public keys via an alternate channel such as a telephone in relatively small environments, or using certificates for large environments like the Internet.

Chapter 9

IMPLEMENTATION

This chapter discusses the implementation details of an elliptic curve cryptosystem over optimal extension fields. It starts by giving a brief description of the ATmega128 microcontroller that is used in the implementation, and then it describes the optimal extension field. The elliptic curve cryptography design architecture, the data structure and the functions used in the implementation are presented. The optimal extension field arithmetic performance and elliptic curve group operation timing are summarized at the end of the chapter.

9.1 ATMEGA128 MICROCONTROLLER

The ATmega128 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega128 achieves throughputs approaching 1 MIPS per MHz allowing the system designer to optimize power consumption versus processing speed.

The AVR core combines a rich instruction set with 32 general-purpose working registers. All the 32 registers are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in one single instruction executed in one clock cycle. The resulting architecture is more code efficient while achieving throughputs up to ten times faster than conventional CISC microcontrollers.

The ATmega128 provides the following features

- 128K bytes of In-System Programmable Flash with Read-While-Write capabilities.

- 4K bytes EEPROM.
- 4K bytes SRAM.
- 53 general-purpose I/O lines.
- 32 general-purpose working registers.
- Real Time Counter (RTC).
- 4 flexible Timer/Counters with compare modes and PWM.
- 2 USARTs.
- A byte oriented Two-wire Serial Interface.
- An 8-channel.
- 10-bit ADC with optional differential input stage with programmable gain
- Programmable Watchdog Timer with Internal Oscillator.
- An SPI serial port.
- IEEE std. 1149.1 compliant JTAG test interface.
- 6 software selectable power saving modes.
- 0 - 16 MHz.

The device is manufactured using Atmel's high-density nonvolatile memory technology. By combining an 8-bit RISC CPU with In-System Self-Programmable Flash on a monolithic chip, the Atmel ATmega128 is a powerful microcontroller that provides a highly flexible and cost effective solution to many embedded control applications [51].

9.2 THE UNDERLYING FINITE FIELD

The first step of implementation is the selection of the underlying finite field. An extension finite field is identified with the notation $GF(p^m)$ for p a prime and m a positive integer. This field is isomorphic to $GF(p)[x]/P(x)$, where $P(x) = x^m + \sum_{i=0}^{m-1} p_i x^i$, $p_i \in GF(p)$, is a monic irreducible polynomial of degree m over $GF(p)$.

In the implementation, the Optimal Extension Field has used as the underlying finite field, in which a special form of p has been chosen as $2^n - c$, where n is the microcontroller word size, which is equal to 8 in ATmega128 microcontroller, and c is an arbitrary positive integers that satisfies $\log_2 c \leq \lfloor \frac{1}{2} n \rfloor$ and is chosen to get a value of p that is close to the microcontroller word value. In addition, m is chosen so that an irreducible binomial $P(x) = x^m - w$ exists, where $w \in GF(p)$. The choices of p, m, w and $P(x)$ can have a dramatic impact on the performance of the ECC. The following parameters are used in the implementation:

$$n = 8$$

$$c = 17$$

$$p = 2^8 - 17 = 239$$

$$m = 17$$

$$w = 2 .$$

So the OEF is

$$GF\left((2^8 - 17)^{17}\right)$$

and the irreducible binomial is

$$P(x) = x^{17} - 2.$$

This field has an order of about 2^{134}. There are some advantages to this selection; for instance, choosing $w = 2$ gives the ability to use shift operation instead of multiplication, and since p satisfies that $p \equiv 3 \pmod 4$ and m is odd with the form $m = 2k + 1$, for some k, then the square root of a finite field element can be computed efficiently, as shown in section (7.2).

The implementation code could be used with the other parameters by modifying the header file and a few simple routines, such as the "Reduction routine".

