PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

Order N° : .....

UNIVERSITY of M'SILA

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Department of Computer Science

Dissertation

Submitted in fulfilment of the requirements for the degree of academic master in computing

Domain: Mathematics and computer science
Field: Computer science
Option: Networks and distributed systems
By: Tayoub Walid

TOPIC

Implementation of Public-Key Cryptosystems on Android Mobiles

Defended publicly on: …./06/2013 by the jury consisted of:

Mr.	University of M'sila President.
Mr. Noureddine Chikouche	University of M'sila Supervisor.
Dr. Linda Belabdelouahab Fernini	University of M'sila Supervisor.
Mr.	University of M'sila Examiner.

Academic year: 2012/2013

Acknowledgement

To the great parents before being great professors

whatever i say, i won't be able to express my deep thanks to the teaching staff in my faculty specially my supervisors ***Mr.Noureddine Chikouche*** *and* ***Dr. Linda Belabdelouahab Fernini*** *who accepted to be my supervisors and helped and supported me with the necessary information in my dissertation. It is impossible for me to list their efforts in preparing me to be a great student in research.*

In addition, I would like to extend my gratitude to my colleague ***Lakehali Somia*** *who helped me and showed me clearly her feeling of responsibility toward me.*

Really, I appreciate this.

dedication

Thanks to ALLAH in the first and last place.

Greeting to the glorious souls of our immortal martyrs who sacrificed them selves to make our country best

Our fathers and mothers are among Allah's greatest gifts to us, let's honor them with respect, love and appreciation and let us not forget them in our prayers.

To my great mother "DJAMILA" and To my great father MOHAMED who has helped, supported, encouraged, advised and sacrificed a lot for us to be a good men, thank you about every thing.

To my dear's brothers Fatima, Hadjira, Abdelrazzak, Mustapha, Amel,Abderrahim, Yacine, who have assisted and shared me the good and the bad times in my life and aided me to pass the critical times.

To the closest people L s, who shared with me this critical period of my life

To my Education Friends b.Ahmed, B.Zaki, and special dedicate to my best friend Sami IBRAHIM who shared me best times in my life.

Thanks to all of our families and our friends who have helped and supported us from near or far across our life...

Thanks to you all

WALID

SUMMARY

Elements | SUMMARY | Page

First chapter

Second chapter

Third chapter

Fourth chapter

LIST OF TABLES

Table | TABLES LISTE | Page

Chapter 3

Chapter 4

LIST OF FIGUERS

Figure | FIGURES LISTE | Page

Chapter 1

Chapter 2

Chapter 3

Chapter 4

GENERAL INTRODUCTION

General Introduction

The world is at the beginning of the mobile data revolution. The convenience and power of mobile applications delivered on emerging smart devices will fuel the rapid growth in subscribers and sheer volume of data. Operators world-wide are racing to add new services and more powerful devices. They are making substantial investments to upgrade the capacity and performance of their networks.

With the growth of wireless networks and the widespread use of mobile devices, the need for development of mobile applications increases. The types of these applications include several areas. Several types of these applications require secure transmissions authentication data between mobile devices and a server such as banking information online, user logins, server authentication, and so on.

Wireless transmission poses security problems. The security of the transmission channel is the most important problem. The information transmitted through the air is not protected against listening and intercepting. Protocols of the Internet were originally designed to disseminate information for flexible use and not for ensures security. Transmitting mobile device to distributed servers provides challenges for securing sensitive user data such as passwords, PINs codes, and credit card information and bank accounts. The Cryptography is the most interesting tool for automatic and secure networks communications.

Since mobile systems are growing quickly, the electronic transactions using computers will change gently to be with the mobile. As a result, mobile security will become one of the most important parts of mobile system and will become the hottest area facing the mobile transactions due to network directness. However, the appropriate encryption scheme for mobile communication must have a small amount of data calculating and quick operation as of its inherent restrictions of small quantity and low calculating ability.

Cryptography enables secure communications and provides the authentication, but the application of cryptography at the software level in mobile applications can slow down the time of execution due to limited resources of these devices. If the cryptography is not well implemented, it can have a performance problem. A bad choice of algorithms and cryptographic key sizes may reduce the security of the system.

Objective of this work:

The objective of the work entitled "**implementation of public key cryptosystems on android mobiles"** is to find a public key cryptosystem fit the android mobiles environments by giving a maximum security, taking a short time to accomplish its processes (encryption, decryption..), and occupy less resource like memory and CPU. We can resume our work in the following steps:

1. Studying and analysing some public-key cryptosystems.
2. Implementing them on different android mobiles.
3. Test the performance and compare between them

Memory organization:

About the methodological plan of our work, our dissertation is divided into four chapters:

- The first chapter presents the different models of crypto systems and identifies the fundamental rules of cryptography
- The second chapter surveys the integer factorization problem, discrete logarithm problem and elliptic curve discrete logarithm problem, includes the current state of algorithms for solving them.
- The third chapter studies some public-key cryptography systems like (RSA, elgamal…..).
- The fourth chapter discusses the implementation of some public-key cryptosystems on different android mobiles, and then we test the performance of each algorithm, compare our results with related works, and discuss the results.

CHAPTER 01

The fundamental rules of cryptography

”Security is a process not a product. If you think technology can solve your security problems, then you don’t understand the problem and you don’t understand the technology”

—Bruce Schneir, Secret and lies

1. Introduction

The use of cryptography went back to the time of the Egyptians. Since 1960s, it was used by governments and the military to protect secrets and national strategies. With the expansion of the Internet in the early 1990s, cryptography widespread and played an important role in the internet security. Security protocols HTTPS, SSL and TLS mechanisms are still the most used in the safety Data communication over the Internet. Cryptography is not only use in security protocols of the Internet, but it uses also in applications that run on servers and mobile devices [1].

The Oxford English Dictionary gives the following definition of the term cryptography:

« A secret manner of writing, either by arbitrary characters, by using letters or characters in other than their ordinary sense, or by other methods intelligible only to those possessing the key, also anything written in this way. Generally, the art of writing or solving ciphers ».

Cryptography is an ancient art, this definition was adequate until a few several decades past. But since 1970s, cryptography was no longer limited to messages encryption [2].

In this chapter, we will present the different models of crypto systems and we will identify the fundamental rules of cryptography, which are necessary to achieve strong and effective security. We can say it is not a totally secure system, but the application of strong cryptosystems reinforces the security of our application.

2. Common goals of cryptography [3]

In essence, cryptography concerns four main goals:

1. Message **confidentiality** (or privacy): Only an authorized recipient should be able to extract the contents of the message from its encrypted form. Resulting from steps to hide, stop or delay free access to the encrypted information.
2. Message **integrity**: The recipient should be able to determine if the message has been altered.
3. **Authentication**: The recipient should be able to verify from the message, the identity of the sender, the origin or the path it travelled (or combinations) so to validate claims from emitter or to validate the recipient expectations.
4. **Non-repudiation**: The emitter should not be able to deny sending the message.

Not all cryptographic systems achieve all of the above goals. Some applications of cryptography have *different* goals; for example, some situations require **repudiation** where a participant can plausibly deny that they are a sender or receiver of a message, or extend these goals to include variations like:

1. Message **access control**: Who are the valid recipients of the message?
2. Message **availability**: By providing means to limit the validity of the message, channel, emitter or recipient in time or space.

3. Definition of cryptography

Cryptography is the science of using mathematics to encrypt and decrypt data. Cryptography enables you to store sensitive information or transmit it across insecure networks (like the Internet) so that it cannot be read by anyone except the intended recipient. While cryptography is the science of securing data, cryptanalysis is the science of analysing and breaking secure communication. Classical cryptanalysis involves an interesting combination of analytical reasoning, application of mathematical tools, pattern finding, patience, determination, and luck. Cryptanalysts are also called attackers. Cryptology embraces both cryptography and cryptanalysis [3].

4. Functionality of work cryptography

A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a

key—a word, number, or phrase—to encrypt the plaintext. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key [3].

5. Encryption and decryption

Data that can be read and understood without any special measures is called plaintext or clear text. The method of disguising plaintext in such a way as to hide its substance is called encryption. Encrypting plaintext results in unreadable gibberish called cipher text. You use encryption to ensure that information is hidden from anyone for whom it is not intended, even those who can see the encrypted data. The process of reverting cipher text to its original plaintext is called decryption [3], this process is illustrated in (Figure 1-1).

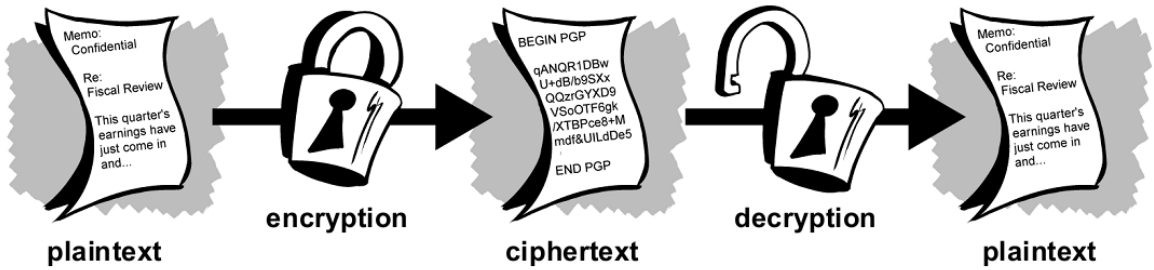

Figure 1.1 Encryption and decryption [3]

6. Conventional cryptography

In conventional cryptography, also called secret-key or symmetric-key encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) is an example of a conventional cryptosystem that is widely employed by the Federal Government [3]. Figure 1-2is an illustration of the conventional encryption process.

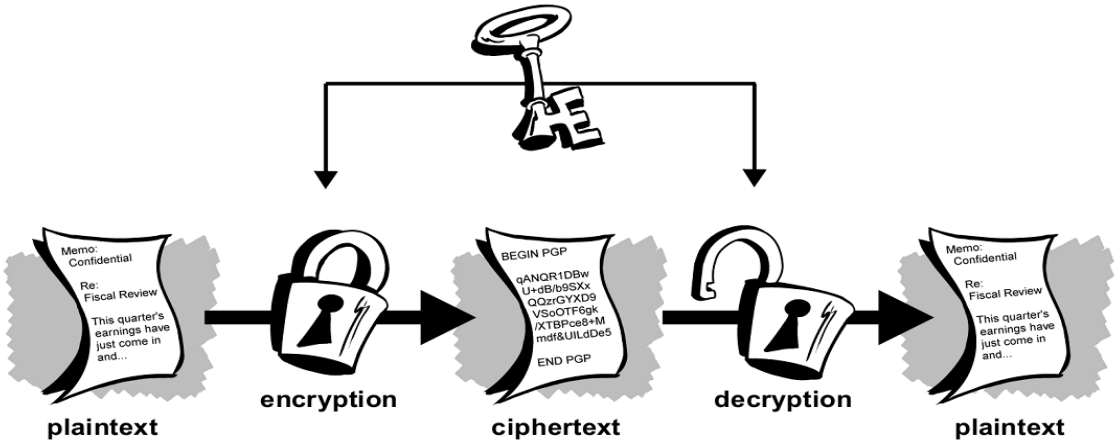

Figure 1.2 Conventional encryption [3]

6.1 Caesar's Cipher

An extremely simple example of conventional cryptography is a substitution cipher. A substitution cipher substitutes one piece of information for another. This is most frequently done by offsetting letters of the alphabet. Two examples are Captain Midnight's Secret Decoder Ring, which you may have owned when you were a kid, and Julius Caesar's cipher. In both cases, the algorithm is to offset the alphabet and the key is the number of characters to offset it.

For example, if we encode the word "SECRET" using Caesar's key value of 3, we offset the alphabet so that the 3rd letter down (D) begins the alphabet.

So starting with

ABCDEFGHIJKLMNOPQRSTUVWXYZ and sliding everything up by 3, you get DEFGHIJKLMNOPQRSTUVWXYZABC where D=A, E=B, F=C, and so on.

Using this scheme, the plaintext, "SECRET" encrypts as "VHFUHW." To allow someone else to read the cipher text, you tell them that the key is 3.

Obviously, this is exceedingly weak cryptography by today's standards, but hey, it worked for Caesar, and it illustrates how conventional cryptography works [3].

7. Key management and conventional encryption

Conventional encryption has benefits. It is very fast. It is especially useful for encrypting data that is not going anywhere. However, conventional encryption alone as a means for transmitting secure data can be quite expensive simply due to the difficulty of secure key distribution.

