



NEAR EAST UNIVERSITY

**GRADEUATE SCHOOL OF APPLIED AND SOCIAL
SCIENCES**

**ELLIPTIC CURVE CRYPTOGRAPHY ANALYSIS
AND IMPLEMENTATION**

Hazem A. Elbaz

MASTER THESIS

DEPARTMENT OF COMPUTER ENGINEERING

Nicosia 2004



NEU

JURY REPORT

DEPARTMENT OF COMPUTER ENGINEERING

Academic Year: 2003-2004



STUDENT INFORMATION

Full Name	Hazem A M Elbaz		
Undergraduate degree	BSc.	Date Received	Spring 1998-2002
University	The Islamic University of Gaza	CGPA	3.10

THESIS

Title	Elliptic Curve Cryptography Analysis and Implementation		
Description	Analysis of Elliptic Curve Cryptography algorithms and implement ElGamal Elliptic Curve over network communication channel to perform Encryption/ Decryption on data transmitting.		
Supervisor	Prof. Dr. Fakhraddin Mamedov	Department	Computer Engineering

DECISION OF EXAMINING COMMITTEE

The jury has decided to accept / ~~reject~~ the student's thesis.
The decision was taken unanimously / ~~by majority~~.

COMMITTEE MEMBERS

Number Attending	3	Date	5/2/2004
Name		Signature	
Assoc. Prof. Dr. Rahib Abiyev, Chairman of the jury			
Assist. Prof. Dr. Dogan Haktanir, Member			
Assoc. Prof. Dr. Ilham Huseynov, Member			

APPROVALS

Date 5/2/2004	 Chairman of Department Assoc. Prof. Dr. Dogan Ibrahim
-------------------------	---

DEPARTMENT OF COMPUTER ENGINEERING
DEPARTMENTAL DECISION

Date: 5/2/2004

Subject: Completion of M.Sc. Thesis

Participants: Prof. Dr. Fakhraddin Mamedov, Assoc. Prof. Dr. Rahib Abiyev, Assist.Prof. Dr. Doğan Haktanir, Assoc.Prof. Dr. İlham Huseynov, Mohammed Abdelal, Mohammed Aldiri.

DECISION

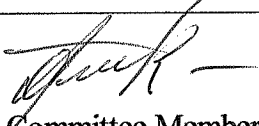
We certify that the student whose number and name are given below, has fulfilled all the requirements for a M .S. degree in Computer Engineering.

CGPA

20021298

Hazem A M Elbaz

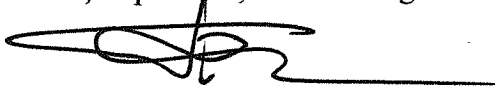

3.857


Assoc. Prof. Dr. Rahib Abiyev, Committee Member, Computer Engineering
Department, NEU


Assist. Prof. Dr. Doğan Haktanir, Committee Member , Electrical and Electronic
Engineering Department, NEU


Assoc. Prof. Dr. İlham Huseynov, Committee Member, Computer Information System
Department, NEU

Prof. Dr. Fakhraddin Mamedov, Supervisor, Dean of Engineering Faculties, NEU



Chairman of Department
Assoc. Prof. Dr. Doğan İbrahim

Hazem A M Elbaz : Elliptic Curve Cryptography Analysis and Implementation.

**Approval of the Graduate School of Applied and
Social Sciences**

Prof. Dr. Fakhraddin Mamedov
Director



**We certify this thesis is satisfactory for the award of the
Degree of Master of Science in Computer Engineering**

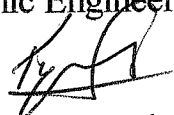
Examining Committee in charge:



**Assoc. Prof. Dr. Rahib Abiyev, Chairman of the jury, Computer
Engineering Department, NEU**

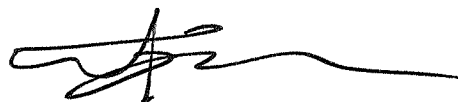


**Assist. Prof. Dr. Dogan Haktanir, Member , Electrical and
Electronic Engineering Department, NEU**



**Assoc. Prof. Dr. Ilham Huseynov, Member, Computer
Information System Department, NEU**

**Prof. Dr. Fakhraddin Mamedov, Supervisor, Dean of Engineering
Faculties, NEU**



ACKNOWLEDGMENTS

First, I would like to thank my supervisor Professor Fakhraddin Mamèdov for giving me the opportunity to work on this interesting project and for the help and guidance

More over I want to pay special regards to my parents who are enduring these all expenses and supporting me in all events. I am nothing without their prayers. They also encouraged me in crises. I shall never forget their sacrifices for my education so that I can enjoy my successful life as they are expecting. They may get peaceful life in Heaven. At the end I am again thankful to those all persons who helped me or even encouraged me to complete me, my project. My all efforts to complete this project might be fruitful.

I want also to pay special thanks to my lovely aunt Najah Elbaz, she have helped me so much in doing my master study. This thesis would not have been possible without her help, encourage, supporting, and her prayers.

I would also like to thank my housemate tamer fatayer, who encouraged me in doing my project.

ABSTRACT

This thesis describes elliptic curve cryptosystems (ECCs), which are expected to become the next-generation public key cryptosystems. ECC requires a shorter key length than RSA cryptosystems, which will be one of standards of public key cryptosystems, but provide equivalent security levels. Because of the shorter key length, ECCs is fast and can be implemented with less hardware.

The application of elliptic curves to the field of cryptography has been relatively recent. It has opened up a wealth of possibilities in terms of security, encryption, and real-world applications. In particular, we are interested in public key cryptosystems that use the elliptic curve discrete logarithm problem to establish security.

The objective of this thesis is to assemble the most important facts and findings into a board, unified overview of this field. To illustrate certain points, we also discuss a sample implementation of the elliptic curve analogue of ElGamal cryptosystem.

CONTENTS

ACKNOWLEDGEMENTS	i
ABSTRACT	ii
TABLE OF CONTENTS	iii
INTRODUCTION	1
1. INTRODUCTION TO CRYPTOGRAPHY	5
1.1 Overview	5
1.2 What is Cryptography?	5
1.3 What Cryptography Can Do?	7
1.4 What Cryptography Cannot Do?	8
1.5 Symmetric Key Cryptography	9
1.6 Asymmetric Key Cryptography	11
1.7 Modern use of cryptography	13
1.8 Cryptanalysis and attacks on Cryptosystems	14
1.9 Summary	16
2. OVERVIEW OF ABSTRACT ALGEBRA	17
2.1 Abstract Algebra	17
2.2 Groups	17
2.3 Rings	19
2.4 Integer module n	20
2.5 Fields	22
2.6 Finite Fields	23
2.7 Elliptic Curve over $GF(p)$	25
2.8 Summary	27
3. ELLIPTIC CURVE CRYPTOGRAPHY	28
3.1 Overview	28
3.2 Elliptic Curve	29
3.3 Addition Law	32
3.4 Factorization and Discrete Logarithm Problem	36
3.4.1 Integer Factorization Problem (IFP)	37
3.4.2 Discrete Logarithm Problem (DLP)	39
3.4.2.1 Cryptosystem based on DLP	39
3.5 Elliptic Curve Discrete Logarithm Problem (ECDLP)	41
3.6 Elliptic Curve Cryptography	42

3.6.1 Elliptic Curve Cryptosystems	42
3.6.2 Security and Efficiency of ECC	45
3.6.2.1 Security of ECC	45
3.6.2.2 Efficiency of ECC	47
3.6.2.2.1 Computational Overheads	48
3.6.2.2.2 Key Size	48
3.6.2.2.3 Bandwidth	48
3.6.3 Comparison between ECC and RSA	49
3.6.3.1 Size key	49
3.6.3.2 Speed	51
3.6.4 Possible to attack ECC	54
3.6.4.1 Naive Approach	54
3.6.4.2 Shanks' Method (baby-step, giant-step)	54
3.6.4.3 Pohling-Hellman Attack	55
3.6.4.4 Index And Xedni Calculus	56
3.6.4.5 Special-Purpose Attacks	56
3.6.4.6 Suggestions to avoid	57
3.6.5 Standards of the ECC algorithms	57
3.7 Elliptic Curve Protocols	59
3.7.1 Elliptic Curve Diffie-Hellman Algorithm (ECDHA)	59
3.7.1.1 Integer Diffie-Hellman key Exchange.	59
3.7.1.2 Elliptic-curve Diffie-Hellman key Exchange.	60
3.7.2 Digital Signature	62
3.7.2.1 Digital Signature Algorithm (DSA)	63
3.7.2.2 Elliptic Curve Digital Signature Algorithm (ECDSA)	65
3.7.3 Encryption (ElGamal Elliptic Curve)	68
3.7.3.1 ElGamal Cryptosystem	68
3.7.3.2 ElGamal Elliptic Curve Cryptosystem	69
3.7.4 Menezes-Vanstone Elliptic Curve Cryptosystem	71
3.8 Summary	72
4. IMPLEMENTATION MENEZES ECC ENCRYPTION USING VISUAL C++ 6.0	73

4.1 Overview	73
4.2 Program Explanation	73
4.3 Design of Program	78
4.3.1 Generate public key	79
4.3.2 Encrypt file	80
4.3.3 Decryption file	82
4.4 Summary	83
5. CONCLUSION	84
6. REFERENCES	85
APPENDICES	91

INTRODUCTION

The word "Cryptography" is derived from the Greek and it literally means "secret writing". Cryptography has been around for more than a thousand years and the Roman Empire was thought to be the masters of cryptography as they used simple cipher techniques to hide the meaning of messages. Some of the earlier and popular cryptographic techniques were Caesar cipher, Substitution cipher and Transposition ciphers. Cryptography is the process of encrypting the plain text into an incomprehensible cipher text by the process of Encryption and the conversion back to plain text by process of Decryption.

The basic of any cryptographic algorithm is the "seed" or the "key" used for encrypting/decrypting information. Many of the cryptographic algorithms are available publicly, though some organizations believe in having the algorithm a secret. The general method is in using a publicly known algorithm while maintaining the key a secret [9] .

Hence the common method adopted is to use a public key system to securely transmit a "secret key". Once we have securely exchanged the Key, we then use this key for encryption and decryption using a Symmetric Key algorithm [9]. So now there is question asked itself; why public key cryptography needed?

Until recently, most users of cryptography were military and/or diplomatic organizations that, by their very nature, were a small, finite number of individuals who would share a system of keys distributed internally.

The relatively recent advent of computer network communication has changed the nature of the average user of cryptography. Now, every time you order that book from some book web sites, do your banking online, or electronically sign your email, you are using some sort of cryptography. Because we may require secure communications with many different parties, these parties constantly changing, the use of classical cryptography quickly becomes unwieldy in all but the smallest of networks.

Therefore, new requirements are made of cryptosystems, such as authentication, non-repudiation, message integrity, and distributed trust, which go beyond mere message hiding [7] .

Elliptic curve cryptography has appeared as a promising new branch of public-key cryptography in recent years, due to its potential for offering similar security to established public-key cryptosystems at reduced key sizes. Improvements in various aspects of implementation, including the generation of elliptic curves, have made elliptic curve cryptography more practical than it was when first introduced in the 1980's. As security of elliptic curve cryptography becomes better understood, a chance is available to develop standards for this technology, thereby promoting interoperability at the same time as implementations are being deployed.

In 1985 Niel Koblitz and Victor Miller independently proposed the Elliptic Curve Cryptosystem (ECC), a method of utilizing a Discrete Logarithm problem over the points on an elliptic curve.

Most cryptosystems based upon the assumed difficulty of the discrete logarithm problem for finite fields have analogous elliptic curve versions [10].

Over the past 12 years, ECC and later ECDLP (Elliptic Curve Discrete Logarithm Problem) has received considerable attention from mathematicians around the world, and no significant breakthroughs have been made in determining weaknesses in the algorithm [10] .

Although critics are still skeptical as to the reliability of this algorithm, several encryption techniques have been developed recently using these properties. The fact that the problem appears so difficult to crack means that key sizes can be dropped in size considerably – even exponentially.

The idea of using Elliptic curves in cryptography as an alternative to established public-key systems such as DSA and RSA. The Elliptical curve Discrete Log Problem (ECDLP) makes it difficult to break an ECC as compared to RSA and DSA where the

problems of factorization or the discrete log problem can be solved in sub-exponential time. This means that significantly smaller parameters can be used in ECC than in other competitive systems such as RSA and DSA. This helps in having smaller key size hence faster computations.

This thesis discusses popular algorithm using Elliptic curve, and comparing the old algorithm that doesn't use Elliptic Curve Schema with the developed algorithm with Elliptic Curve Schema. The result of the thesis is implementing Encryption/Decryption algorithm that use Elliptic Curve Schema, which is ElGamal Elliptic Curve, it implemented with VC++ 6.0. I choice this algorithm because all resarchs and publishes concerns on Key Exchange using Diffie-Helman and Digital Signature using DSA, where developed using Elliptic Curve Schema.

The aim of this thesis is: to analyze Elliptic Curve Cryptography algorithms and to apply ElGamal Elliptic Curve over network communication channel to perform Encryption/ Decryption on data transmitting.

This thesis includes four chapters covering the main topics related in the following structure:

Chapter 1, Discusses the cryptography as whole; definition and types of cryptography, mechanism of public key cryptography, techniques used in cryptography and the key management process, what can cryptography do and what can't do, Modern using of cryptography and at last cryptanalysis and attack on cryptosystems

Chapter 2, Processes the abstract of algebra as, Groups, Rings, Fields and Finite Fields, and the properties of these concepts and the behavior of Elliptic Curve over it, also how can we use it in our purpose.

Chapter 3, Gives the Elliptic Curve Cryptography as, history of elliptic curve, what is elliptic curve in mathematics concepts, the problems the elliptic curve depends on, Elliptic Curve Cryptosystem, and at last elliptic curve protocols and how it different between new one and old one.

Chapter 4, Presents the developed application of elliptic curve cryptography based on the ElGamal Elliptic Curve Algorithm.

Finally in conclusion the obtained important results for the thesis are given.

1. INTRODUCTION TO CRYPTOGRAPHY

1.1 Overview

This chapter plans to give the reader a bottoms-up introduction to the basics of cryptography and this is the goal. Special emphasis will be given to the differences, advantages, and disadvantages of the various methods used in cryptography, without delving too deeply into the mathematical foundations of cryptography [41].

The origin of the word cryptology lies in ancient Greek. The word cryptology is made up of two components: "kryptos", which means hidden and "logos" which means word. Cryptology is as old as writing itself, and has been used for thousands of years to safeguard military and diplomatic communications. For example, the famous Roman emperor Julius Caesar used a cipher to protect the messages to his troops. Within the field of cryptology one can see two separate divisions: cryptography and cryptanalysis. The cryptographer seeks methods to ensure the safety and security of conversations while the cryptanalyst tries to undo the former's work by breaking his systems [43].

The basic idea behind cryptography is as follows. The message passes through a filter to encrypt the message into the ciphertext. The ciphertext goes to the receiver who passes the ciphertext through a related filter to decrypt the message and obtain the plaintext.

1.2 What is Cryptography?

The word "cryptography" is derived from Greek and when literally translated, means "secret writing." Before the advent of digital communications, cryptography was used primarily by the military for the purposes of espionage. With the advances in modern communication, technology has enabled businesses and individuals to transport information at a very low cost via public networks such as the Internet. This development comes at the cost of potentially exposing the data transmitted over such a medium. Therefore, it becomes imperative for businesses to make sure that sensitive data is transferred from one point to another in an airtight, secure manner over public networks. Cryptography can help us achieve this goal by making messages unintelligible to all but the intended recipient [41].

Encryption refers to the transformation of data in “plaintext” form into a form called “ciphertext,” which renders it almost impossible to read without the knowledge of a “key,” which can be used to reverse this transformation. The recovery of plaintext from the ciphertext requires the key, and this recovery process is known as decryption. This key is meant to be secret information and the privacy of the ciphertext depends on the cryptographic strength of the key.

Types of Cryptography

There are two types of cryptographic algorithms: Secret Key Cryptography and Public Key Cryptography.

Secret Key Cryptography:

- This crypto-system uses the same key for both encryption and decryption (this is also referred to as “symmetric” cryptography).
- Both the sender and the receiver need to have the same key in order to communicate successfully.
- Examples: DES, 3-DES, RC4, RC5, etc [41].

Advantages:

- o Very fast relative to public key cryptography;
- o Considered secure, provided the key is relatively strong;
- o The ciphertext is compact (that is, encryption does not add much excess “baggage” to the ciphertext);
- o Widely used and very popular.

Disadvantages:

- o The administration of the keys can become extremely complicated;
- o A large number of keys is needed to communicate securely with a large group of people;
- o Non-repudiation is not possible.
- o The key is subject to interception by hackers.

Public Key Cryptography

- This crypto-system uses one key for encryption and another key for decryption (also known as “asymmetric” cryptography).

- Each user has two keys – one **public** key, which is revealed to all users, and one **private** key, which remains a secret. The private key and the public key are mathematically linked.
- Encryption is performed with the public key and decryption is performed with the private key.
- Examples: RSA, Elliptic Curve Cryptography (ECC) [41].

Advantages:

- o Considered very secure;
- o No form of secret sharing is required, thus reducing key administration to a minimum;
- o Supports non-repudiation;
- o The number of keys managed by each user is much less compared to secret key cryptography.

Disadvantages:

- o Much slower compared to secret key cryptography;
- o The ciphertext is much larger than the plaintext, relative to secret key cryptography.

1.3 What Cryptography Can Do

Potentially, cryptography can hide information while it is in transit or storage. In general, cryptography can:

- Provide secrecy.
- Authenticate that a message has not changed in transit.
- Implicitly authenticate the sender.

Cryptography hides *words*. At most, it can only hide *talking about* contraband or illegal actions. But in a country with "freedom of speech," we normally expect crimes to be more than just "talk."

Cryptography can kill in the sense that boots can kill; that is, as a part of some other process, but that does not make cryptography like a rifle or a tank. Cryptography is defensive, and can *protect* ordinary commerce and ordinary people. Cryptography may be to our private information as our home is to our private property, and our home is our "castle."

Potentially, cryptography can hide *secrets*, either from others, or during communication. There are many good and non-criminal reasons to have secrets: Certainly, those engaged in commercial research and development (R&D) have "secrets" they must keep. Business often needs secrecy from competitors while plans are laid and executed, and the need for secrecy often continues as long as there are business operations. Professors and writers may want to keep their work private, until an appropriate time. Negotiations for new jobs are generally secret, and romance often is as well, or at least we might prefer that detailed discussions not be exposed. And health information is often kept secret for good reason.

One possible application for cryptography is to secure on-line communications between work and home, perhaps leading to a society-wide reduction in driving, something we could all appreciate.

1.4 What Cryptography Can Not Do

Cryptography can only hide information *after* it is encrypted and *while* it remains encrypted. But secret information generally does not *start out* encrypted, so there is normally an original period during which the secret is not protected. And secret information generally is not *used* in encrypted form, so it is again outside the cryptographic envelope every time the secret is used.

Secrets are often related to public information, and subsequent activities based on the secret can indicate what that secret is.

And while cryptography can hide *words*, it cannot hide:

- Physical contraband,
- Cash,
- Physical meetings and training,
- Movement to and from a central location,
- An extravagant lifestyle with no visible means of support, or
- Actions.

And cryptography simply cannot protect against:

- Informants,

- Undercover spying,
- Bugs,
- Photographic evidence, or
- Testimony.

It is a joke to imagine that cryptography alone could protect most information against Government investigation. Cryptography is only a small part of the protection needed for "absolute" secrecy [45].

1.5 Symmetric Key Cryptography

Symmetric key algorithms, known as secret-key algorithms, use a the same key for both encryption and decryption. Symmetric-key systems are simpler and faster than Asymmetric-key (public-key) systems, but their main drawback is that the two parties must somehow exchange the key in a secure way.

Symmetric algorithms can be divided into stream ciphers and block cipher. Stream ciphers can encrypt a single bit of plaintext at a time, whereas block ciphers take a number of bits (typically 64 bits in modern ciphers), and encrypt them as a single unit.

The most popular symmetric-key system is the Data Encryption Standard (DES) developed in 70s. DES is a block cipher with 64-bit block size. It uses 56-bit key. With this key length, DES is considered as unsafe for the future use. There is a variant of DES, Triple-DES or 3DES. It is based on using DES three times (in an encrypt-decrypt-encrypt sequence with three different, unrelated keys) Since November 1998, DES was no longer allowed for US government use.

A block cipher is a type of symmetric-key encryption algorithm that transforms a fixed-length block of plaintext data into a block of ciphertext data of the same length. This transformation takes place under the action of a user-provided secret key. Applying the reverse transformation to the ciphertext block using the same secret key performs decryption. The fixed length is called the block size, and for many block ciphers, the block size is 64 bits, for example DES. This means that they take a fixed-size block of data, an transform it to another 64 bit block using a function selected by the key. The

cipher basically defines a one-to-one mapping from 64-bit integers to another permutation of 64-bit integers [46].

The following list summarizes the private key systems in common use today.

ROT13

A simple cryptography algorithm, which is used, among other things, to obscure the content of risqué jokes on various Usenet groups. The ROT13 encryption algorithm has no key, and it is not secure.

Crypt

The original UNIX encryption program which is modeled on the German Enigma encryption machine. Crypt uses a variable-length key. Some programs can automatically decrypt *crypt*-encrypted files without prior knowledge of the key or the plaintext. *crypt* is not secure. (This program should not be confused with the secure one-way *crypt* program that UNIX uses for encrypting passwords.)

DES

The Data Encryption Standard (DES), an encryption algorithm developed in the 1970s by the National Bureau of Standards and Technology (since renamed the National Institute of Standards and Technology, or NIST) and IBM. DES uses a 56-bit key. Technically, we should refer to it as the DEA: Data Encryption Algorithm. Standard-conforming implementations are certified by NIST, and usually require a hardware implementation. However, nearly everyone refers to it as the DES, so we will too.

RC2

A block cipher originally developed by Ronald Rivest and kept as a trade secret by RSA Data Security. This algorithm was revealed by an anonymous Usenet posting in 1996 and appears to be reasonably strong (although there are some particular keys that are weak). RC2 is sold with an implementation that allows keys between 1 and 2048 bits. The RC2mail key length is often limited to 40 bits in software that is sold for export. Unfortunately, a 40-bit key is vulnerable to a brute force attack.

RC4

A stream cipher originally developed by Ronald Rivest and kept as a trade secret by RSA Data Security. This algorithm was revealed by an anonymous Usenet posting in 1994 and appears to be reasonably strong (although there are some particular keys that are weak). RC4 is sold with an implementation that allows keys between 1 and 2048

bits. The RC4 key length is often limited to 40 bits in software that is sold for export. Unfortunately, a 40-bit key is vulnerable to a brute force attack.

RC5

A block cipher developed by Ronald Rivest and published in 1994. RC5 allows a user-defined key length, data block size, and number of encryption rounds.

IDEA

The International Data Encryption Algorithm (IDEA), developed in Zurich, Switzerland by James L. Massey and Xuejia Lai and published in 1990. IDEA uses a 128-bit key, and is believed to be quite strong. IDEA is used by the popular program PGP (described later in this chapter) to encrypt files and electronic mail. Unfortunately, wider use of IDEA may be hampered by a series of software patents on the algorithm, which is currently held by Ascom-Tech AG, in Solothurn, Switzerland. Ascom-Tech supposedly will allow IDEA to be used royalty free in implementations of PGP outside the U.S., but concerned users should verify the terms with Ascom-Tech or their licensees directly.

Although we are generally in favor of intellectual property protection, we are opposed to the concept of software patents, in part because they hinder the development and use of innovative software by individuals and small companies.

Skipjack

A classified (SECRET) algorithm developed by the National Security Agency (NSA). Reportedly, a Top Secret security clearance is required to see the algorithm's source code and design specifications. Skipjack is the algorithm used by the Clipper encryption chip. It uses an 80-bit key [42].

1.6 Asymmetric Key Cryptography

Unlike symmetric key algorithms, public key algorithms use a different key for encryption and decryption. The decryption key cannot (practically) be derived from the encryption key. The merit of public key algorithms is that they can be used to transmit encryption keys or other data securely even when the parties have no opportunity to agree on a secret key in private.

RSA (Rivest-Shamir-Adelman) is developed by Ron Rivest, Adi Shamir, and Leonard Adleman in 1977. This public-key cryptosystem offers both encryption and digital signatures (authentication). It is generally considered to be secure when sufficiently

long keys are used (512 bits is insecure, 768 bits is moderately secure, and 1024 bits is good) [46].

The following list summarizes the public key systems in common use today:

Diffie-Hellman

A system for exchanging cryptographic keys between active parties. Diffie-Hellman is not actually a method of encryption and decryption, but a method of developing and exchanging a shared private key over a public communications channel. In effect, the two parties agree to some common numerical values, and then each party creates a key. Mathematical transformations of the keys are exchanged. Each party can then calculate a third session key that cannot easily be derived by an attacker who knows both exchanged values.

Several versions of this protocol exist, involving a differing number of parties and different transformations. Particular care must be exercised in the choice of some of the numbers and calculations used or the exchange can be easily compromised. If you are interested, consult the references for all the gory details.

The Diffie-Hellman algorithm is frequently used as the basis for exchanging cryptographic keys for encrypting a communications link. The key may be any length, depending on the particular implementation used. Longer keys are generally more secure.

RSA

The well-known public key cryptography system developed by (then) MIT professors Ronald Rivest and Adi Shamir, and by USC professor Leonard Adleman. RSA can be used both for encrypting information and as the basis of a digital signature system. Digital signatures can be used to prove the authorship and authenticity of digital information. The key may be any length, depending on the particular implementation used. Longer keys are generally considered to be more secure.

ElGamal

Another algorithm based on exponentiation and modular arithmetic. ElGamal may be used for encryption and digital signatures in a manner similar to the RSA algorithm. Longer keys are generally considered to be more secure.

DSA

The Digital Signature Algorithm, developed by NSA and adopted as a Federal Information Processing Standard (FIPS) by NIST. Although the DSA key may be any length, only keys between 512 and 1024 bits are permitted under the FIPS. As specified, DSA can only be used for digital signatures, although it is possible to use DSA implementations for encryption as well. The DSA is sometimes referred to as the DSS, in the same manner as the DEA is usually referred to as the DES [42].

1.7 Modern use of cryptography

Actually, public key cryptography is really interesting because it is easy to use and it solves many security problems unsolved so far. More precisely, it solves a few authentication problems:

- **Identifying individuals:** using anonymous communications means of today, Alice wants to be sure the person with whom she is talking is not cheating and impersonating Bob. To do so, she uses an identifying protocol. Multiple identifying protocols exist and commonly rely on the principles of RSA or of discrete logarithm.
- **Document authentication:** an authority authenticates documents through a *digital signature*. Signing consists in appending a few bits which are the result of some processing with document and authority as input, and which are generally hashed by a hash algorithm such as MD5 or SHA. Moreover, any person with access to the document should be able to verify the authority has really issued that signature. To do so, signature schemas are used. One of the most famous signature schemes is ElGamal - once more based on discrete logarithm problems.

Besides, as secret key cryptography, public key cryptography provides encryption-based cryptosystems, guaranteeing confidentiality of communications.

Let's imagine Alice wants to communicate secretly with Bob. Alice retrieves Bob's public key in a public directory, and enciphers her message with this key. When Bob receives the ciphertext, he uses his private key to decipher the ciphertext and read initial clear text. Both keys have very different roles, this explains why such systems are called

asymmetric cryptosystems - referring to secret key cryptosystems, which use the same key for encipherment and decipherment and are also known as symmetric cryptosystems.