9.3 SOFTWARE IMPLEMENTATION

The algorithms shown in the previous chapters have implemented in C using the Visual Micro Lab (VMLab) simulator version 3.9, which is a virtual prototyping design framework that simulates several types of AVR microcontrollers, such as ATmega128. The WinAVR software has used as a third party GNU C compiler (GCC) to compile C Code. The WinAVR software is open source software development tools for the Atmel AVR series of RISC microprocessors hosted on the Windows platform, and it includes the GNU compiler for C and C++.

9.3.1 Design Architecture

Elliptic Curve Cryptography implementation is very intensive, and it consists of several routines and functions, so a proper design should be used to accomplish an efficient implementation. The design has been used in this dissertation divides the implementation into three levels, as depicted in figure (9.1):

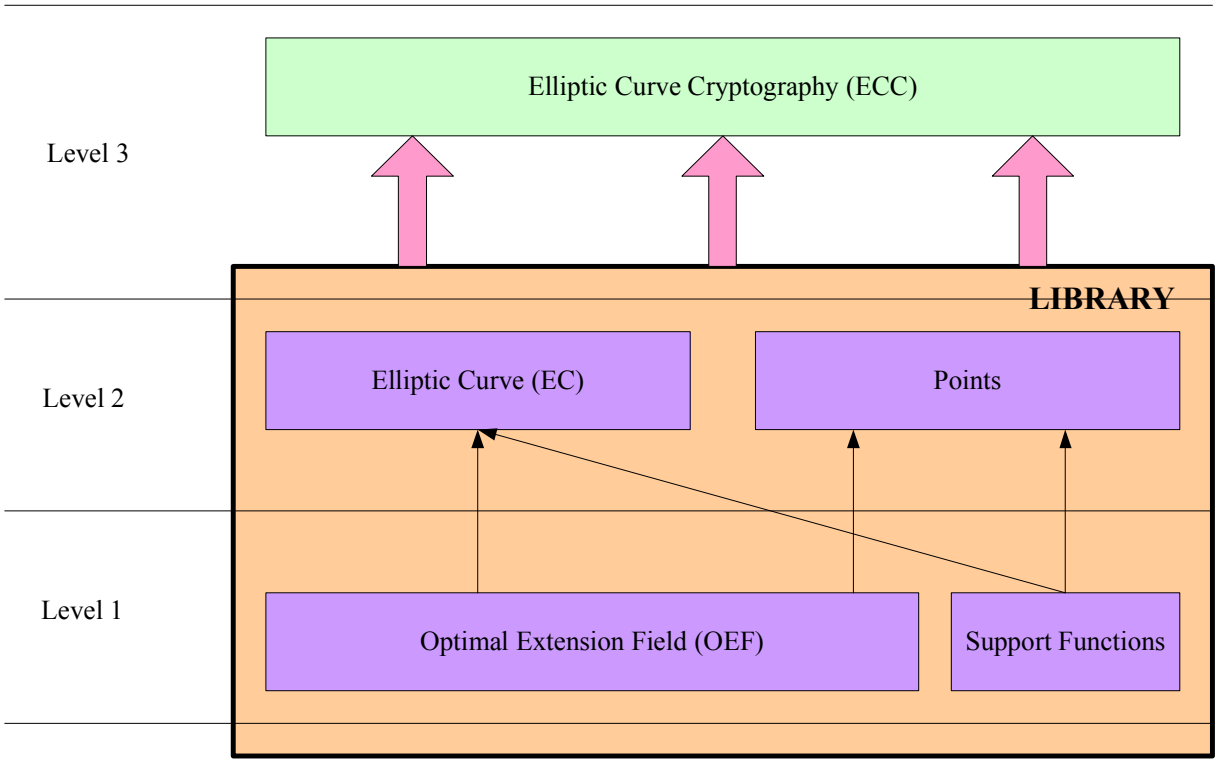


Figure 9.1 Elliptic Curve Cryptography Design Architecture

The first level consists of two groups: the support functions group includes basic functions, such as conversion functions between OEF elements and long integer string, and other basic arithmetic functions. The second group in this level includes the required functions to handle OEF elements and the arithmetic operations in this field such as addition, subtraction, multiplication, and other operations. This level is the only abstract level and it could be considered as the core level, since the other levels depend on it.