For a sender and recipient to communicate securely using conventional encryption, they must agree upon a key and keep it secret between themselves. If they are in different physical locations, they must trust a courier, the Bat Phone, or some other secure communication medium to prevent the disclosure of the secret key during transmission. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all information encrypted or authenticated with that key. From DES to Captain Midnight's Secret Decoder Ring [3], the persistent problem with conventional encryption is key distribution: how do you get the key to the recipient without someone intercepting it?

8. Public key cryptography

The problems of key distribution are solved by public key cryptography, the concept of which was introduced by Whitfield Diffie and Martin Hellman in 1975. (There is now evidence that the British Secret Service invented it a few years before Diffie and Hellman, but kept it a military secret—and did nothing with it.)

Public key cryptography is an asymmetric scheme that uses a pair of keys for encryption: a public key, which encrypts data, and a corresponding private, or secret key ford encryption. You publish your public key to the world while keeping your private key secret. Anyone with a copy of your public key can then encrypt information that you can read only. Even people you have never met.

It is computationally infeasible to deduce the private key from the public key. Anyone who has a public key can encrypt information but cannot decrypt it. Only the person who has the corresponding private key can decrypt the information [3].

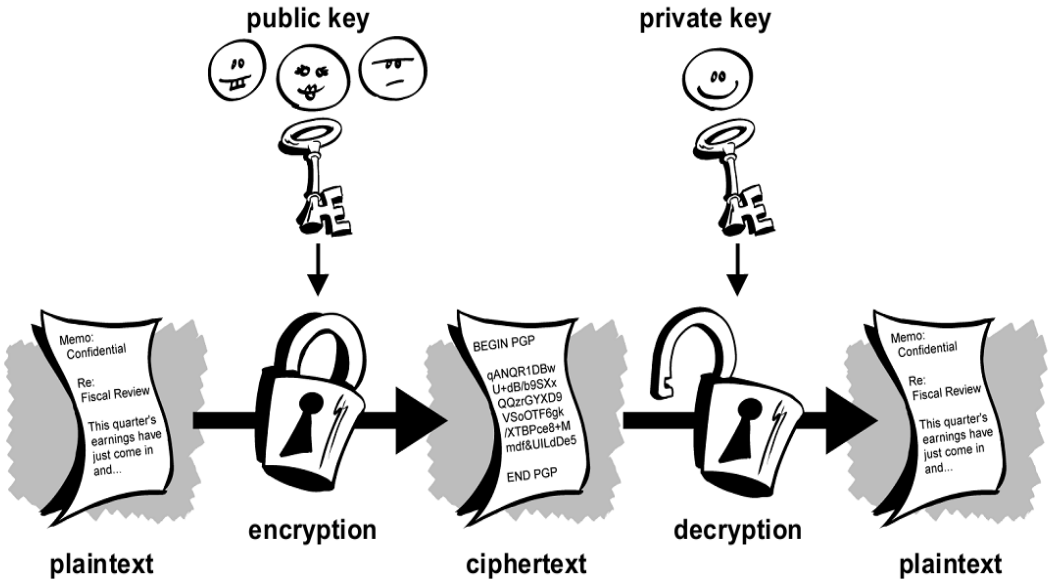

Figure 1.3 Public key encryption [3]

The primary benefit of the public key cryptography is that it allows people who have no pre-existing security arrangement to exchange messages securely. The need for sender and receiver to share secret keys via some secure channel is eliminated. All communications involve only public keys, and no private key is ever transmitted or shared. Some examples of public-key cryptosystems are Elgamal (named for its inventor, Taher Elgamal), RSA (named

for its inventors, Ron Rivest, Adi Shamir, and Leonard Adleman), Diffie-Hellman (named, you guessed it, for its inventors), and DSA, the Digital Signature
Algorithm (invented by David Kravitz).

Because conventional cryptography was once the only available means for relaying secret information, the expense of secure channels and key distribution relegated its use only to those who could afford it, such as governments and large banks (or small children with secret decoder rings). Public key encryption is the technological revolution that provides strong cryptography to the adult masses. Remember the courier with the locked briefcase handcuffed to his wrist? Public-key encryption puts him out of business (probably to his relief).

9. Keys

A key is a value that works with a cryptographic algorithm to produce a specific cipher text. Keys are basically really, big numbers. Key size is measured in bits; the number representing a 1024-bit key is darn huge. In public key cryptography, the bigger the key, the more secure the cipher text.

However, public key size and conventional cryptography's secret key size are totally unrelated. A conventional 80-bit key has the equivalent strength of a 1024-bit public key. A conventional 128-bit key is equivalent to a 3000-bit public key. Again, the bigger the key, the more secure, but the algorithms used for each type of cryptography are very different and thus comparison is like that of apples to oranges.

While the public and private keys are mathematically related, it's very difficult to derive the private key given only the public key; however, deriving the private key is always possible given enough time and computing power. This makes it very important to pick keys of the right size; large enough to be secure, but small enough to be applied fairly quickly. Additionally, you need to consider who might be trying to read your files, how determined they are, how much time they have, and what their resources might be.

Larger keys will be cryptographically secure for a longer period of time. If what you want to encrypt needs to be hidden for many years, you might want to use a very large key. Of course, who knows how long it will take to determine your key using tomorrow's faster, more efficient computers? There was a time when a 56-bit symmetric key was considered extremely safe [3].

10. Digital Signatures

A major benefit of public key cryptography is that it provides a method for employing *digital signatures*. Digital signatures enable the recipient of information to verify the authenticity of the information's origin, and also verify that the information is intact. Thus, public key digital signatures provide *authentication* and data *integrity*. A digital signature also provides *non-repudiation*, which means that it prevents the sender from claiming that he or she did not actually send the information. These features are every bit as fundamental to cryptography as privacy, if not more.

A digital signature serves the same purpose as a handwritten signature. However, a handwritten signature is easy to counterfeit. A digital signature is superior to a handwritten signature in that it is nearly impossible to counterfeit, plus it attests to the contents of the information as well as to the identity of the signer.

[2]Some people tend to use signatures more than they use encryption. For example, you may not care if anyone knows that you just deposited $1000 in your account, but you do want to be darn sure it was the bank teller you were dealing with.

The basic manner in which digital signatures are created is illustrated in **Figure 1-5**. Instead of encrypting information using someone else's public key, you encrypt it with your private key. If the information can be decrypted with your public key, then it must have originated with you [3].

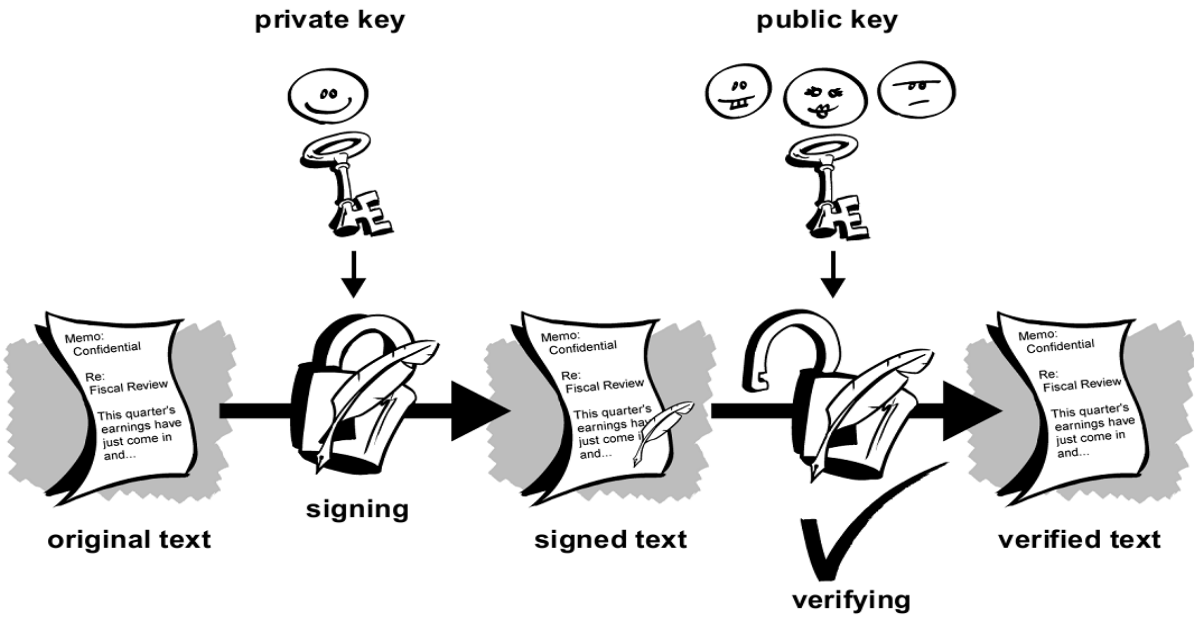

Figure 1.4 Simple digital signature [3]

11. Conclusion

In this chapter we introduced the cryptography science and its useful in our life in this days, we presented the rules of public key crypto-systems and how they work and in the next chapter we will go deeply into the public key cryptosystems.

CHAPTER 02

Mathematical Concepts

1. Introduction

There are numerous cryptosystems whose security is based on the difficullty of solving the logarithm problem such as discret and eliptic curve logarithm prblem. This chapter is a survey of the integer factorization problem,discret logarithm problem and eliptic curve discret logarithm problem, including the current state of algorithms for solving them , complexity isssues related to them and their application in the cryptography.

2. Integer factorization[4]

In number theory, integer factorization or prime factorization is the decomposition of a composite number into smaller non-trivial divisors, when multiplied together equal the original integer.

When the numbers are very large, no efficient, non-quantum integer factorization algorithm is known; an effort concluded in 2009 by several researchers factored a 232-digit number (RSA-768), utilizing hundreds of machines over a span of 2 years. The presumed difficulty of this problem is at the heart of widely used algorithms in cryptography such as RSA. Many areas of mathematics and computer science have been brought to bear on the problem, including elliptic curves, algebraic number theory, and quantum computing.

Not all numbers of a given length are equally hard to factor. The hardest instances of these problems (for currently known techniques) are semi primes, the product of two prime numbers. When they are both large, for instance more than 2000 bits long, randomly chosen, and about the same size (but not too close, e.g. to avoid efficient factorization by Fermat's factorization method), even the fastest prime factorization algorithms on the fastest computers can take enough time to make the search impractical; that is, as the number of digits of the primes being factored increases, the number of operations required to perform the factorization on any computer increases drastically.

Many cryptographic protocols are based on the difficulty of factoring large composite integers or a related problem, the RSA problem. An algorithm that efficiently factors an arbitrary integer would render RSA-based public-key cryptography insecure.

2.1 Prime decomposition [4]

By the fundamental theorem of arithmetic, every positive integer has a unique prime factorization. (A special case for 1 is not needed using an appropriate notion of the empty product.) However, the fundamental theorem of arithmetic gives no insight into how to obtain an integer's prime factorization; it only guarantees its existence. Given a general algorithm for integer factorization, one can factor any integer down to its constituent prime factors by repeated application of this algorithm. However, this is not the case with a special-purpose factorization algorithm, since it may not apply to the smaller factors that occur during decomposition, or may execute very slowly on these values. For example, if N is the number

$(2^{521} - 1) \times (2^{607} - 1)$, then trial division will quickly factor 10N as 2 × 5 × N, but will not quickly factor N into its factors.

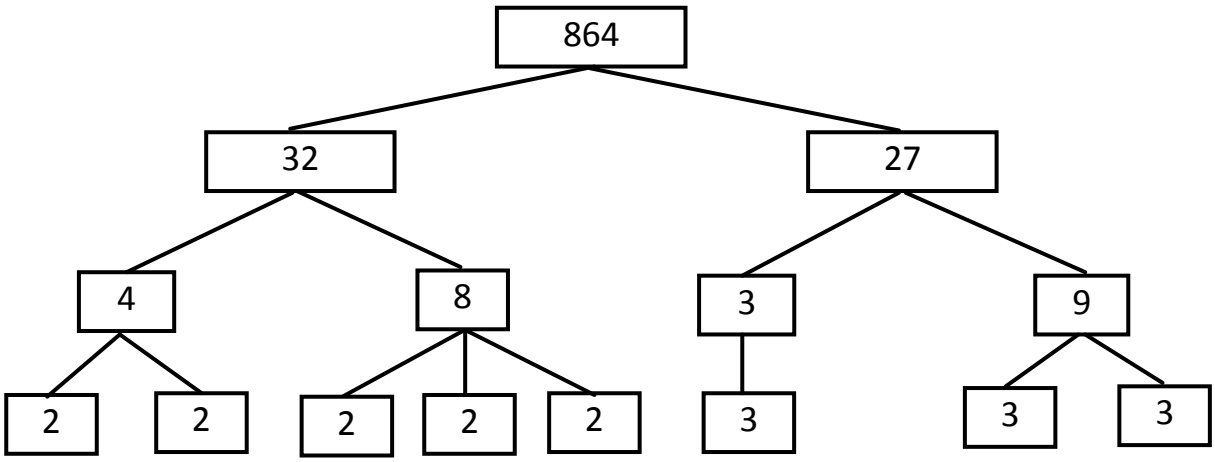

Figure 2.1 prime decomposition of 864.