Public key cryptography offers another major benefit over secret key cryptography. As a matter of fact, if n users communicate through a secret key cryptosystem, each of them needs one different secret key for each person in the group. So, $n(n-1)$ keys need to be managed. If n is over thousands of users, then millions of keys need to be managed... Furthermore, adding a new user to the group is not an easy task, because n new keys need to be generated for the user to communicate with all members of the group. Then, those new keys need to be sent over to the group. On the contrary, in asymmetric cryptosystems, the n public keys of the members are stored in a public directory. Adding a new user simply consists in adding his public key to the directory [44].

1.8 Cryptanalysis and attacks on Cryptosystems

Cryptanalysis is the art of deciphering encrypted communications without knowing the proper keys. There are many cryptanalytic techniques. Some of the more important ones for a system implementer are described below.

Ciphertext-only attack: This is the situation where the attacker does not know anything about the contents of the message, and must work from ciphertext only. In practice it is quite often possible to make guesses about the plaintext, as many types of messages have fixed format headers. Even ordinary letters and documents begin in a very predictable way. It may also be possible to guess that some ciphertext block contains a common word.

- **Known-plaintext attack:** The attacker knows or can guess the plaintext for some parts of the ciphertext. The task is to decrypt the rest of the ciphertext blocks using this information. This may be done by determining the key used to encrypt the data, or via some shortcut.
- **Chosen-plaintext attack:** The attacker is able to have any text he likes encrypted with the unknown key. The task is to determine the key used for encryption. Some encryption methods, particularly RSA, are extremely vulnerable to chosen-plaintext attacks. When such algorithms are used, extreme

care must be taken to design the entire system so that an attacker can never have chosen plaintext encrypted.

- **Man-in-the-middle attack:** This attack is relevant for cryptographic communication and key exchange protocols. The idea is that when two parties are exchanging keys for secure communications (e.g., using Diffie-Hellman), an adversary puts himself between the parties on the communication line. The adversary then performs a separate key exchange with each party. The parties will end up using a different key, each of which is known to the adversary. The adversary will then decrypt any communications with the proper key, and encrypt them with the other key for sending to the other party. The parties will think that they are communicating securely, but in fact the adversary is hearing everything.
- One way to prevent man-in-the-middle attacks is that both sides compute a cryptographic hash function of the key exchange (or at least the encryption keys), sign it using a digital signature algorithm, and send the signature to the other side. The recipient then verifies that the signature came from the desired other party, and that the hash in the signature matches that computed locally. This method is used e.g. in Photuris.
- **Timing Attack:** This very recent attack is based on repeatedly measuring the exact execution times of modular exponentiation operations. It is relevant to at least RSA, Diffie-Hellman, and Elliptic Curve methods. More information is available in the original paper and various follow up articles.

There are many other cryptographic attacks and cryptanalysis techniques. However, these are probably the most important ones for a practical system designer. Anyone contemplating to design a new encryption algorithm should have a much deeper understanding of these issues. One place to start looking for information is the excellent book *Applied Cryptography* by Bruce Schneier [47].

1.9 Summary

This chapter gave basic concepts of what is cryptography and what can cryptography do and can't do, it describe the both type of cryptography Symmetric and Asymmetric methods, also it gave brief idea of algorithms of each type, it show also the modern using of cryptography and the cryptanalysis of attacks of cryptography.

2. OVERVIEW OF ABSTRACT ALGEBRA

2.1 Abstract Algebra

Abstract algebra is the field of mathematics concerned with the study of algebraic structures such as groups, rings and fields. The term "abstract algebra" is used to distinguish the field from "elementary algebra" or "high school algebra" which teaches the correct rules for manipulating formulas and algebraic expressions involving real and complex numbers.

Historically, algebraic structures usually appear first in some other field of mathematics, were specified axiomatically, and were then studied in their own right in abstract algebra. Because of this, abstract algebra has numerous fruitful connections to all other branches of mathematics.

Examples of some algebraic structures with a single binary operation are:

- groups
- rings
- modules
- fields

In universal algebra, all those definitions and facts are collected that apply to all algebraic structures alike. All the above classes of objects, together with the proper notion of homomorphism, form categories, and category theory frequently provides the formalism for translating between and comparing different algebraic structures [30].

2.2 Groups

A great many of the objects investigated in mathematics turn out to be groups, including familiar number systems, such as the integers, rational, real, and complex numbers under addition, non-zero rational, real, and complex numbers under multiplication, non-singular matrices under multiplication, invertible functions under composition, and so on. Group Theory allows for the properties of these systems and many others to be investigated in a more general setting, and its results are widely applicable. Group theory is also a rich source of theorems in its own right. Groups underlie the other

algebraic structures such as fields and vector spaces and are also important tools for studying symmetry in all its forms. For these reasons, group theory is considered to be an important area in modern mathematics [31].

A group G is a finite or infinite set of elements together with a binary operation, which together satisfy the four fundamental properties of closure, associativity, the identity property, and the inverse property. The operation with respect to which a group is defined is often called the "group operation" and a set is said to be a group "under" this operation. Elements A, B, C, \dots with binary operation between A and B denoted AB form a group if:

- a. Closure: If $A, B \in G$, then $AB \in G$:
- b. Associativity: For all $A, B, C \in G$, $(AB)C = A(BC)$.
- c. Identity: There exists an element I such that $AI = IA = A$ for all $A \in G$.
- d. Inverse: For every $A \in G$, there exists an element $B = A^{-1}$ such that $AB = BA = I$ [7].

2.2.1 Abelian Groups

Abelian groups are groups where all elements in that group commute. That is, for group G , $AB = BA$ for all $A, B \in G$ [7].

If a group is abelian, we usually write the operation as $+$ instead of $*$, the identity element as 0 (often called the *zero element* in this context) and the inverse of the element a as $-a$.

Examples of abelian groups include all cyclic groups such as the integers \mathbf{Z} (with addition) and the integers modulo n \mathbf{Z}_n (also with addition). The real numbers form an abelian group with addition, as do the non-zero real numbers with multiplication. Every field gives rise to two abelian groups in the same fashion. Another important example is the factor group \mathbf{Q}/\mathbf{Z} , an injective cogenerator [32].

If n is a natural number and x is an element of an abelian group G , then nx can be defined as $x + x + \dots + x$ (n summands) and $(-n)x = -(nx)$. In this way, G becomes a module over the ring \mathbf{Z} of integers. In fact, the modules over \mathbf{Z} can be identified with

the abelian groups. Theorems about abelian groups can often be generalized to theorems about modules over principal ideal domains. An example is the classification of finitely generated abelian groups.

Any subgroup of an abelian group is normal, and hence factor groups can be formed freely. Subgroups, factor groups, products and direct sums of abelian groups are again abelian. If $f, g : G \rightarrow H$ are two group homomorphisms between abelian groups, then their sum $f+g$, defined by $(f+g)(x) = f(x) + g(x)$, is again a homomorphism. (This is not true if H is a non-abelian group). The set $\text{Hom}(G, H)$ of all group homomorphisms from G to H thus turns into an abelian group in its own right.

The abelian groups, together with group homomorphisms, form a category, the prototype of an abelian category.

Somewhat akin to the dimension of vector spaces, every abelian group has a *rank*. It is defined as the cardinality of the largest set of linearly independent elements of the group. The integers and the rational numbers have rank one, as well as every subgroup of the rationals. While the rank one torsion-free abelian groups are well understood, even finite-rank abelian groups are not well understood. Infinite-rank abelian groups can be extremely complex and many open questions exist, often intimately connected to questions of set theory [32].

2.3 Rings

A ring is a set S together with two binary operators $+$ and $*$ (addition and multiplication, respectively) satisfying the following conditions:

Additive associativity: For all $a, b, c \in S$, $(a + b) + c = a + (b + c)$.

1. Additive commutativity: For all $a, b \in S$, $a + b = b + a$.
2. Additive identity: There exists an element $0 \in S$ such that for all $a \in S$, $0 + a = a + 0 = a$.
3. Additive inverse: For every $a \in S$ there exists $-a \in S$ such that $a + (-a) = (-a) + a = 0$.
4. Multiplicative associativity: For all $a, b, c \in S$, $(a * b) * c = a * (b * c)$.
5. Left and right distributivity: For all $a, b, c \in S$, $a * (b + c) = (a * b) + (a * c)$

$$\text{and } (b + c) * a = (b * a) + (c * a).$$

This means that a ring is an abelian group under addition [7, 33].

2.4 Integer module \mathbb{Z}

A left R -module consists of an abelian group $(M, +)$ together with a ring of scalars $(R, +, *)$ and an operation $R \times M \rightarrow M$ (scalar multiplication, usually just written by juxtaposition, i.e. as rx for r in R and x in M) such that for all r, s in R , x, y in M , we have:

1. $(rs)x = r(sx)$
2. $(r+s)x = rx + sx$
3. $r(x+y) = rx + ry$
4. $1x = x$

Usually, we simply write "a left R -module M " or ${}_R M$.

Some authors omit condition 4 for the general definition of left modules, and call the above defined structures "unital left modules". In this encyclopedia however, all modules are assumed to be unital.

A right R -module M or M_R is defined similarly, only the ring acts on the right, i.e. we have a scalar multiplication of the form $M \times R \rightarrow M$, and the above three axioms are written with scalars r and s on the right of x and y . If R is commutative, then the left R -module is the same as the right R -module and is simply called an R -module [34].

If R is a field, then an R -module is also called a vector space. Modules are thus generalizations of vector spaces, and much of the theory of modules consists of recovering desirable properties of vector spaces in the realm of modules over certain rings. However, in general, an R -module may not have a basis [35].

Examples

- Every abelian group M is a module over the ring of integers \mathbb{Z} if we define $nx = x + x + \dots + x$ (n summands) for $n > 0$, $0x = 0$, and $(-n)x = -(nx)$ for $n < 0$.

- If R is any ring and n a natural number, then the cartesian product R^n is a module over R if we use the component-wise operations.
- If M is a smooth manifold, then the smooth functions from M to the real numbers form a ring R . The set of all vector fields defined on M form a module over R , and so do the tensor fields and the differential forms on M .
- The square n -by- n matrices with real entries form a ring R , and the Euclidean space \mathbf{R}^n is a left module over this ring if we define the module operation via matrix multiplication.
- If R is any ring and I is any left ideal in R , then I is a left module over R [35].

Submodules and homomorphisms :

Suppose M is an R -module and N is a subgroup of M . Then N is a **submodule** (or R -submodule, to be more explicit) if, for any n in N and any r in R , the product rn is in N (or nr for a right module) [34, 35].

If M and N are left R -modules, then a map $f: M \rightarrow N$ is a **homomorphism or R -modules** if, for any m, n in M and r, s in R , $f(rm + sn) = rf(m) + sf(n)$. This, like any homomorphism of mathematical objects, is just a mapping which preserves the structure of the objects.

Alternative definition as representations :

If M is a left R -module, then the *action* of an element r in R is defined to be the map $M \rightarrow M$ that sends each x to rx (or xr in the case of a right module), and is necessarily a group endomorphism of the abelian group $(M, +)$. The set of all group endomorphisms of M is denoted $\text{End}_{\mathbf{Z}}(M)$ and forms a ring under addition and composition, and sending a ring element r of R to its action actually defines a ring homomorphism from R to $\text{End}_{\mathbf{Z}}(M)$.

Such a ring homomorphism $R \rightarrow \text{End}_{\mathbf{Z}}(M)$ is called a *representation* of R over the abelian group M ; an alternative and equivalent way of defining left R -modules is to say that a left R -module is an abelian group M together with a representation of R over it.

A representation is called *faithful* if and only if the map $R \rightarrow \text{End}_{\mathbf{Z}}(M)$ is injective. In terms of modules, this means that if r is an element of R such that $rx=0$ for all x in M , then $r=0$. Every abelian group is a faithful module over the integers or over some modular arithmetic $\mathbf{Z}/n\mathbf{Z}$ [34].

2.5 Fields

A field is an algebraic structure in which the operations of addition, subtraction, multiplication, and division (except division by zero) may be performed and the associative, commutative, and distributive rules hold, which are familiar from the arithmetic of ordinary numbers.

Fields are important objects of study in algebra since they provide the proper generalization of number domains, such as the sets of rational numbers, real numbers, or complex numbers. Fields used to be called **rational domains**.

The concept of a field is of use, for example, in defining vectors and matrices, two structures in linear algebra whose components can be elements of an arbitrary field. Galois theory studies the symmetry of equations by investigating the ways in which fields can be contained in each other [36].

Definition: A field F is a nonempty set together with two binary operations $+$: $F \times F \rightarrow F$ and \cdot : $F \times F \rightarrow F$ such that:

1. $(F, +)$ is an abelian group.
2. $(F - \{0\}, \cdot)$ is an abelian group. (Here 0 represents the identity element for the $+$ operation)
3. For all $a, b, c \in F$, $a \cdot (b + c) = a \cdot b + a \cdot c$.
4. $0 \neq 1$. (The identity elements for the addition and multiplication operations are distinct) [18].

As an example of a finite field, let p be a prime number and consider the set

$$\mathbf{Z}_p = \{0, 1, 2, \dots, p-1\}$$

give with the usual operations of addition and multiplication modulo p .

The only part of the definition that is not clear satisfied is that each element of $Z_p - \{0\}$ has a multiplicative inverse. Clearly 1 is its own inverse, so consider n , where $2 \leq n \leq p - 1$. Since $\gcd(n, p) = 1$, there exist integers x and y such that $xn + yp = 1$, so $xn \equiv 1 \pmod{p}$, i.e. $xn = 1$ and so x is a multiplicative inverse for n [18].

We can also talk about elliptic curves over a finite field. For example, we can consider $E : y^2 = x^3 + 1$ as an elliptic curve over Z_3 . A short calculation shows that $E(Z_3) = \{O, (0,1), (0,2), (2, 0)\}$.

2.6 Finite Fields

Finite field play a crucial role in many cryptographic algorithms. It can be shown that the order of finite (number of elements in the field) must be a power of a prime p^n , where n positive integer.

The finite field of order p^n is generally written $GF(p^n)$; GF stands for Galois field, in honor of the mathematical who first studied finite field.

A **finite field** or **Galois field** is a field that contains only finitely many elements. Finite fields are important in cryptography and coding theory. The finite fields are completely known, as will be described below.

Since every field of characteristic 0 contains the rationals and is therefore infinite, all finite fields have prime characteristic [37].

If p is a prime, the integers modulo p form a field with p elements, denoted by Z_p , F_p or $GF(p)$. Every other field with p elements is isomorphic to this one.

If $q = p^n$ is a prime power, then there exists up to isomorphism exactly one field with q elements, written as F_q or $GF(q)$. It can be constructed as follows: find an irreducible polynomial $f(Z)$ of degree n with coefficients in $GF(p)$, then define $GF(q) = GF(p)[Z] / (f(Z))$. Here, $GF(p)[Z]$ denotes the ring of all polynomials with coefficients in $GF(p)$, and the quotient is meant in the sense of factor rings. The polynomial $f(Z)$ can be found

by factoring the polynomial $T^q - T$ over $\mathbf{GF}(p)$. The field $\mathbf{GF}(q)$ contains $\mathbf{GF}(p)$ as a subfield. There are no other finite fields [37].

Examples

The polynomial $\mathcal{A}(T) = T^2 + T + 1$ is irreducible over $\mathbf{GF}(2)$, and $\mathbf{GF}(4)$ can therefore be written as the set $\{0, 1, \iota, \iota+1\}$ where the multiplication is defined (modularly) by $\iota^2 + \iota + 1 = 0$. For example, to determine ι^3 , note that $\mathcal{A}(\iota^2 + \iota + 1) = 0$; so $\iota^3 + \iota^2 + \iota = 0$, and thus $\iota^3 + \iota^2 + \iota + 1 = 1$, so $\iota^3 = 1$. Similarly, since the characteristic of the field is 2, $\iota^2 = \iota + 1$.

In order to find the multiplicative inverse of ι in this field, we have to find a polynomial $\mathcal{A}(T)$ such that $T^* \mathcal{A}(T) = 1$ modulo $T^2 + T + 1$. The polynomial $\mathcal{A}(T) = T + 1$ works, and hence $1/\iota = \iota + 1$. Note that the field $\mathbf{GF}(4)$ is completely unrelated to the ring \mathbf{Z}_4 of integers modulo 4.

To construct the field $\mathbf{GF}(27)$, we start with the irreducible polynomial $T^3 + T^2 + T - 1$ over $\mathbf{GF}(3)$. We then have $\mathbf{GF}(27) = \{a\iota^2 + b\iota + c : a, b, c \text{ in } \mathbf{GF}(3)\}$, where the multiplication is defined by $\iota^3 + \iota^2 + \iota - 1 = 0$. [37]

Properties and facts:

If F is a finite field with $q = p^n$ elements (where p is prime), then $x^q = x$ for all x in F . Furthermore, the Frobenius homomorphism $f: F \rightarrow F$ defined by $f(x) = x^p$ is bijective, and is therefore an automorphism. The Frobenius homomorphism has order n , and the cyclic group it generates is the full group of automorphisms of the field.

The field $\mathbf{GF}(p^m)$ contains a copy of $\mathbf{GF}(p^n)$ if and only if n divides m . The reason for this is that there exist irreducible polynomials of every degree over $\mathbf{GF}(p^n)$.

Applications:

The multiplicative group of every finite field is cyclic, a special case of a theorem mentioned in the article about fields. This means that if F is a finite field with q elements, then there always exists an element x in F such that

$$F^* = \{0, 1, x, x^2, \dots, x^{q-2}\}.$$

The element x is not unique. If we fix one, then for any non-zero element a in F_q , there is a unique integer n in $\{0, \dots, q-2\}$ such that $a = x^n$. The value of n for a given a is called the *discrete log* of a (in the given field, to base x). In practice, although calculating x^n is relatively trivial given n , finding n for a given a is (under current theories) a computationally difficult process, and so has many applications in cryptography [37].

Finite fields also find applications in coding theory: many codes are constructed as subspaces of Vector spaces over finite fields.

Finite fields may be used to create a coordinate system for finite geometry, in the same way that the set of real numbers can be used as coordinates for Euclidean geometry.

2.7 Elliptic Curve over Galois Field

2.7.1 Elliptic Curves over Binary Finite Fields:

We start work in the field $GF(2^m)$ where we have characteristic = 2. Here we only consider so called "nonsupersingular curves". They have the property $a_1 \neq 0$. So we can perform the following change of variables:

$$\begin{aligned} X &\rightarrow a_1^2 X + \frac{a_3}{a_1} \\ Y &\rightarrow a_1^3 + \frac{a_1^2 a_4 + a_3^2}{a_1^3} \end{aligned}$$

This leads us to the following definition [16].

Definition 3. A (nonsupersingular) elliptic curve E over the finite field F_{2^m} is given through an equation of the form

$$Y^2 + XY = X^3 + aX^2 + b, \quad a, b \in F_{2^m}$$

Before starting with the arithmetic of the points on an elliptic curve, we take a final look at the coefficients in the following equation:

Definition: An elliptic curve E over the field F is a smooth curve in the so-called "long Weierstrassform"

$$Y^2 + a_1 XY + a_3 Y = X^3 + a_2 X^2 + a_4 X + a_6, \quad a_i \in F.$$

The subscripts of these coefficients seem to be a little bit strange. But consider following: For big values of X we can say that the equation is very close to $F: Y = X^{3/2}$. This function can be parameterized by setting $X = T^2$, $Y = T^3$. One says, "X has degree 2" and "Y has degree 3". The subscripts of the coefficients in previous equation indicate the degrees that must be given to the coefficients in order that the equation be homogeneous (this means that each term has the same total degree which is 6 in this case) [16].

2.7.2 Elliptic Curves over Prime Finite Fields:

Now we work with F_p ($p \in \mathbf{P}$, $p > 3$, $\text{char}(F_p) \neq 2, 3$) and we can make the following change of variables:

$$\begin{aligned} X &\rightarrow X - \frac{a_2}{3} \\ Y &\rightarrow Y - \frac{a_1 X + a_3}{2} \end{aligned}$$

Let's take a look what is happening to the left side after the substitution for Y :

$$\begin{aligned} (Y - (a_1 X + a_3)/2)^2 + a_1 X(Y - (a_1 X + a_3)/2) + a_3(Y - (a_1 X + a_3)/2) = \dots \\ \dots = Y^2 - a_1^2 X^2 / 4 - a_1 a_3 X / 2 - a_3^2 / 4 \end{aligned}$$

Both, XY and Y have vanished, so their coefficients a_1 and a_3 must equal zero! That reduces the left side to a single Y^2 . If we make the substitution for X and take a look at the right side of $Y^2 + XY = X^3 + aX^2 + b$ we get:

$$\begin{aligned} (X - a_2/3)^3 + a_2(X - a_2/3)^2 + a_4(X - a_2/3) + a_6 = \dots \\ \dots = X^3 + (a^2/9 + a_4)X + 2a_2^3/27 - a_2/3a_4a_6 \end{aligned}$$

Setting $(\frac{1}{9}a^2 + a_4) = a$ and $\frac{2}{27}a_2^3 - \frac{1}{3}a_2a_4a_6 = b$ we have the much nicer form

$X^3 + aX + b$. In F_p the equation:

$$Y^2 + a_1XY + a_3Y = X^3 + a_2X^2 + a_4X + a_6, \quad a_i \in F.$$

reduces to $Y^2 = X^3 + aX + b$.

What can we say about the smoothness of this equation? Consider the partial derivative of the equation $y^2 = f(x)$, which is $f'(x) = 2y \frac{dy}{dx}$. The expression of $\frac{dy}{dx}$ is undefined in (x_0, y_0) if and only if $f'(x_0) = f'(x_0) = y_0 = 0$. In other words, the function $f(x)$ must have a multiple root at the point x_0 . In the case that $f(x) = x^3 + ax + b$, this is equivalent to $\text{disc}(f(x)) = -(4a^3 + 27b^2) = 0$. We give now our definition for an elliptic curve over the finite field F_p :

Definition: *An elliptic curve E over the finite field F_p is given through an equation of the Form:*

$$Y^2 = X^3 + aX^2 + b, \quad a, b \in F_p \quad \text{and} \quad -(4a^3 + 27b^2) \neq 0$$

Please note that as stated in the beginning of the section, the "=" should be replaced by an " \equiv " in the above definition. Another remark is that when we talk about partial derivatives we mean the "formal partial derivate" which can be defined over an arbitrary field [16].

2.8 Summary

This chapter planned to give the reader a bottoms-up introduction to the basics of algebra that using in elliptic curve cryptography and this is the goal. Special emphasis will be given to what is the groups, Rings, fields, finite field, and the properties of each of them, this topic are important to reader of this thesis to give him imagin how the elliptic curve works.

3. ELLIPTIC CURVE CRYPTOGRAPHY

3.1 Overview

With the need for information security in today's digital systems both acute and growing, cryptography has become one of their critical components. Cryptographic services are required across a variety of platforms in a wide range of applications such as secure access to private networks, stored value, electronic commerce, and health care. Incorporating these services into solutions presents an ongoing challenge to manufacturers, systems integrators, and service providers because applications must meet the market requirements of mobility, performance, convenience, and cost containment.

Neal Koblitz and Victor Miller first proposed elliptic curve cryptography in 1985 independently. Elliptic curve cryptosystems (ECCs), which are expected to become the next-generation public key cryptosystems. Elliptic curves and elliptic curve discrete logarithm problem have been used in cryptography system for the last 12 years. (ECC) is based on the properties of the elliptic curve, which define and set constraints on the set of public and private keys. An elliptic curve is a set of solutions (x, y) to an equation of the form:

$$y^2 = x^3 + ax + b \pmod{p}$$

in which x, y, a, b are elements of the set of integers modulo p .

The primary advantage of elliptic curve systems over the "conventional" public key cryptosystems based on factoring or on the discrete logarithm problem is that there is no known sub-exponential algorithm to solve the discrete logarithm problem on elliptic curves. Also, elliptic curve systems need smaller key sizes to ensure the same level of security, and thus also less memory and processor time for calculation. Furthermore, many cryptographic systems (e.g. Digital Signature Algorithm, ElGamal encryption scheme, Diffie-Hellman key exchange protocol) have analogues for the elliptic curves.

Some of these algorithms are also included in standards of American National Standards Institute (ANSI X9.62, ANSI X9.63), Institute of Electrical and Electronics

Engineers (IEEE P1363), International Standards Organization (ISO/IEC 14888-3, ISO/IEC 15946) and National Institute of Standards and Technology (NIST FIPS 186-2) we present it below [1].

In this chapter we describe in details 4 protocols based on elliptic curve cryptography techniques, and the result of our implementation of ECC over Galois Field over prime $GF(p)$, where p is prime number in the next chapter.

3.2 Elliptic Curve

An elliptic curve is not an ellipse! The reason for the name is a little more indirect. It has to do, as we shall explain shortly, with "elliptic integrals", which arise in computing the arc length of an ellipse. But this happenstance of nomenclature isn't too significant, since an elliptic curve has different, and much more interesting, properties as compared to an ellipse [38, 5].

An elliptic curve is an object that is easily definable with simple high school algebra. Its amazing fruitfulness as an object of investigation may well depend on this simplicity, which makes possible the study of a number of much more sophisticated mathematical objects that can be defined in terms of elliptic curves.

The purpose of this section is to provide sufficient background material in ECC to understand the remainder of this document.

Elliptic curves are mathematical constructs that have been studied by mathematicians since the seventeenth century. In 1985, Neal Koblitz and Victor Miller independently proposed public-key systems using a group of points on an elliptic curve, and elliptic curve cryptography (ECC) was born. Since that time, numerous researchers and developers have spent several years researching the strength of ECC and improving techniques for its implementation. Today it offers those looking for a smaller, faster public-key system a practical and secure technology for even the most constrained environments [3].

Elliptic curves arise from algebra and number theory, and also make use of groups from which we can see how these would be related to both modular arithmetic and the discrete logarithm problem.

ECC delivers the highest strength per bit of any known public-key system because of the difficulty of the hard problem upon which it is based. This greater difficulty of the hard problem “the elliptic curve discrete logarithm problem (ECDLP)” means that smaller key sizes yield equivalent levels of security. The following Table compares the key sizes needed for equivalent strength security in ECC with RSA and DSA. Given the best-known algorithms to factor integers and compute elliptic curve logarithms, the key sizes are considered to be equivalent strength based on MIPS years needed to recover one key [3].

Table 3.1 Key length Equivalent Strength Comparison [3].

Time to break in MIPS years	RSA/DSA Key size	ECC key size	RSA/ECC key size ratio
10^4	512	106	5 : 1
10^8	768	132	6 : 1
10^{11}	1024	160	7 : 1
10^{20}	2048	210	10 : 1
10^{78}	21000	600	35 : 1

The first thing to note is that an elliptic curve is not an ellipse! An elliptic curve is a mathematical equation: $y^2 = x^3 + ax + b$, where all calculations are performed modulo p , and $4a^3 + 27b^2 \neq 0$ modulo p , for some odd prime p .

The mathematical property that makes elliptic curves useful for cryptography is simply that if, in general, we take two (distinct) points on the curve then the chord joining them intercepts the curve in a third point (because we have a cubic curve). If we then reflect that point in the x-axis we get another point on the curve (since the curve is symmetric about the x-axis).