Level 2 also consists of two groups: the first group is the curve functions, which include the required routines to generate a curve and to embed data as a point on a curve. A point functions group includes addition, subtraction and doubling of elliptic curve points and scalar point multiplication.

In the last level any elliptic curve cryptosystem can be implemented, such as ECDH, EC analogue to ElGamal cryptosystem, ECDSA, and other cryptosystems. As shown in figure

(9.1), the first two levels can be considered as a library that contains all the definitions and routines required to implement any elliptic curve cryptosystem.

The advantages of this design are as follows:

- The library can be used to implement any elliptic curve cryptosystem.
- An elliptic curve cryptosystem can be implemented independent of a chosen curve or on a selected finite field.
- For security reasons several curves can be used on the same finite field.
- The underlying finite field can be changed without changing the top level routines.
- Future improvement can be accomplished easily.
- This design makes it easy to understand the implementation of different algorithms.

9.3.2 Data Representation

The main groups in the design are OEF, curve and points blocks. Each of these blocks consists of two parts: data and functions. This section shows how the data could be represented.

Representation of OEF elements

The elements in $GF(p^m)$ are represented by polynomials of degrees less than m as

$$A(x) = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots + a_1 x + a_0 \qquad (9.1)$$

where all the coefficients a_i are in $GF(p)$. Thus the OEF elements can be represented by a structure that consists of an array of the element's coefficients as follows:

```
typedef struct
{
    uint8_t Element[m];
}OEFPoly;
```

Notice that the type of Element array is uint8_t (unsigned char), so one can use the arithmetic operations, e.g. addition, subtraction, multiplication, and so on, which are implemented using C-type functions. This structure will be the prototype for most of the routines in the three levels.

Representation of the Elliptic curve

The elliptic curve equation over optimal extension field $GF(p^m)$ where $p > 3$ is given as

$$y^2 = x^3 + ax + b \qquad (9.2)$$

where $a, b \in GF(p^m)$.

Any elliptic curve of this form can be uniquely determined by a and b. Thus an elliptic curve can be given by the following structure:

```
typedef struct
{
  OEFPoly a;
  OEFPoly b;
}Curve;
```

This structure consists of the two parameters of the elliptic curve equation, which are of OEFPoly type.

Representation of Points

Each point on an elliptic curve E over $GF(p^m)$ has the form (x, y), where $x, y \in GF(p^m)$. Both coordinates x and y are represented as OEFPoly type; thus the structure for a point on an elliptic curve is given by:

```
typedef struct
{
  OEFPoly x;
  OEFPoly y;
}Point;
```

This structure can be used to represent any point on an elliptic curve, but there is still another point that the point at infinity O. In the implementation this point has represented as $(0, 0)$.

9.3.3 Functions Representation

The first two levels in figure (9.1) are a library that contains most of the required functions to implement any elliptic curve cryptosystem. This section gives a brief description of these functions.

OEF functions

The first task of the implementation is to implement the required functions to generate an underlying finite field, and to perform all the arithmetic operations on this field, which are discussed in chapter 5. Table (9.1) shows these functions, gives a brief description of each function task and shows the related algorithm or equation of these functions