2.2 Current state of the art [4]

The most difficult integer to factor in practice using existing algorithms are those that are products of two large primes of similar size, and for this reason, these are the integers used in cryptographic applications. The largest such semi prime yet factored was RSA-768, a 768-bit number with 232 decimal digits, on December 12, 2009.This factorization was a collaboration of several research institutions, spanning two years and taking the equivalent of almost 2000 years of computing on a single-core 2.2 GHz AMD Opteron. Like all recent factorization records, this factorization was completed with a highly optimized implementation of the general number field sieve run on hundreds of machines.

3. Algorithms of integer factorization [5]

In this section we will take look on some integer factorization algorithms:

3.1 Algorithm: Trial Division

Trial division is the simplest algorithm for factoring an integer. Assume that s and t are nontrivial factors of N such that st = N and s £ t. To perform the trial division algorithm, one simply checks whether s | N for s = 2, … . , $[\sqrt{N}]$. When such a divisor s is found, then t = N / s is also a factor, and a factorization has been found for N. The upper bound of s≤ $[\sqrt{N}]$ is provided by the following theorem:

Theorem. If N has nontrivial factors s, t with st = N and ss ≤ t, then s ≤ $\sqrt{N}$.

3.2 Pseudo-code: Trial Division

```
Function trial Division (N)
    for s from 2 to floor(sqrt(N))
        if s divides N then
            return s, N/s
        end if
    end for
end function
```

If this algorithm is given composite N, then it returns a pair of nontrivial factors s, t with s ≤ t. The statement s | N is equivalent to s ≡ 0 (mod N) , and so it can be implemented via modular arithmetic in most languages.

3.2.1 Algorithm: Fermat Factorization [5]:

This algorithm was discovered by mathematician Pierre de Fermat in the 1600s Fermat factorization rewrites a composite number N as the difference of squares

$$N = x^2 - y^2$$

This difference of squares leads immediately to the factorization of N:

$$N = (x + y)(x - y)$$

Assume that s and t are nontrivial odd factors of N such that st = N and s≤t. We can find x and y such that s = (x - y) and t = (x + y). Solving this equation, we find that x = (s + t) / 2 and y = (t -s) / 2. Here x and y are integers, since the difference between any two odd numbers is even, and an even number is divisible by two. Since s > 1 and t ≥s , we find that x≥1 and y≥ 0

0. For particular x, y satisfying s = (x . y) and t = (x + y), we thus know that $x = \sqrt{N + y^2}$, and hence $x \geq \sqrt{N}$ .Also, $x \leq (s + t) / 2 \leq 2t / 2 \leq N$.

3.2.2. Pseudocode: Fermat Factorisation :

```
Function Fermat Factor(N)
    for x from ceil(sqrt(N)) to N
        ySquared := x * x - N
        if is Square(ySquared) then
            y := sqrt(ySquared)
            s := (x - y)
            t := (x + y)
            if s <> 1 and s <> N then
            return s, t
            end if
        end if
    end for
end function
```

4. Discrete logarithm problem [6, 7]

In mathematics, specifically in abstract algebra and its applications, **discrete logarithms** are group-theoretic analogues of ordinary logarithms. In particular, an ordinary logarithm $\log_a(b)$ is a solution of the equation $a^x = b$ over the real or complex numbers. Similarly, if g and h are elements of a finite cyclic group G then a solution x of the equation $g^x = h$ is called a discrete logarithm to the base g of h in the group G.

4.1 Example

Discrete logarithms are perhaps simplest to understand in the group $(Z_p)^\times$. This is the set $\{1, \ldots, p-1\}$ of congruence classes under multiplication modulo the prime p.

If we want to find the kth power of one of the numbers in this group, we can do so by finding its kth power as an integer and then find the remainder after division by p. This process is called discrete exponentiation. For example, consider $(Z_{17})^\times$. To compute 3^4 in this group, we first compute $3^4 = 81$, and then we divide 81 by 17, obtain a remainder of 13. Thus $3^4 = 13$ in the group $(Z_{17})^\times$.

Discrete logarithm is just the inverse operation. For example, take the equation $3^k \equiv 13 \pmod{17}$ for k. As shown above k=4 is a solution, but it is not the only solution. Since $3^{16} \equiv 1 \pmod{17}$ — which we know from Fermat's little theorem — it also follows that if n is an integer then $3^{4+16n} \equiv 3^4 \times (3^{16})n \equiv 13 \times 1^n \equiv 13 \pmod{17}$. Hence the equation has infinitely many solutions of the form 4 + 16n. Moreover, since 16 is the smallest positive integer m satisfying $3^m \equiv 1 \pmod{17}$, i.e. 16 is the order of 3 in $(Z_{17})^\times$, these are the only solutions. Equivalently, the solution can be expressed as $k \equiv 4 \pmod{16}$.

4.2 Definition [6, 7]

In general, let G be a finite cyclic group with n elements. We assume that the group is written multiplicatively. Let b be a generator of G; then every element g of G can be written in the form $g = b^k$ for some integer k. Furthermore, any two such integers k_1 and k_2 representing g will be congruent modulo n. We can thus define a function $\log_b : G \to Z_n$

(Where Z_n denotes the ring of integers modulo n) by assigning to each g the congruence class of k modulo n. This function is a group isomorphism, called the **discrete logarithm** to base b.

The familiar base change formula for ordinary logarithms remains valid: If c is another generator of G, then we have

$\text{Log}_c(g) = \text{Log}_c(b).\text{Log}_b(g)$.

4.3 Algorithms [7]

Can the discrete logarithm be computed in polynomial time on a classical computer?

No efficient classical algorithm for computing general discrete logarithms $\log_b g$ is known. The naive algorithm is to raise b to higher and higher powers k until the desired g is found; this is sometimes called trial multiplication. This algorithm requires running time linear in the size of the group G and thus exponential in the number of digits in the size of the group. There exists an efficient quantum algorithm due to Peter Shor.

More sophisticated algorithms exist, usually inspired by similar algorithms for integer factorization. These algorithms run faster than the naive algorithm, but none of them runs in polynomial time (in the number of digits in the size of the group).

• Baby-step giant-step

• Pollard's rho algorithm for logarithms

• Pollard's kangaroo algorithm (aka Pollard's lambda algorithm)

• Pohlig–Hellman algorithm

• Index calculus algorithm

• Number field sieve

• Function field sieve

4.4 Comparison with integer factorization [6]

While the problem of computing discrete logarithms and the problem of integer factorization are distinct problems they share some properties:

• both problems are difficult (no efficient algorithms are known for non-quantum computers),

• for both problems efficient algorithms on quantum computers are known,

• algorithms from one problem are often adapted to the other, and

• the difficulty of both problems has been used to construct various cryptographic systems.

4.5 Cryptography

There exist groups for which computing discrete logarithms is apparently difficult. In some cases (e.g. large prime order subgroups of groups $(Z_p)^\times$) there is not only no efficient algorithm known for the worst case, but the average-case complexity can be shown to be about as hard as the worst case using random self-reducibility.

At the same time, the inverse problem of discrete exponentiation is not difficult (it can be computed efficiently using exponentiation by squaring, for example). This asymmetry is analogous to the one between integer factorization and integer multiplication. Both asymmetries have been exploited in the construction of cryptographic systems.

Popular choices for the group G in discrete logarithm cryptography are the cyclic groups $(Z_p)^\times$; see ElGamal encryption, Diffie–Hellman key exchange, and the Digital Signature Algorithm.

Newer cryptography applications use discrete logarithms in cyclic subgroups of elliptic curves over finite fields.

5. Discrete logarithm records [6, 7]

Discrete logarithm records are the best results achieved to date in solving the discrete logarithm problem, which is the problem of finding solutions x to the equation $g^x = h$ given elements g and h of a finite cyclic group G. The difficulty of this problem is the basis for the security of several cryptographic systems, including Diffie–Hellman key agreement, ElGamal encryption, the ElGamal signature scheme, the Digital Signature Algorithm, and the elliptic curve cryptography analogs of these. Common choices for G used in these algorithms include the multiplicative group of integers modulo p, the multiplicative group of a finite field, and the group of points on an elliptic curve modulo p or over a finite field.

Definition of Elliptic Curve

6. Elliptic Curves discrete logarithm problem [8]

6.1 Problem definition

Let E be an elliptic curve over some finite field $GF_{(p)}$ and $G = E(F_q)$ be a cyclic additive group, where the group operation is addition modulo p. The elliptic curve discrete logarithm problem is defined as follows**.**

Given P ϵ G and an element Q ϵ<P>, find the integer m, such that Q = [m]P [8] .

ECC relies on a group structure induced on an elliptic curve. A set of points on an elliptic curve together with the point at infinity, denoted ∞ and with point addition as a binary operation has the structure of an abelian group. Here we consider finite fields of characteristic two. A non-super singular elliptic curve E over $F2^n$ is defined as the set of solutions (x, y) $\in$ $F2^n$ x $F2^n$ to the equation:

$y^2+xy=x^3+ax^2+b$ where a, b$\in$ $F2^n$, b$\neq$0, together with ∞.

Suppose p is a prime number finite field *F*p includes

P elements 0,1,2….. p −1

Addition is : a+ b ≡ c mod p (1)

Multiplication is： a* b ≡ c mod p

Law is : a * b^{-1}

Unit element is 1, Zero element is 0. The elliptic curve point is defined as: $e_p(a,b) = \{(x,y)|y^2 = x^3+ax+b \bmod b\}$, such that $(x,y) \in Z_p$

$$Z_p = \{0,1 \ldots., p-1$$

∞, express infinite for point. a, b are no-negative integer less than p

a,b are no-negative integer less than p

Examples

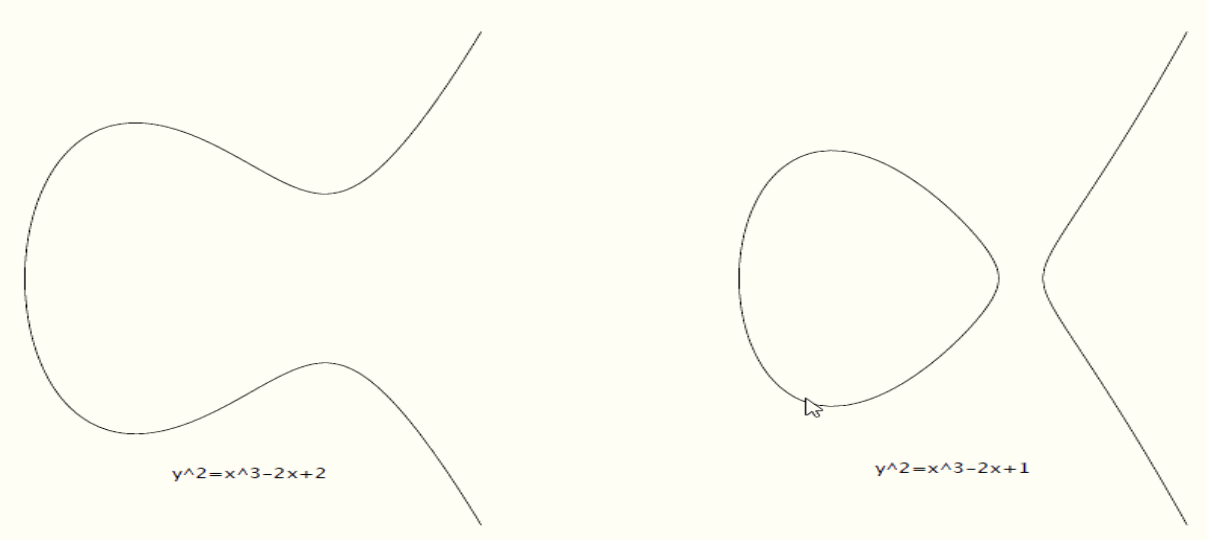

Figure 2.2 elliptic curve example

7. Conclusion

In this chapter, we explained the problems that some public-key cryptosystems are based on, also saw some algorithms to solve these problems. In the next chapter, we will study some public key cryptosystems to prepare their implementation in android mobile phones.