This allows us to define a form of arithmetic on the curve. If we denote the two original points by P and Q then we will denote the final (reflected) point by $P+Q$ (see

Figure 3.1). It turns out that this “addition” satisfies all the usual algebraic properties that we associate with integers, provided we define a single additional point “the point at infinity”, which plays the role of 0 in the integers [5].

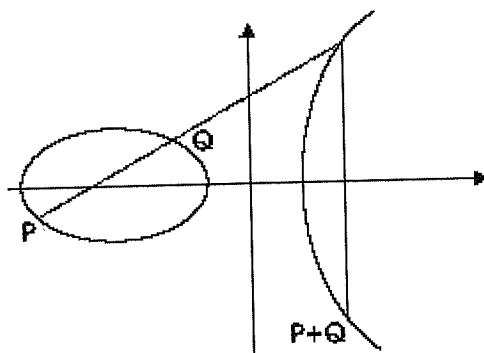


Figure 3.1 Addition of Elliptic Curve Points [5].

The “point at infinity” is a “virtual” point, not a point on the curve. It is needed for completeness of the newly defined arithmetic system. For example, if the points P and Q are mirror images of each other in the x -axis then the chord joining P and Q does not actually meet the curve again, so in this case we say $P+Q = \Phi$, where Φ denotes the point at infinity. If we identify Φ with the zero of the system then this naturally leads to the idea of denoting the mirror image of P by $-P$.

In other words, we can define a form of arithmetic on the points of an elliptic curve (plus the point at infinity) that lends itself to normal algebraic manipulation. In mathematical terms, we can define a *finite additive abelian group* on the points of the curve, with the zero being the point at infinity. In particular, if we let the points P and Q coincide, we can define $P+P$, naturally denoted $2P$. Extending this idea, we can define kP , for any integer k , and hence define the order of P , being the smallest integer k such that $kP = \Phi$ [5].

We are now in a position to define the *Elliptic Curve Discrete Logarithm Problem* (ECDLP) which is the reason we are considering these systems: *Given a “base point” P and the point kP , lying on the curve, find the value of k* It is believed that, for suitable elliptic curves and base points, this is a really, really hard problem! From a cryptographic point of view, we are in a position to define new cryptographic systems based on elliptic curves. In particular, any standard system that relies on the discrete

logarithm problem has a direct analogy based on the ECDLP. For example, *Elliptic Curve DSA* (ECDSA) has already been standardized (ANSI X9.62). Diffie-Hellman key exchange can be easily implemented in an elliptic curve framework, so in section 3.4 you can find more details on Elliptic Curve Discrete Logarithm Problem ECDLP [5].

3.3 Addition Law

ECC involves several areas of mathematics including finite fields, representations of field elements, and group theory. In this section we describe the mathematics necessary to understand the main algorithms being investigated in this research.

In its most simple form, an elliptic curve is a set of elements of the form (x, y) that satisfy the equation $y^2 = x^3 + ax + b \pmod n$.

Where a , b and n are predetermined numbers. In cryptographic applications, we specify that $4a^3 + 27b^2 \not\equiv 0 \pmod n$ and that n be prime. See an example of an elliptic curve at the end of the section.

In addition, another single element denoted \mathcal{O} is also part of the set, and represents "the point at infinity" (the top and bottom of every vertical line).

These curves can be defined over any *field*: real, fractional or complex. The majority of elliptic curves used for cryptographic purposes are defined over *finite fields*.

A finite field F_n is simply a finite set of elements with two operations, addition and scalar multiplication, where the operations are performed modulo n and satisfy the following properties: [2]

1. **Closure under addition.** If x and y are elements of a field F_n , then $x + y \in F_n$.
2. **Closure under scalar multiplication.** If x is an element of a field F_n , and λ is any integer, then $\lambda x \in F_n$.

For most cryptographic applications, finite fields of the form F_p , where p is prime, or F_{2^k} where k is a positive integer are most common.

In the case of an elliptic curve, addition is defined in the following way (**illustrated in Figure 3.2**):

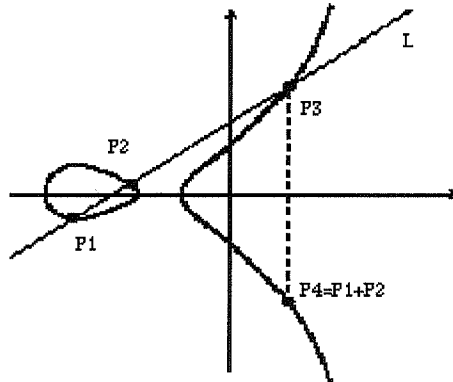


Figure 3.2 An example in addition in an elliptic curve [2].

1. Given points $p1$ and $p2$ on the elliptic curve, find $p3$, which is the third point of intersection with the elliptic curve of a line through $p1$ and $p2$.
2. Let $p4 = (x, -y)$ where $p3 = (x, y)$.
3. Define $p1 + p2 = p4$.

Multiplication of a point $p1$ by a positive integer λ is then simply defined by repeated addition as:

$$\lambda p1 = \begin{cases} 0 & \text{if } \lambda = 0 \\ x + (\lambda - 1)x & \text{if } \lambda > 0 \end{cases}$$

It may be unclear how $2p1 = p1 + p1$ is determined, since there are an infinite number of lines that pass through just the point $p1$. To find this result (which is known as the *double* of $p1$), we simply do the following (**illustrated in Figure 3.3**):

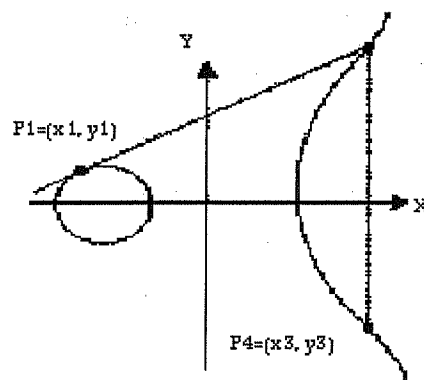


Figure 3.3 An example of the double of a point on an elliptic curve [2].

1. Draw a line that is tangent to the elliptic curve at point p_1 . This line will intersect the curve again at point p_3 .
2. Let $p_4 = (x, -y)$ where $p_3 = (x, y)$.
3. Define $2p_1 = p_1 + p_2 = p_4$.

Now the avid reader may wonder how we can be so sure that our straight lines are going to intersect the elliptic curve at a new point in both the above cases. Suppose that the straight line in question has the form $y=mx+c$.

If we now substitute into the elliptic curve equation, we obtain

$$\begin{aligned} (mx+c)^2 &= x^3 + ax + b \\ \Rightarrow m^2 x^2 + 2mxc + c^2 &= x^3 + ax + b \\ \Rightarrow 0 &= x^3 - m^2 x^2 + (a - 2mc)x + b - c^2 \end{aligned}$$

And so we are left with a cubic equation in x that we have to solve. In both the cases above, we already have two roots of this equation (in the second case, p_1 is a repeated root since the straight line is tangential to the elliptic curve), and since these roots are real-valued it follows that the third must be real-valued. To find the root, one would factorize the cubic as:

$$0 = \begin{cases} (x-x_1)(x-x_2)(x-x_3) & \text{where we have two points } p_1 \text{ and } p_2 \\ (x-x_1)(x-x_1)(x-x_3) & \text{where we have one point } p_1 \end{cases}$$

Where $p_1=(x_1, y_1)$, $p_2=(x_2, y_2)$ and we wish to find point $p_3=(x_3, y_3)$.

Further mathematics can be used to show that, in general

$$\begin{aligned} x_3 &= \lambda^2 - x_1 - x_2 \\ y_3 &= \lambda(x_1 - x_3)y_1 \end{aligned}$$

Where $p_1 = (x_1, y_1)$ and $p_2=(x_2, y_2)$ for any points p_1 and p_2 on the elliptic curve (they may be equal), $p_3=(x_3, y_3)$, $p_3 = p_1 + p_2$ and where: [2]

$$\lambda = \begin{cases} \frac{y_2 - y_1}{x_2 - x_1} & \text{if } P \neq Q \\ \frac{3x_1^2 + a}{2y_1} & \text{if } P = Q \end{cases}$$

See the following example using these formulae:

Example:

Let E be the elliptic curve $y^2 = x^3 + x + 6$ over \mathbf{Z}_{11} . For a given x we can test to see if $z = x^3 + x + 6 \bmod 11$ is a quadratic residue by applying Euler's criterion. Applying this formula, we have that the square roots of a quadratic residue z are:

$$\pm z^{(11+1)/4} \bmod 11 = \pm z^3 \bmod 11$$

The results of these computations in this table:

X	$X^3+X+6 \bmod 11$	In QR(11)?	Y
0	6	No	
1	8	No	
2	5	Yes	4,7
3	3	Yes	5,6
4	8	No	
5	4	Yes	2,9
6	8	No	
7	4	Yes	2,9
8	9	Yes	3,8
9	7	No	
10	4	Yes	2,9

Thus E has 13 points on it. Suppose we take the generator $a=(2,7)$. Then we can compute the "powers" of a (which we will write as multiples of a , since the group operation is additive). To compute $2a=(2,7)+(2,7)$, we first compute

$$\begin{aligned}\lambda &= (3 \cdot 2^2 + 1)(2 \cdot 7)^{-1} \bmod 11 \\ &= 2 \cdot 3^{-1} \bmod 11 \\ &= 2 \cdot 4 \bmod 11 \\ &= 8\end{aligned}$$

Then we have

$$x_3 = 8^2 - 2 - 2 \bmod 11 = 5$$

And

$$y_3 = 8(2 - 5) - 7 \bmod 11 = 2$$

So

$$2a=(5,2).$$

The next multiple would be $3a = 2a + a = (5,2) + (2,7)$. Again, we begin by computing 1, which in this solution is done as follows:

$$\begin{aligned}\lambda &= (7 - 2)(2 - 5)^{-1} \bmod 11 \\ &= 5 \cdot 8^{-1} \bmod 11 \\ &= 5 \cdot 7 \bmod 11 \\ &= 2\end{aligned}$$

Then we have

$$x_3 = 2^2 - 5 - 2 \bmod 11 = 8$$

And

$$y_3 = 2(5 - 8) - 2 \bmod 11 = 3$$

So

$$3a = (8,3).$$

3.4 Factorization and Discrete Logarithm Problem

Over the years, many of the proposed public-key cryptographic systems have been broken and many others have been demonstrated to be impractical. Today, only three types of systems are considered both secure and efficient. Examples of such systems and the mathematical problems, on which their security is based, are:

1. **Integer factorization problem (IFP):** RSA and Rabin-Williams.
2. **Discrete logarithm problem (DLP):** the U.S. government's Digital Signature Algorithm (DSA), the Diffie-Hellman key agreement scheme, the ElGamal encryption and signature schemes, the Schnorr signature scheme, and the Nyberg-Rueppel signature scheme.
3. **Elliptic curve discrete logarithm problem (ECDLP):** the elliptic curve analog of the DSA (ECDSA), and the elliptic curve analogs of the Diffie-Hellman key agreement scheme, the ElGamal encryption and signature schemes, the Schnorr signature scheme, and the Nyberg-Rueppel signature scheme.

It must be emphasized that none of these problems have been *proven* to be intractable (i.e., difficult to solve in an efficient manner). Rather, they are *believed* to be intractable because years of intensive study by leading mathematicians and computer scientists have failed to yield efficient algorithms for solving them [4].

3.4.1 Integer Factorization Problem (IFP)

In mathematics, the **integer prime-factorization** (also known as **prime decomposition**) problem is this: given a positive integer, write it as a product of prime numbers. For example, given the number 45, the prime factorization would be $3^2 \cdot 5$. The factorization is always unique, according to the fundamental theorem of arithmetic. This problem is of significance in mathematics, cryptography, complexity theory, and quantum computers.

The complete list of factors can be derived from the prime factorization by incrementing the exponents from zero until the number is reached. For example, since $45 = 3^2 \cdot 5$, 45 is divisible by $3^0 \cdot 5^0$, $3^0 \cdot 5^1$, $3^1 \cdot 5^0$, $3^1 \cdot 5^1$, $3^2 \cdot 5^0$, and $3^2 \cdot 5^1$, or 1, 5, 3, 15, 9, and 45. In contrast, the prime factorization only includes prime factors.

Given two large prime numbers, it is easy to multiply them together. However, given their product, it appears to be difficult to find the factors. This is relevant for many modern systems in cryptography. If a fast method were found for solving the integer factorization problem, then several important cryptographic systems would be broken, including the RSA public-key algorithm, and the Blum Blum Shub random number generator.

Although fast factoring is *one* way to break these systems, there may be *other* ways to break them that don't involve factoring. So it is possible that the integer factorization problem is truly hard, yet these systems can still be broken quickly. A rare exception is the Blum Blum Shub generator. It has been proved to be exactly as hard as integer factorization. There is no way to break it without also solving integer factorization quickly.

If a large, n -bit number is the product of two primes that are roughly the same size, then no algorithm is known that can factor in polynomial time. That means there is no known algorithm that can factor it in time $O(n^k)$ for any constant k . There are algorithms, however, that are faster than $\Theta(e^n)$. In other words, the best known algorithms are sub-exponential, but super-polynomial. In particular, the best known asymptotic running time is for the *General Number Field Sieve* (GNFS) algorithm, which is:

$$\Theta \left(\exp \left(\left(\frac{64}{9} n \right)^{\frac{1}{3}} (\log n)^{\frac{2}{3}} \right) \right)$$

For an ordinary computer, GNFS is the best known algorithm for large n . For a quantum computer, however, Peter Shor discovered an algorithm in 1994 that solves it in polynomial time! This will have significant implications for cryptography if a large quantum computer is ever built. Shor's algorithm takes only $O(n^3)$ time and $O(n)$ space. Forms of the algorithm are known that use only about $2n$ qubits. In 2001, the first 7-qubit quantum computer became the first to run Shor's algorithm. It factored the number 15.

It is not known exactly which complexity classes contain the integer factorization problem. The decision-problem form of it ("does N have a factor less than M ?") is known to be in both NP and co-NP. This is because both YES and NO answers can be checked if given the prime factors along with their primality proofs. It is known to be in BQP because of Shor's algorithm. It is suspected to be outside of all three of the complexity classes P, NP-Complete, and co-NP-Complete. If it could be proved that it is in either NP-Complete or co-NP-Complete, that would imply $NP = co-NP$. That would be a very surprising result, therefore integer factorization is widely suspected to be outside both of those classes. Many people have tried to find polynomial-time algorithms for it and failed, therefore it is widely suspected to be outside P.

Interestingly, the decision problem "is N a composite number?" (or equivalently: "is N a prime number?") appears to be much easier than the problem of actually finding the factors of N . Specifically, the former can be solved in polynomial time (in the number of digits of N), according to a recent preprint given in the references, below. In addition, there are a number of probabilistic algorithms that can test primality very quickly if one is willing to accept the small possibility of error. The easiness of prime testing is a crucial part of the RSA algorithm, as it is necessary to find large prime numbers to start with [39].

3.4.2 Discrete Logarithm Problem (DLP)

Taher ElGamal was the first mathematician to propose a public-key cryptosystem based on the Discrete Logarithm problem. He in fact proposed two distinct cryptosystems, one for encryption and the other for digital signatures. Since then, many variations have been made on the digital signature system to offer improved efficiency over the original system.

The **discrete logarithm problem (DLP)** is the following: given a prime p , a generator α of Z_p , and a non-zero element $\beta \in Z_p$, find the unique integer x , $0 \leq x \leq p-2$, such that $\beta \equiv \alpha^x \pmod{p}$. The integer x is called the **discrete logarithm** of β to the base α [8].

Based on the difficulty of this problem, Diffie and Hellman proposed the well-known Diffie-Hellman key agreement scheme in 1976. Since then, numerous other cryptographic protocols whose security depends on the DLP have been proposed, including: the ElGamal encryption and signature schemes, the U.S. government digital signature algorithm (DSA), the Schnorr signature scheme, and the Nyberg-Rueppel signature scheme.

And ElGamal encryption is the best-known example of a large system where the Discrete Logarithm algorithm is used.

Due to interest in these applications, mathematicians have extensively studied the DLP for the past 20 years [4].

3.4.2.1 Cryptosystem based on DLP

Let F_q be a finite field of q elements so that $q = p^n$ for some prime p and integer n . It is well known that the multiplicative group of nonzero elements of F_q , denoted by F_q^* , is a cyclic group of order $q-1$. Thus if α is a generator of this multiplicative group, then every nonzero element β in F_q is given by $\beta = \alpha^x$ for some integer x ; in fact for each β there is a unique integer in the range $0 \leq x \leq q-1$ with this property. For a given x and α , the power α^x can be quickly computed by the square-and-multiply method as demonstrated in Example below. The inverse problem, i.e., the problem of finding, for a given α and β , the x in the range $0 < x < q-1$ satisfying $\beta = \alpha^x$, is the discrete

logarithm problem; it is believed to be hard for many fields. Thus, exponentiation in finite fields is a candidate for a one-way function.

Example:

For the prime $p = 1999$, the ring Z_p is a finite field and the nonzero elements Z_p^* and Z_p form a group G under multiplication modulo p :

$$G = Z_p^* = \{1, 2, \dots, p-1\}.$$

Furthermore, the element $\alpha = 3$ is a generator of G , and is also known as a primitive element modulo p :

$$G = \{1, \alpha, \alpha^2, \dots, \alpha^{p-2}\} \bmod p.$$

It is easy to compute that

$$3^{789} \equiv 1452 \bmod p.$$

however, it is not nearly so easy to determine that $x = 789$, given only that x is in the range from 0 to 1997 and satisfies the equation

$$3^x \equiv 1452 \bmod 1999.$$

A more realistic challenge is to find an integer x such that

$$3x \equiv 2 \bmod p, \text{ where } p = 142 \cdot (10^{301} + 531) + 1.$$

We know a solution exists but we don't know the value.

The above discussion can be generalized to any group G (whose operation is written multiplicatively). The *discrete logarithm problem* for G is to find, for given $\alpha, \beta \in G$, a nonnegative integer x (if it exists) such that $\beta = \alpha^x$. The smallest such integer x is called the discrete logarithm of β to the base α , and is written $x = \log_\alpha \beta$. In previous Example, $\log_3 1452 = 789$. Clearly, the discrete logarithm problem for a general group G is exactly the problem of inverting the exponentiation function $\exp: Z_N \rightarrow G$ defined by $\exp(x) = \alpha^x$ where N is the order of α [14].

The Discrete Logarithm Problem consists mainly of transformations of the form $g^x \bmod p$, for some integer x and a fixed number g (between 0 and $p-1$). As with the RSA algorithm, these transformations raise the computational complexity of the problem. The discrete logarithm system relies on the discrete logarithm problem modulo p for security, and the speed of calculating the modular exponentiation for efficiency.

3.5 Elliptic Curve Discrete Logarithm Problem (ECDLP)

The elliptic curve discrete logarithm problem can be stated as follows. Fix an elliptic curve. xP represents the point P added to itself x times. Suppose Q is a multiple of P , so that $Q = xP$ for some x . Then the elliptic curve discrete logarithm problem is to determine x given P and Q .

The security of the ECC rests on the difficulty of the elliptic curve discrete logarithm problem. As is the case with the integer factorization problem and the discrete logarithm problem modulo p , no efficient algorithm is known at this time to solve the elliptic curve discrete logarithm problem.

One of the advantages of ECC is that the elliptic curve discrete logarithm problem is believed to be harder than both the integer factorization problem and discrete logarithm problem modulo p . This extra difficulty implies that ECC is one of the strongest public key cryptographic systems known today.

The elliptic curve discrete logarithm problem is relatively easy for a small class of elliptic curves, known as supersingular elliptic curves and also for certain anomalous elliptic curves. In both cases, the weak instances of the problem are easily identified, and an implementation merely checks that the specific instance selected is not one of the classes of easy problems.

Basically, elliptic curve cryptography is constructed on similar concepts to those used for discrete logarithm systems, but the discrete logarithm functions are performed on elliptic curves over finite fields.

A major factor in accepting ECC is the fact of smaller cryptographic key sizes. With small, electronic commerce and banking type transactions this may be an important consideration in overall system performance.

There are many possible algorithms to use for encryption with elliptic curves. As ElGamal Algorithm and Menezes-Vanstone Algorithm, many discrete logarithm problems can be converted to use elliptic curves. The newest version of IEEE's P1363 standard does not however define any encryption algorithm to use with elliptic curves.

In one of the earlier version of P1363 they recommended to use the ElGamal cryptosystem, but now it has been removed from the document. Because there is no recommendations which encryption algorithm to use [28].

Only two elliptic curve signature schemes are given in the IEEE P1363 standard: Nyberg-Rueppel and ECDSA. They are similar in overall security. The security of both schemes depends on the order of the base point being a large prime number.

3.6 Elliptic Curve Cryptography

3.6.1 Elliptic Curve Cryptosystems

Elliptic curve cryptosystems (ECCs) include key distribution, encryption, and digital signature algorithms. The key distribution algorithm is used to share a secret key, the encryption algorithm enables confidential communication, and the digital signature algorithm is used to authenticate the signer and validate the integrity of the message.

ECCs is based on the addition of rational points on a chosen elliptic curve.

An elliptic curve E over a Galois field $GF(p)$, where $p > 3$ and is prime, is the set of all (x, y) ($x, y \in GF(p)$) that satisfy the following equation:

$E: y^2 = x^3 + ax + b$ where $a, b \in GF(p)$, and $4a^3 + 27b^2 \neq 0$.

The rational points on the elliptic curve E are the points over $GF(p)$ that satisfy the defining equation. If the set of parameters $\{a, b, p\}$ are specified, the number of rational points on the elliptic curve is determined uniquely; this number is called the order of the elliptic curve E and is denoted by $\#E$. It is known that rational points form an additive group in the addition over the elliptic curve shown in **Figure 3.4**.

Instead of giving a hard definition of the addition of two rational points on an ECC over $GF(p)$, we will give an simple definition of addition

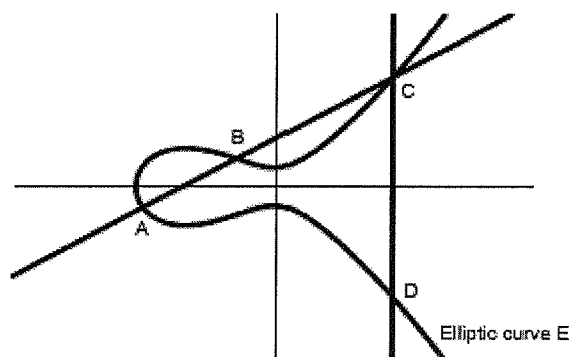


Figure 3.4 Addition rule over an elliptic curve.

by using an illustrative model of an ECC over real numbers.

When points A and B on the elliptic curve E shown in Figure above are added, the result is defined as the point D obtained by inverting the sign of the y-coordinate of point C, where point C is the intersection of E and the line passing through A and B. If A and B are at the same position, the line is the tangent of E at A.

Moreover, an ideally defined point O, namely the point at infinity, is also recognized as a point on E. The sum of the point at infinity and a point P is defined as point P itself.

We define the k scalar multiplication of a point G as the operation by which point G is added to itself k times. We denote the resulting point as kG . We can easily calculate $W = kG$ from a given k and G, but it is computationally difficult to calculate the scalar k from points W and G. If a prime p as large as 160 bits long is selected, we cannot find k within a reasonable time, even if we use the most efficient algorithms known so far and the world's most powerful computers. The problem of calculating k from given points G and W is called "the discrete logarithm problem over the elliptic curve." The security of ECC derives from the difficulty of solving the problem.

Moreover, when a point G on an elliptic curve E is given, there is a minimum positive integer n such that $nG = O$. Integer n is called the order of the point G. It is known that n is a divisor of the order of the curve E [17].

Elliptic curve cryptography is one of several ways to accomplish the objectives of security. Like any other cryptosystem, it isn't perfect, but it has definite advantages over other systems. The cryptosystem about to be described is called a public key cryptosystem, in that it involves the divulging of public information while allowing the individuals to retain and use private pieces of information ("keys").

Here's how it works. Alice and Bob agree upon a fixed elliptic curve E over a finite field Z_p and a "base point" $P \in E(Z_p)$. Nobody else knows these quantities. They also agree upon an encryption system; namely, given an "encoding key" z (a point on the elliptic curve) and a "message letter" m (another point on the elliptic curve), they determine how one might construct an "encoded message" $z * m$. (Please note that this $*$ has nothing to do with the group law on the elliptic curve; just think of $z * m$ as "m encoded by z".

Now suppose Alice wants to send Bob a message. First, she should assign letters of the alphabet to (distinct) points of $E(Z_p)$ to turn her (English) message into a sequence of points of $E(Z_p)$. Alice will then alert Bob that she is about to send a message. In response, Bob will choose a (probably fairly large) integer S_p , known as his secret, or private key. He will not divulge this to Alice or anyone else, however, he will send the quantity $S_p \cdot P = P + P + \dots + P$ (This is P added to itself S_p times) to Alice.

This is, of course, another point on the elliptic curve.

Now Alice picks an integer k as her private key and sends Bob two pieces of data: kP and $(kS_pP) * m$. Clearly she can compute kP since she knows k , P , and the elliptic curve on which P resides; she can also compute the latter quantity since she knows S_pP from Bob's transmission and can hence calculate $k(S_pP)$, her encryption key. Finally, she knows how to use this key to encode the message m , so she can compute $(kS_pP) * m$.

Bob then receives kP and $(kS_pP) * m$ from Alice. He doesn't know k , but he does know S_p and kP (the latter from Alice's transmission), so he can calculate the decryption key $S_p kP = kS_pP$. Hence he can find the decryption key, call it $(kS_pP)^{-1}$

and apply that to encrypted message $(kS_pP)^*m$ sent by Alice to recover $(kS_pP)^{-1}(kSPP)^*m = m$; the original message sent by Alice [18].

3.6.2 Security and Efficiency of ECC.

3.6.2.1 Security of ECC

One of the advantages of using elliptic curve based cryptographic systems instead of integer factorization or discrete logarithm based methods is that they provide similar security levels using smaller key lengths.

Why is this? As mentioned in the previous sections, the security of any public-key based cryptography is based upon the difficulty of solving certain mathematical problems. Thus, we can determine the amount of effort that would be required to break one of these public-key systems by looking at the effort required to solve these hard problems, using the best algorithms, software and hardware which are known. It should be noted that in the future new solutions to any of these problems might be discovered that drastically change the amount of effort required to solve them. The analysis below is based on the best methods known today.

Most people consider the integer factorization and discrete logarithm problems to have approximately equivalent security. Both of these problems have had intensive review and study by many of the world's top mathematicians and cryptographers. This can give us a sense of comfort that these problems are, in fact, difficult to solve. Actually, the best method known to solve each of these problems is the Number Field Sieve (NFS). The NFS is what is known as a **sub-exponential time method**. This means that the problem can be considered hard to solve, but not as hard as problems that only allow fully exponential solutions.

It is generally accepted that, based on the difficulty of solving the integer factorization problem and discrete logarithm problem, RSA, DSA and Diffie-Hellman keys should be at least 1024 bits long and that for very long-term security (20 years or more) 2048 bit keys should be used. Recently a large-scale effort was able to factor a 512-bit integer, thus showing that keys of this size are vulnerable to attack by large, sophisticated adversaries.