Table 9.1 OEF functions

Function	Description	Note
PolyRandom	Generate a random OEF Poly	
PolyNull	Null out an OEF Poly	
PolyNeg	Negate an OEF Poly	
PolyCopy	Copy an OEF Poly to another one	
PolyIsNull	Check if an OEF Poly is Null	
PolyIsEqual	Check if 2 OEF Poly are equal	
PolyAdd	Add 2 OEF Poly	Algorithm (5.1)
PolySub	Subtract 2 OEF Poly	Algorithm (5.2)
PolyDouble	Double OEF Ploly	
PolyMulti	Multiply 2 OEF Poly	Algorithm (5.5)
PolyKaratsuba	Multiply 2 OEF Poly using Karatsuba algorithm	Algorithm (5.9)
PolySquare	Square an OEF Poly	Algorithm (5.6)
PolyScale	Scale an OEF Poly by 8-bit value	
PolyScale2n	Scale an OEF Poly by 8-bit power 2 value	
PolyInverse	Inverse an OEF Poly	Algorithm (5.7)
PolyFrobenius	Apply Frobenius map on an OEF Poly	Equation (4.37)
PolyExponent	Exponent an OEF Poly	Algorithm (5.8)
PolyQR	Check if an OEF Poly is Quadratic Residue	Algorithm (7.3)
PolySqrRoot	Find a square root of an OEF Poly	Algorithm (7.2)

Elliptic Curve Functions

This group includes functions required to generate an elliptic curve and to embed data on an elliptic curve. Table (9.2) describes this functions group.

Table 9.2 Elliptic Curve functions

Function	Description
CurveRandom	Generate a random Elliptic Curve
CurveCheck	Check equation (2.4)
CurveFx	Compute $f(x)$ value in equation (7.2)
CurveEmbedData	Embed an OEF Poly as a point on an Elliptic Curve
CurveEmbedMesg	Embed a plain text as a point on an Elliptic Curve
CurveExtractMesg	Recover a plain text from an Elliptic Curve

Points Functions

As shown in table (9.3), these functions are used to generate random points and to perform the arithmetic operation of points.

Table 9.3 Point functions

Function	Description
PointRandom	Generate a random Point
PointNull	Null out a Point
PointNeg	Negate a Point
PointCopy	Copy a Point to another one
PointIsInfinity	Check if a Point is at Infinity
PointIsEqual	Check if 2 Points are equal
PointAdd	Add 2 Points
PointSub	Subtract 2 Points
PointDouble	Double a Point
PointMulti	Multiply a Point by 32-bit value
PointPolyMulti	Multiply a Point by an OEF Poly (Algorithm 6.1)

Support Functions

This group consists of several support functions that could be called from different levels. However, table (9.4) shows only those functions that could be called from level three while implementing an elliptic curve cryptosystem.

Table 9.4 Support Functions

Function	**Description**
Str2OEF	Convert long integer to an OEF Poly
OEF2Str	Convert an OEF Poly to long integer
Mesg2OEF	Convert a plain text to an OEF Poly
OEF2Mesg	Convert an OEF Poly to a plain text
Reduction	Compute the Reduction.
SubfieldReduct	Compute subfield reduction using algorithm (5.4)
RandGenerat	Generate 8-bit random number
Euclidean	Inverse of 8-bit value using Euclidean algorithm

9.4 RESULTS AND TIMING

This section describes the performance of some algorithms and finite field operations, which are discussed in the previous chapters; it also shows the execution time of an elliptic curve group operation. Table (9.5) shows the finite field arithmetic performance on the ATmega128 microcontroller.

Table 9.5 OEF arithmetic performance on ATmega128 microcontroller

Description	**Operation**	**Time**
Addition	$C(x) = A(x) + B(x)$	0.1 msec
Subtraction	$C(x) = A(x) - B(x)$	0.1 msec
Doubling	$C(x) = 2A(x)$	0.08 msec
Multiplication	$C(x) = A(x) \cdot B(x)$	2.94 msec
Squaring	$C(x) = A^2(x)$	2.19 msec
Kartsuba Multiplication	$C(x) = A(x) \cdot B(x)$	6.43 msec
Inversion	$C(x) = A^{-1}(x)$	16.39 msec
Scale by 8-bit	$C(x) = s \cdot A(x)$	0.27 msec
Scale by 8-bit of 2^n	$C(x) = 2^n A(x)$	0.19 msec
Frobenius Map	$C(x) = A^{p^i}(x)$	0.29 msec
Exponent	$C(x) = A^e(x)$	13.9 msec
Square Root	$C(x) = \sqrt{A(x)}$	79.37 msec

Table (9.5) shows that the Kartsuba multiplication is two times slower than normal multiplication (using the Schoolbook method), this is because the Kartsuba algorithm requires 817 arithmetic operations (153 multiplications and 664 additions), while the Schoolbook method requires 545 arithmetic operations (289 multiplications and 256 additions) as shown in table (4.2) for $m = 17$. The ratio between the cost of one multiplication and one addition r should be greater than three in order to the Kartsuba algorithm to be more efficient than the Schoolbook method.