CHAPTER 03

Public-Key Cryptography Systems

1. Introduction

In traditional cryptography, the sender and receiver of a message know and use the same secret key. The sender uses the secret key to encrypt the message, and the receiver uses the same secret key to decrypt the message. This method is known as secret key or symmetric cryptography. The main challenge is getting the sender and receiver to agree on the secret key without anyone else finding out. If they are in separate physical locations, they must trust a courier, a phone system, or some other transmission medium to prevent the disclosure of the secret key. Anyone who overhears or intercepts the key in transit can later read, modify, and forge all messages encrypted or authenticated using that key. The generation, transmission and storage of keys is called key management; all cryptosystems must deal with key management issues. Because all keys in a secret-key cryptosystem must remain secret, secret-key cryptography often has difficulty providing secure key management, especially in open systems with a large number of users [9].

In order to solve the key management problem, Whitfield Diffie and Martin Hellman [DH76] introduced the concept of public-key cryptography in 1976. Public-key cryptosystems have two primary uses, encryption and digital signatures. In their system, each person gets a pair of keys, one called the public key and the other called the private key. The public key is published, while the private key is kept secret. The need for the sender and receiver to share secret information is eliminated, all communications involve only public keys, and no private key is ever transmitted or shared. In this system, it is no longer necessary to trust the security of some means of communications. The only requirement is that public keys be associated with their users in a trusted (authenticated) manner (for instance, in a trusted directory). Anyone can send a confidential message by just using public information, but the message can only be decrypted with a private key, which is in the sole possession of the intended recipient. Furthermore, public-key cryptography can be used not only for privacy (encryption), but also for authentication (digital signatures) and other various techniques,[9].

In this chapter, with the adoption of the mathematical problems explained in the previous, chapter we will studied some public-key cryptography systems like (RSA, elgamal…..).

2. The mathematics of the RSA public-key cryptosystem [11]

RSA is an algorithm for public-key cryptography that is based on the presumed difficulty of factoring large integers, the factoring problem. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first described and publicly the algorithm in 1977, [10].

2.1 Prime Generation and Integer Factorization

Two basic facts and one conjecture in number theory prepare the way for today's RSA public-key cryptosystem.

Fact 1: Prime generation is easy: It is easy to find a random prime number of a given size.

This is a result of two other points: Prime numbers of any size are very common, and it is easy to test whether a number is a prime – even a large prime. To generate a random prime, one can simply generate random numbers of a given size and test them for primarily until a prime is found.

It has not always been easy to test whether a number is a prime. In fact, it might seem that testing for primarily would require one to determine all the factors of the number to see if there are others beside the number itself and 1. Faster methods for primarily testing were discovered in the 1970s that test for certain properties held by prime numbers but not by composites, rather than finding the factors. Without these results, much of public-key cryptography today would not be practical because of the dependence on efficient methods of generating primes.

In the following, let p and q be two large, randomly genera ted primes. "Large" cryptographic context typically means 512 bits (155 decimal digits) or more.

Fact 2. Multiplication is easy: Given p and q, it's easy to find their product, n = p*q.

There are many efficient ways to multiply two large numbers, starting with the "grade-school" method that multiplies one number by the other digit-by-digit, and sums the table of intermediate results.

Conjecture 3. Factoring is hard: Given such an n, it appears to be quite hard to recover the prime factors p and q.

Despite hundreds of years of study of the problem, finding the factors of a large number still takes a long time in general. The fastest current methods are much faster than the simple

approach of trying all possible factors one at a time. (Such a method would take on the order of n steps.) However, they are still expensive. For instance, it has been estimated recently that recovering the prime factors of a 1024-bit number would take a year on a machine costing US $10 million. A 2048-bit number would require several billion times more work.

These estimates are much less than would have been expected in the 1970s when the problem was first proposed in cryptography. The recommended sizes have accordingly increased over the years, due to the discovery of faster factoring methods as well as steady advances in computing power.

No one knows whether faster methods might be discovered in the coming years. On the other hand, no one has proved that they can't be. Both aspects remain important research areas in mathematics.

2.2 Modular Exponentiation and Roots

Given this background, n will hereafter denote the product of two large, randomly generated primes. Let m and c be integers between 0 and n-1, and let e be an odd integer between 3 and n-1 that is relatively prime to p-1 and q-1.

The encryption and decryption operations in the RSA public-key cryptosystem are based on two more facts and one more conjecture:

Fact 1. Modular exponentiation is easy: Given n, m, and e, it's easy to compute

$$c = m^e \bmod n.$$

The value $m^e \bmod n$ is formally the result of multiplying e copies of m, dividing by n, and keeping the remainder. This may seem to be an expensive computation, involving e -1 multiplications by m with increasingly large intermediate results, followed by a division by n. However, two optimizations make the operation easy:

- Multiplying by an appropriate sequence of previous intermediate values, rather than only by m, can reduce the number of multiplications to no more than twice the size of e in binary.
- Dividing and taking the remainder after each multiplication keeps the intermediate results the same size as n.

Fact 2. Modular root extraction – the reverse of modular exponentiation – is easy given the prime factors: Given n, e , c , and the prime factors p and q, it's easy to recover the value m such that $c = m^e \bmod n$.

The value m can be recovered from c by a modular exponentiation operation with another odd integer d between 3 and n-1. In particular, for this d, the following holds for all m:

$$m = (m^e)^d \bmod n$$

This integer d is easy to compute given e , p, and q; see below for details.

Conjecture 3. Modular root extraction is otherwise hard: Given only n, e, and c, but not the prime factors, it appears to be quite hard to recover the value m.

3. The RSA Cryptosystem

The various observations just stated form the basis for the RSA public-key cryptosystem, which was invented at MIT in 1977 by Ronald Rivest, Adi Shamir and Leonard Adleman.

The public key in this cryptosystem consists of the value n, which is called the modulus, and the value e , which is called the public exponent. The private key consists of the modulus n and the value d, which is called the private exponent .

An RSA public-key / private-key pair can be generated by the following steps:

1. Generate a pair of large, random primes p and q.
2. Compute the modulus n as $n = p*q$.
3. Select an odd public exponent e between 3 and n-1 that is relatively prime to p-1 and q-1.
4. Compute the private exponent d from e , p and q. (See below.)
5. Output (n, e) as the public key and (n, d) as the private key

3.1 RSA encryption

The encryption operation in the RSA cryptosystem is exponentiation to the e^{th} power modulo n:

$$\mathbf{c = ENCRYPT\ (m)^e = m \bmod n\ .}$$

The input m is the message; the output c is the resulting cipher text. In practice, the message m is typically some kind of appropriately formatted key to be shared. The actual message is encrypted with the shared key using a traditional encryption algorithm. This construction makes it possible to encrypt a message of any length with only one exponentiation.

3.2 RSA decryption

The decryption operation is exponentiation to the d^{th} power modulo n:

$$\mathbf{m = DECRYPT\ (c, d) = c \bmod n.}$$

The relationship between the exponents e and d ensures that encryption and decryption are inverses, so that the decryption operation recovers the original message m. Without the private key (n, d) (or equivalently the prime factors p and q), it's difficult (by CONJECTURE 6) to recover m from c. Consequently, n and e can be made public without compromising security, which is the basic requirement for a public-key cryptosystem.

3.3 RSA signature

The fact that the encryption and decryption operations are inversed and operated on the same set of inputs also means that the operations can be employed in reverse order to obtain a digital signature scheme following Diffie and Hellman's model. A message can be digitally signed by applying the decryption operation to it, i.e., by exponentiating it to the d^{th}power:

$$\mathbf{s = SIGN\ (m) = m^d \bmod n}$$

3.4 RSA signature verifying

The digital signature can then be verified by applying the encryption operation to it and comparing the result with and/or recovering the message:

$$\mathbf{m = VERIFY\ (s) = s^e \bmod n.}$$

Example 3.1

Key Pair Public key: n = 55, e = 3 Private key: n = 55, d = 7			**Key Pair Generation** Primes: p = 5, q = 11 Modulus: n = p*q = 55 Public exponent: e = 3 Private exponent: d = $3^{-1} \bmod 20 = 7$			
Message	**Encryption** $c = m^3 \bmod n$		**Decryption** $m = c^7 \bmod n$			
M	$m^2 \bmod n$	$m^3 \bmod n$	$c^2 \bmod n$	$C^3 \bmod n$	$c^6 \bmod n$	$c^7 \bmod n$
0	0	0	0	0	0	0
1	1	1	1	1	1	1
2	4	8	9	17	14	2
3	9	27	14	48	49	3
4	16	9	26	14	31	4
5	25	15	5	20	15	5
6	36	51	16	46	26	6
7	49	13	4	52	9	7

Table 3.1 RSA example

4. The Rabin cryptosystem

The Rabin cryptosystem specifications were published in January 1979 by Michael Rabin, in an effort to improve the already existing RSA cryptosystem, by presenting a cryptographic solution whose security was mathematically proven to be based on the difficulty of the integer factorization problem [16].

4.1 The Rabin key generation

Summary: Entity A generates the keys required for the encryption and decryption processes.

1) Generate two random and distinct prime numbers, p and q, of roughly the same size.

2) Compute : n= p*q, n ϵN.

3) The keys are:

The public key: n

the private key:**(p, q)**

4.2 The Rabin encryption

Summary: Entity B encrypts a message m, using the public key n received from entityA.

1) Receive the public key n from A.

2) Express the message plaintext as a number m ϵN, m<n.

3) Compute, c≡m² mod n, cϵ N.

4) Send the cipher text to B.

4.3 The Rabin decryption

Summary: Entity A decrypts the encrypted message c received from B , using the private key (p, q)

1) Receive the cipher text from B.

2) Recover the 4 message plaintexts m_1 ,m_2 ,m_3 ,m_4 satisfying , $m_i^2 \equiv$ c mod n

3) Discern which of the 4 message plaintexts m_1 ,m_2 ,m_3 ,m_4 resembles the original message .m

.**Example 6.1**

- English alphabet enriched with character “space” m ⇒27
- English alphabet enriched with character “space” to Z_{27} correspondence:

-	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Table 3.2 English alphabet.

- N = p.q = 439.631 = 277'009
- redundancy of 2 *decimal* digits
- The initial plaintext $m_i$is split into blocks of length k , while the ciphertext is divided intoblocks of length , fulfilling the conditions:

$$\begin{cases} 27k \prec mi \prec n \prec 27 \\ 27^k \prec m_r = (m_i \,.\, 10^2)(m_i \bmod 10^2) \prec n \prec 27^l \end{cases} \Rightarrow k=2,\ l=1$$

Encryption

Plaintext: rabin

Splitting: ra / bi / n_

Numerical correspondence: 487 63 379

ra → $18.27^1 + 1.27^0 = 487$

bi → $2.27^1 + 9.27^0 = 63$

n_ → $1.27^1 + 0.27^0 = 378$

Adding the replication (2 decimal digits): 48'787 6363 37'878

63 → 6363

378 → 37878

Encryption (using e_k): 110'041 44'455 113'273

$48'787^2 \bmod 277'009 \equiv 110'041 \bmod 277'009$

$6363^2 \bmod 277'009 \equiv 44'455 \bmod 277'009$

$37'878^2 \bmod 277'009 \equiv 113'273 \bmod 277'009$

Cipher text: EOYPBFZMETJH

$110'041 = 5 \,.\, 27^3 + 15 \,.\, 27^2 + 25 \,.\, 27^1 + 16 \,.\, 27^0$→ EOYP

$44'455 = 2 \,.\, 27^3 + 6 \,.\, 27^2 + 26 \,.\, 27^1 + 13 \,.\, 27^0$→ BFZM

$44'455 = 5 \,.\, 27^3 + 20 \,.\, 27^2 + 10 \,.\, 27^1 + 8 \,.\, 27^0$→ ETJH

Decryption

Cipher text: EOYPBFZMETJH

Splitting: EOYP / BFZM / ETJH

Numerical correspondence:

EOYP → $5 \cdot 27^3 + 15 \cdot 27^2 + 25 \cdot 27^1 + 16 \cdot 27^0 = 110'041$

BFZM → $2 \cdot 27^3 + 6 \cdot 27^2 + 26 \cdot 27^1 + 13 \cdot 27^0 = 44'455$

ETJH → $5 \cdot 27^3 + 20 \cdot 27^2 + 10 \cdot 27^1 + 8 \cdot 27^0 = 113'273$

Decryption (using d_k):487 63 378

4 possible roots: {48'787, 111' 887, 165'122, 228'222}mod 277'009 → 48787 is the *only* value to match the replication → 487

4 possible roots: {6363, 44' 117, 232'892, 270'646}mod 277'009 → 6363 is the *only* value to match the replication → 63

4 possible roots:{37'878, 48' 605, 228'404, 239'131}mod 277'009 → 37878 is the *only* value to match the replication → 378

Plain text: rabin

$487 = 18 \cdot 27^1 + 1 \cdot 27^0$→ ra

$63 = 2 \cdot 27^1 + 9 \cdot 27^0$→ bi

$378 = 14 \cdot 27^1 + 0 \cdot 27^0$→ n

5. The El Gamal Cryptosystem

We have seen that the security of the RSA cryptosystem is related to the difficulty of factoring large numbers. It is possible to construct cryptosystems based on other difficult number-theoretic problems. We now consider the El Gamal cryptosystem, named after its inventor, Taher El Gamal, which is based on the difficulty of a problem called the discrete logarithm.