On the other hand, solving the ECDLP is generally considered to be a much more difficult problem than factoring integers or solving the discrete logarithm problem. Because of the structure that is inherent within an elliptic curve, the types of solutions to these problems do not seem to apply to the ECDLP. The best method known to solve the ECDLP is an elliptic curve version of an attack developed by Entrust researchers for the discrete logarithm problem, known as the parallel collision search method [VOW]. This method is **fully exponential**, which means that the ECDLP can be considered among the hardest types of problems to solve, using the best methods known today. One of the consequences of the ECDLP only having a fully exponential solution is that for every two additional bits of key used, attacking that key requires twice as much effort. Thus, attacking a 193-bit elliptic curve public key requires twice as much effort as attacking a 191-bit key.

Because it is relatively new, the ECDLP has not received as much attention from mathematicians and cryptographers as the integer factorization and discrete logarithm problem. Although, within the past few years, that has begun to change and a great deal of effort has been made at attempting to solve this problem.

Since a great deal of research is still ongoing, it is difficult to directly compare the security levels provided by ECC with those provided by RSA, DSA and Diffie-Hellman, for example. However, it seems reasonable that for security equivalent to an RSA key with 1024 bits, one should use an elliptic curve with about 170 bits and that for security equivalent to an RSA key with 2048 bits, one should use an elliptic curve with about 230 bits.

The above discussion on the difficulty of attacking an ECC public key assumes that certain weak cases have been avoided when constructing the elliptic curve parameters. There are certain elliptic curves that are known to produce cryptographic systems with a substantially lower security level than the general case described above. These weak cases include:

- A class of curves known as super singular elliptic curves;
- Elliptic curves modulo p which contain exactly p points; and
- Elliptic curves defined over a finite field with 2^m elements where m is not a prime.

Fortunately, each of these classes of weak curves is easy to identify and most standards bodies forbid their use. In order to guarantee that a given curve has not been intentionally constructed to somehow be weaker than expected and also to guard against possible future attacks against additional classes of weak curves, it is generally recommended to use elliptic curves which have been verifiably generated at random. This is the most conservative, or safest, option when choosing elliptic curves on which to base a system.

There exists another class of special elliptic curves, which requires attention. This class of even characteristic curves, called **Koblitz curves**, allows more efficient implementations. For very resource constrained environments these curves are an attractive option. However, some cryptographers are concerned that the additional structure exploited in these curves to obtain the efficient implementations may also be used to efficiently attack them. In fact, Entrust researchers were one of two independent groups, which were able to show that these curves provide a few bits less security than randomly generated curves. While the slight weakness demonstrated should not, in itself, stop people from using these curves, it does raise the question of how secure these curves really are. Entrust recommends that these curves be used with caution [19].

3.6.2.2 Efficiency of ECC

When talking about the efficiency of a public-key cryptographic system, there are three distinct factors to take into account:

- **Computational overheads** - how much computation is required to perform the public key and private key transformations.
- **Key size** - how many bits are required to store the key pairs and any system parameters.
- **Bandwidth** - how many bits must be communicated to transfer an encrypted message or a signature.

Clearly the comparisons should be made between systems offering similar levels of security, so in order to make the comparisons as concrete as possible, 160-bit ECC is compared with 1024-bit RSA and DSA. As indicated in Section 5.1, these parameter sizes offer comparable levels of security.

3.6.2.2.1 Computational Overheads

In each of the systems, considerable computational savings can be made. In RSA, a short public exponent can be employed (although this does incur some security risks) to speed up signature verification and encryption. In both DSA and ECC, a large proportion of the signature generation and encrypting transformations can be pre-computed. Also, various special bases for the finite field F_2^m can be employed to perform more quickly the modular arithmetic involved in ECC operation. State-of-the-art implementations of the systems show that with all of these efficiencies in place, ECC is an order of magnitude (roughly 10 times) faster than either RSA or DSA. The use of a short public exponent in RSA can make RSA encryption and signature verification timings (but not RSA decryption and signature generation timings) comparable with timings for these processes using the ECC.

3.6.2.2.2 Key Size

Table 3.2 compares the size of the system parameters and key pairs for the different systems.

Table 3.2 Size of system parameters and key pairs (approx)

	System parameter (bits)	Public key (bits)	Private key (bits)
RSA	N/A	1088	2048
DSA	2208	1024	160
ECC	481	161	160

It is clear from the figure that the system parameters and key pairs are shorter for the ECC than for either RSA or DSA.

3.6.2.2.3 Bandwidth

All three types of systems have similar bandwidth requirements when they are used to encrypt or sign long messages. However, the case when short messages are being transformed deserves particular attention, because public-key cryptographic systems are often employed to transmit short messages – for example to transport session keys for use in a symmetric-key cryptographic system. For the sake of a concrete comparison, suppose that each is being used to sign a 2000-bit message, or to encrypt

a 100-bit message. Table 3.3 compares the lengths of the signatures and encrypted messages respectively.

Table 3.3 signature size on long messages and size of encrypted 100-bits messages

	Signature size (bits)	Encrypted message (bits)
RSA	1024	1024
DSA	320	
ECC	320	321
ElGamal	--	2048

Therefore ECC offers considerable bandwidth savings over the other types of public-key cryptographic systems when being used to transform short messages.

In summary, the ECC provides greater efficiency than either integer factorization systems or discrete logarithm systems, in terms of computational overheads, key sizes, and bandwidth. In implementations, these savings mean higher speeds, lower power consumption, and code size reductions [40].

3.6.3 Comparison between ECC and RSA.

This section compares ECC public key sizes, signature and encryption lengths, and speed with those of RSA. Typical usage scenarios will be used to describe the effect these have on various implementations.

The ECC system under consideration will use an odd characteristic 192-bit elliptic curve, which is the default used by the Entrust product line. The RSA system will use 1024-bit keys, which is also the default in the Entrust product line.

3.6.3.1 Size key

The size of an ECC public key, ECDSA signature and an ECIES encryption will be compared with those produced by an RSA system.

3.6.3.1.1 Public Key Size

An RSA public key consists of an ordered pair (n, e) where n is a composite number, called the modulus, and e is the public exponent. In a 1024-bit RSA system, n will have 1024 bits. A common value for the public exponent is $e=216+1$. This is the value

that Entrust uses. Thus, an Entrust RSA public key would require 128 bytes for the modulus and 3 bytes for the public exponent. The total size is then 131 bytes.

An ECC public key consists of a point on the elliptic curve. Each point is represented by an ordered pair of elements (x, y) each with 192 bits. For a 192-bit elliptic curve, the public key would then be represented by two 24-byte values, giving a total key size of 48 bytes. As can be seen from the numbers above, ECC does provide a significant reduction in public key size. This reduction can be crucial in many severely constrained environments where large public keys are not possible. However, in a PKI using X.509 certificates, the effect of using the smaller public keys is minimal. A typical size for an X.509 certificate would be about 1K (~1000 bytes). Thus, changing a user's public key from RSA or DSA to ECC would reduce his/her certificate size by less than 10%.

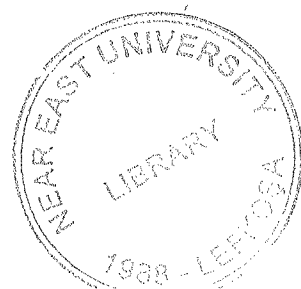
Another important point to keep in mind is that each ECC public key is only valid in the context of certain parameters. These parameters must also be specified and transferred with integrity to the public key recipient (e.g. within an X.509 certificate). While there do exist certain curves which can be represented using short identifiers, in the general case, it will require an additional five 192-bit (24-byte) quantities to specify these parameters. Thus, it could take up to 110 additional bytes. RSA does not require any parameters be transferred with the public key.

3.6.3.1.2 Signature Size

An RSA signature consists of a single 1024 bit value. Thus, it can be represented in 128 bytes.

An ECDSA signature consists of two 192-bit values. Thus, it can be represented using two 24-byte values, for a total signature size of 48 bytes. Again, the reduction in signature size is substantial and may be important for many constrained environments. However, as with public key size, the difference represents less than 10% of the size of a public key certificate. For larger signed messages, the difference would represent an even smaller percentage of the overall message.

3.6.3.1.3 Encryption Size



This section will compare the difference in size in transporting a 128 bit symmetric key using RSA and ECIES. This is the typical scenario when files are encrypted, for example. The encryption algorithm ECIES is specified in the ANSI X9.63 draft .

A 128 bit symmetric key encrypted using RSA will consist of one 1024-bit value. Thus, it can be represented using 128 bytes.

A 128 bit symmetric key encrypted using ECIES will consist of an elliptic curve point, a 128-bit value and a 160-bit value. The elliptic curve point consists of two 192-bit values, so it can be represented using two 24-byte values, or 48 bytes.³ the 128-bit value can be represented using 16 bytes and the 160-bit value can be represented using 20 bytes. Thus the encrypted symmetric key requires 84 bytes.

While ECIES does indeed produce smaller encrypted values than RSA, the difference is not as dramatic as for public keys and signature values. When considering that the symmetric key will then usually be used to encrypt much larger files, the advantage may become inconsequential.

3.6.3.2 Speed

This section will compare the time required to perform ECC signature and encryption operations with the time required for RSA signatures and encryption. Absolute timings for different cryptographic implementations can vary widely. These variations can be caused by a number of factors, including the quality of the implementation, the platform used, optimizations made which exploit certain special cases, or the use of proprietary or patented techniques not available to other implementers. For this reason it can often be misleading to directly compare any individual timings. It is better to consider general trends in timings for types of cryptographic systems. That is what we do here.

The first issue to consider is whether the implementation is in software or hardware. For a hardware implementation in custom silicon, even characteristic elliptic curves clearly allow the fastest implementations. This is due to the fact that the underlying arithmetic for even characteristic curves can be implemented using fewer gates, and thus in a smaller area, than the arithmetic for odd characteristic curves or for RSA.

Thus, the advantage these curves have is lost if the implementation is in software or firmware on an embedded processor. Odd characteristic elliptic curves and RSA however can take advantage of the integer mathematics routines that inherently exist on most computers and therefore they should be used for software implementations. The decision as to which of these two to use in a software implementation will depend on the particular environment in which the cryptography will be used. The remainder of this section will examine some common, general software environments in which this choice must be made.

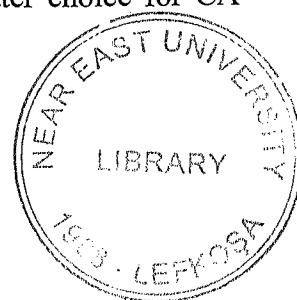
Using a small public exponent value (e.g. $e=216+1$), the RSA public key operations can be made very fast.

This means that RSA signature verification and RSA encryption can be performed up to 40 times faster than an ECDSA verify or ECIES encryption operation. The RSA private key operations (i.e. signature generation and decryption) are generally slower than the public key operations. The ECC private key operations are generally faster than the public key operations. The situation can be summarized in Table 3.4.

Table 3.4 Generation of Key in ECC and RSA

	RSA	ECC
Private Key Operations (Signature Generation and Decryption)	LESS FAST	FAST
Public Key Operations (Signature Verification and Encryption)	VERY FAST	LESS FAST

Let's now consider four common situations. First, let's consider the question of which algorithm would be best for CA signing keys. CA signing keys are used to sign certificates and CRLs. Each certificate and CRL only gets signed once, but is verified many times. In particular, every time that an end user's signature is verified or something is encrypted at least one certificate and CRL must be verified. Thus, it makes sense for CAs to try and minimize the amount of time end users spend on signature verification. In this situation, RSA appears to be a better choice for CA public keys.



Now let's consider end user signing keys. Each signature, regardless of the number of people to whom it will be sent or the number of times that it will be verified, will only be created once. In the general case however, many different people may potentially verify it many times. Thus, it makes sense in this situation, again, to try and minimize the time that signature verification takes. Since signature generation will only be performed once, we are not as concerned about the amount of time that operation takes. Thus, again, RSA appears to be a more optimal choice for end user signature keys.

Let's now look at what happens with end user encryption keys. When encrypting a document, the actual document will get encrypted with a symmetric algorithm (e.g. DES, CAST, IDEA) and then the symmetric key will be individually encrypted using the public-key algorithm for each recipient. Thus, the person encrypting the document must perform a separate public-key encryption for each recipient and therefore may need to perform many public-key encryptions each time a document is encrypted. Each recipient, however, will only need to perform one decryption operation each time access to the document is required.

Thus, in this situation it makes sense to try and minimize the amount of time the person encrypting the document must spend and, therefore, minimize the amount of time spent on encryption. Again, using RSA seems to make the most sense in this situation.

Finally, let's consider a general email application where a user wishes to digitally sign and encrypt an email for one recipient. Digitally signing the email requires one private key operation. Encrypting the email requires one public key operation to encrypt the symmetric key, which will be used to encrypt the contents of the email. However, before using the recipient's encryption public key, the sender must first validate the recipient's public key certificate. Validating the certificate will involve verifying the signature on at least one certificate and also verifying the signature on at least one CRL. Thus, at least three public key operations must be performed before the email can get sent. In this situation, the difference between the time required for an RSA-based system and an ECC-based system is very small. Either choice will result in a very fast implementation [19].

3.6.4 Possible to attack ECC.

In this section we overview known attacks on the discrete logarithm elliptic curve discrete logarithm problems and discuss how to avoid them in practice.

3.6.4.1 Naive Approach

The most basic method for solving the ECDLP (or the DLP) is to compute multiples of the generators Q (say, of order n) and store them in a table.

Considering an operation to be an elliptic curve addition, then this method take $O(n)$ time to build the table and $O(n)$ space to store it. The following methods are improvements upon these worst-case bounds [10].

3.6.4.2 Shanks' Method (baby-step, giant-step)

An algorithm of Daniel Shanks reduces the running time to $O(n^{1/2} \log n)$ and the space to $O(n^{1/2})$ points. The elliptic curve version of his algorithm is as follows. Assume that Q is a generator of order n , and, given A , we want to compute k such that $kQ = A$. Let $m = \lceil \sqrt{n} \rceil$. Store in a list L_1 the pairs (j, jmQ) for $0 \leq j \leq m-1$. Sort this list by the second element of the pairs.

Create a second list L_2 from the pairs $(i, -iQ + A)$ for $0 \leq i \leq m-1$. Sort this list by the second element. Search the lists until pairs $(j; P) \in L_1$ and $(i; P) \in L_2$ are found. Then

$$P = jmQ$$

$$P = -iQ + A$$

$$jmQ = -iQ + A$$

$$(jm + i)Q = A$$

Thus, the multiple of Q such that $kQ = A$ is $k = jm + i$. Of course, A must be a multiple of Q for this algorithm to work. One simple way to be certain of this is to choose an elliptic curve with order prime p . Since the order of any element must divide the order of the group, every element must have either order 1, namely the identity, or order p . The space requirement is for storage of the two lists, each composed of $n^{1/2}$ points. The time complexity is dominated by the sorting of these $n^{1/2}$ elements.

As an example consider $y^2 = x^3 + 2 \pmod{19}$. This curve has order 13. Choose as a generator $Q = (4, 16)$ and as the element of unknown index $A = (8, 18)$. Let $m = |\sqrt{13}| = 4$. Lists L_1 and L_2 work out as follows.

$$\text{List 1} = \{ (1, (9, 3)), (2, (12, 18)), (3, (4, 3)) \}$$

$$\text{List 2} = \{ (1, (8, 1)), (2, (12, 1)), (3, (9, 3)) \}$$

Item 1 in L_1 matches item 3 in L_2 . This indicates that $4Q = -3Q + A$ or $7Q = A$. This is correct [10].

3.6.4.3 Pohling-Hellman Attack

Pohling-Hellman attack is an elliptic curve equivalent of the standard DLP attack. In this attack the DLP is reduced to DLP in prime order subgroups and then the result is synthesized using the Chinese Remainder Theorem.

Let $n = p_1^{e_1} p_2^{e_2} \dots p_r^{e_r}$ be the prime factorization of the order of the group generator. If $Q = [k]P$, then first all $k_i = k \pmod{p_i^{e_i}}$ (for $i = 1, \dots, r$) has to be determined. Let $k_i = k_{i,0} + k_{i,1} p_i + \dots + k_{i,e_i-1} p_i^{e_i-1}$ is the p_i -ary representation of k_i and $\gamma_j = k_{i,0} + k_{i,1} p_i + \dots + k_{i,j-1} p_i^{j-1}$. Then

$$\begin{aligned} Q_{i,j} &= \left[\frac{n}{p_i^{j+1}} \right] (Q - [\gamma_j]P) = [k - \gamma_j] \left[\frac{n}{p_i^{j+1}} \right] P \\ &= [k_{i,j} p_i^j + \dots + k_{i,e_i-1} p_i^{e_i-1}] \left[\frac{n}{p_i^{j+1}} \right] P \\ &= [k_{i,j} p_i^j + \dots + k_{i,e_i-1} p_i^{e_i-1}] \left[\frac{n}{p_i} \right] P \\ &= [k_{i,j}] P' \end{aligned}$$

where $P' = [n/p_i] P$. (The last equality holds since P' has order p_i .) Hence, after calculating ECDLPs for $Q'_{i,j} = [k_{i,j}] P'$ for all j from 0 to $e_i - 1$ using another method (e.g. exhaustive search), $k_i = k \pmod{p_i^{e_i}}$ can be determined. And after obtaining all values k_i ($i = 1, \dots, r$), the solution is computed using the Chinese Remainder

Theorem as $k = \sum_{i=1}^r k_i N_i M_i \pmod{n}$, where $N_i = n/p_i$ and $M_i = N_i^{-1} \pmod{p_i}$.

Given the factorization of n , the Pohling-Hellman algorithm needs

$O(\sum_{i=1}^r e_i (\lg n + \sqrt{p_i}))$ steps to solve an ECDLP.

In order to avoid this attack, the order of the group generator must be divisible by a large prime. Preferably, the order should be a prime or a prime multiplied by a small integer [1].

3.6.4.4 Index And Xedni Calculus

Index method is the most efficient method known to solve the standard DLP — it runs in sub exponential time. Although most methods for the standard DLP are applicable also to the ECDLP, no practical index calculus method has been found for the ECDLP, and there are theoretical as well as practical reasons to believe that the ECDLP is not susceptible to this kind of attacks.

There is another attack based on the idea of the index calculus — Xedni calculus (Xedni is index backwards) —, which might be used to solve the ECDLP. This method is applicable also to the standard DLP as well as to the integer factorization problem, and because the main step of the method resisted quantification necessary to evaluate its complexity, it seemed to be a big threat for all nowadays widely used cryptographic systems. However, it turned out later that this method is unpractical [1].

3.6.4.5 Special-Purpose Attacks

Although there is no known general sub-exponential algorithm to solve the ECDLP, two classes of elliptic curves over F_p have special properties, which disqualify them from the cryptographic use at all (they are also prohibited by all standards considering ECC):

1. Supersingular curves. A curve over F_p is super singular if $\#E(F_p) = p + 1$. There are reduction attacks (Menezes-Okamoto-Vanstone and Frey-Ruck) which reduce the discrete logarithm problem in these curves to the standard DLP in an extension field F_{p^B} for some $B > 1$, where the sub-exponential number field sieve algorithm applies. The attacks are only practical if B is small. Supersingular curves can be excluded by the MOV condition: For $k = 1, 2, \dots, B$ verify that n does not divide $p^k - 1$. (This

condition does not exclude only supersingular curves, but all curves susceptible to the mentioned attacks.) For an implementation, it is important to choose appropriate value for the constant B . ANSI X9.62 recommends to use $B \geq 20$ to ensure resistance against the reduction attacks.

2. Anomalous curves. A curve over F_p is anomalous if $\#E(F_p) = p$. There are methods (Semaev, Smart, Satoh-Akari), which solve the ECDLP for these curves efficiently. They can be excluded by the Anomalous condition: Verify that $\#E(F_p) \neq p$ [1].

3.6.4.6 Suggestions to avoid

In order to avoid attacks mentioned in this section, the elliptic curve E over a prime field F_p and the order n of the generator of the group of elliptic curve points on E should satisfy this properties:

1. The curve is not anomalous, i.e. $n \neq p$.
2. The smallest value of B such that $p^B \equiv 1 \pmod{n}$ is large (at least 20).
3. The group of all points on the curve has a subgroup with a large prime order [1].

3.6.5 Standards of the ECC algorithms.

The development of standards is a very important point for the use of a cryptosystem. Standards help ensure security and interoperability of different implementations of one cryptosystem. There are several major organizations that develop standards.

The most important for security in information technology are the:

- International Standards Organization (ISO),
- American National Standards Institute (ANSI),
- Institute of Electrical and Electronics Engineers (IEEE),
- Federal Information Processing Standards (FIPS).

The most famous ECC algorithm, the ECDSA was accepted 1998 as ISO standard (ISO14888-3), 1999 as ANSI standard (ANSI X9.62), and 2000 as IEEE (P1363) and FIPS (186-2) standard. Several other standardization efforts are in progress. The most famous is the ANSI X9.63, which includes dozens of key transport and key agreement

schemes. Table 3.5 provides an overview of the various standards and the included algorithms [25].

Table 3.5 Standards and algorithms [16].

Standard	Schemes	Status
ANSI X9.62	ECDSA	Approved
ANSI X9.63	ECIES,ECDH,ECMQV	Draft
FIPS 186-2	ECDSA	Approved
IEEE P1363	ECDSA,EDDH,ECMQV	Approved
IEEE P1363A	ECIES	Draft
ISO 14888-3	ECDSA	Approved
ISO 15946	ECDSA,EDDH,ECMQV	Draft

The relationship between these standards is visualized in Figure 3.5.

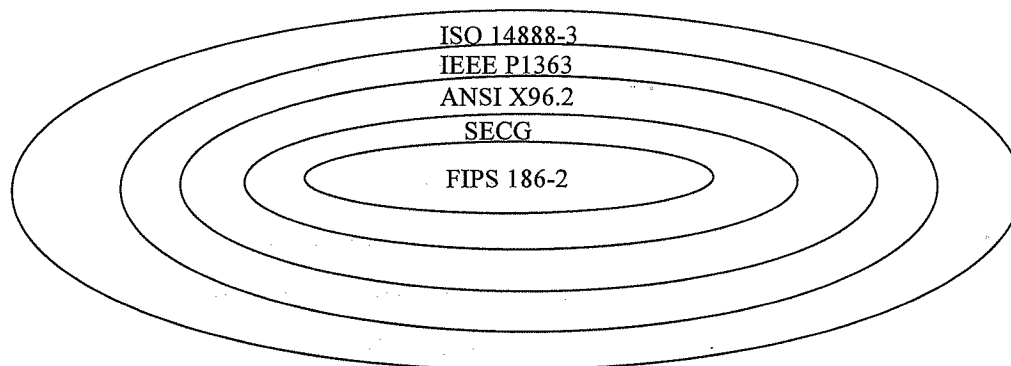


Figure 3.5 Standards [16].

There is a company with a certain interest in elliptic curve cryptography, which enforces the integration of ECC in several commonly used protocols. We will mention some of them. For example in the recent version of the TLS/SSL (Transport Layer Security/Secure Socket Layer) protocol basically ECC is supported. In its recent implementation of this protocol, Certicom has already included this feature.

Another important application is the Wireless Application Protocol (WAP/WTLS). It ensures secure wireless communications and includes in its recent version ECC in the WTLS layer. Also in the ATM Security Specification 1.0 ECC is included. They used modified variants of the ANSI algorithms in order to avoid the costly inversion in the signature generation [16].

Table 3.6 compares the two algorithms.

Table 3.6 ECDSA vs. ECDSA-Like

ECDSA-Like	ECDSA
$r = x \bmod$	$r = x \bmod$
$s = (kr - e)d^{-1}$	$s = k^{-1} (e + dr)$

The Secure/Multipurpose Internet Mail Extensions (S/MIME) standard specifies security mechanisms for electronic communications. S/MIME can be applied to all compatible MIME formats, such as http for example. The internet Draft Elliptic Curve S/MIME tries to embed ECC in S/MIME [25, 16].

3.7 Elliptic Curve Protocols

3.7.1 Elliptic Curve Diffie-Hellman Algorithm (ECDHA)

The first public key cryptosystem ever invented was Diffie-Hellman. Because the computation behind public key cryptosystems takes a relatively long time when compared to classical cryptosystems, public key cryptography is often used in a modified role along with a symmetric cryptosystem to transmit hidden messages. In 1976, Diffie and Hellman provided a detailed method of agreeing upon a key for a symmetric system using public key methods, the mathematics of which were based on the discrete log problem.

This section focuses on the elliptic-curve Diffie-Hellman key exchange as an example to illustrate the differences of elliptic-curve algorithms from integer algorithms. We first review the integer version.

3.7.1.1 Integer Diffie-Hellman key Exchange.

Diffie-Hellman key exchange (DHKE) is used to establish a shared key between two parties over a public channel. It is based on the multiplicative group Z_p^* of integers, where p is a prime. **Figure 3.6** shows how a secret key, K , is agreed upon between Alice and Bob by exchanging two quantities a_T and b_T publicly. We assume that the

two parties Alice and Bob have agreed on p and α values in advance, where α is a primitive element of the group Z_p^* . [4]

Step	Alice		Bob
1	Choose random a $a \in [2, p-2]$		Choose random b $b \in [2, p-2]$
2	Compute a_T $A_T = \alpha^a \bmod p$		Compute b_T $b_T = \alpha^b \bmod p$
3	Send a_T Receive b_T	$a_T \rightarrow$ $\leftarrow b_T$	Send b_T Receive a_T
4	Compute key K $K = (b_T)^a \bmod p$ $= \alpha^{ab} \bmod p$		Compute key K $K = (a_T)^b \bmod p$ $= \alpha^{ab} \bmod p$

Figure 3.6 Diffie-Hellman key exchange (integer) [4].

The security of the integer Diffie-Hellman is based on the Discrete Logarithm problem: given p , α and a_T , it is computationally infeasible to compute a (for sufficiently large p). Therefore, even though an eavesdropper may capture the intermediate values a_T and b_T as they are exchanged over the public channel, neither a nor b will be exposed, and therefore the final key K remains known only to Alice and Bob.

3.7.1.2 Elliptic-curve Diffie-Hellman key Exchange.

This protocol establishes a shared key between two parties. The original Diffie-Hellman algorithm is based on the multiplicative group modulo p , while the elliptic curve Diffie-Hellman (ECDH) protocol is based on the additive elliptic curve group.

Elliptic-curve Diffie-Hellman key exchange (ECDHKE) is similar to the integer version except that it uses the points on an elliptic-curve rather than integers. We assume that Alice and Bob have previously agreed on a binary field $GF(p)$, a common elliptic curve E with suitable coefficients, and a base point $P=(x,y)$, which lies on E and has order n .

To generate a key, first Alice chooses a random $a \in \mathbb{F}_p$ (of high order) which she keeps secret. Next she calculates $aP \in E$ which is public and sends it to Bob. Bob does the same steps, i.e. he chooses a random integer b (secret) and calculates bP , which is sent to Alice. Their secret common key is then $K = abP \in E$. The **Figure 3.7** show the idea in simple word.