The ATmega128 microcontroller executes powerful instructions in a single clock cycle and achieves throughput approaching 1 MIPS per MHZ, this means the cost of one multiplication is approach to the cost of one addition, so r is less than three and the Schoolbook method is more efficient than the Kartsuba algorithm.

Using the arithmetic functions listed in table (9.5) to implement the elliptic curve group operation gives the performance results shown in table (9.6):

Table 9.6 Elliptic Curve group operation

Description	Operation	Time
Point Addition	$Q = P_1 + P_2$	25.2 msec
Point Doubling	$Q = 2\,P$	27.44 msec
Point Multiplication by 32-bit value	$Q = k \cdot P$	1.022 sec
Point Multiplication by an OEF Poly	$Q = A(x) \cdot P$	4.972 sec

Table (9.7) shows the size required for the software implementation:

Table 9.7 Program Size

File	Size (KB)
OEF	10.6
Point	4.98
Curve	1.71
Support	7.20
Header	3.47
Hex	40.3

As shown in table (9.7), the size of the hex file is 40.3 kB. In fact, one can reduce the size by selecting only the most suitable functions for a certain microcontroller. In the implementation there is more than one function to perform the same task; the performance of these functions depends on the selected microcontroller. For example, to perform the multiplication of two OEF elements, one has two options: using the Schoolbook method or the Karatsuba algorithm, for the ATmega128 microcontroller the first method is better, as it is shown in table (9.5). Another example, to find the subfield inversion one could use look-up table or the Extended Euclidean algorithm depending on the available memory size. Even though a lot of microcontrollers have a random number generator, the implementation includes a random number generator that could be more secure than the built-in one.

The variety of the functions gives a user more flexibility to choose the best functions, according to the memory size, speed, and other features of the selected microcontroller. As a result, this gives the most efficient performance.

Chapter 10

CONCLUSION

10.1 SYMMARY

Embedded systems are essential parts of most communications systems, which makes them especially attractive as a potential platform for implementing cryptographic algorithms. Several algorithms were implemented, although previous implementations of arithmetic intensive cryptographic algorithms seem to indicate that they can achieve acceptable performance on an embedded microcontroller and on a constrained platform.

This dissertation has demonstrated that elliptic curve cryptosystems are well suited for cryptographic applications and can be used in embedded system. Furthermore, elliptic curve cryptosystems are a logical alternative to other systems based on the discrete logarithm problem over finite fields because of their security and their efficiency, derived from their short key lengths. The main advantages of implementing elliptic curve cryptosystems are:

- Small key size as compared with traditional schemes.
- Wide range of applications including network security, smart cards, electronic banking, digital signatures, and other applications.
- High flexibility and enhanced security through periodically changing the curve.
- Can be used with any public-key cryptosystem.

The underlying finite field is the core of the elliptic curve cryptosystem. This dissertation evaluated and discussed different types of finite fields: the prime finite field, binary finite field, binary composite finite field, extension finite field and optimal extension finite field; then it compared these types. The merits of prime extension fields and their role in ECCs were investigated further, and the theory for an ECC based over a special class of prime extension fields, known as Optimal Extension Fields, was derived using a polynomial basis representation.

This dissertation described all of the OEF arithmetic and the operations for the group of points on an elliptic curve over such a field. Algorithms based on this theory were detailed, and several techniques used to speed up the OEF arithmetic and elliptic curve group operation were explained.