5.1 The Discrete Logarithm

When we are working with the real numbers, $\log_b y$ is the value x, such that $b^x = y$. We can define an analogous discrete logarithm. Given integers b and n, with b<n, the discrete logarithm of an integer y to the base b is an integer x, such that

$$\mathbf{b^x \equiv y \bmod n:}$$

The discrete logarithm is also called index and we write

$$\mathbf{x = ind_{b;n}\, y.}$$

While it is quite efficient to raise numbers to large powers modulo p (recall the repeated squaring algorithm), the inverse computation of the discrete logarithm is much harder. The El-Gamal system relies on the difficulty of this computation,[12].

5.2 El Gamal Encryption

Let p be a prime and g be a generator of Z_p. The private key x is an integer between 1 and p − 2. Let $y = g^x \bmod p$. The public key for El Gamal encryption is the triplet (p, g, y).

If taking discrete logarithms is as difficult as it is widely believed, releasing $y = g^x \bmod p$ does not reveal x.

To encrypt a plaintext M , a random integer k relatively prime to p − 1 is selected, and

the following pair of values is computed:

$$\mathbf{a \leftarrow g^k \bmod p}$$

$$\mathbf{b \leftarrow My^k \bmod p}$$

The ciphertextC consists of the pair (a; b) computed above.

5.3 El Gamal Decryption

The decryption of the cipher text C =(a; b) in the El Gamal scheme, to retrieve the plaintext M ,is simple:

$$\mathbf{M \leftarrow b/a^x \bmod p}$$

In the above expression, the "division" by a^x should be interpreted in the context of modular arithmetic, that is, M is multiplied by the inverse of a^x in Z p. The correctness of the El Gamal encryption scheme is easy to verify. Indeed, we have

$$\mathbf{b/a^x \bmod p = My^k(a^x)^{-1} \bmod p}$$

$$\mathbf{= Mg^{xk}(g^{kx})^{-1} \bmod p}$$

$$\mathbf{= M.}$$

5.4 El Gamal signature

A variation of the above scheme provides a digital signature. Namely, a signature for message M is a pair S =(a; b) obtained by selecting a random integer k relatively prime to p – 1and computing:

$$\mathbf{a \leftarrow g^k \bmod p}$$

$$\mathbf{b \leftarrow k^{-1}(M - xa) \bmod (p - 1)}$$

5.5 El Gamal signature verification

To verify a digital signature S =(a; b), we check that

$$y^a a^b \equiv g^M \pmod{p}$$

Example 4.1: Alice chooses PA = 107, $\alpha_A = 2$, $d_A = 67$, and she computes $\beta_A = 267 \equiv 94 \pmod{107}$. Her public key is $(PA, \alpha_A, \beta A) = (2, 67, 94)$, and her private key is $d_A = 67$.

Bob wants to send the message "B" (66 in ASCII) to Alice. He chooses a random integer $k = 45$ and encrypts $M = 66$ as $(r, t) = (\alpha_A^k, \beta_A^k M) \equiv (2^{45}, 94^{45}66) \equiv (28, 9) \pmod{107}$. He sends the encrypted message (28, 9) to Alice.

Alice receives the message $(r, t) = (28, 9)$, and using her private key $d_A = 67$ she decrypts to $tr^{-dA} = 9 \cdot 28^{-67} \equiv 9 \cdot 28^{106-67} \equiv 9 \cdot 43 \equiv 66 \pmod{107}$.

6. Elliptic Curve Cryptosystem (ECC)

In 1985, Neal Koblitz and Victor Miller independently proposed the concept of elliptic curve cryptography (ECC). It is based on the DLP in a group defined by points on an elliptic curve over a finite field [13].

6.1 Elliptic Curve Cryptography

Elliptic curve encryption was introduced in 1985 by Victor Miller and Neil Koblitz as a different scheme for using public key encryption. Public key encryption generates a method for exchanging keys between numbers of entities in a complicated system. Unlike other common schemes such as RSA, elliptic curve cryptography is relied on discrete a logarithm that is harder to face at the same key size [14]. Also, its key bytes are less than RSA scheme. It can allow computer operation and network broadcast is sound and fast, figure 1 shown the key size comparison.

ECC KEY SIZE (Bits)	RSA KEY SIZE (Bits)	KEY SIZE RATION
163	1024	1 :6
256	3072	1 :12
384	7680	1 :20
		1 :30

Figure 3.1 NIST Key size comparison for public key

In addition, Elliptic curve cryptography needs less bandwidth, less storage space and less computing time, compared with the other schemes. This lets to apply encryption in platforms that are restricted, such as wireless devices, smart cards, and thin-clients. It also gives a large win in states where efficiency is significant [14]. Elliptic curve cryptography is shown in figure 2.

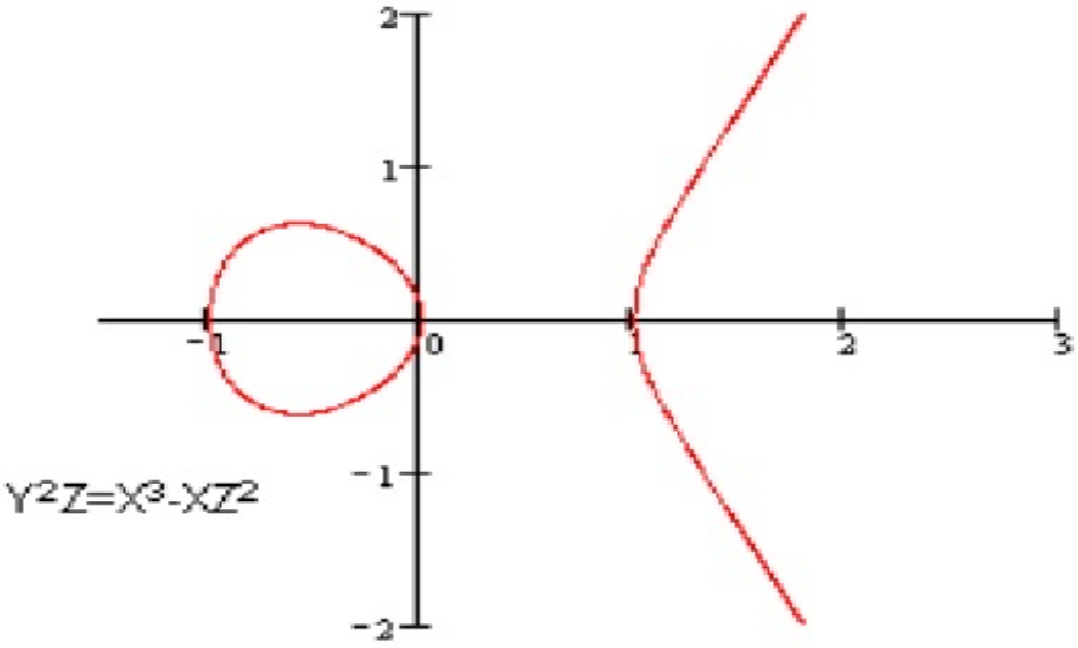

Figure 3.2 Elliptic curve cryptography [14].

6.1.1 EC on Prime field F_p

The equation of the elliptic curve on a prime field Fp is $y^2 \bmod p = x^3 + ax + b \bmod p$, where $4a^3 + 27b^2 \bmod p \neq 0$. Here the elements of the finite field are integers between 0 and p - 1. All the operations such as addition, subtraction, division, multiplication involves integers between 0 and p - 1. This is modular arithmetic and is defined in section Modular Arithmetic. The prime number p is chosen such that there is finitely large number of points on the elliptic curve to make the cryptosystem secure. SEC specifies curves with p ranging between 112-521 bits [15].

6.1.2 EC on Binary field F_{2^m}

The equation of the elliptic curve on a binary field F_{2^m} is $y^2 + xy = x^3 + ax^2 + b$, where $b \neq 0$. Here the elements of the finite field are integers of length at most m bits. These numbers can be considered as a binary polynomial of degree m - 1. In binary polynomial the

coefficients can only be 0 or 1. All operations such as addition, subtraction, division, multiplication involves polynomials of degree m-1 or lesser. The polynomial arithmetic is defined in section Polynomial Arithmetic. The m is chosen such that there is a finitely large number of points on the elliptic curve to make the cryptosystem secure. SEC specifies curves with m ranging between 113-571 bits [15].

6.1.3 ECC encryption/ decryption

Suppose entity A wants to send an encrypted message x to entity B. Thus entity B chooses a large prime p and an integer p mod a. Also, entity B chooses a secret integer i and computes:

$c \equiv a^i mod\ p$ Entity B then makes c, a, p public and keeps i secret. Entity A chooses a random k and computes:

$$y_1 \equiv a^k \bmod p$$

$$y_2 \equiv c^k \bmod p$$

Entity A sends (y1,y2) to entity B, which decrypts by calculating $x \equiv y_1 * {y_1}^{-i}$ mod p, Now we describe the elliptic curve version. Entity B chooses an elliptic curve E mod p where p is a large prime. Entity B chooses a point a on E and a secret integer i .Entity B computes c = a*a (= a + a +...+ a). The points a and c are made public, while i kept secret. Entity A expresses its message as a point *x* on *E* .Then entity *A* chooses a random integer *k* ,computes $y_1 = k * a$ mod p and $y_2 = x + k * c$ then sends the pair (*y*1, *y*2) to Entity *B that it* decrypts by calculating $x = y_2 - a * y_1$.

6.1.4 ECC signature

6.1.4.1 Signing

Entity A needs to sign a message m (which might actually be the hash of a long message). We assume m is an integer. Entity A fixes an Elliptic Curve E mod p where p is a large prime, and a point A on E. Assume that the number of points n on E has been calculated and assume $0 \leq m < n$ (if not, choose a larger p). Entity A also choses a private integer i and computes c = i * A . The prime p the curve E, the integer n, and the points A and c are made public. To sign the message, Entity A does the following:

1. Chooses a random integer k with $1 \leq k < n$, pgcd (k, n) = 1 , and computes R = kA = (x, y)

2. Computes $s \equiv k^{-1}(m - i * x) mod n$

3. Sends the signed message (m, R, s) to entity B

Note that R is a point on E, m and s are integers.

6.1.4.2 Verifying

Entity *B* verifies the signature as follows:

1. Downloads Entity *A* public information *p*, *E*, *n*, *A*, *c*

2. Computes *v1 = x*c + s *R and v2 = m* A*

3. Declares the signature valid if *v1 = v2*

7. The NTRU cryptosystem

The NTRU cryptosystem [17, 18] was presented in 1996 and published in 1998. The security of NTRU is based on two difficult problems, SVP (Shortest Vector Problem) and CVP (Closest Vector Problem). His domain calculation is the ring of polynomials Z [X] = (XN -1) where N is an integer. It consists of two schemes: Scheme of encryption-decryption NTRUEncrypt and NTRU Signature Scheme of signature (NSS). NTRU is now considered reliable by the IEEE P1363.1 standard [14]. Concerning of security, Unlike RSA and Elliptic Curve Cryptography, NTRU is not known to be vulnerable to quantum computer based attacks.

7.1 Mathematical concepts

The NTRU public-key cryptosystem, also known as the NTRU encryption algorithm, is a lattice-based alternative to RSA and ECC and is based on the shortest $R = Z[X]/(X^N - 1)$ vector problem in a lattice (which is not known to be breakable using quantum computers). Operations are based on objects in a truncated polynomial $a = a_0 + a_1X + a_2X^2 + \ldots + a_{N-1}X^{N-1}$ ring with convolution multiplication and all polynomials in the ring have integer coefficients and degree at most *N*-1.

Different descriptions of NTRU and different proposed parameter sets have been in circulation since 1996. However, for the purposes of this overview, we will concentrate on the three most important:

N- the polynomials in the truncated polynomial ring have degree *N*-1.

p- small modulu. p and q are relatively prime.

q- is much larger modulus than p.

In order to ensure security, it is essential that *p* and *q* have no common factors. The following table gives some possible values for NTRU parameters at various security levels.