Step	Alice		Bob
1	Choose random a $a \in [2, n-1]$		Choose random b $b \in [2, n-1]$
2	Compute A_T $A_T = P \times a$		Compute B_T $B_T = P \times b$
3	Send A_T Receive B_T	$A_T \rightarrow$ $\leftarrow B_T$	Send B_T Receive A_T
4	Compute key K $K = B_T \times a$ $= P \times a \times b$		Compute key K $K = A_T \times b$ $= P \times a \times b$

Figure 3.7 Elliptic Curve Diffie-Hellman key exchange

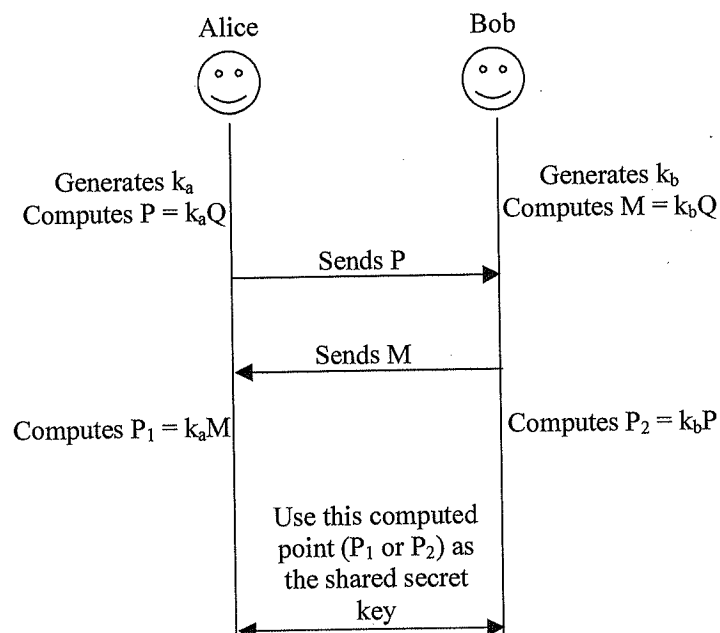


Figure 3.8 Illustration of Elliptic Curve Diffie-Hellman Protocol [9].

At the end of the protocol the communicating parties end up with the same value K that is a point on the curve. A part of this value can be used as a secret key to a secret-key encryption algorithm.

The security of the EC-DHKE is based on the elliptic curve Discrete Logarithm problem: given $GF(p)$, E , P , A , it is computationally infeasible to compute a (for sufficiently large k and n). Unlike the integer discrete logarithm problem, the elliptic curve discrete logarithm problem has no known sub-exponential time solutions (for a well-chosen set of system parameters) [4].

Accordingly, a given level of security can be achieved with a k smaller than the number of bits required to encode the p in the integer Diffie-Hellman key exchange.

Table 3.7 provides a comparison of the basic features of 1024-bit integer Diffie-Hellman and 160-bit elliptic-curve Diffie-Hellman key exchanges.

Table 3.7 Comparison of integer Diffie-Hellman and elliptic curve Diffie-Hellman

	DHKE	EC-DHKE
Group	Integers in Z_p^*	Points on the elliptic curve E
Base object	$\alpha \in Z_p^*$	Elliptic Curve point $P \in E$
Primary operation	Exponentiation ($\alpha^a \bmod p$)	Point multiplication ($P \times a$)
Key length	1024 bits	160 bits

3.7.2 Digital Signature

The ECDSA is the elliptic curve analog of the DSA. Vanstone first proposed ECDSA in 1992 in response to NIST's (National Institute of Standards and Technology) request for comments on their first proposal for DSS. Digital signature schemes are the counterpart to handwritten signatures. A digital signature is a number that depends on the secret key only known by the signer and on the contents of the message being signed. Signatures must be verifiable without access to the signer's private key. Signatures should be existentially unforgeable under chosen-message attacks. This

asserts that an adversary who is able to obtain Alice's signatures for any messages of his choice cannot forge Alice signature on a single other message.

3.7.2.1 Digital Signature Algorithm (DSA)

The DSA was proposed in August 1991 by the U.S. National Institute of Standards and Technology (NIST) and became a U.S. Federal Information Processing Standard (FIPS 186) in 1993. It was the first digital signature scheme to be accepted as legally binding by a government.

The algorithm is a variant of the ElGamal signature scheme. It exploits small subgroups in \mathbb{Z}_p^* in order to decrease the size of signatures. The key generation, signature generation, and signature verification procedures for DSA [24].

The following figures describe the DSA in details. There are three parameters that are public and can be common to a group of users. A 160-bit prime number q is chosen. Next, a prime number p is selected with a length between 512 and 1024 bits such that q divides $(p-1)$. Finally, g is chosen to be of the form $h^{(p-1)/q} \bmod p$, where h is an integer between 1 and $(p-1)$ with the restriction that g must be greater than 1.

With these numbers in hand, each user selects private key and generates a public key. The private key x must be a number from 1 to $(q-1)$ and should be chosen randomly or pseudorandomly. The public is calculated from the private key as $y = g^x \bmod p$. The calculation of y given x is relatively straightforward. However, given the public key, it is believed to be computationally infeasible to determine x , which is the discrete logarithm of y to the base g , mod p . we summarize the in the following:

Global Public-Key Components	
P	prime number where $2^{L-1} < p < 2^L$ for $512 \leq L \leq 1024$ and L a multiple of 64; i.e bit length of between 512 and 1024 bits in increment of 64 bits.
q	prime divisor of $(p-1)$, where $2^{159} < q < 2^{160}$; i.e bits length of 160 bits.
g	$= h^{(p-1)/q} \bmod p$; where h is any integer with $1 < h < (p-1)$ such that $h^{(p-1)/q} > 1$.

User's Private Key	
x	random or pseudorandom integer with $0 < x < q$

User's Public Key	
y	$=g^x \bmod p$

User's Per-message Select Number	
k	random or pseudorandom integer with $0 < k < q$

To create signature, a user calculates two quantities, r and s, that are functions of the public key components (p, q, g), the user's private key (x), the hash code of the message, H(M), and additional integer k that should be generated randomly or pseudorandomly and be unique for each signing.

Signing	
r	$= (g^x \bmod p) \bmod p$
s	$= [k^{-1}(H(M) + xr)] \bmod p$
signature = (r,s)	

At receiving end, verification is performed using the formulas shown in the following figure. The receiver generates a quantity v that is a function of the public key components, the sender's public key, and hash code of the incoming message. If this quantity matches the r components of the signature, then the signature is validated [29].

Verifying	
w	$= (s^{-1}) \bmod p$
u1	$= [H(M')w] \bmod p$
u2	$= (r')w \bmod p$
v	$= [(g^{u1}y^{u2}) \bmod p] \bmod p$
TEST: $v=r'$	

3.7.2.2 Elliptic Curve Digital Signature Algorithm (ECDSA)

ECDSA is the elliptic curve analogue of the DSA. That is, instead of working in a subgroup of order q in \mathbb{Z}_p^* , we work in an elliptic curve group $E(\mathbb{Z}_p)$. The ECDSA is currently being standardized within the ANSI X9F1 and IEEE P1363 standards committees. The following table shows the correspondence between some math notation used in DSA and ECDSA [24].

Table 3.8 Correspondence between DSA and ECDSA notation [24].

DSA notation	ECDSA notation
Q	n
G	P
X	d
Y	Q

We next describe the elliptic curve analogue (ECDSA) of the U.S. government digital signature algorithm (DSA). The ECDSA is an ANSI standard and is also being considered by the ANSI X9F1 and IEEE P1363 standards committees as a digital signature standard [25].

Suppose Alice wants to send a digitally signed message to Bob. They first choose a finite field \mathbb{F}_q , an elliptic curve E , defined over that field and a base point P with order n . Alice's key pair is (d, Q) , where d is her private and Q is her public key. To sign a message M Alice does the following:

ECDSA Key Generation. E is an elliptic curve defined over \mathbb{F}_q , and P is a point of prime order n in $E(\mathbb{F}_q)$, these are system-wide parameters. For simplicity, we shall suppose that q is a prime, although the construction can easily be adapted to a prime power q as well. Each entity A does the following:

1. Select a random integer d in the interval $[1, n - 1]$.
2. Compute $Q = dP$.
3. A 's public key is Q , A 's private key is d .

ECDSA Signature Generation. To sign a message m , A does the following:

1. Select a random integer k in the interval $[1, n - 1]$.
2. Compute $kP = (x_1, y_1)$ and $r = x_1 \bmod n$ (where x_1 is regarded as an integer between 0 and $q - 1$). If $r = 0$ then go back to step 1.
3. Compute $k^{-1} \bmod n$.
4. Compute $s = k^{-1} \{h(m) + dr\} \bmod n$, where h is the Secure Hash Algorithm (SHA-1). If $s = 0$, then go back to step 1.
5. The signature for the message m is the pair of integers (r, s) .

ECDSA Signature Verification. To verify A 's signature (r, s) on m , B should do the following:

1. Obtain an authenticated copy of A 's public key Q .
2. Verify that r and s are integers in the interval $[1, n - 1]$.
3. Compute $w = s^{-1} \bmod n$ and $h(m)$.
4. Compute $u_1 = h(m)w \bmod n$ and $u_2 = rw \bmod n$.
5. Compute $u_1P + u_2Q = (x_0, y_0)$ and $v = x_0 \bmod n$.
6. Accept the signature if and only if $v = r$. [25].

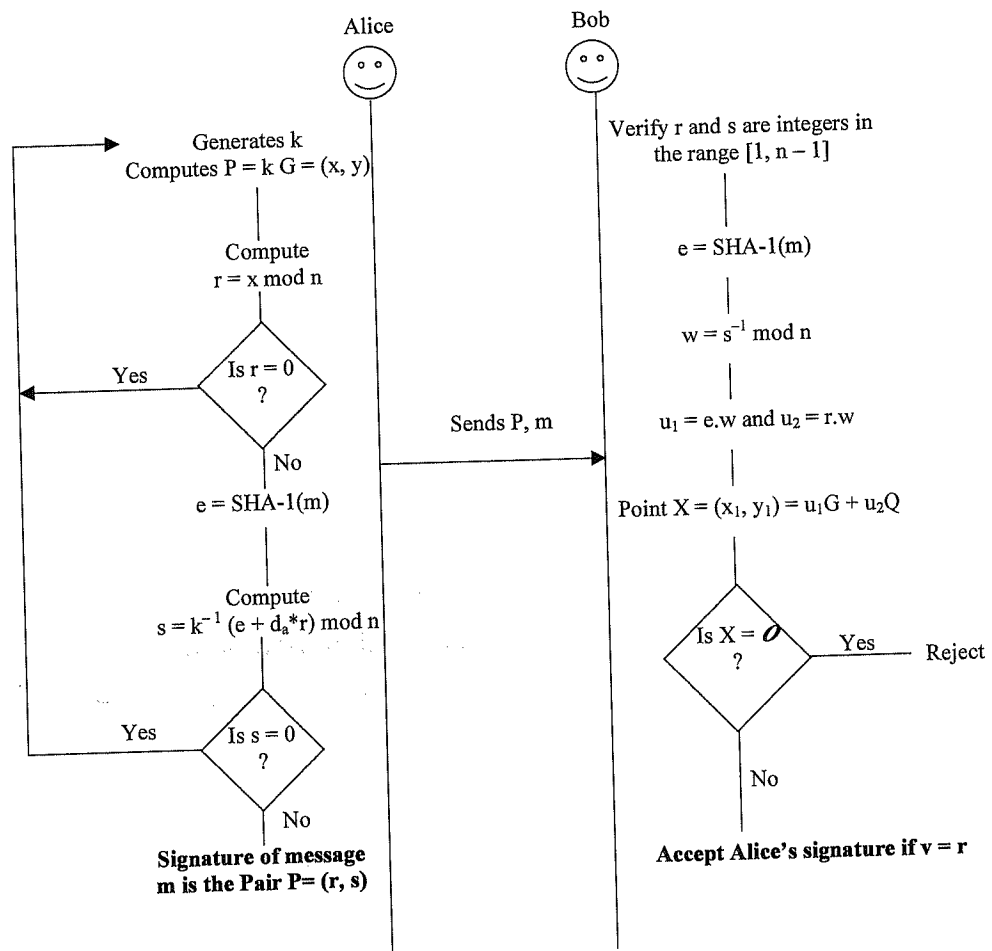


Figure 3.8 Illustration of Elliptic Curve Digital Signature Algorithm [9].

The only significant difference between ECDSA and DSA is in the generation of r . The DSA does this by taking the random element $(\alpha^k \bmod p)$ and reducing it modulo q , thus obtaining an integer in the interval $[1, q - 1]$. (In the DSA, q is a 160-bit prime divisor of $p - 1$, and α is an element of order q in F_p^* .) The ECDSA generates the integer r in the interval $[1, n - 1]$ by taking the x -coordinate of the random point kP and reducing it modulo n .

To obtain a security level similar to that of the DSA, the parameter n should have about 160 bits. If this is the case, then DSA and ECDSA signatures have the same bitlength (320 bits).

Instead of using system-wide parameters, we could fix the underlying finite field F_q for all entities, and let each entity select its own elliptic curve E and point $P \in E(F_q)$. In this case, the defining equation for E , the point P , and the order n of P must also be

included in the entity's public key. If the underlying field F_q is fixed, then hardware or software can be built to optimize computations in that field. At the same time, there are an enormous number of choices of elliptic curves E over the fixed F_q [25].

Other features of ECDSA

- Since the public key of an individual is not solely determined by the elliptic curve $E(\mathbb{Z}_n)$ used, nor on the point P , many users can use the same curve $E(\mathbb{Z}_n)$ and point P , and thus an individual's private key consists solely of Q . This is good as it reduces the size of the public-key.
- Since ECDSA defines private keys as per-signature values as statistically unique and unpredictable rather than being simply random, as per the DSA standard, it is more secure than DSA (as only "non-weak" keys are considered).
- ECDSA makes use of deterministic primality testing, while DSA only requires probabilistic testing. Thus numbers which are claimed to be prime can be verified to indeed be prime, and thus avoid potential security hazards [2].

3.7.3 Encryption (ElGamal Elliptic Curve)

3.7.3.1 ElGamal Cryptosystem

This cryptosystem is based on the difficulty of the discrete logarithm problem. Given g , and p , it is difficult to find a when: $g^a \equiv (\text{mod } p)$, a in which p is a prime number larger than two; $1 < a < p-1$; $2 \leq a \leq p-2$; and g is a primitive element of a group of numbers. Note that $a \neq 1$. In addition, $a \neq 1$, for reasons as mentioned later.

Hence a is limited to values less than $p-2$, since by the little Fermat's theorem $g^{p-1} \equiv (\text{mod } p)$ is always true if g and p are co-prime. As shown below, having a value of a less than 2 is of no practical use, as a trapdoor.

The public key here is (p, g, a) and it is used in the encryption and decryption processes. To send the plaintext x , the sender performs the following operations to generate the ciphertext $(y1, y2)$:

Encryption function: $y1 \equiv g^k \pmod{p}$
 $y2 \equiv x d^k \pmod{p}$;

Decryption function: $x \equiv y2 (y1^a)^{-1} \pmod{p}$

which is feasible because

$$\begin{aligned} y2 (y1^a)^{-1} &\equiv x d^k (g^{ak})^{-1} \pmod{p} \\ &\equiv x (g^a)^k (g^{ak})^{-1} \pmod{p} \\ &\equiv x, \end{aligned}$$

in which k is a randomly chosen integer?

Thus cannot be equal to 1 if the plaintext x is to be effectively masked [26].

Security

Breaking ElGamal is believed to be, by most informed observers, generally as difficult as solving the discrete logarithm problem. If the discrete logarithm problem could be solved efficiently, then ElGamal could be broken. However, it remains possible that there may be some way to break ElGamal without having to solve that problem.

3.7.3.2 ElGamal Elliptic Curve Cryptosystem

How can we set up a public-key cryptosystem using an elliptic curve? The only public-key cryptosystem that we've studied so far is the RSA cryptosystem; unfortunately, there is no analogue of RSA for elliptic curves!

The original ElGamal cryptosystem was developed on the standard discrete logarithm problem.

This algorithm has expansion factor of two, i.e. for each plaintext field element, the corresponding ciphertext consists of a pair of field elements. An elliptic curve analogue of the ElGamal cryptosystem has a message expansion factor of four, i.e. for each plaintext field element, the corresponding ciphertext consists of four field elements. Moreover, the plaintext space has to consist of points on the elliptic curve

and there is no known deterministic method of generation of points on an elliptic curve.

Now we describe the ElGamal elliptic curve version in simple words. Bob choose an elliptic curve $E(\text{mod } p)$, where p is a large prime α on E and a secret integer a . he computes

$$\beta = a * \alpha \quad (= \alpha + \alpha + \dots + \alpha).$$

The points α and β are public, while a is kept secret. Alice expresses her message as a point x on E . she choose a random integer k , and computes

$$y1 = k * \alpha \text{ and}$$

$$y2 = x + k * \beta,$$

and sends the pair $PC[y1, y2]$ to Bob. Bob decrypts by calculating

$$x = y2 - ay1.$$

Example: (Elliptic curve encryption):

Consider the following elliptic curve:

$$y^2 = x^3 + ax + b \text{ mod } p$$

$$y^2 = x^3 - x + 188 \text{ mod } 751$$

that is: $a = -1$, $b = 188$, and $p = 751$. The elliptic curve group generated by the above elliptic curve is then $E_A(a, b) = E_{751}(-1, 188)$.

Let the generator point $\alpha = (0, 376)$. Then the multiples $k\alpha$ of the generator point α are (for $1 \leq k \leq 751$):

$\alpha = (0, 376)$	$2\alpha = (1, 376)$	$3\alpha = (750, 375)$	$4\alpha = (2, 373)$
$5\alpha = (188, 657)$	$6\alpha = (6, 390)$	$7\alpha = (667, 571)$	$8\alpha = (121, 39)$
$9\alpha = (582, 736)$	$10\alpha = (57, 332)$...	
$761\alpha = (565, 312)$	$762\alpha = (328, 569)$	$763\alpha = (677, 185)$	
$764\alpha = (196, 681)$	$765\alpha = (417, 320)$	$766\alpha = (3, 370)$	
$767\alpha = (1, 377)$	$768\alpha = (0, 375)$	$769\alpha = \mathcal{O}(\text{point at infinity})$	

If Alice wants to send to Bob the message M which is encoded as the plaintext point $x = (443, 253) \in E_{751}(-1, 188)$. She must use Bob public key to encrypt it. Suppose that Bob secret key is $a_B = 85$, then his public key will be:

$$\beta = a_B \alpha = 85(0, 376)$$

$$\beta = (671, 558)$$

Alice selects a random number $k = 113$ and uses Bob's public key $\beta = (671, 558)$ to encrypt the message point into the ciphertext pair of points:

$$PC = [(k\alpha), (x + k\beta)]$$

$$PC = [113 \times (0, 376), (443, 253) + 113 \times (671, 558)]$$

$$PC = [(34, 633), (443, 253) + (47, 416)]$$

$$PC = [(34, 633), (217, 606)]$$

Upon receiving the ciphertext pair of points, $PC = [(34, 633), (217, 606)]$, Bob uses his private key, $a_B = 85$, to compute the plaintext point, x , as follows

$$(x + k\beta) - [a_B(k\alpha)] = (217, 606) - [85(34, 633)]$$

$$(x + k\beta) - [a_B(k\alpha)] = (217, 606) - [(47, 416)]$$

$$(x + k\beta) - [a_B(k\alpha)] = (217, 606) + [(47, -416)] \text{ (since } -P = (x_1, -y_1))$$

$$(x + k\beta) - [a_B(k\alpha)] = (217, 606) + [(47, 335)]$$

$$\text{(since } -416 \equiv 335 \pmod{751})$$

$$(x + k\beta) - [a_B(k\alpha)] = (443, 253)$$

and then maps the plaintext point $PM = (443, 253)$ back into the original plaintext message M [27].

3.7.4 Menezes-Vanstone Elliptic Curve Cryptosystem

The Menezes-Vanstone elliptic curve cryptosystem is defined as follows. Let E be an elliptic curve defined over Z_p ($p > 3$ prime), or in $GF(p^n)$ with $n > 1$, such that E contains a cyclic subgroup H in which the discrete logarithm problem is intractable.

Let $P = Z^*_p \times Z^*_p$, $C = E \times Z^*_p \times Z^*_p$, and define $K = \{(E; \alpha; a; \beta) : \beta = a\alpha\}$;

Where $\alpha \in E$. the values α and β are public and a is secret. For $K = (E, \alpha, a, \beta)$, for a (secret) random number $k \in Z_{|H|}$, and for $x = (x_1, x_2) \in Z^*_p \times Z^*_p$, define $e_K(x, k) = (y_0, y_1, y_2)$,

Where

$$y_0 = k\alpha,$$

$$(c_1, c_2) = k\beta$$

$$y_1 = c_1 x_1 \pmod{p}$$

$$y_2 = c_2 x_2 \pmod{p}.$$

For a ciphertext $y = (y_0, y_1, y_2)$, define

$$d_K(y) = (y_1 c_1^{-1} \pmod{p}, y_2 c_2^{-1} \pmod{p}),$$

where

$$ay_0 = (c_1, c_2).$$

The Menezes-Vanstone cryptosystem is a more efficient variation of the well-known ElGamal cryptosystem [28].

There are some practical difficulties in implementing an ElGamal cryptosystem on an elliptic curve. The ElGamal cryptosystem, when implemented in Z_p , has a message expansion factor of two. An elliptic curve implementation has a message expansion factor of (about) four. This happens because there are approximately p plaintexts, but each ciphertext consists of four field elements. A more serious problem is that the plaintext space consists of the points on the curve E , and there is no convenient method known to deterministically generate points on E .

In the Menezes-Vanstone variation of the ElGamal cryptosystem an elliptic curve is used for masking, and plaintexts and ciphertexts are allowed to be arbitrary ordered pairs of (nonzero) field elements (i.e. they are not required to be points on E). This yields a message expansion factor of two, the same as original ElGamal cryptosystem.

In addition, using point compression can reduce the message expansion. That is, the y -coordinate of the point can be recovered given its x -coordinate and a single bit of extra information. This yields a message expansion factor of 1.5 [28.]

3.8 Summary

This chapter described the Elliptic Curve Cryptography as bellow; the chapter gave the history of elliptic curve, elliptic curve in mathematics side, the problems that public key algorithms depends on and described it in deep, then it study the elliptic curve cryptosystems in more details as showed general idea how elliptic curve cryptography works, elliptic curve discrete logarithm problem and how the security if ECC depends on it, the showed the security and efficiency of ECC compared with RSA algorithm, then possible attack to ECC and how we can avoid from these ways, at last described the ECC protocols and how developed the some algorithm that worked without elliptic curve cryptography schema.

4. IMPLEMENTATION ELGAMAL ECC ENCRYPTION USING VISUAL C++ 6.0

4.1. Overview

With the rapid growth of the Internet technology and its application especially in electronic commerce and electronic mail systems, the need for more secure communication channel has become critical for surviving in this open world. As we have mentioned before cryptography systems is the solution for this problem of security and the Elliptic Curve Cryptography using ElGamal ECC encryption protocol has helped to fill that need by providing an easy to understand implementation of Public key encryption.

This chapter is to introduce an application developed to secure transaction over communication channel and by using public key cryptography and the ElGamal ECC protocol.

4.2. Program Explanation

What we will introduce here is how to use my application to secure your data transfer over communication channel, this application was developed by using VISUAL C++ 6.0 programming language and the source code of this application is found in Appendix A.

The reason for using VISUALC++ 6.0 programming language is, VISUAL C++ is a powerful and complex tool for building 32-bit application for Windows 9X and Windows NT. These applications are much larger and more complex than their predecessors for 16-bit windows or older program that didn't use a graphical user interface. Yet, as program size and complexity has increased, programmer effort has decreased, at least for programmers who are using right tools.

Visual C++ 6.0 is one of the right tools. With its code-generating wizards, it can produce the shell of a working Windows application in second. The classes library include with visual C++, the Microsoft Foundation Classes (MFC), has become the industry standard for windows software development in a variety of C++ compilers. The visual editing tools make layout of menus and dialogs and snap. The time you invest in

learning to use this product will pay for itself on your first windows programming project.

The following flow chart will explain the step of how work on the application:

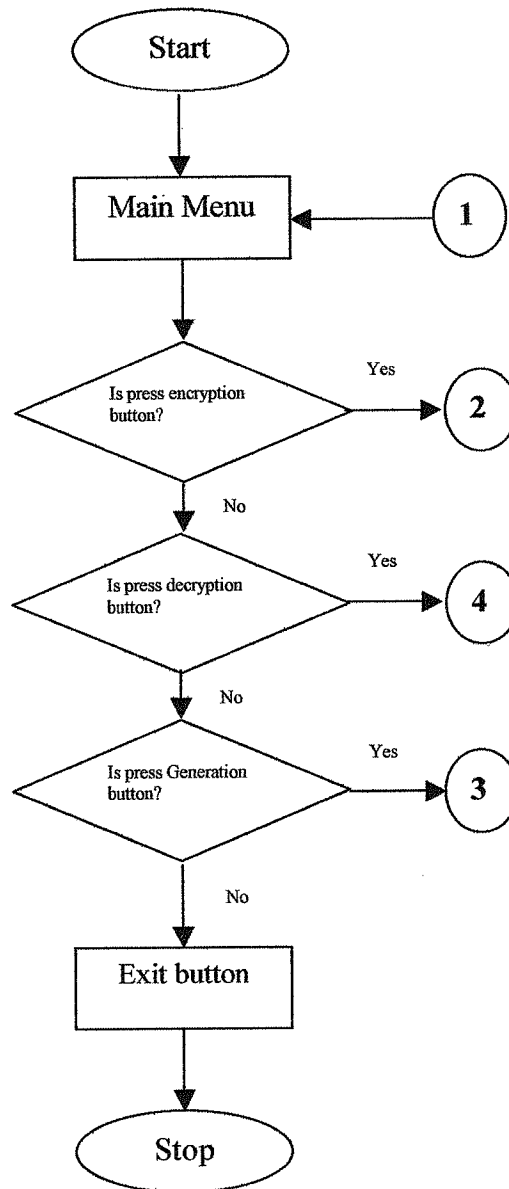


Figure 4.1 Start Maim Menu

In Figure 4.1 we explain that the program when it start, it contain three steps and the exit that are 2, 3 and 4 where step 2 explain in Figure 4.2 to illustrated the encryption operation, step 3 explain in Figure 4.3 to illustrated the Generation public key operation, and step 4 explain in Figure 4.4 to illustrated the decryption operation.