An implementation based on these algorithms was discussed, with emphasis on the high level design and representation of the elements in the OEF and point on the curve. The result of this implementation is a library that contains all the definitions and functions required to perform the OEF arithmetic and elliptic curve group operation in embedded system; this library can be used to implement an elliptic curve cryptosystem in embedded system in a simple manner without involving the complicated operation of the underlying finite field. The library performance was evaluated in chapter 9, which shows that OEFs are especially attractive for use in an elliptic curve cryptosystem.

10.2 SUMMARY OF ACHIEVEMENTS

The main achievements of this dissertation include the following:

- The underlying finite field, and a combination of optimization techniques, can result in acceptable performance of ECCs in an embedded system.

- The dissertation shows that OEFs are good arithmetic structures to use as underlying finite field in ECCs in an embedded system.

- During finite field multiplication, performing one reduction for each coefficient instead of a subfield reduction after each subfield multiplication is an efficient practice in an embedded system.

- It demonstrates an efficient implementation of the Itoh and Tsujii Inversion algorithm for computing inverses in Optimal Extension Field.

- It implements a library of functions that can be used to implement any elliptic curve cryptosystem in a low end embedded system.

10.3 FUTURE DEVELOPMENT

Finally, this is not the end; it is just the end of the beginning. The work performed in this dissertation has made available many interesting opportunities for further research and development. There include the following:

- Instead of choosing elliptic curves at random, a more structured approach could be taken.

- Further improvements could be made on the algorithms used in the finite field implementation. The OEF arithmetic, especially multiplication and inversion algorithm, could be made more efficient, which would significantly improve the performance of the cryptosystem.

- Because point scalar multiplication is the base of any elliptic curve cryptosystem, the improvement of this operation could directly speed up the complete cryptosystem.

- Any one interested in implementing a cryptosystem based on elliptic curve could use the library that was implemented in this dissertation. This introduces functions dealing with long integers that make the implementation as simple as those cryptosystems based on small integers.

REFRENCES

[1] W. Stallings, *Cryptography and Network Security*, Prentice Hall, New Jersey, USA, 2003.

[2] M.E. Smid and D.K Branstad, Data Encryption Standard: past and future, *Proceedings of the IEEE*, vol. 76, no. 5, pp. 550-559, May 1988.

[3] M.P. Leong, O.Y.H Cheung, K.H. Tsoi and P.H.W. Leong, A bit-serial implementation of the international data encryption algorithm IDEA, *2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 122-131, Apr 2000.

[4] W.E. Burr, Selecting the Advanced Encryption Standard, *IEEE Security & Privacy Magazine*, vol. 1, no. 2, pp. 43-52, Mar-Apr 2003.

[5] A. Selby and C. Mitchell, Algorithms for software implementations of RSA, *IEE Proceedings Digital Techniques*, vol. 136, no. 3, pp. 166-170, May 1989.

[6] W. Diffie and M. Hellman, New Directions in Cryptography, *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644-654, Nov 1976

[7] Bruce Schneier, *Applied Cryptography*, John Wiley and Sons, USA, 1996.

[8] Internet.com, http://www.webopedia.com/TERM/E/embedded_system.html, visited on Aug 30, 2004.

[9] N. Koblitz, Elliptic Curve Cryptosystems, Mathematics of Computation, vol. 48, pp. 203-209, 1987.

[10] V.S. Miller, Uses of Elliptic Curves in Cryptograph, *Advances in cryptology - CRYPTO '85,* Lecture Notes in Computer Science (LNCS), vol. 218, pp. 417-426, Springer-Verlag, New York, USA, 1986.

[11] N. Koblitz, A. Menezes and S. Vanstone, The State of Elliptic Curve Cryptography, *Designs, Codes and Cryptography*, vol. 19, no. 2-3, pp. 173-193, Mar 2000.

[12] J. Guajardo and C. Paar, Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography and Codes, *Designs, Codes and Cryptography*, vol. 25, no. 2, pp. 107-216, Feb 2002.