	N	q	P
Moderate Security	167	128	3
Standard Security	251	128	3
High Security	347	128	3
Highest Security	503	256	3

Table 3.3 NTRU parameters [14]

7.2 NTRU key pair generation

To create a NTRU key, one randomly chooses two small polynomials **f** and **g** in the ring of truncated polynomials *R*. The polynomial f must satisfy the additional requirement that it has an inverse f_p modulo p and an inverse f_q modulo q, that is

$$\mathbf{f}*\mathbf{f}_q = 1 \text{ (modulo } q\text{) and } \mathbf{f}*\mathbf{f}_p = 1 \text{ (modulo } p\text{).}$$

Then the private key is the pair of polynomials **f** and f_p and the public key is the polynomial **h**.

$$\mathbf{h} = p\ \mathbf{f}_q * g \text{ (modulo } q\text{).}$$

We recall that N, p, q are also public.

7.4 NTRU Encryption

To encrypt a message m, firstly put this message in the form of a polynomial m whose coefficients are chosen modulo p, say between -p/2 and p/2 (in other words, **m** is a small polynomial mod *q*). The next one randomly chooses another small polynomial, **r**. This is the "blinding value", which is used to obscure the message.

The cipher text is the polynomial e:

$$\mathbf{e} = \mathbf{r}*\mathbf{h} + \mathbf{m} \text{ (modulo } q\text{).}$$

7.5 NTRU Decryption

To decrypt an encrypted message **e** using the private key **f**, one computes:

$$\mathbf{a} = \mathbf{f}*\mathbf{e} \text{ (modulo } q\text{).}$$

Since a compute *a* modulo *q*, we can choose the coefficients of *a* to lie between *-q/2* and *q/2*. The next one computes the polynomial

b = **a** (modulo *p*).

That is, one reduces each of the coefficients of *a* modulo *p*. Finally, we use his other private polynomial $\mathbf{f}_p$ to compute **c** = $\mathbf{f}_p$***b** (modulo *p*).

Then, the polynomial **c** is the original message **m**.

7.6 NTRU Signature

Signing

$\omega \epsilon \mathcal{F}_\omega$Bob's document is a polynomial mmodulo p. Bob chooses a polynomial. of the form w = m + w_1 + pw_2; where w_1 and w_2 are small polynomials

He then computes: $\delta \equiv \mathcal{F} * \omega \ (modq)$

Bob's signed message is the pair (m; s).

Verification

In order to verify Bob's signature s on the message m, Alice checks that s ≠0 and then verifies the following two conditions:

1) Alice compares s to f_0*m by checking if their deviation satisfies

$$D_{min} \leq Dev(s, f_0 * m) \leq D_{max}$$

2) Alice uses Bob's public verification key h to compute the polynomial $t \equiv h * s \ (mod\ q)$. She then checks if the deviation of t from g_0*m satisfies

$$D_{min} \leq Dev(t, g_0 * m) \leq D_{max}$$

If Bob's signature passes tests 1) and 2), then Alice accepts it as valid.

Example 7.1

- *(N, p, q, d) = (7, 3, 41, 2)*
- Alice sends a message m to Bob:

$$\mathbf{m}(x) = -x5 + x3 + x2 - x + 1$$

- *using the* ephemeral key

$$r(x) = x\text{^}6 - x\text{^}5 + x - 1.$$

$$e = r*h + m \ (modulo\ q)$$

$$e(x) = 31x^6+19x^5+4x^4+2x^3+40x^2+3x+25 \ (mod\ 41)$$

- Step 1: B calculate a: a= f*e(mod q)

$$a = x^6 + 10x^5 + 33x^4 + 40x^3 + 40x^2 + x + 40 \ (mod\ 41).$$

- Step2:B then center lifts modulo *q to obtain:*

$$b = a(mod\ p)$$

$$b = x^6 + 10x^5 - 8x^4 - x^3 - x^2 + x - 1 \ (mod\ 3).$$

- Step3:B reduces a(*x) modulo p and computes :*

$$c = Fp(x)*b(x)$$

$$c = 2x^5 + x^3 + x^2 + 2x + 1 \ (mod\ 3).$$

$$\mathbf{m(x) = -x5 + x3 + x2 - x + 1.}$$

8. Conclusion

We presented in this chapter some public-key cryptography systems by taking in consideration the different processes of them like encryption, decryption and digital signature

In preparation their implementation, testing their performance and analysing the obtained results in android mobiles in the last chapter.

CHAPTER 04

Implementation and experimental results

1. Introduction

In this chapter, we will implement the public-key cryptosystems studied in the previous chapter on different android mobiles, and then we will test the performance of each algorithm, compare our results with related works, and discuss the results.

2. The developpement environnement

2.1 Key Terms [19]

- **SDK** (Software Development Kit) - A set of tools and libraries that allow the user to create an application based on a product.
- **IDE** (Integrated Development Environment) – A software application that consists of a source code editor, A compiler, build automation tools and a debugger. It makes programing and running applications easier.
- **ADT** (Android Development Tools) – A plugin for eclipse that extends the Eclipse IDE by providing more tools to develop Android Applications
- **AVD** (Android Virtual Device) – An Android emulator that allows you to simulate how the application will run on an actual Android device.
- **JDK** (Java SE Development Kit) – A popular Java SDK that is used to program Android applications.

2.2 Eclipse [19]

Eclipse Software Development Kit (Eclipse SDK) is a vast extendable set of tools for software development. Here we are interested in Eclipse's Integrated Development Environment (IDE) component for writing Java software. Eclipse is an open source project of Eclipse Foundation; you can find information about Eclipse Project at http://www.eclipse.org/eclipse. Eclipse is available free of charge under the Eclipse Public License.

Eclipse was developed by software professionals for software professionals; it may seem overwhelming to a novice.

Eclipse runs on multiple platforms including Windows, Linux, and Mac OS. There may be minor differences between Eclipse versions for different platforms and operating systems, but the core features work the same way. Here we will use examples and screen shots from Windows.

2.3 Android [20]

Android, as an operating system, is a Java-based operating system that runs on the Linux 2.6 kernel. The system is very lightweight and full featured. Figure 1-1 shows the unmodified Android home screen.

Figure 4.1 The current Android home screen as seen on the Android Emulator.

Android applications are developed using Java and can be ported rather easily to the new platform, other features of Android include an accelerated 3-D graphics engine (based on hardware support), database support powered by SQLite, and an integrated web browser.

If you are familiar with Java programming or are an OOP developer of any sort, you are likely used to program a user interface (UI), UI placement that is handled directly within the program code. Android, while recognizing and allowing for programmatic UI development, also supports the newer, XML-based UI layout.

XML UI layout is a new concept to the average desktop developer.

One of the most exciting and compelling features of Android is its architecture, third party applications including those that are “home grown” are executed with the same system priority as those that are bundled with the core system.

This is a major departure from most systems, which give embedded system apps a greater execution priority than the thread priority available to apps created by third party developers. In addition, each application is executed within its own thread using a very lightweight virtual machine.

A side from the very generous SDK and the well-formed libraries that are available to us to develop with, the most exciting feature for Android developers is that we now have access to anything the operating system has access to. In other words, if you want to create an application that dials the phone, you have access to the phone’s dialer; if you want to create an application that utilizes the phone’s internal GPS (if equipped), you have access to it. The potential for developers to create dynamic and intriguing applications is now wide open.

On top of all the features that are available from the Android side of the equation, Google has thrown in some very tantalizing features of its own. Developers of Android applications will be able to tie their applications into existing Google offerings such as

Google Maps and the omnipresent Google Search. Suppose you want to write an application that pulls up a Google map of where an incoming call is emanating from, or you want to be able to store common search results with your contacts; the doors of possibility have been flung wide open with Android.

2.4 Installation and Configuration

2.4.1 Download Android SDK

1. Go to http://developer.android.com/sdk/index.html and click on the Windows installer.
2. This should automatically check to see if you have the proper JDK installed. If you do not you can download the newest JDK here: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

2.4.2 Install ADT plugin for eclipse

1. Open Eclipse and go to Help->Install new Software.

2. Enter https://dl-ssl.google.com/android/eclipse/ in the Work with box.

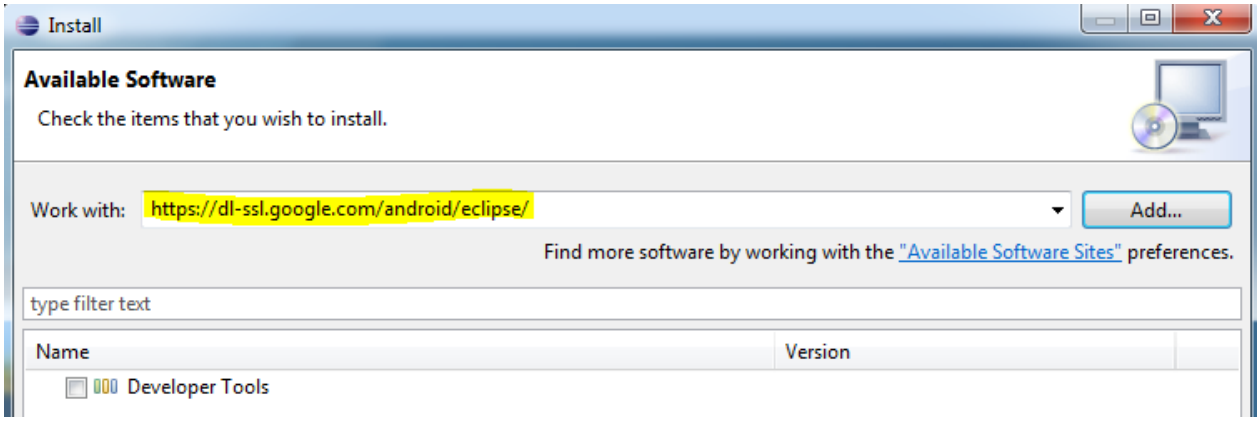

Figure 4.2 Example of options when downloading the SDK.

3. Then put a check mark next to the Developer Tools and click Next.

4. On the next screen it will show all of the tools that will be downloaded. Click Next.

5. Read and agree the license agree to the license agreements then click Finish.

6. Restart Eclipse.

2.4.3 Configure the ADT plugin

After installing the ADT it will need to be configured to point to the SDK directory.

1. Click on Window->Preferences->Android.

2. Click on Browse to find the location of your SDK directory.

3. Click Apply, and then OK.

2.4.4 Adding SDK Components

1. Click on Window->Android SDK Manager.

2. This will allow you to choose the Android platform versions, add-ons tools and other components. Choose the version of Android that you would like you application to work on.

3. Click Install Selected and wait for the components to download. When the download is finished, verify and accept the new components.

2.4.5 Create an AVD

1. In Eclipse go to Window->AVD Manager->New.

2. Type in the Name of the AVD and choose a Target. The target is the version of the Android SDK that you would like to run on the emulator.

3. Click Create AVD.

2.4.6 Create a new Android Application

1. In Eclipse go to File->New->Project…

2. Select an Android Project from the Android Folder and press Next.

3. Fill in the details of your Android application.

   a) Project Name: The project name and folder that Eclipse will store the project files
   b) Build Target: The version of the Android SDK that will be used when you build your program. Select a platform that is equal to or lower than the target chosen for the AVD.
   c) Application Name: This is the name of the application.
   d) Package Name: The namespace that all of the source code will reside under.
   e) Create Activity: The name for that class stub that is generated by the plugin.

4. The values that are used in this example are:

   a) Project Name: MainActivity
   b) Build Target: 2.3.3
   c) Application Name: MainActivity
   d) Package Name: mimoire.sample.example
   e) Create Activity: MainActivity

5. Click on Finish.

2.5 The cryptographic library in JAVA

2.5.1 Introduction

In our application, we used FlexiProvider to implement the deferent public-key cryptosystems schemes like RSA, ECC, El-Gamal…

We chosed FlexiProvider because it is the only provider that fits the android mobile environment and supplies fast and secure implementations of cryptographic algorithms which are easy to use even for developers who are not well-footed in the field of cryptography.

2.5.2 FlexiProvider, [24]

The FlexiProvider is a powerful toolkit for the Java Cryptography Architecture (JCA/JCE). It provides cryptographic modules that can be plugged into every application that is built on top of the JCA.

The FlexiProvider has been developed by the Theoretical Computer Science Research Group of Prof. Dr. Johannes Buchmann at the Department of Computer Science at Technische Universität Darmstadt, Germany.