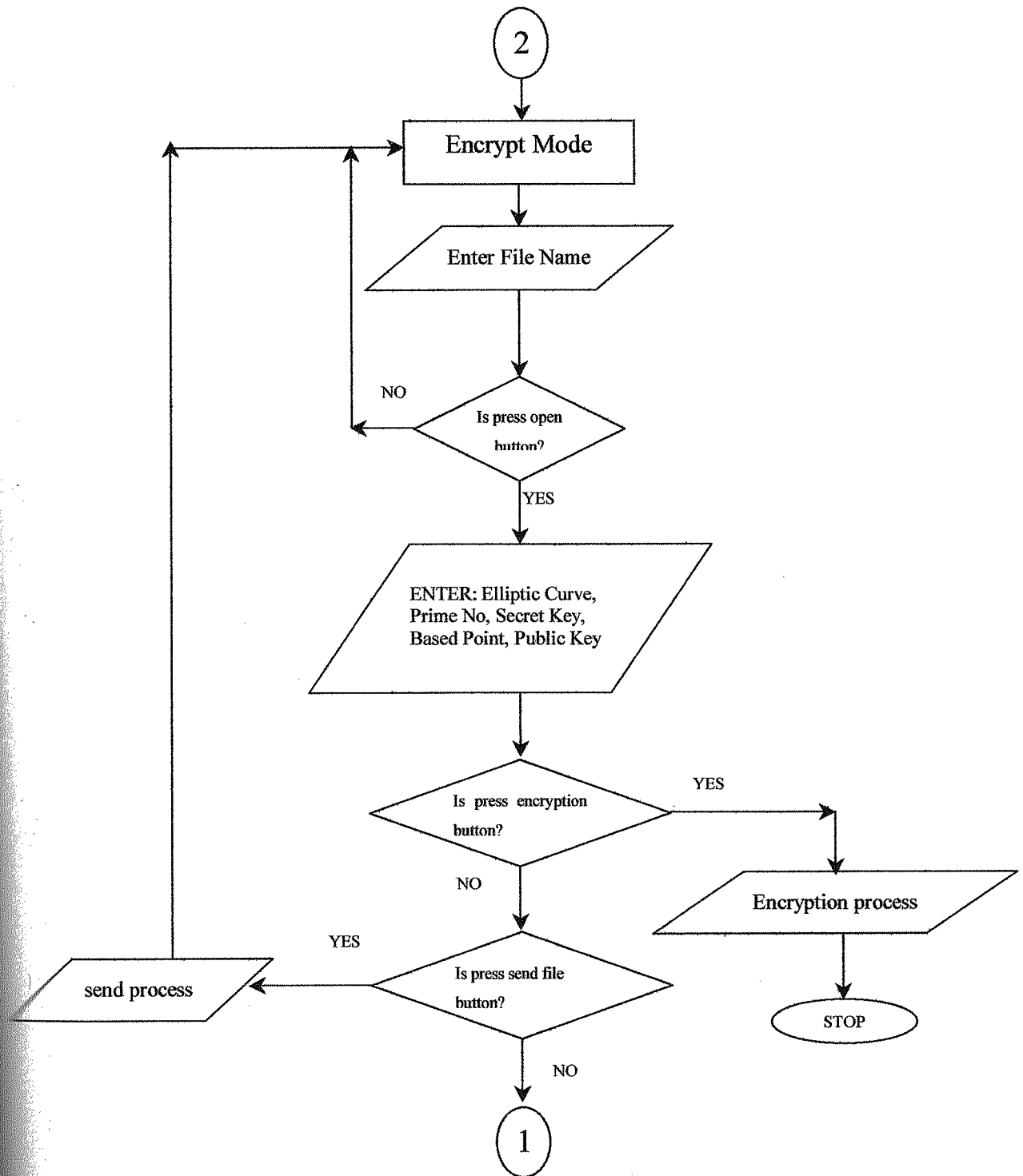


Figure 4.2 Encryption operation

In Encryption operation you work as follow, first you enter the file name that you want to encrypt, if you don't press open you can't do any thing, if press yes the you go to the step, you enter the Elliptic curve Equation, Prime number, Based Point, and enter the Public key, after that you face 3 choices encrypt, send file and exit, to encrypt you must do the last steps, to send file you must encrypt file then send it, and you have exit if you don't like to any thing and you change your mind.

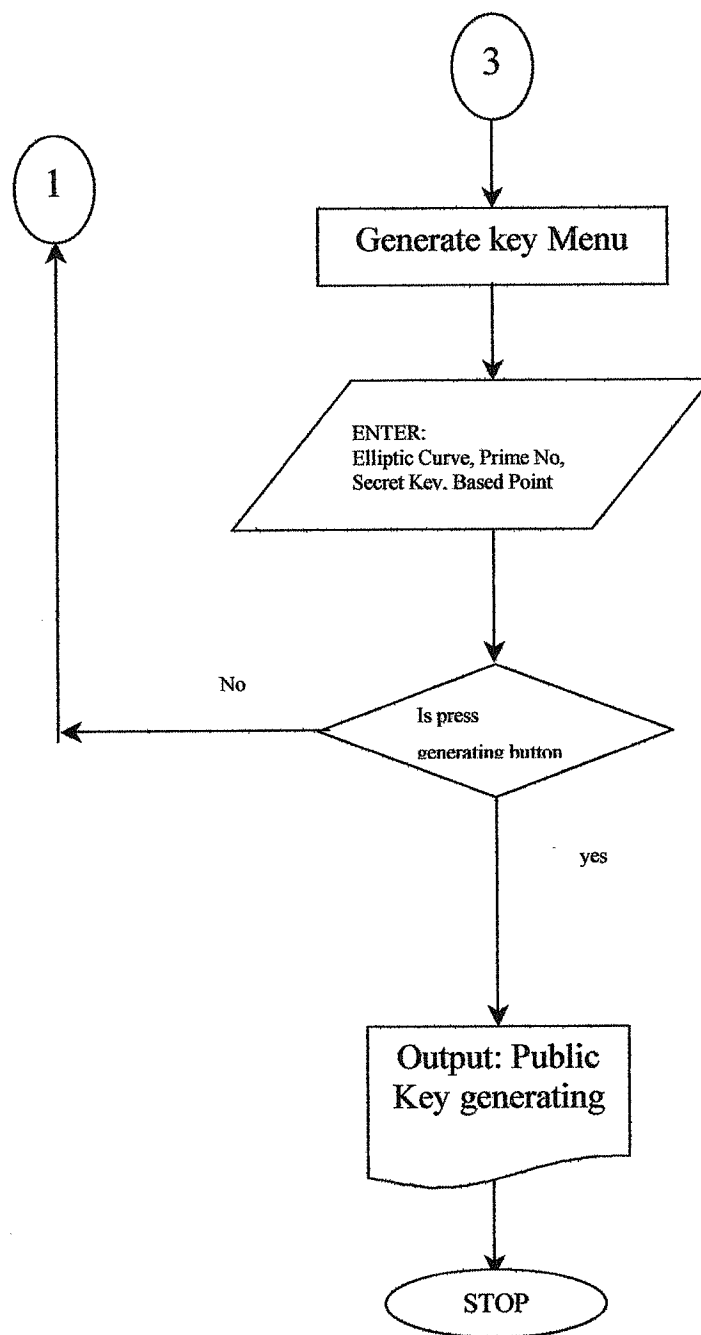


Figure 4.3 Generation Public key

In generation public key you works as follow, you enter the coefficients of elliptic curve equation A and B, the prime number, based point, all these you must first agree on it with other side, and enter your secret key also, then you face 2 choices generate key and exit, after you did the last steps and press the generate key you get a public key as a point, if you change your mind and don't like to any thing press exit.

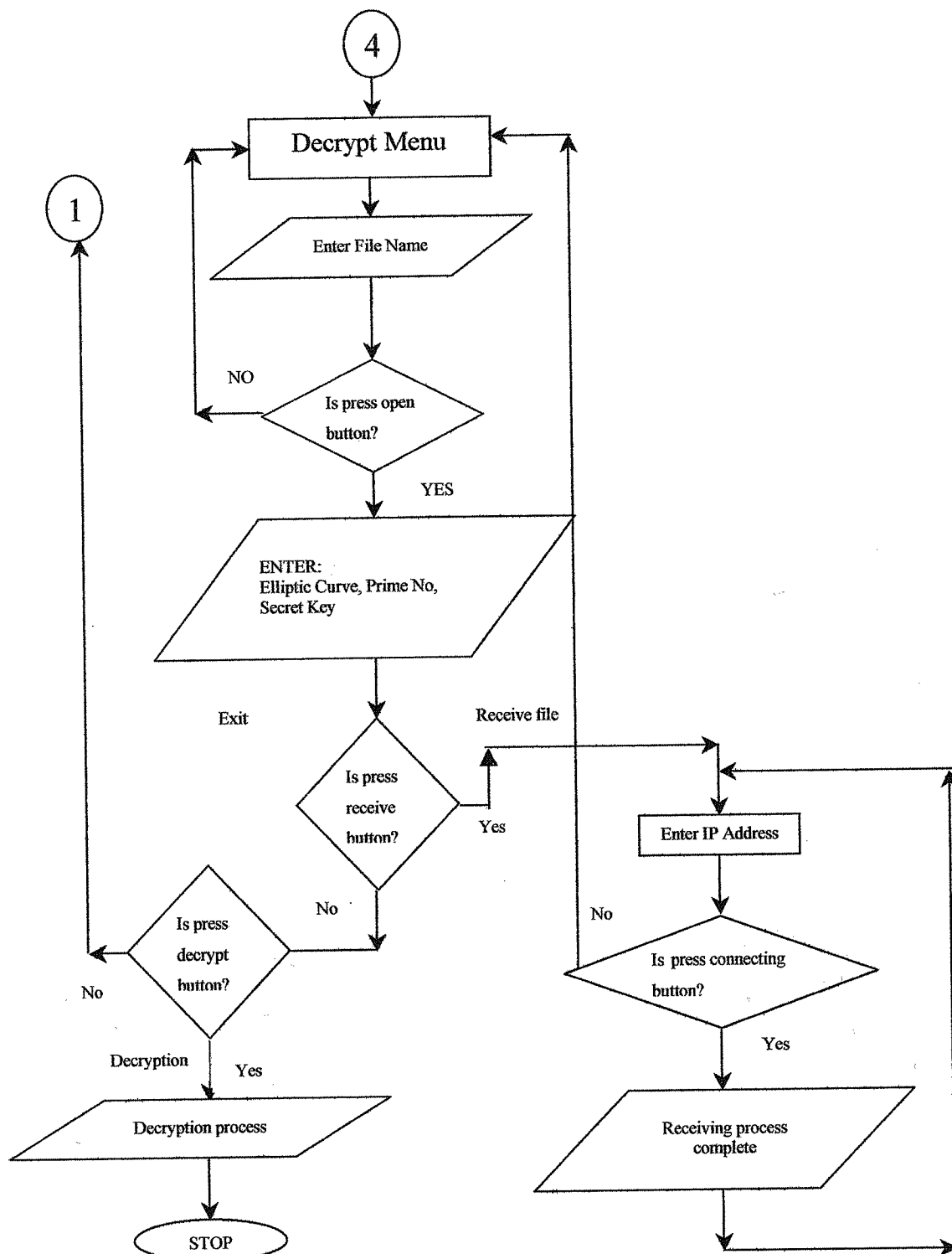


Figure 4.4 Decryption operation

In Decryption operation you work as follow, first you enter the file name that you want to decrypt, if you don't press open you can't do any thing, if press yes the you go to, enter the Elliptic curve Equation, Prime number, and secret key, after that you face 3 choices decrypt, receive file and exit, to decrypt you must do the last steps, to receive file you must enter the IP of the sender, and press connect to do receiving operation, and you have exit if you don't like to any thing and changing your mind.

4.3. Design of the program

The usage of the application is very easy and the interface of the application that is shown in figure 4.5 to figure 4.11, by following direction stated below the user will be able to use efficiently. To complete the encryption, transfer and decryption and the program can be used the following general asks; Generate public key, Encrypt file, Decrypt file, as you see in figure 4.5.

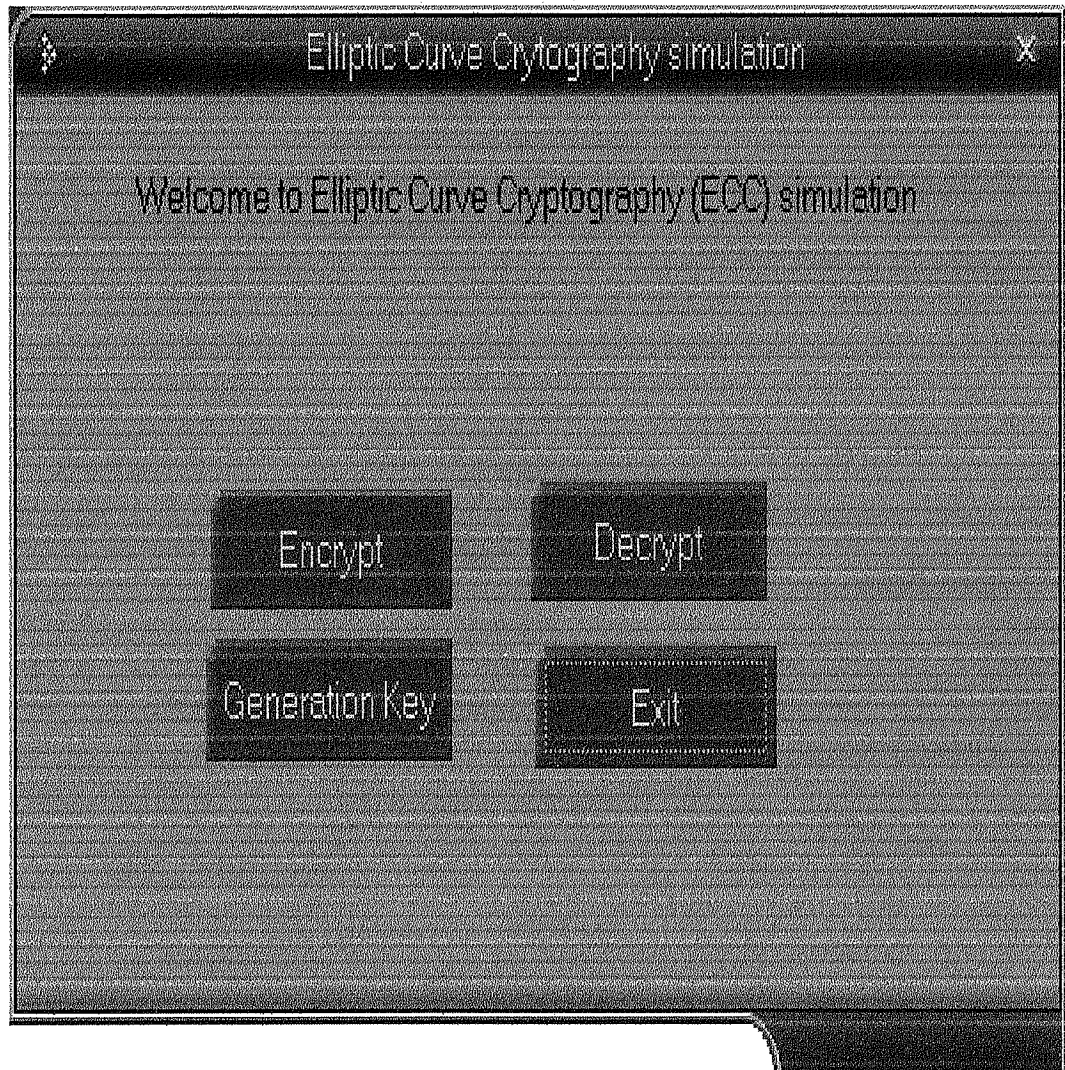


Figure 4.5 Main Menu

4.3.1 Generate public key

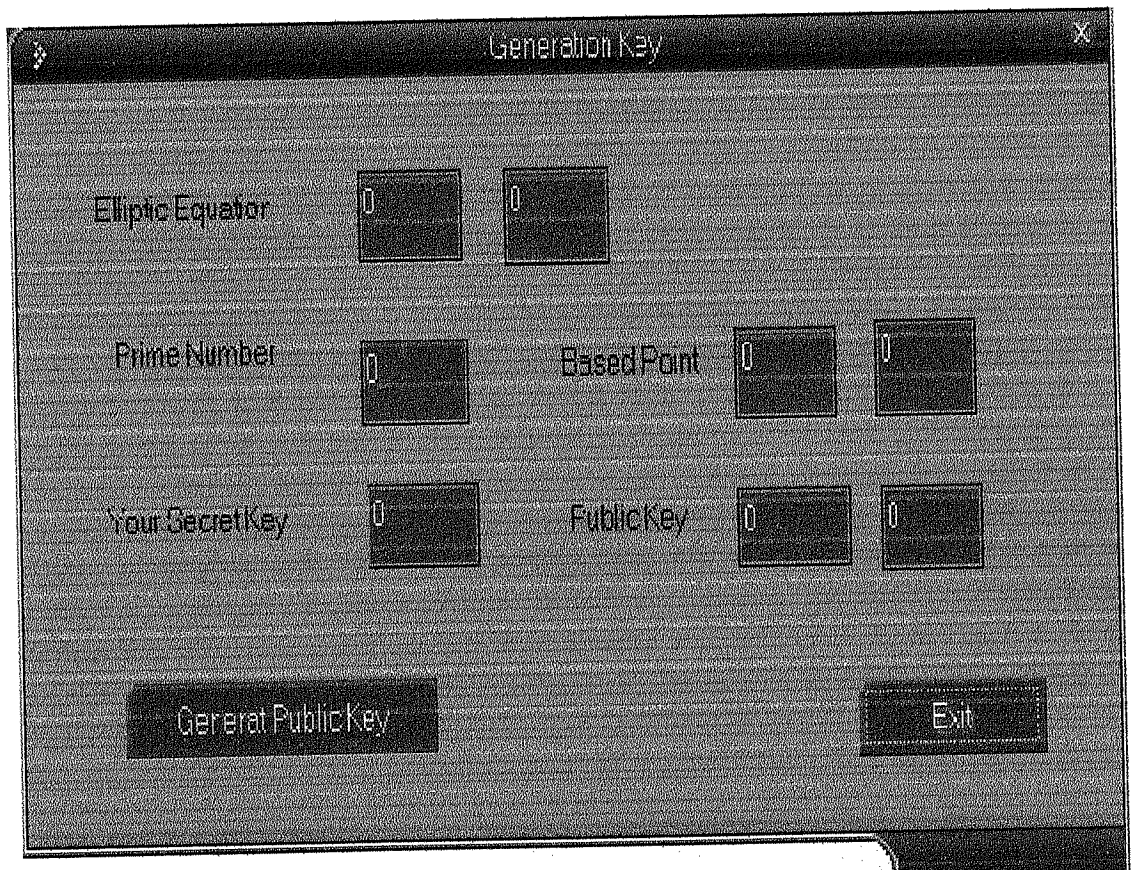


Figure 4.6 Generation Key

In the mode the user generate his public key, to use the other side that will encrypt the file to him. As we know the public key using to encrypt the message and the private key use to decrypt the message.

To generate the public key the user must agree with the other side on the following:

1. Elliptic Curve Equation $y^2 = x^3 + Ax + B$,
2. Prime Number.
3. Based Point it belong to Elliptic curve.

The user enters the A and B from the Equation, the prime number, based point, and his secret key, to get the Public Key as point.

4.3.2 Encrypt file

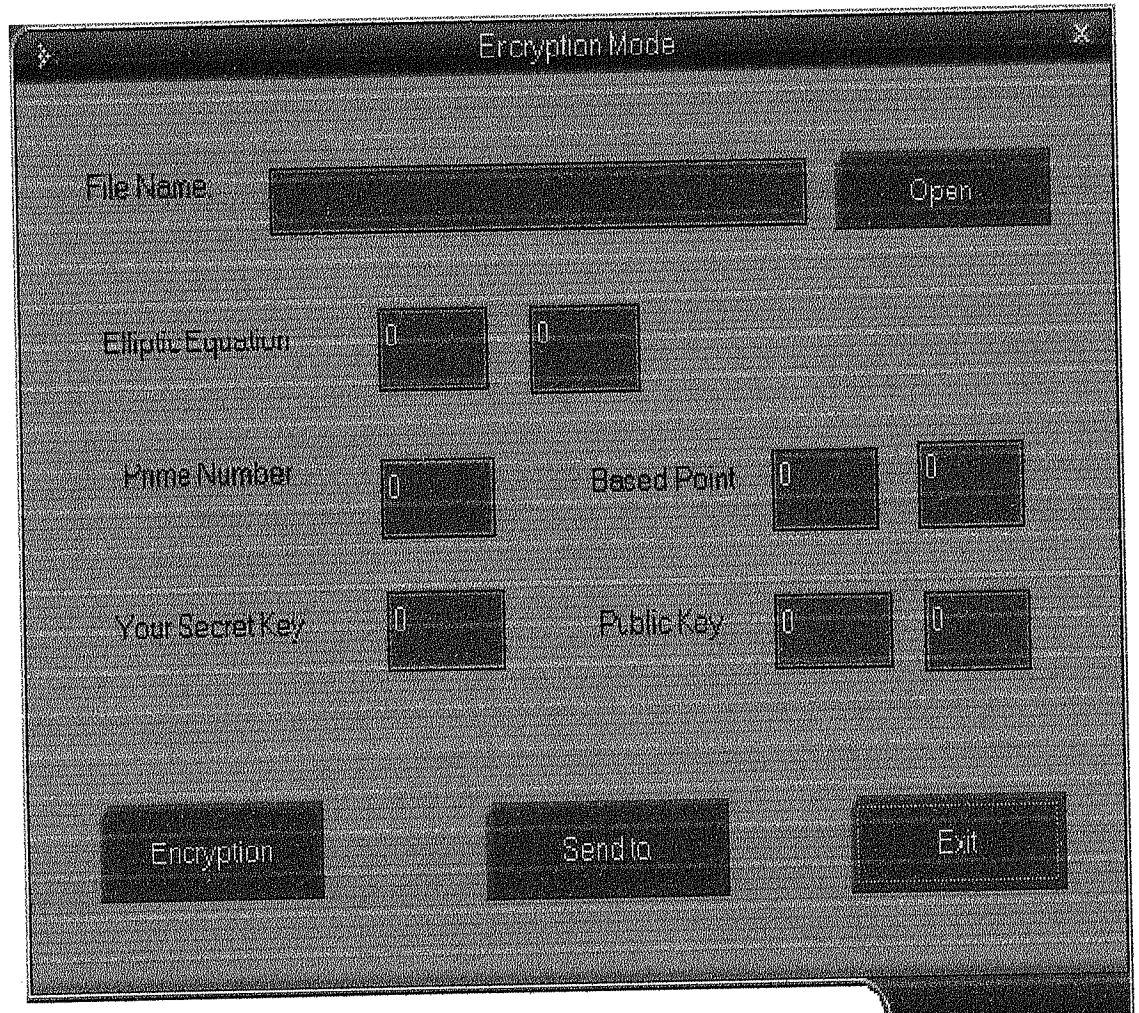


Figure 4.7 Encryption Mode

In this mode the user will encrypt the file, as following: first of all he will select the text file that he want to encrypt as you see in figure 4.8, then he will enter the A and B of elliptic curve equation, the prime number, the based point, the secret key where it is generate automatically, and the public key.

In this mode there is option to send the file over communication channel, so if you are connected to networks, press Send to it will appear the figure 4.9 to select the encrypt file to send, then if the other side receive the file the file the program will give message "Sending file operation successfully", else it will give you message "Sending file operation failed".

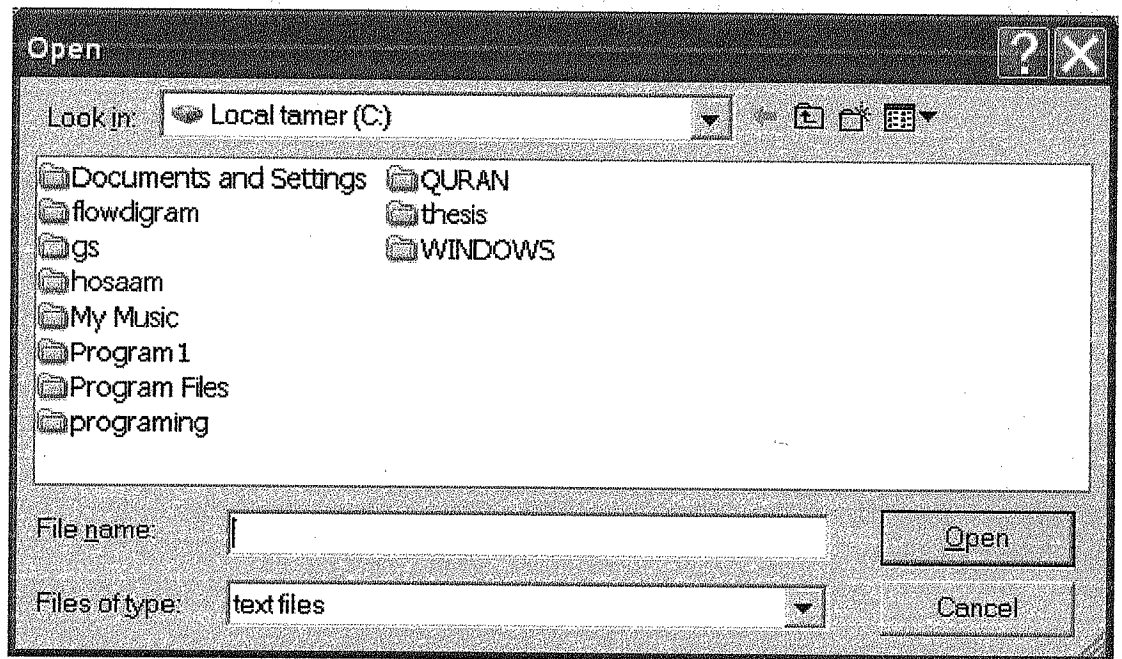


Figure 4.8 Select Text file

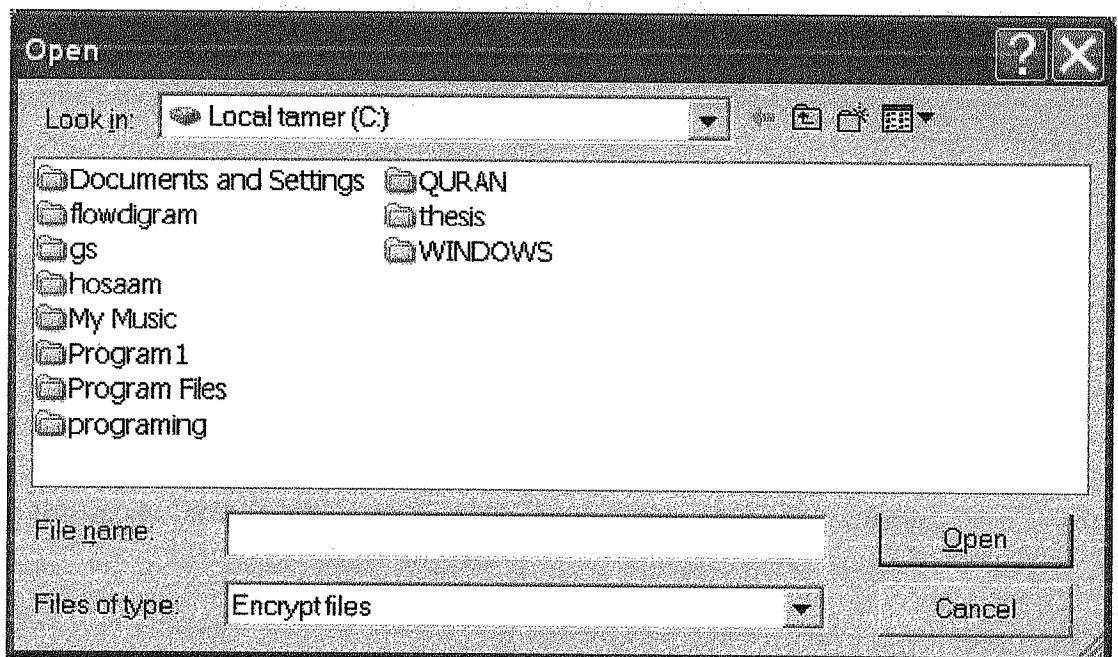


Figure 4.9 Select Encrypt file

4.3.3 Decryption file

The screenshot shows a graphical user interface window titled "Decryption Mode". The window contains the following elements:

- File Name:** A text input field followed by an **Open** button.
- Elliptic Equation:** Two side-by-side input fields.
- Prime Number:** A single input field.
- Your Secret Key:** A single input field.
- Buttons:** Three buttons at the bottom labeled **Decryption**, **Receive from**, and **Exit**.

Figure 4.10 Decryption Mode

In this mode the user will decrypt the file, as following: first of all he will select the Encrypt file as you see in figure 4.9, that he want to decrypt, then he will enter the A and B of elliptic curve equation, the prime number, the based point, and the secret key.