[13] Certicom, ECC Tutorial, http://www.certicom.com/index.php?action=ecc_tutorial,home, visited on Aug 30, 2004.

[14] J.P. Escofier, *Galois Theory*, Springer-Verlag, New York, USA, 2001.

[15] M. Rosing, *Implementing Elliptic Curve Cryptography*, Manning Publication Co, CT, USA, 1999.

[16] N. Koblitz, *A Course in Number Theory and Cryptography*, Springer-Verlag, New York, USA, 1994.

[17] *IEEE Standard Specifications for Public-Key Cryptography- Amendment 1: Additional Techniques*, IEEE Std 1363a – 2004, IEEE Computer Society, 2 Sep. 2004.

[18] R. Lidl and H. Niederreiter, *Introduction to Finite Fields and their Applications,* Cambridge University Press, UK, 1986.

[19] J. Lopez and R. Dahab, An Overview of Elliptic Curve Cryptography, Institute of Computing State University of Campinas, SP, Brazil, May 2000, http://citeseer.ist.psu.edu/cache/papers/cs/16063/http:zSzzSzwww.dcc.unicamp.brzSz~stolfizSzEXPORTzSzbibliographyzSz..zSzpaperszSzby-tagzSzgom-sto-00-polsp-tr.pdf/lop00overview.pdf, visited on Aug 30, 2004.

[20] B. Sunar, E. Savas and C.K. Koc, Constructing composite field representations for efficient conversion, *IEEE Transactions on Computers*, vol. 52, no. 11, pp. 1391-1398, Nov 2003.

[21] D. Beauregard, Efficient Algorithms for Implementing Elliptic Curve Public-Key Schemes, Master's Thesis, Worcester Polytechnic Institute, MA, USA, May 1996.

[22] J. Guajardo and C. Paar, Efficient Algorithms for Elliptic Curve Cryptosystems, *Advances in Cryptology - CRYPTO '97,* Lecture Notes in Computer Science (LNCS), vol. 1294, pp. 342-356, Springer-Verlag, Aug 1997.

[23] V. Daniel, C. Bailey and C. Paar, Optimal Extension Fields for Fast Arithmetic in Public-Key Algorithms, *Advances in Cryptology – CRYPTO '98 ,* Lecture Notes in Computer Science (LNCS), vol. 1462, pp. 472-485, Springer-Verlag, Aug 1998.

[24] P.L. Montgomery, Modular Multiplication without Trial Division, *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, 1985.

[25] P. Gaudry, F. Hess, and N. Smart, Constructive and Destructive Facets of Weil Descent on Elliptic Curves, HP Laboratories Bristol, Technical Report HPL-2000-10, Jan 2000, http://www.hpl.hp.com/techreports/2000/HPL-2000-10.pdf, visited on Aug 30, 2004.

[26] E. De Win, A. Bosselaers, S. Vandenberghe, P. De Gersem and J. Vandewalle, A fast software implementation for arithmetic operations in $GF(2^n)$. *Advances in Cryptology - Asiacrypt '96,* Lecture Notes in Computer Science (LNCS), vol. 1163, pp. 65-76, Springer-Verlag, 1996.

[27] E. De Win, S. Mister, B. Preneel and M. Wiener, On the Performance of Signature Schemes Based on Elliptic Curves, *Algorithmic Number Theory,* Lecture Notes in Computer Science (LNCS), vol. 1423, pp. 252-266, Springer-Verlag, Jun 1998.

[28] R. Schroeppel, H. Orman, S. O'Malley and O. Spatscheck, Fast Key Exchange with Elliptic Curve Systems, *Advances in Cryptology – CRYPTO '95,* Lecture Notes in Computer Science (LNCS), vol. 973, pp. 43-56, Springer-Verlag, 1995.

[29] D.E. Knuth, The Art of Computer Programming, Seminumerical Algorithms, vol. 2, Addison-Wesley, 1981.