The FlexiProvider contains four sub libraries: **CoreProvider**, **ECProvider, PQCProvider, NFProvider:**

- **CoreProvider:** The CoreProvider contains well-known public key algorithms such as the RSA cipher and signature in different flavours according to PKCS #1, a multitude of symmetric block ciphers, most prominent among them the AES cipher Rijndael and 3-DES, password-based encryption (PBE) according to PKCS #5, hash functions such as MD5, SHA1, and RIPEMD, MACs such as CBC-MAC, CMAC, HMAC, plus its own pseudo-random number generator (BBS).
- **ECProvider:** The ECProvider is a provider for cryptographic algorithms which are based on elliptic curves. The provider includes the digital signature schemes ECDSA and ECNR, the Elliptic Curve Diffie-Hellman (ECDH) key agreement scheme, and the Elliptic Curve Integrated Encryption Scheme (ECIES). The schemes are implemented according to following standards :

ANSI X9.62-1998:	ECDSA
IEEE 1363-2000 and 1363a-2004:	ECDSA, ECNR, ECDH, ECIES
Certicom SEC 1, SEC 2:	ECDSA, ECDH, ECIES

Table 4.1 Ecc standards.

- **PQCProvider:** The PQCProvider is a provider for cryptographic algorithms which are secure even against quantum adversaries. Most of the algorithms are not standardized, but instead are topic of current research. The provider currently contains the CMSS signature scheme, the McEliece cryptosystem in four variants, the Niederreiter cryptosystem and the CFS signature scheme.

NFProvider: This provider has been developed as a proof of concept for the viability of Number Field Cryptography in imaginary quadratic orders (IQ) by our research group. It contains two DSA variants (IQRDSA and IQDSA) as well as the Guillou-Quisquater signature scheme (IQGQ). We have also devised an ASN.1 module for describing the domain parameters, keys, and signatures.

3. The application characteristics:

3.1 The architecture of the application

Our application is constituted of several interfaces independently of each other, and the main interface is linked with each sub interface.

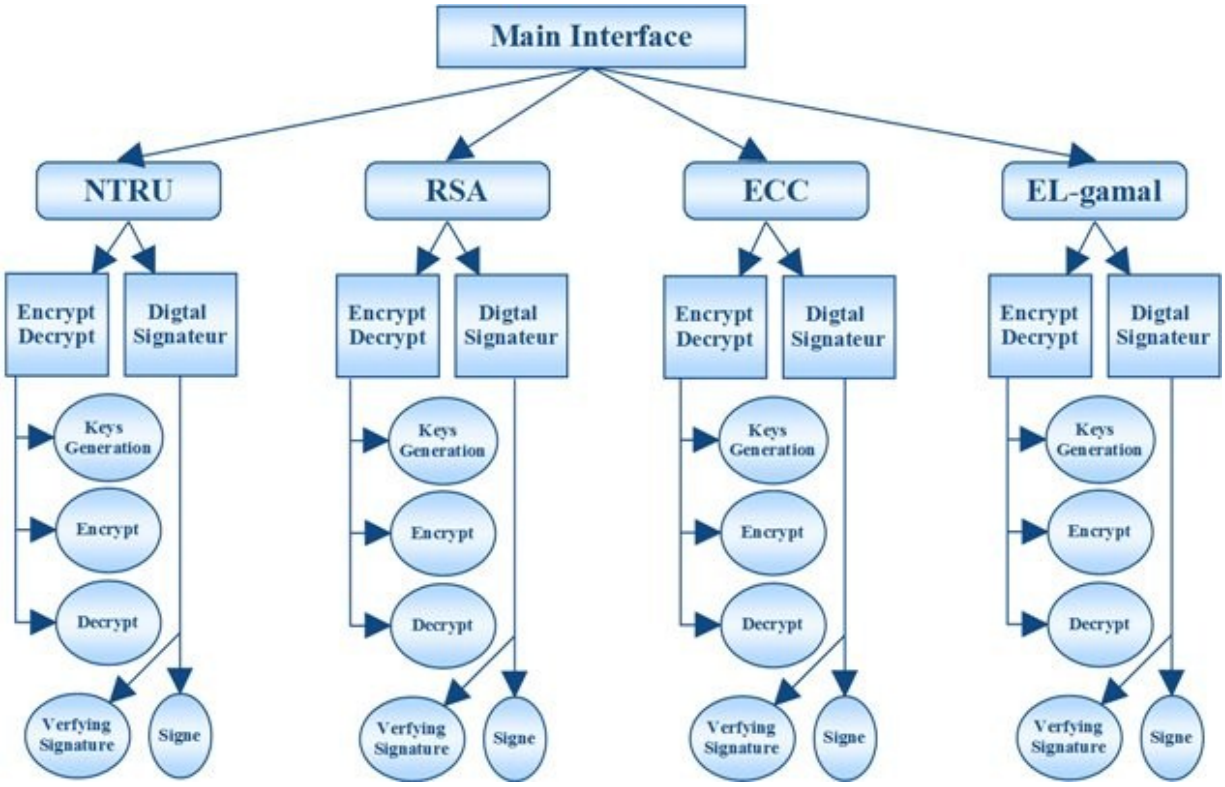

Figure 4.3 The architecture of the application

3.2 The specifications of the application

Our application can just run on android mobile phones with the android version 2.2 or higher, and it consists of 11 java classes with their own xml layout.

3.3 Some application screenshots

The main interface of the application contains the buttons that allows us to access to the different sub interfaces like NTRU, ECC…,

Figure 4.4 The application main interface

If we click for example on ECC button we will browse the following interface:

Figure 4.5 Ecc cryptosystem interface

When I click on any button from the main interface it takes me to the selected cryptosystem interface like ECC in this screenshot, and we find two button the first one is for encryption/decryption and the second one is for the digital signature , if we click on encryption/decryption button we go to the following interface :

Figure 4.6 ECC Encryption Decryption interface.

In this interface we chose the parameters of the cryptosystem, for example in this shot the selected parameters is **secp160r1,** then we specify a an input text in the text field, after this we generate the two key according to the selected parameter, when we generate the keys we can encrypt or decrypt the text by clicking on the specific button, and we can also sign or verify the signature.

4. Implementation and experimental results

4.1 The selected android mobiles

We implemented the ECC, NTRU, RSA, El-Gamal cryptosystems on a number of different platforms of android mobile phones, which are illustrated with their characteristics in the table below:

Device	Model	CPU	Fabrication date	Android version	Internal memory
Samsung	GT-I9100	Dual-Core 1.2 GHz	2011, February	4.2	831 GB
Samsung	GT-S6102	832 MHz	2011, December	2.2	270 MB

Table 4.2 the selected android mobiles

4.2 Choice of parameters

The table below presents different parameters we chosed for ECC, NTRU, RSA, El-Gamal schemes in order to implements them on the mobiles mentioned previously, and we have chosen the parameters since they are on closely spaced of security without taking in consideration the key length:

Parameters	Security	Size (bits)	Algorithm
secp160r1	80	160	**ECC**
secp224k1	112	224	
secp521r1	256	521	
APR2011-439	80	439	**NTRU**
APR2011-743	112	467	
EES1087EP2	256	1087	
P512	80	1024	**RSA**
P2048	112	2048	
P15360	256	15360	
P512	80	512	**El-Gamal**
P2048	112	2048	
P15360	256	15360	

Table 4.3 the selected paramaters [25, 26]

4.3 Calculation of execution time and memory occupation

- To calculate the execution time, we use the method of java.lang.System class:
  - public static long currentTimeMillis()

We can calculate a method executing time by using the following code:

```
long start, finish;
start = System.currentTimeMillis();
appel_Methode();
finish = System.currentTimeMillis();
long duration = finish - start long size = before - after;
```

- To calculate the memory usage, we can use the methods of the class **java.lang.Runtime**:

To find the amount of memory used by an object, we can proceed like this:

```
Runtime runtime = Runtime.getRuntime()
long before, after;
System.gc();
before = runtime.freeMemory();
Object newObject = new String();
after = runtime.freeMemory();
```

4.4 Experimental results

In the implementation of the cryptosystems, we used a fixed clear text, its size is 16 bit, and the registered results is the average of 5 excusions.

4.4.1 The time of execution:

- The time of execution of the ECC cryptosystem on different android mobiles with different parameters are shown in the following table

Device	***GT-S6102***			***GT-I9100***		
Size key Operation	**160**	**224**	**521**	**160**	**224**	**521**
Generation 2 key pairs	661.33	598.66	577	373.2	362	389.33
Encryption	1193.33	1179	1161.33	653.66	622.66	633.33
Decryption	631.33	610	638.33	391.33	400	348.66
ECDSA - Signature	601.33	594	552.33	396.33	377.33	331
ECDSA - Verification	644	619	696	373	339	307

Table 4.4 ECC CRYPTOSYSTEMS EXECUTION TIME (in ms)

- The time of execution of the **NTRU** cryptosystem on different android mobiles with different parameters are shown in the following table

Device	***GT-S6102***			***GT-I9100***		
Size key Operation	**439**	**743**	**1087**	**439**	**743**	**1087**
Generation 2 key pairs	1264.33	2395.33	5043.33	353	708	2384
Encryption	47	40	46.33	50.33	24.33	26
Decryption	180	185.33	189.66	47.66	68.33	101.66
NTRU - Signature	42699	49045	44189	19696	13645.33	14859.66
NTRU - Verification	104	120.33	143.66	79.5	50.66	68.89

Table 4.5 NTRU CRYPTOSYSTEMS EXECUTION TIME (in ms)

- The time of execution of the RSA cryptosystem on different android mobiles with different parameters are shown in the following table

Device	***GT-S6102***			***GT-I9100***		
key Size Operation	**1024**	**2048**	**15360**	**1024**	**2048**	**15360**
Generation 2 key pairs	1088.66	3896	>1h	1150	2341.66	>1h
Encryption	10.66	8	/	6.33	16.33	/
Decryption	12.66	75	/	8.33	91	/
RSA - Signature	14.33	77.66	/	28	95	/
RSA - Verification	1	1	/	0.33	1	/

Table 4.6 RSA CRYPTOSYSTEMS EXECUTION TIME (in ms)

- The time of execution of the ECC cryptosystem on different android mobiles with different parameters are shown in the following table

Device	***GT-I9100***			***GT-S6102***		
Size key Operation	**1024**	**2048**	**15360**	**1024**	**2048**	**15360**
Generation 2 key pairs	>1h	>2h	>6h	>1h	>2h	>6h
Encryption	/	/	/	/	/	/
Decryption	/	/	/	/	/	/
El-Gamal- Signature	/	/	/	/	/	/
El-Gamal Verification	/	/	/	/	/	/

Table 4.7 EL-GAMAL CRYPTOSYSTEMS EXECUTION TIME (in ms)

4.5 Comparisons and performance tests

The comparison between two cryptosystems is to calculate the number of blocks encrypted and decrypted by each one in one second than compare between them, and we avoid El-Gamal cryptosystem because it does not fit the mobile environment

The comparison results between the ECC, RSA and NTRU are illustrated in the following tables:

4.5.1 Mobile GT-I9100

4.5.1.1 NTRU vs ECC

Function	**Unite**	**NTRU**			**ECC**			**comparisons**		
Paramaters		**439**	**743**	**1087**	**160**	**224**	**521**			
Encryption	Bloc/s	19.8	41	38	1.5	1.6	1.5	**19 :1**	**41 :1**	**38 :1**
Decryption	Bloc/s	20.9	14.5	9.8	2.5	2.5	2.8	**8 :1**	**5 :1**	**3 :1**

Table 4.8 NTRU and ECC comparison

4.5.1.2 RSA vs ECC

Function	**Unite**	**RSA**			**ECC**			**comparisons**		
Paramaters		**1024**	**2048**	**16360**	**160**	**224**	**521**			
Encryption	Bloc/s	151.9	61	/	1.5	1.6	1.5	**151 :1**	**61 :1**	/
Decryption	Bloc/s	120	10	/	2.5	2.5	2.8	**48 :1**	**4 :1**	/

Table 4.9 NTRU and ECC comparison

4.5.1.3 NTRU vs RSA

Function	**Unite**	**RSA**			**NTRU**			**comparisons**		
Paramaters		**1024**	**2048**	**16360**	**439**	**743**	**1087**			
Encryption	Bloc/s	151.9	61	/	19.8	41	38	**7 :1**	**1.5 :1**	/
Decryption	Bloc/s	120	10	/	20.9	14.5	9.8	**6 :1**	**1 :2**	/