In this mode there is option to receive file over communication channel, so if you are connected to networks, press Receive from it will appear the figure 4.11 to add the address or computer's IP for the other side, then he will press connect to accomplish the receiving process, then if the file is received the program will give message "Receiving file operation successfully", else it will give you message "Receiving file operation failed".

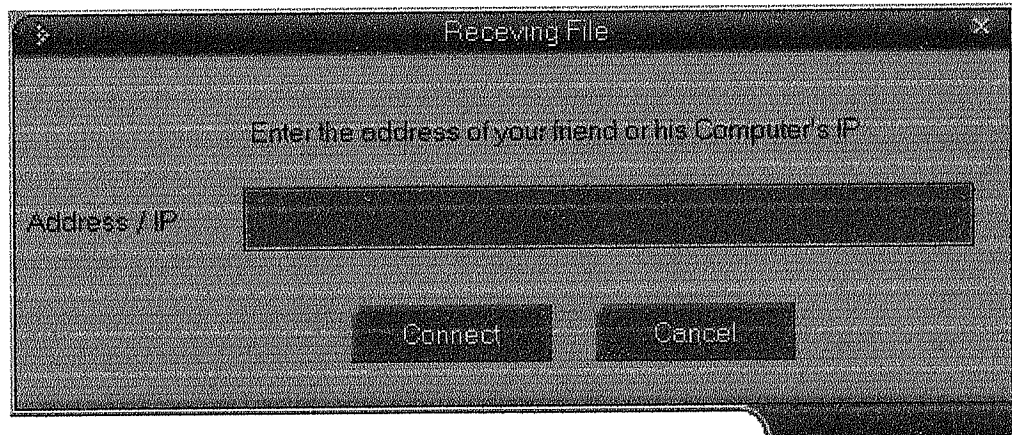


Figure 4.11 Receiving file

4.4. Summary

This chapter has described the how the application works, which is an implementation of the Elliptic Curve Cryptography in public key cryptography concepts using ElGamal ECC algorithm.

CONCLUSION

In this thesis, it has given an analysis the most important facts of Elliptic Curve Cryptography and findings into a board. It also illustrated certain points of this filed. It has given a detailed view on the protocols of Elliptic Curve cryptography (ECC); it also has provided a sample program using ElGamal Elliptic Curve protocol a practical implementation of the pervious theoretical background.

Next paragraph the important result obtained from this thesis:

- Basic of cryptography and where it is used, the basics of the cryptography system and its types, techniques and algorithms used by any cryptography system were discussed in more details, and finally we talked about modern using of cryptography and possible ways to attacks.
- Before cryptographic systems, the corresponding mathematical problems discussed, the difficulty of a problem defined. What did it mean for a mathematical problem to be difficult?
- Elliptic Curve Cryptography (ECC) is the next generation techniques of public key cryptography systems, describe history of ECC, prosperities of ECC using algebra concepts, describe ECDLP and how the security depend on it, compare with RSA, describe the protocols of ECC.
- An application developed to secure data transmission over communication channels using Elliptic Curve Cryptography technology (ECC) implemented using Visual C++ 6.0.

But we have observed that:

- The usage of this technology is grown by time to the interest of the public key cryptography due to the growing of client/server application and needs more secure in this filed.

REFERENCES

- [1] Lenka Fibikova', "Elliptic Curve Cryptography Over Prime Fields", Institute for Experimental Mathematics University of Essen. Essen, Germany, September, 2002.
- [2] Carl Hultquist, "Elliptic Curve Encryption The Immediate Future of Cryptography", National Institute for Research in Computer Science and Control, 8 April 2003
- [3] Certicom Corp., "The Elliptic Curve cryptosystem for smart cards", A Certicom White Paper, The seventh in a series of ECC white papers, Published: May 1998, You can download these papers from Certicom's Web site at <http://www.certicom.com>.
- [4] Certicom Corp., "The Elliptic Curve Cryptosystem" remarks on the security of the elliptic curve cryptosystem, Published: September 1997 Updated: July 2000
- [5] Michael J Ganley, "Elliptic Curve Cryptography- an Introduction", CHI Publishing Ltd, June 2001.
- [6] Montserrat B. Ros, "Hyperelliptic curve cryptosystems over optimal extension fields", Department of Computer Science and Electrical Engineering, University of Queensland, October, 2000.
- [7] Jeffrey L. Vagle, "A Gentle Introduction to Elliptic Curve Cryptography", BBN Technologies, November 21, 2000.
- [8] Richard Southern, "Elliptic Curve Cryptosystems The Future of Public-Key Encryption", sthric003, 20 April 1998.
<http://www.cs.uct.ac.za/courses/CS400W/NIS/papers98/rsouther/ecc.htm>

- [9] Vikram V Kumar, Satish Doraiswamy, Zabeer Jainullabudeen, "Design and Analysis of Algorithms Elliptic Curve Cryptography".
- [10] Jeff Hamblin, "Elliptic Curve Cryptography", May,12,1999.
- [11] Mugino Saeki, "Elliptic Curve Cryptosystems", school of computer science McGill University, Montreal, February 1997.
- [12] Daniel R.L. Brown, "Generic Groups, Collision Resistance and ECDSA", Certicom Research, Canada. Feb 26 2002.
- [13] Jacques Stern, "Evaluation Report on the Discrete Logarithm Problem over finite fields".
- [14] Shuhong Gao, "Cryptosystems Based on Discrete Logarithms", Sun Oct 17 11:04:53 EDT 1999,
http://www.math.clemson.edu/faculty/Gao/crypto_mod/node2.html
- [15] Tim Kerins¹, Emanuel Popovici¹, William Marnane¹, and Patrick Fitzpatrick², "Fully Parameterizable Elliptic Curve Cryptography Processor over GF(2^m)", ¹Dept of Electrical and Electronic Engineering, University College Cork, College Rd., Cork City, Ireland.
ftimk,emanuel,liamg@rennes.ucc.ie, ²Dept of Mathematics, University College Cork, College Rd., Cork City, Ireland. p.fitzpatrick@ucc.ie
- [16] Elisabeth Oswald, "Introduction to Elliptic Curve Cryptography", Institute for Applied Information Processing and Communication A-8010 Inffeldgasse 16a, Graz, Austria Elisabeth.Oswald@iaik.at, July 3,2002.

- [17] Naoya Torii, Kazuhiro Yokoyama, "Elliptic Curve Cryptosystem", FUJITSU Sci. Tech. J.,**36**,2,pp.140-146, Manuscript received June 6, 2000.
- [18] Dr. Reza Akhtar, "Elliptic Curve Cryptography", SUMSRI Algebra Short Course Note.
- [19] Robert Zuccherato, "Elliptic Curve Cryptography Support in Entrust", Entrust Securing Digital Identities & Information, May 9, 2000 Version: 1.0
- [20] Darrel Hankerson (Auburn University) Alfred Menezes (University of Waterloo), "Elliptic Curve Discrete Logarithm Problem", February 25, 2003.
- [21] M. Aydos, E. Savaş, and Ç. K. Koç, "Implementing Network Security Protocols based on Elliptic Curve Cryptography", Electrical & Computer Engineering Oregon State University Corvallis, Oregon 97331, USA.
- [22] A.Murat Fiskiran and Ruby B. Lee, "Workload Characterization of Elliptic Curve Cryptography and other Network Security Algorithms for Constrained Environments", Princeton Architecture Laboratory for Multimedia and Security (PALMS) Department of Electrical Engineering Princeton University {fiskiran,rblee}@ee.Princeton.edu. November 2002.
- [23] M. Aydos, B. Sunar, and Ç. K. Koç, "An Elliptic Curve Cryptography based Authentication and Key Agreement Protocol for Wireless Communication", Electrical & Computer Engineering Oregon State University Corvallis, Oregon 97331.
- [24] Aleksandar Jurisic, Alfred J. Menezes, "Elliptic Curves and Cryptography". Certicom Corp. (Canada), Publishers, 1993

- [25] Neal Koblitz, Alfred Menezes, Scott Vanstone, "The State of Elliptic Curve Cryptography", Designs, Codes and Cryptography, Kluwer Academic, Boston. Manufactured in The Netherlands. Publishers (2000), 173–193
- [26] Chue Peck Har, Luo Yanying, Wong Xiu Ming, "Information Security and Cryptographic Systems", Special Program in Science A Y 2002/2003.
- [27] Wade Trappe, and Lawrence C. Washington. Cryptography Coding Theory. Prentice-Hall, Upper Saddle River, 2002.
- [28] Henna Pietiläinen, "Elliptic curve cryptography on smart cards", Faculty of Information Technology, October 30, 2000.
- [29] Willim Stallings. Cryptography and Network Security principles and practice. Third Edition. Prentice-Hall, Upper Saddle River, 2002.
- [30] Abstract algebra, From Wikipedia, the free encyclopedia.
http://www.wikipedia.org/wiki/Abstract_algebrahttp
- [31] Group (mathematics), From Wikipedia, the free encyclopedia.
[http://www.wikipedia.org/wiki/Group_\(mathematics\)](http://www.wikipedia.org/wiki/Group_(mathematics))
- [32] Abelian group, From Wikipedia, the free encyclopedia.
http://www.wikipedia.org/wiki/Abelian_group
- [33] Abelian group, From Wikipedia, the free encyclopedia.
[http://www.wikipedia.org/wiki/Ring_\(algebra\)](http://www.wikipedia.org/wiki/Ring_(algebra))
- [34] Module, From Wikipedia, the free encyclopedia.
<http://www.wikipedia.org/wiki/Module>
- [35] Modular arithmetic, From Wikipedia, the free encyclopedia.
http://www.wikipedia.org/wiki/Modular_arithmetic

- [36] Field, From Wikipedia, the free encyclopedia.
<http://www.wikipedia.org/wiki/Field>
- [37] Finite field, From Wikipedia, the free encyclopedia.
http://www.wikipedia.org/wiki/Finite_field
- [38] Charles Daney, "Elliptic Curves and Elliptic Functions", March 28, 1996
<http://cgd.best.vwh.net/home/ft/ft01.htm>
- [39] integer factorization, From Wikipedia, the free encyclopedia.
http://www.wikipedia.org/wiki/integer_factorization
- [40] A Certicom Whitepaper, The Elliptic Curve Cryptosystem "Current Public-Key Cryptographic Systems", Published: April 1997 Updated: July 2000
- [41] Mohan Atreya, "Introduction to Cryptography".
- [42] Simson Garfinkel, and Gene Spafford, Practical UNIX and Internet Security Second Edition, April 1996.
- [43] Introduction to Cryptography Mark Vandenwauver Katholieke Universiteit Leuven, Laboratorium ESAT-Groep COSIC,
<http://www.esat.kuleuven.ac.be/cosic/intro/>
- [44] Pierre Loidreau, "Introduction to cryptography",
<http://www.security-labs.org/index.php3>
- [45] Terry Ritter, "Learning About Cryptography" A Basic Introduction to Crypto, Ciphers By Ritter Page, 2003 September 2,
<http://www.ciphersbyritter.com/>

[46] Alan S H Lam, "Overview of Cryptographic Algorithms" Last Updated
Tuesday, 23-Nov-1999 15:05:59 HKT,
<http://itec.erg.cuhk.edu.hk/index.html>

[47] Tatu Ylonen, "Introduction to Cryptography",
<http://burks.brighton.ac.uk/burks/pcinfo/security/criptint.htm>

APPENDICES

```
// ECCDlg.cpp : implementation file
//
```

```
#include "stdafx.h"
#include "ECC.h"
#include "ECCDlg.h"
#include "EncrDlg.h"
#include "DecrDlg.h"
#include "GkDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CAboutDlg dialog used for App About
```

```
class CAboutDlg : public CDialog
{
public:
    CAboutDlg();
```

```
// Dialog Data
//{{AFX_DATA(CAboutDlg)
enum { IDD = IDD_ABOUTBOX };
//}}AFX_DATA
```

```
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CAboutDlg)
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL
```

```
// Implementation
protected:
//{{AFX_MSG(CAboutDlg)
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};
```

```
CAboutDlg::CAboutDlg() : CDialog(CAboutDlg::IDD)
{
   //{{AFX_DATA_INIT(CAboutDlg)
    }}AFX_DATA_INIT
}
```

```

void CAboutDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CAboutDlg)
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CAboutDlg, CDialog)
   //{{AFX_MSG_MAP(CAboutDlg)
    // No message handlers
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CECCDlg dialog

CECCDlg::CECCDlg(CWnd* pParent /*=NULL*/)
: CDialog(CECCDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CECCDlg)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
    // Note that LoadIcon does not require a subsequent DestroyIcon in Win32
    m_hIcon = AfxGetApp()->LoadIcon(IDR_MAINFRAME);
}

void CECCDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CECCDlg)
    // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CECCDlg, CDialog)
   //{{AFX_MSG_MAP(CECCDlg)
    ON_WM_SYSCOMMAND()
    ON_WM_PAINT()
    ON_WM_QUERYDRAGICON()
    ON_BN_CLICKED(IDC_ENC, OnEnc)
    ON_BN_CLICKED(IDC_DEC, OnDec)
    ON_BN_CLICKED(IDC_GK, OnGk)
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CECCDlg message handlers

```



```

BOOL CECCDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // Add "About..." menu item to system menu.

    // IDM_ABOUTBOX must be in the system command range.
    ASSERT((IDM_ABOUTBOX & 0xFFF0) == IDM_ABOUTBOX);
    ASSERT(IDM_ABOUTBOX < 0xF000);

    CMenu* pSysMenu = GetSystemMenu(FALSE);
    if (pSysMenu != NULL)
    {
        CString strAboutMenu;
        strAboutMenu.LoadString(IDS_ABOUTBOX);
        if (!strAboutMenu.IsEmpty())
        {
            pSysMenu->AppendMenu(MF_SEPARATOR);
            pSysMenu->AppendMenu(MF_STRING, IDM_ABOUTBOX,
strAboutMenu);
        }

        // Set the icon for this dialog. The framework does this automatically
        // when the application's main window is not a dialog
        SetIcon(m_hIcon, TRUE);           // Set big icon
        SetIcon(m_hIcon, FALSE);        // Set small icon

        // TODO: Add extra initialization here

        CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN1)-
>GetControlUnknown();
        pSkin->ApplySkin((long)m_hWnd);

        return TRUE; // return TRUE unless you set the focus to a control
    }

    void CECCDlg::OnSysCommand(UINT nID, LPARAM lParam)
    {
        if ((nID & 0xFFF0) == IDM_ABOUTBOX)
        {
            CAboutDlg dlgAbout;
            dlgAbout.DoModal();
        }
        else
        {

```

```

        CDialog::OnSysCommand(nID, lParam);
    }
}

// If you add a minimize button to your dialog, you will need the code below
// to draw the icon. For MFC applications using the document/view model,
// this is automatically done for you by the framework.

void CECCDlg::OnPaint()
{
    if (IsIconic())
    {
        CPaintDC dc(this); // device context for painting

        SendMessage(WM_ICONERASEBKGND, (WPARAM)
            dc.GetSafeHdc(), 0);

        // Center icon in client rectangle
        int cxIcon = GetSystemMetrics(SM_CXICON);
        int cyIcon = GetSystemMetrics(SM_CYICON);
        CRect rect;
        GetClientRect(&rect);
        int x = (rect.Width() - cxIcon + 1) / 2;
        int y = (rect.Height() - cyIcon + 1) / 2;

        // Draw the icon
        dc.DrawIcon(x, y, m_hIcon);
    }
    else
    {
        {
            CDialog::OnPaint();
        }
    }
}

// The system calls this to obtain the cursor to display while the user drags
// the minimized window.
HCURSOR CECCDlg::OnQueryDragIcon()
{
    return (HCURSOR) m_hIcon;
}

void CECCDlg::OnEnc()
{
    // TODO: Add your control notification handler code here
    CEncrDlg dlg;
    dlg.DoModal();
}

void CECCDlg::OnDec()

```

```

{
    // TODO: Add your control notification handler code here
    CDecrDlg dlg;
    dlg.DoModal();

```

```

}

```

```

void CECCDlg::OnGk()

```

```

{
    // TODO: Add your control notification handler code here
    CGkDlg dlg;
    dlg.DoModal();

```

```

}

```

```

/*****

```

```

// EncrDlg.cpp : implementation file

```

```

//

```

```

#include "stdafx.h"
#include "ECC.h"
#include "EncrDlg.h"
#include "GkDlg.h"

```

```

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>

```

```

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

```

```

////////////////////////////////////
// CEncrDlg dialog

```

```

CEncrDlg::CEncrDlg(CWnd* pParent /*=NULL*/)
: CDialog(CEncrDlg::IDD, pParent)

```

```

{
   //{{AFX_DATA_INIT(CEncrDlg)
    m_eeA = 0;
    m_eeB = 0;
    m_prime = 0;

```

```

        m_BPX = 0;
        m_BPY = 0;
        m_SK = 0;
        m_PKX = 0;
        m_PKY = 0;
        m_enc_file = _T("");
        //}}AFX_DATA_INIT
    }

void CEncrDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CEncrDlg)
    DDX_Text(pDX, IDC_EDIT2, m_eeA);
    DDX_Text(pDX, IDC_EDIT3, m_eeB);
    DDX_Text(pDX, IDC_EDIT4, m_prime);
    DDX_Text(pDX, IDC_EDIT6, m_BPX);
    DDX_Text(pDX, IDC_EDIT7, m_BPY);
    DDX_Text(pDX, IDC_EDIT8, m_SK);
    DDX_Text(pDX, IDC_EDIT10, m_PKX);
    DDX_Text(pDX, IDC_EDIT11, m_PKY);
    DDX_Text(pDX, IDC_EDIT1, m_enc_file);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CEncrDlg, CDialog)
    //{{AFX_MSG_MAP(CEncrDlg)
    ON_BN_CLICKED(IDC_Encrypt, OnEncrypt)
    ON_BN_CLICKED(IDC_Send1, OnSend1)
    ON_BN_CLICKED(IDC_Brows, OnBrows)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CEncrDlg message handlers

void CEncrDlg::OnEncrypt()
{
    //***** Encryption Mode
    //*****
    //Here we are making the encryption operation

    CGkDlg D;

    FILE *input, *output;

```

```

char ch, ch_out;
int prime, A;
Assign assign[256], ass;
Point alpha={0,0};
Point beta={0,0};

UpdateData(TRUE);

alpha.x = m_BPX;
alpha.y = m_BPY;

beta.x = m_PKX;
beta.y = m_PKY;

prime = m_prime;
A = m_eeA;

// here k is secret key for Alice uses in encryption operation
int k=6;

/*srand(100);
do
{
    k = rand();
}while (k>2*D.E|| k<0);*/

m_SK = k; UpdateData(FALSE);

//Here clear the buff where we put PlainText
for (int i=0; i<256; i++)
{
    assign[i].PT.x=0;
    assign[i].PT.y=0;
}

// Here we open file to read from
if ((input = fopen(m_enc_file, "r")) == NULL)
{
    MessageBox("Cannot open input file");
    exit(1);
}
fseek(input, SEEK_SET, 0);

CString Save_file;

CFileDialog m_IdFile (FALSE, "*.ENC", NULL,
OFN_HIDEREADONLY, "Encrypt files|*.ENC");

```



```

m_IdFile.m_ofn.lpstrInitialDir = "c:\\";

if (m_IdFile.DoModal() != IDCANCEL )
{
    Save_file = m_IdFile.GetPathName();
    UpdateData(FALSE);
}

// Here we open file to write into
if ((output = fopen(Save_file, "w+")) == NULL)
{
    MessageBox("Cannot open output file");
    exit(1);
}

char my, mz;
Encrypt enc;

MessageBox("You now enter thye loop in file");
do
{
    /* read a char from the file */
    ch = fgetc(input);
    ch = CAP (ch); //convetr char to Capetail

    putch(ch);

    ass.ch = ch;
    ass.PT = map(ass.ch); //function converts char to point

    enc = encrypt (ass.PT, alpha, beta, k, prime, A);
    ch_out = cmap(enc.x);

    fputc(ch_out, output);
    fputc(enc.y, output);
    fputc(enc.z, output);

} while (ch != EOF);

fclose(input);
fclose(output);

MessageBox("Your Encryption opertion Successfully");

int x = MessageBox("Do You Want To Send The File", "Send
File", MB_YESNO);

```

```

        if (x == IDYES)
        {
            MessageBox("Press Send");
            GetDlgItem ( IDC_Send1 ) -> EnableWindow ( TRUE );
        }

        if (x == IDNO)
            MessageBox("You are offline");
    }

```

```

char CEncrDlg::CAP(char a)
{
    char A;
    int aa;

    aa = a;

    if (aa >= 97 && a <= 122)
    {
        aa = aa - 32;
        A = aa;
    }
    else
        A = a;
    return A;
}

```

```

Point CEncrDlg::map(char c)
{
    char ch;
    int ascii;
    char str[2];
    float num;
    double fraction, integer;
    Point p={0,0};

    ch = c;
    ascii = ch;

    num = ascii;

    num = num/10;

    fraction = modf(num, &integer);

```

```

    p.x = integer;
    p.y = int(fraction*10+0.1);
    return p;
}

Encrypt CEncrDlg::encrypt(Point X, Point alpha, Point beta, int k, int prime, int A)
{
    Point y0={0,0}, C1_C2={0,0};
    int y1=0,y2=0;
    Encrypt enc;

    CGkDlg d;

    y0 = d.Mult(alpha,A,prime,k);
    C1_C2 = d.Mult(beta,A,prime,k);

    y1 = (C1_C2.x * X.x); y1 = fmod(y1,prime);
    y2 = (C1_C2.y * X.y); y2 = fmod(y2,prime);

    enc.x=y0;
    enc.y=y1;
    enc.z=y2;

    return enc;
}

char CEncrDlg::emap(Point p)
{
    int x;
    char c;

    x = (p.x*10) + p.y;
    c = x;

    return c;
}

void CEncrDlg::OnSend1()
{
    // TODO: Add your control notification handler code here

    SendFile();

    //GetDlgItem ( IDC_Send1 ) -> EnableWindow ( FALSE );
}

void CEncrDlg::SendFile()

```

```

{

#define PORT 34000 /// Select any free port you wish

AfxSocketInit(NULL);

CSocket sockSrvr;
CSocket sockRecv;

sockSrvr.Create(PORT); // Creates our server socket

sockSrvr.Listen(); // Start listening for the client at PORT

if (sockSrvr.Accept(sockRecv)) // Use another CSocket to accept the connection
    MessageBox("you'r online now ...");
else
    MessageBox("you'r offline now ...");

// Select the file to send
CFileDialog m_IdFile (TRUE, NULL, NULL,
OFN_HIDEREADONLY, "Encrypt files|*.ENC");
m_IdFile.m_ofn.lpstrInitialDir = "c:\\";

if (m_IdFile.DoModal() != IDCANCEL )
{
    m_enc_file=      m_IdFile.GetPathName();
    UpdateData(FALSE);
}

CFile myFile;
myFile.Open(m_enc_file, CFile::modeRead | CFile::typeBinary);

int myFileLength = myFile.GetLength(); // Going to send the correct File Size

sockRecv.Send(&myFileLength, 4); // 4 bytes long

byte* data = new byte[myFileLength];

myFile.Read(data, myFileLength);

sockRecv.Send(data, myFileLength); //Send the whole thing now

myFile.Close();
delete data;

sockRecv.Close();

```

```

        MessageBox("Sending file operation successfully");
    }

void CEncrDlg::OnBrows()
{
    // TODO: Add your control notification handler code here

    CFileDialog m_IdFile (TRUE,NULL, NULL, OFN_HIDEREADONLY,"text
files|*.txt");
    m_IdFile.m_ofn.lpstrInitialDir = "c:\\";

    if (m_IdFile.DoModal() != IDCANCEL )
    {
        m_enc_file= m_IdFile.GetPathName();
        UpdateData(FALSE);
    }

}

BOOL CEncrDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN1)-
>GetControlUnknown();
    pSkin->ApplySkin((long)m_hWnd);

    //GetDlgItem ( IDC_Send1 ) -> EnableWindow ( FALSE );

    return TRUE; // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}

void CEncrDlg::OnOK()
{
    // TODO: Add extra validation here

    CDialog::OnOK();
}

/*****

// GkDlg.cpp : implementation file
//

```



```
#include "stdafx.h"
#include "ECC.h"
#include "GkDlg.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include "ECCDlg.h"
```

```
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
```

```
////////////////////////////////////
// CGkDlg dialog
```

```

/*****
CGkDlg::CGkDlg(CWnd* pParent /*=NULL*/)
: CDialog(CGkDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CGkDlg)
    m_eeA = 0;
    m_eeB = 0;
    m_prime = 0;
    m_BPX = 0;
    m_BPY = 0;
    m_SK = 0;
    m_PKX = 0;
    m_PKY = 0;
    m_E = 0;
    //}}AFX_DATA_INIT
}

```

```
void CGkDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{{AFX_DATA_MAP(CGkDlg)
    DDX_Text(pDX, IDC_EDIT2, m_eeA);
    DDX_Text(pDX, IDC_EDIT3, m_eeB);
    DDX_Text(pDX, IDC_EDIT4, m_prime);
    DDX_Text(pDX, IDC_EDIT6, m_BPX);
    DDX_Text(pDX, IDC_EDIT7, m_BPY);
    DDX_Text(pDX, IDC_EDIT8, m_SK);

```

```

        DDX_Text(pDX, IDC_EDIT10, m_PKX);
        DDX_Text(pDX, IDC_EDIT11, m_PKY);
        DDX_Text(pDX, IDC_EDIT9, m_E);
        //}}AFX_DATA_MAP
    }

BEGIN_MESSAGE_MAP(CGkDlg, CDialog)
   //{{AFX_MSG_MAP(CGkDlg)
        ON_BN_CLICKED(IDC_GenKey, OnGenKey)

   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CGkDlg message handlers

void CGkDlg::OnGenKey()
{
    // TODO: Add your control notification handler code here

    UpdateData(TRUE);

    int prime, A, B, y1, y2, tst;
    int Nset; // No of QR elements
    int QR[5000], Ysqr[5000];
    Point point[8000];
    Point alpha={0,0}; //PT={9,1}; // x and y must be less than prime
    Point beta={0,0}; //dec={0,0};

    E=0;

    prime = m_prime;
    A = m_eeA ;
    B = m_eeB ;

    //to test if A and B are multiple of prime or not?
    tst = tst_AB(prime,A,B);
    if (tst == 0)
    {
        MessageBox("your have wrong equation TRY AGAIN");
        EndDialog(0);
    }
    else
        MessageBox("your have right equation");

    /*****

```

```

        Nset = (prime-1)/2;

        for (int i=1; i<=Nset; i++)
        {
            QR[i] = (i*i)%prime;
        }
//***** New Step *****

        int j=0; // index of point

        for (int x=0; x<prime; x++)
        {

            Ysqr[x] = ( (x*x*x)+(A*x)+B ) % prime;

            for (i=1; i<=Nset; i++)
            {
                if (Ysqr[x] == QR[i])
                {
                    E++;
                    y1 = i;

                    y2 = prime - i;

                    point[j].x=x; point[j].y=y1;
                    j++;
                    point[j].x=x; point[j].y=y2;
                    j++;
                }
            }

        }

        m_E = 2*E;
        UpdateData(FALSE);
//***** New Step *****
//test the alpha if it belongs to elliptic curve or not

        int chk = 0;
        alpha.x = m_BPX ;
        alpha.y = m_BPY;

        for (i=0; i<(2*E); i++)
        {
            if (point[i].x == alpha.x && point[i].y == alpha.y)
                chk = 1;
        }