[30] T. Itoh, S. Tsujii and Y. Asano, Generalised fast algorithm for computing multiplicative inverses in $GF(2^m)$, *Electronics Letters*, vol. 25, no. 10, pp. 664-665, May 1989.

[31] T. Itoh and S. Tsujii, Effective recursive algorithm for computing multiplicative inverses in $GF(2^m)$, *Electronics Letters*, vol. 24, no. 6, pp. 334-335, Mar 1988.

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

[33] 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, no. 5, pp. 394-498, May 2001.

[34] O. Jin Young, K. Young-Gern, P. Dong-Young and K. Heung-Su, Efficient multiplier architecture using optimized irreducible polynomial over $GF((3^n)^3)$, *Proceedings of the IEEE Region 10 Conference TENCON 99*, vol. 1, pp. 383-386, Sep 1999.

[35] A. Karatsuba and Y. Ofman, Multiplication of multidigit numbers on automata. Soviet Physics-Doklady 7, pp. 595–596, 1963.

[36] D. Page and N. P. Smart. Hardware implementation of finite fields of characteristic three. *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems,* Lecture Notes in Computer Science (LNCS), pp. 529-539, Springer-Verlag, Aug 2002

[37] M.G. Parker and M. Benaissa, $GF(2^m)$ multiplication using polynomial residue number systems, *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 42, no. 11, pp. 718-721, Nov 1995.

[38] T. Hasegawa, J. Nakajima and M. Matsui, A Practical Implementation of Elliptic Curve Cryptosystems over $GF(p)$ on a 16-bit Microcomputer, *Public Key Cryptography - PKC '98,* Lecture Notes in Computer Science (LNCS), vol. 1431, pp. 182-194, Springer-Verlag, Feb 1998.

[39] E. Al-Daoud, R. Mahmod, M. Rushdan and A. Kilicman, A new addition formula for elliptic curves over GF(2n), *IEEE Transactions on Computers*, vol. 51, no. 8, pp. 972-975, Aug 2002.

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

[41] K.H. Rosen, *Elementary Number Theory and its Applications*, AT&T Bell Laboratories, New Jersey, USA, 1993.

[42] A. Weimerskirch and C. Paar, Generalizations of the Karatsuba Algorithm for Efficient Implementations, 2003, http://www.crypto.ruhr-uni-bochum.de/imperia/md/content/texte/kaweb.pdf, visited on Aug 30, 2004.

[43] J.R. Bastida, *Field Extensions and Galois Theory*, Addison-Wesley, California, USA, 1984.

[44] N. Koblitz, CM-Curves with Good Cryptographic Properties, *Advances in Cryptology - CRYPTO '91*, Lecture Notes in Computer Science (LNCS), vol. 576, pp. 279-287, Springer-Verlag, Aug 1991.

[45] J.A. Solinas, An Improved Algorithm for Arithmetic on a Family of Elliptic Curves, *Advances in Cryptology - CRYPTO '97*, pp 357-371, Springer-Verlag, 1997.

[46] S.B. Mohan and B.S. Adiga, Fast Algorithms for Implementing RSA Public Key Cryptosystem, *Electronics Letter*, vol. 21, no. 17, pp. 761, Aug 1985.

[47] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer-Verlag, New York, USA, 1995.

[48] P.S Barreto, H.Y. Kim and M. Scoot, Efficient Algorithms for Pairing-Based Cryptosystems, http://eprint.iacr.org/2002/008.pdf, visited on Aug 30, 2004.

[49] Dr. Marsaglia's Web page, http://stat.fsu.edu/~geo/, visited on Aug 30, 2004.

[50] Certicom, An Elliptic Curve Cryptography (ECC) Primer, Jun 2004, http://www.certicom.com/content/live/resources/docs/WP-ECCprimer.pdf, visited on Aug 30, 2004.

[51] ATMEL, 8-bit Microcontroller with 128K Bytes In-System Programmable Flash, Dec 2003, http://www.atmel.com/dyn/resources/prod_documents/2467S.pdf, visited on Aug 30, 2004.