```

```

        //MessageBox("YES alpha is belong to Elliptic curve");
    }

```

```

    if (chk == 0)
    {
        MessageBox("No alpha is NOT belong to Elliptic curve");

        EndDialog(0);
    }

```

```

//***** New Step *****

```

```

//Generation key Mode.
//Here I compute beta is public key
//Here Bob make generation public key and give it to Alice

    // here a is secret key for Bob and the [a*(alpha)] make the public key
    int a=0;
    a = m_SK;

    if (a>2*E+1 || a<2)
    {
        MessageBox("Choose Bobs secret key [a], it must  $2 \leq a < \text{Number of}$ 
points");
        MessageBox("Try again ... BYE BYE");

        EndDialog(0);
    }

```

```

    beta = Mult(alpha, A, prime, a);

```

```

    m_PKX = beta.x; UpdateData(FALSE);
    m_PKY = beta.y; UpdateData(FALSE);

```

```

}

```

```

int CGkDlg::tst_AB(int P, int A, int B)
{

```

```

    int equation;

```

```

    equation = (4*A*A*A) + (27*B*B) % P;

```

```

    if (equation != 0)
    {

```

```

        return 1;
    }
    else
    {
        return 0;
    }
}

Point CGkDlg::Mult(Point alpha, int A, int prime, int a)
{
    int x3=0, y3=0;
    float lamda=0.0,L1=0.0,L2=0.0;
    Point Q={0,0};

    L1 = (3*alpha.x*alpha.x+A); L1 = fmod(L1,prime);
    if (L1<0)
        L1=prime+L1;

    //cout<<"\nL1 is: "<<L1;

    L2 = (2*alpha.y); L2 = fmod(L2,prime);
    if (L2<0)
        L2=prime+L2;

    //cout<<"\nL2 is: "<<L2;

    lamda = L1 * inverse(prime,L2); ;
    lamda = fmod(lamda,prime);
    if (lamda<0)
        lamda=prime+lamda;

    //cout<<"\nthe lamda is: "<<lamda<<"\n";

    x3 = (lamda*lamda) - 2*alpha.x; x3 = fmod(x3,prime);
    if (x3<0)
        x3=prime+x3;

    //cout<<"X3= "<<x3<<endl;

    y3 = lamda * (alpha.x - x3) - alpha.y; y3 = fmod(y3,prime);
    if (y3<0)
        y3=prime+y3;

    //cout<<"Y3= "<<y3<<endl;

    Q.x=x3;
    Q.y=y3;
}

```



```

if (a>2)
for (int i=1; i<(a-1); i++)
{
    L1=0;L2=0;x3=0;y3=0;

    //cout<<"Now P("<<alpha.x<<","<<alpha.y<<") ";
    //cout<<"Q("<<Q.x<<","<<Q.y<<") ";

    L1 = (Q.y-alpha.y); L1 = fmod(L1,prime);
    if (L1<0)
        L1=prime+L1;

    //cout<<"\nL1 is: "<<L1;

    L2 = (Q.x-alpha.x); L2 = fmod(L2,prime);
    if (L2<0)
        L2=prime+L2;

    //cout<<"\nL2 is: "<<L2;

    lamda = L1 * inverse(prime,L2);
    lamda = fmod(lamda,prime);
    if (lamda<0)
        lamda=prime+lamda;

    //cout<<"\nthe lamda is: "<<lamda<<"\n";

    x3 = (lamda*lamda) - alpha.x - Q.x; x3 = fmod(x3,prime);
    if (x3<0)
        x3=prime+x3;

    //cout<<"X3= "<<x3<<endl;

    y3 = lamda * (alpha.x - x3) - alpha.y; y3 = fmod(y3,prime);
    if (y3<0)
        y3=prime+y3;

    //cout<<"Y3= "<<y3<<endl;

    //getch();

    Q.x=x3;
    Q.y=y3;

}

return Q;
}

```

```

int CGkDlg::inverse(int prime, int B3)
{
    int A1=1,A2=0,A3=5;
    int B1=0,B2=1;
    int T1=0,T2=0,T3=0;
    float Q=0;

    A3= prime;
    do
    {
        if (B3 == 0)
        {
            //cout<<" No inverse so A3: "<<A3<<endl;
            return A3;
        }

        if (B3 == 1)
        {
            //cout<<" The inverse in it self: "<<B2<<endl;
            return B3;
        }

        Q = A3/B3;
        Q = int(Q);

        T1= A1-(Q*B1);
        T2= A2-(Q*B2);
        T3= A3-(Q*B3);

        A1 = B1;
        A2 = B2;
        A3 = B3;

        B1 = T1;
        B2 = T2;
        B3 = T3;

    }while (B3 != 1);

    if (B2<0)
    {
        B2 = prime+B2;
    }

    //cout<<" The inverse is: "<<B2<<endl;

    return B2;
}

```

```

}

BOOL CGkDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN1)-
>GetControlUnknown();
    pSkin->ApplySkin((long)m_hWnd);

    return TRUE; // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}
/*****
// ECC.cpp : Defines the class behaviors for the application.
//

#include "stdafx.h"
#include "ECC.h"
#include "ECCDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CECCApp

BEGIN_MESSAGE_MAP(CECCApp, CWinApp)
   //{{AFX_MSG_MAP(CECCApp)
        // NOTE - the ClassWizard will add and remove mapping macros here.
        // DO NOT EDIT what you see in these blocks of generated code!
    }}AFX_MSG
    ON_COMMAND(ID_HELP, CWinApp::OnHelp)
END_MESSAGE_MAP()

////////////////////////////////////
// CECCApp construction

CECCApp::CECCApp()
{
    // TODO: add construction code here,
    // Place all significant initialization in InitInstance
}

```

```

////////////////////////////////////
// The one and only CECCApp object

CECCApp theApp;

////////////////////////////////////
// CECCApp initialization

BOOL CECCApp::InitInstance()
{
    if (!AfxSocketInit())
    {
        AfxMessageBox(IDP_SOCKETS_INIT_FAILED);
        return FALSE;
    }

    AfxEnableControlContainer();

    // Standard initialization
    // If you are not using these features and wish to reduce the size
    // of your final executable, you should remove from the following
    // the specific initialization routines you do not need.

#ifdef _AFXDLL
    Enable3dControls();           // Call this when using MFC in a shared
DLL
#else
    Enable3dControlsStatic();     // Call this when linking to MFC statically
#endif

    CECCDlg dlg;
    m_pMainWnd = &dlg;
    int nResponse = dlg.DoModal();
    if (nResponse == IDOK)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with OK
    }
    else if (nResponse == IDCANCEL)
    {
        // TODO: Place code here to handle when the dialog is
        // dismissed with Cancel
    }

    // Since the dialog has been closed, return FALSE so that we exit the
    // application, rather than start the application's message pump.
    return FALSE;
}
/*****

```

```

// Recive.cpp : implementation file
//

#include "stdafx.h"
#include "ECC.h"
#include "Recive.h"
#include "ECCDlg.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CRecive dialog

CRecive::CRecive(CWnd* pParent /*=NULL*/)
: CDialog(CRecive::IDD, pParent)
{
   //{{AFX_DATA_INIT(CRecive)
    m_ip = _T("");
    m_dec_file = _T("");
   //}}AFX_DATA_INIT
}

void CRecive::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    {{{AFX_DATA_MAP(CRecive)
    DDX_Text(pDX, IDC_EDIT1, m_ip);
    DDX_Text(pDX, IDC_EDIT2, m_dec_file);
    }}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CRecive, CDialog)
    {{{AFX_MSG_MAP(CRecive)
    ON_BN_CLICKED(IDC_RConnect, OnRConnect)
    ON_BN_CLICKED(IDC_BUTTON1, OnSave)
    }}}AFX_MSG_MAP
END_MESSAGE_MAP()

//////////////////////////////////////
// CRecive message handlers

```



```

void CRecive::OnRConnect()
{
    // TODO: Add your control notification handler code here

    ReciveFile();
}

void CRecive::ReciveFile()
{
    int chk=0;

    CString x;

    #define PORT 34000 /// Select any free port you wish

    AfxSocketInit(NULL);

    CSocket sockClient;
    sockClient.Create();

    // "127.0.0.1" is the IP to your server, same port
    UpdateData(TRUE);
    chk = sockClient.Connect(m_ip, PORT);

    if(chk == 0)
        MessageBox("You connection faild");
    else
        MessageBox("Your connection successfully, you now reciving file");

    int dataLength;
    sockClient.Receive(&dataLength, 4); //Now we get the File Size first

    byte* data = new byte[dataLength];
    sockClient.Receive(data, dataLength); //Get the whole thing

    OnSave();

    CFile destFile(m_dec_file, CFile::modeCreate | CFile::modeWrite |
CFile::typeBinary);

    destFile.Write(data, dataLength); // Write it

    destFile.Close();

    delete data;
}

```

```

        sockClient.Close();

        MessageBox("Receiving Operation Successfully");
    }

    BOOL CReceive::OnInitDialog()
    {
        CDialog::OnInitDialog();

        // TODO: Add extra initialization here

        CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN1)-
        >GetControlUnknown();
        pSkin->ApplySkin((long)m_hWnd);

        return TRUE; // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
    }

    void CReceive::OnSave()
    {
        // TODO: Add your control notification handler code here

        CFileDialog m_IdFile (FALSE, "*.ENC", NULL,
        OFN_HIDEREADONLY, "Encrypt files|*.ENC");

        //CFileDialog m_IdFile (FALSE, NULL, "*.ENC",
        OFN_HIDEREADONLY, "Encrypt files|ENC");
        m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
        if (m_IdFile.DoModal() != IDCANCEL )
        {
            m_dec_file= m_IdFile.GetPathName();
            UpdateData(FALSE);
        }
    }
}
/*****/

// Send.cpp : implementation file
//

#include "stdafx.h"
#include "ECC.h"
#include "Send.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;

```

```

#endif

////////////////////////////////////
// CSend dialog

CSend::CSend(CWnd* pParent /*=NULL*/)
    : CDialog(CSend::IDD, pParent)
{
   //{{AFX_DATA_INIT(CSend)
    // NOTE: the ClassWizard will add member initialization here
   //}}AFX_DATA_INIT
}

void CSend::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
   //{{AFX_DATA_MAP(CSend)
    // NOTE: the ClassWizard will add DDX and DDV calls here
   //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CSend, CDialog)
   //{{AFX_MSG_MAP(CSend)
    // NOTE: the ClassWizard will add message map macros here
   //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CSend message handlers
/*****

// ECCDlg.h : header file
//

#if
!defined(AFX_ECCDLG_H__78BD108D_F35E_4023_B801_34489B80B63F__INCL
UDED_)
#define
AFX_ECCDLG_H__78BD108D_F35E_4023_B801_34489B80B63F__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#import "actskin4.ocx" no_implementation raw_interfaces_only raw_native_types
using namespace ACTIVESKINLib;

```

```
#include "atlbase.h"
```

```
////////////////////////////////////  
// CECCDlg dialog
```

```
class CECCDlg : public CDialog
```

```
{
```

```
// Construction
```

```
public:
```

```
    CECCDlg(CWnd* pParent = NULL);    // standard constructor
```

```
// Dialog Data
```

```
//{{AFX_DATA(CECCDlg)
```

```
enum { IDD = IDD_ECC_DIALOG };
```

```
    // NOTE: the ClassWizard will add data members here
```

```
//}}AFX_DATA
```

```
// ClassWizard generated virtual function overrides
```

```
//{{AFX_VIRTUAL(CECCDlg)
```

```
protected:
```

```
    virtual void DoDataExchange(CDataExchange* pDX);    // DDX/DDV
```

```
support
```

```
    //}}AFX_VIRTUAL
```

```
// Implementation
```

```
protected:
```

```
    HICON m_hIcon;
```

```
// Generated message map functions
```

```
//{{AFX_MSG(CECCDlg)
```

```
    virtual BOOL OnInitDialog();
```

```
    afx_msg void OnSysCommand(UINT nID, LPARAM lParam);
```

```
    afx_msg void OnPaint();
```

```
    afx_msg HCURSOR OnQueryDragIcon();
```

```
    afx_msg void OnEnc();
```

```
    afx_msg void OnDec();
```

```
    afx_msg void OnGk();
```

```
//}}AFX_MSG
```

```
DECLARE_MESSAGE_MAP()
```

```
};
```

```
//{{AFX_INSERT_LOCATION}}
```

```
// Microsoft Visual C++ will insert additional declarations immediately before the  
previous line.
```

```
#endif //
```

```
!defined(AFX_ECCDLG_H__78BD108D_F35E_4023_B801_34489B80B63F__INCL  
UDED_)
```

```

/*****/

#ifdef AFX_ENCRDLG_H__FC8F4E2D_0AA7_40B0_8535_72B1F57A3B02__IN
CLUDED_
#define AFX_ENCRDLG_H__FC8F4E2D_0AA7_40B0_8535_72B1F57A3B02__INCLUDED
-

#include "GkDlg.h" // Added by ClassView
#include "ECCDlg.h"
#ifdef _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// EncrDlg.h : header file
//

////////////////////////////////////
// CEncrDlg dialog

class CEncrDlg : public CDialog
{
// Construction
public:
    void SendFile();
    char cmap (Point p);
    Encrypt encrypt(Point X, Point alpha, Point beta, int k, int prime, int A);
    Point map (char c);
    char CAP (char a);
    CEncrDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
    #ifdef AFX_DATA(CEncrDlg)
    enum { IDD = IDD_ENCRYPTION };
    int m_eeA;
    int m_eeB;
    int m_prime;
    int m_BPX;
    int m_BPY;
    int m_SK;
    int m_PKX;
    int m_PKY;
    CString m_enc_file;
    #endif AFX_DATA

// Overrides
    // ClassWizard generated virtual function overrides
    #ifdef AFX_VIRTUAL(CEncrDlg)

```



```
protected:
virtual void DoDataExchange(CDataExchange* pDX);  // DDX/DDV support
//}}AFX_VIRTUAL
```

```
// Implementation
protected:
```

```
    // Generated message map functions
    //{{AFX_MSG(CEncrDlg)
    afx_msg void OnEncrypt();
    afx_msg void OnSend1();
    afx_msg void OnBrows();
    virtual BOOL OnInitDialog();
    virtual void OnOK();
    //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
```

```
};
```

```
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.
```

```
#endif //
!defined(AFX_ENCRDLG_H__FC8F4E2D_0AA7_40B0_8535_72B1F57A3B02__IN
CLUDED_)
/*****
```

```
#if
!defined(AFX_GKDLG_H__60B8FD97_BE34_4EC3_9184_BD14219AD261__INCL
UDED_)
#define
AFX_GKDLG_H__60B8FD97_BE34_4EC3_9184_BD14219AD261__INCLUDED_
```

```
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// GkDlg.h : header file
//
/*****
*****/
```

```
typedef struct
```

```
{
    float x;
    float y;
}Point;
```

```
typedef struct
```

```
{
```

```

    Point x;
    int y;
    int z;
}Encrypt;

typedef struct
{
    char ch;
    Point PT;
}Assign;
/*****
*****/
////////////////////////////////////
// CGkDlg dialog

class CGkDlg : public CDialog
{
// Construction
public:
    int E;// to count #E ... No of points on elliptic curve;

    int inverse (int prime, int B3);
    Point Mult(Point alpha, int A, int prime, int a);
    int tst_AB (int P, int A, int B);
    CGkDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
//{{AFX_DATA(CGkDlg)
enum { IDD = IDD_GENERAT_KEY };
int         m_eeA;
int         m_eeB;
int         m_prime;
int         m_BPX;
int         m_BPY;
int         m_SK;
int         m_PKX;
int         m_PKY;
int         m_E;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CGkDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation

```

protected:

```
// Generated message map functions
//{{AFX_MSG(CGkDlg)
afx_msg void OnGenKey();
virtual BOOL OnInitDialog();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_GKDLG_H__60B8FD97_BE34_4EC3_9184_BD14219AD261__INCL
UED_)
/*****

// DecrDlg.cpp : implementation file
//

#include "stdafx.h"
#include "ECC.h"
#include "DecrDlg.h"
#include "EncrDlg.h"
#include "GkDlg.h"
#include "Recive.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

////////////////////////////////////
// CDecrDlg dialog

CDecrDlg::CDecrDlg(CWnd* pParent /*=NULL*/)
: CDialog(CDecrDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CDecrDlg)
    m_eeA = 0;

```

```

    m_eeB = 0;
    m_prime = 0;
    m_SK = 0;
    m_dec_file = _T("");
    //}}AFX_DATA_INIT
}

void CDecrDlg::DoDataExchange(CDataExchange* pDX)
{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CDecrDlg)
    DDX_Text(pDX, IDC_EDIT2, m_eeA);
    DDX_Text(pDX, IDC_EDIT3, m_eeB);
    DDX_Text(pDX, IDC_EDIT4, m_prime);
    DDX_Text(pDX, IDC_EDIT8, m_SK);
    DDX_Text(pDX, IDC_EDIT1, m_dec_file);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CDecrDlg, CDialog)
    //{{AFX_MSG_MAP(CDecrDlg)
    ON_BN_CLICKED(IDC_Decrypt, OnDecrypt)
    ON_BN_CLICKED(IDC_Send2, OnSend2)
    ON_BN_CLICKED(IDC_Brows2, OnBrows2)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CDecrDlg message handlers

void CDecrDlg::OnDecrypt()
{
    // TODO: Add your control notification handler code here

    //***** Decryption Mode
    //*****
    //Here we are making the decryption operation

    FILE *decrypt1;
    FILE *output1;

    CEncrDlg EN;
    int A, prime, a;

    UpdateData (TRUE);

    A = m_eeA;

```

```
prime = m_prime;
a = m_SK;
```

```
if ((output1 = fopen(m_dec_file, "r")) == NULL)
{
    MessageBox("Cannot open input file to read from");
    exit(1);
}
fseek(output1, SEEK_SET, 0);
```

```
CString Save_file;
```

```
CFileDialog m_IdFile (FALSE, "*.txt", NULL,
OFN_HIDEREADONLY, "Decrypt files|*.txt");
m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
```

```
if (m_IdFile.DoModal() != IDCANCEL )
{
    Save_file = m_IdFile.GetPathName();
    UpdateData(FALSE);
}
```

```
if ((decrypt1 = fopen(Save_file, "w+")) == NULL)
{
    MessageBox("Cannot open output file to write on");
    exit(1);
}
```

```
char c1, c2, c3, ch_dec;
Encrypt Enc1;
```

```
Point dec;
```

```
do
{
    c1 = fgetc(output1);
    Enc1.x = EN.map(c1);

    c2 = fgetc(output1);
    Enc1.y = c2;

    c3 = fgetc(output1);
    Enc1.z = c3;
```



```

        dec = decrypt (Enc1, A, prime, a);

        ch_dec = EN.cmap(dec);

        putchar(ch_dec);

        fwrite (&ch_dec, sizeof(char), 1, decrypt1);

    } while (c1 != EOF);

    fclose(output1);
    fclose(decrypt1);

    MessageBox("Your Decryption Operation Successfulys");
}

Point CDecrDlg::decrypt(Encrypt Enc, int A, int prime, int a)
{
    Point res={0,0};
    Point dec;
    int x1,x2;

    CGkDlg d;

    res = d.Mult(Enc.x,A,prime,a);
    getch();
    x1 = (d.inverse(prime,res.x) * Enc.y); x1 = fmod(x1,prime);
    x2 = (d.inverse(prime,res.y) * Enc.z); x2 = fmod(x2,prime);

    dec.x = x1;
    dec.y = x2;

    return dec;
}

void CDecrDlg::OnSend2()
{
    // TODO: Add your control notification handler code here

    CRecv receive;
    MessageBox("Enter the IP of your friend");

    receive.DoModal();
}

void CDecrDlg::OnBrows2()

```

```

{
    // TODO: Add your control notification handler code here

    CFileDialog m_IdFile (TRUE, NULL, NULL, OFN_HIDEREADONLY, "text
files|*.Enc");
    m_IdFile.m_ofn.lpstrInitialDir = "c:\\";
    if (m_IdFile.DoModal() != IDCANCEL )
    {
        m_dec_file= m_IdFile.GetPathName();
        UpdateData(FALSE);
    }
}

BOOL CDecrDlg::OnInitDialog()
{
    CDialog::OnInitDialog();

    // TODO: Add extra initialization here

    CComQIPtr<ISkin> pSkin = GetDlgItem(IDC_SKIN1)-
>GetControlUnknown();
    pSkin->ApplySkin((long)m_hWnd);

    return TRUE; // return TRUE unless you set the focus to a control
        // EXCEPTION: OCX Property Pages should return FALSE
}
/*****
# if
!defined(AFX_RECVIE_H__1728CEAA_CADA_450A_A089_B6B3D8783167__INC
LUDED_)
#define
AFX_RECVIE_H__1728CEAA_CADA_450A_A089_B6B3D8783167__INCLUDED
-

# if _MSC_VER > 1000
# pragma once
# endif // _MSC_VER > 1000
// Recive.h : header file
//

////////////////////////////////////////////////////////////////
// CRecive dialog

class CRecive : public CDialog
{
// Construction
public:
    void ReciveFile();

```

```

CRecive(CWnd* pParent = NULL); // standard constructor

// Dialog Data
//{{AFX_DATA(CRecive)
enum { IDD = IDD_DIALOG1 };
CString      m_ip;
CString      m_dec_file;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CRecive)
protected:
virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

// Generated message map functions
//{{AFX_MSG(CRecive)
afx_msg void OnRConnect();
virtual BOOL OnInitDialog();
afx_msg void OnSave();
//}}AFX_MSG
DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
#ifndef AFX_RECIVE_H__1728CEAA_CADA_450A_A089_B6B3D8783167__INC
LUDED_ )

/*****/

//{{NO_DEPENDENCIES}}
// Microsoft Developer Studio generated include file.
// Used by ECC.rc
//
#define IDM_ABOUTBOX                0x0010
#define IDD_ABOUTBOX                100
#define IDS_ABOUTBOX                101
#define IDD_ECC_DIALOG              102
#define IDP_SOCKETS_INIT_FAILED    103
#define IDR_MAINFRAME               128

```

```

#define IDD_ENCRYPTION 129
#define IDD_DECRYPTION 130
#define IDD_GENERAT_KEY 131
#define IDD_DIALOG1 133
#define IDC_ENC 1000
#define IDC_DEC 1001
#define IDC_GK 1002
#define IDC_Encrypt 1003
#define IDC_Send1 1004
#define IDC_EDIT1 1005
#define IDC_Brows 1006
#define IDC_EDIT2 1007
#define IDC_EDIT3 1008
#define IDC_EDIT4 1009
#define IDC_EDIT6 1011
#define IDC_EDIT7 1012
#define IDC_EDIT8 1013
#define IDC_EDIT10 1015
#define IDC_EDIT11 1016
#define IDC_Decrypt 1017
#define IDC_Send2 1018
#define IDC_Brows2 1019
#define IDC_GenKey 1020
#define IDC_RConnect 1023
#define IDC_SKIN1 1024
#define IDC_BUTTON1 1025
#define IDC_E 1027
#define IDC_EDIT9 1029

```

// Next default values for new objects

//

```

#ifdef APSTUDIO_INVOKED
#ifndef APSTUDIO_READONLY_SYMBOLS
#define _APS_NEXT_RESOURCE_VALUE 143
#define _APS_NEXT_COMMAND_VALUE 32771
#define _APS_NEXT_CONTROL_VALUE 1030
#define _APS_NEXT_SYMED_VALUE 101
#endif
#endif

```

/*****/

```

// stdafx.h : include file for standard system include files,
// or project specific include files that are used frequently, but
// are changed infrequently
//

```

```

#if
!defined(AFX_STDAFX_H_FE507C59_E3CE_49FF_83CA_3B01BD8A0C85__INC
LUDED_)

```

```

#define
AFX_STDAFX_H__FE507C59_E3CE_49FF_83CA_3B01BDBA0C85__INCLUDED
-

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#define VC_EXTRALEAN           // Exclude rarely-used stuff from Windows
headers

#include <afxwin.h>           // MFC core and standard components
#include <afxext.h>           // MFC extensions
#include <afxdisp.h>         // MFC Automation classes
#include <afxdtctl.h>         // MFC support for Internet Explorer 4 Common Controls
#ifndef _AFX_NO_AFXCMN_SUPPORT
#include <afxcmn.h>           // MFC support for Windows Common Controls
#endif // _AFX_NO_AFXCMN_SUPPORT

#include <afxsock.h>          // MFC socket extensions

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
!defined(AFX_STDAFX_H__FE507C59_E3CE_49FF_83CA_3B01BDBA0C85__INC
LUDED_)
/*****
*/
#if
!defined(AFX_DECRDLG_H__6D8CAF83_8B2A_44FE_B3E1_F39804D5AC90__IN
CLUDED_)
#define
AFX_DECRDLG_H__6D8CAF83_8B2A_44FE_B3E1_F39804D5AC90__INCLUDE
D_

#include "GkDlg.h" // Added by ClassView
#include "ECCDlg.h"
#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
// DecrDlg.h : header file
//

////////////////////////////////////
// CDecrDlg dialog

class CDecrDlg : public CDialog
{

```



```

// Construction
public:
    Point decrypt (Encrypt Enc, int A, int prime, int a);
    CDecrDlg(CWnd* pParent = NULL); // standard constructor

// Dialog Data
//{{AFX_DATA(CDecrDlg)
enum { IDD = IDD_DECRYPTION };
int         m_eeA;
int         m_eeB;
int         m_prime;
int         m_SK;
CString     m_dec_file;
//}}AFX_DATA

// Overrides
// ClassWizard generated virtual function overrides
//{{AFX_VIRTUAL(CDecrDlg)
protected:
    virtual void DoDataExchange(CDataExchange* pDX); // DDX/DDV support
//}}AFX_VIRTUAL

// Implementation
protected:

    // Generated message map functions
    {{{AFX_MSG(CDecrDlg)
    afx_msg void OnDecrypt();
    afx_msg void OnSend2();
    afx_msg void OnBrows2();
    virtual BOOL OnInitDialog();
    }}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

#endif //
#ifndef AFX_DECRDLG_H_6D8CAF83_8B2A_44FE_B3E1_F39804D5AC90__IN
    CLUDED_
    /*****
    // ECC.h : main header file for the ECC application
    */

```

```

#if
!defined(AFX_ECC_H__B19448DB_0E6A_40FC_9A8E_5ED8B69FA381__INCLUD
ED_)
#define
AFX_ECC_H__B19448DB_0E6A_40FC_9A8E_5ED8B69FA381__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#ifndef __AFXWIN_H__
    #error include 'stdafx.h' before including this file for PCH
#endif

#include "resource.h"           // main symbols

////////////////////////////////////
// CECCApp:
// See ECC.cpp for the implementation of this class
//

class CECCApp : public CWinApp
{
public:
    CECCApp();

// Overrides
    // ClassWizard generated virtual function overrides
   //{{AFX_VIRTUAL(CECCApp)
public:
    virtual BOOL InitInstance();
   //}}AFX_VIRTUAL

// Implementation

   //{{AFX_MSG(CECCApp)
        // NOTE - the ClassWizard will add and remove member functions here.
        // DO NOT EDIT what you see in these blocks of generated code !
   //}}AFX_MSG
    DECLARE_MESSAGE_MAP()
};

////////////////////////////////////

//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the
previous line.

```

```
#endif//  
!defined(AFX_ECC_H__B19448DB_0E6A_40FC_9A8E_5ED8B69FA381__INCLUD  
ED_)  
/*****/
```