Table 4.10 NTRU and RSA comparison

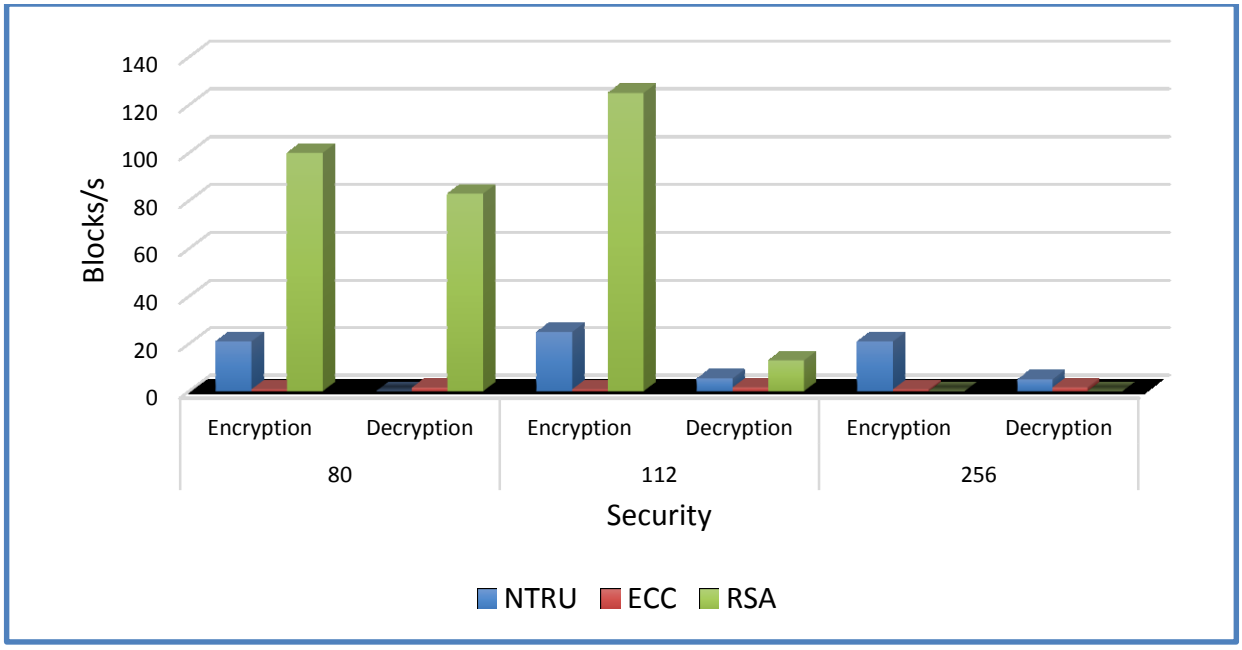

Figure 4.6 Comparison between ECC, RSA, NTRU in Mobile GT-I9100

4.5.2 Mobile GT-S6102

4.5.2.1 NTRU vs ECC

Function	**Unite**	**NTRU**			**ECC**			**comparisons**		
Paramaters		**439**	**743**	**1087**	**160**	**224**	**521**			
Encryption	Bloc/s	21	25	21	0.8	0.8	0.8	**26 :1**	**31 :1**	**26 :1**
Decryption	Bloc/s	5 .5	5.4	5	1.5	1.6	1.5	**3:1**	**3 :1**	**3:1**

Table 4.11 NTRU and ECC comparison

4.5.2.2 RSA vs ECC

Function	**Unite**	**RSA**			**ECC**			**Comparisons**		
Paramaters		**1024**	**2048**	**16360**	**160**	**224**	**521**			
Encryption	Bloc/s	100	125	/	0.8	0.8	0.8	**125 :1**	**156 :1**	/
Decryption	Bloc/s	83	13	/	1.5	1.6	1.5	**55 :1**	**8 :1**	/

Table 4.12 NTRU and ECC comparison

4.5.2.3 NTRU vs RSA

Function	**Unite**	**RSA**			**NTRU**			**Comparisons**		
Paramaters		**1024**	**2048**	**16360**	**439**	**743**	**1087**			
Encryption	Bloc/s	100	125	/	21	25	21	**4 :1**	**5 :1**	
Decryption	Bloc/s	83	13	/	5 .5	5.4	5	**15 :1**	**2 :1**	

Table 4.13 NTRU and RSA comparison

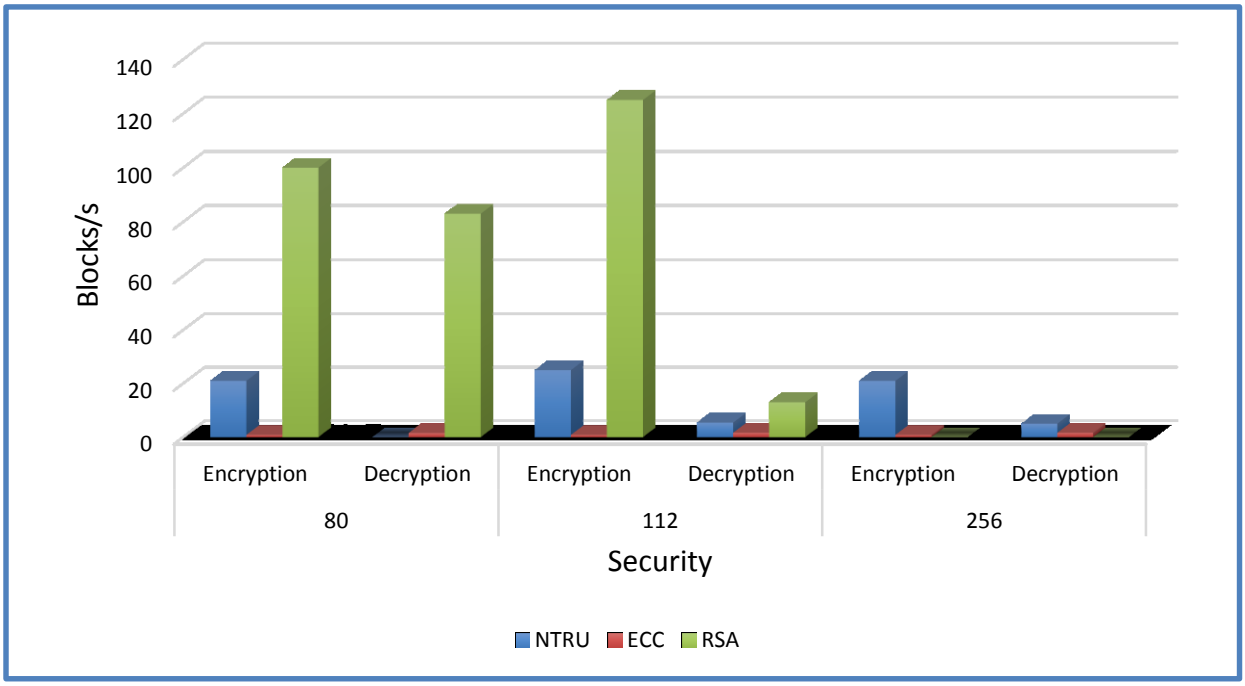

Figure 4.7 Comparison between ECC, RSA, NTRU in Mobile GT-S6102

4.6 Results discussion

The results obtained from (table 4.8) to table (4.13) show clearly that the generation key in ECC cryptosystem is faster than the other cryptosystems in both mobiles and this due to the small key length, in the other hand the encryption and decryption in RSA is faster than the other cryptosystems but the key length of RSA is very large and does not take in consideration the limited resources of mobiles phones, and the encryption/decryption in NTRU is something exciting, in addition it also challenges RSA, moreover the key length is smaller than RSA.

According to the previous results we noticed that El-Gamal cryptosystem is not suitable with the mobile environment because the key generation process takes more than 6 hours in the two mobiles, so we judge that El-Gamal public key cryptosystem does not fit the mobile environment because it does not respect the limited resources, also the other registered observation is that the key generation in RSA takes more than 1 hours when the key length is 15360 bit, so we can say that RSA cryptosystem is suitable for mobile environment in low security requirements cases.

We noticed also that NTRU cryptosystem is very suitable for mobile environment because its key length is very small and its encryption and decryption is very fast, but it has only one drawback which is digital signature because actually it takes a while.

Finally, the ECC stays the ideal cryptosystem for the mobile environment but the encryption and decryption process is slow and this push us to discover a new cryptosystems that give us a maximum security and occupy small resources.

4.7 Related works

Stefan Tillich and Johann Großsch¨adl, [21] this work investigated the feasibility of public-key implementations on modern mid-range to high-end devices, with the focus set on elliptic Curve Cryptography (ECC). They implemented the Elliptic Curve Digital Signature Algorithm (ECDSA) for both signature generation and verification and they showed that both could be done on a J2ME-enabled cell phone depending on the device in times of a few seconds or even under a second. They also compared the performance of ECDSA with RSA signatures and provide some key issues for selecting one protocol type for implementation in a constrained device.

Eric Neidhardt, [22], this work gave the reader a general overview about the application of asymmetric cryptography in communication, particular in mobile devices. The basics principles of a cryptosystem are addressed, as well as the idea of symmetric and asymmetric cryptography. The functional principles of RSA encryption and the Diffie-Hellman key exchange scheme, as well as the general idea of digital signatures are shortly described. Furthermore, some challenges and solution approaches for the application of asymmetric encryption in low power mobile devices are named. On completing, some of the future developments in cryptography are shortly described.

Johann Großsch¨adl, Dan Page, and Stefan Tillich, [23], in this work they introduced a Java implementation of scalar multiplication on a GLV curve over a 174-bit prime field. Their implementation supports arbitrary base points (making it suitable for ECDH key exchange) and is optimized to reach high performance, low memory footprint and small byte code size. Furthermore, the described implementation is highly self-contained as it needs only a few classes from the standard Java class library. On a Think pad T60 with a 1.66 GHz Core Duo processor, their GLV method reaches a throughput of 3,000 scalar multipli-cations per second, which is about 45 times higher than the throughput of the widely-used Bouncy Castle library for J2ME. The scalar multiplication time on mobile phones ranges from less than 2 ms (HTC Desire S) to roughly 400.2 ms (Nokia 6610). They attributed the high performance of their GLV technique to the efficiency of both the field and group arithmetic.

5. Conclusion:

In this chapter, we presented the different development environments and programming language we used it to implement public-key cryptosystems we studied in the previous chapter on android mobile, than we presented our application and we explained its functionalities. After this, we tested the performance of the different procedures of each algorithm in different android mobiles phones we mentioned their characteristics than we registered the results and we compared between the algorithms. Finally, we discussed the obtained results.

GENERAL CONCLUSION

General conclusion

In this dissertation, we showed the security techniques used to secure particularly the channels (wireless and wired) transmission of transaction data. These security techniques have problems and do not ensure the necessary security rules.

The cryptographic techniques are the most interesting tools to encrypt mobile applications data we presented in our dissertation the basic rules of public-key cryptosystems like encryption, decryption, digital signature...

ECC and NTRU Cryptosystem offer equivalent security levels of traditional cryptosystems (RSA, El-Gamal) but with smaller keys. They provide better performance of CPU computing and energy consumption. These features make the ECC and NTRU cryptosystems more suitable for mobile devices and wireless environments.

In the practical part, we used FlexiProvider cryptography library. This Library are required to implement cryptographic techniques in mobile applications, because it is light white library and provides high and effective security for mobile applications. The development of mobile applications is very complicated in apple Smartphone environment (with Visual C + +). The Android platform with java is the best environment for developing mobile application.

We implemented ECC, NTRU, RSA and El-Gamal cryptosystems in different android mobiles, we tested the CPU computing speed and memory occupation of them and we compared between them,

One of our perspectives is to continue working on cryptographic services in the Android environments . Many cryptography works can be made in the Android environments:

- Optimize the algorithms of FlexiProvider library
- Implement a fast and efficient light provider just for ECC and NTRU cryptosystems
- Extend the internet security protocols for mobiles and wireless networks ...

REFERENCES

References

[1] William Stallings, Cryptography and network security principles and practice fifth edition, Copyright © 2011, 2006 Pearson Education, Inc., publishing as Prentice Hall, pages 23, 98.

[2] http://oald8.oxfordlearnersdictionaries.com/ 19/05/2013

[3] Santa Clara, An Introduction to Cryptography, Copyright © 1990-1999 Network Associates, Inc. and its Affiliated Companies, pages 11, 15.

[4] Richard Crandall and Carl Pomerance (2001). Prime Numbers: A Computational Perspective. Springer. ISBN 0-387-94777-9. Chapter 5: Exponential Factoring Algorithms, pp. 191–226. Chapter 6: Subexponential Factoring Algorithms, pp. 227–284. Section 7.4: Elliptic curve method, pp. 301–313.

[5] Donald Knuth. The Art of Computer Programming, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89684-2. Section 4.5.4: Factoring into Primes, pp. 379–417.

[6] Richard Crandall; Carl Pomerance. Chapter 5, Prime Numbers: A computational perspective, 2nd ed., Springer.

[7] Stinson, Douglas Robert (2006), Cryptography: Theory and Practice (3rd ed.), London: CRC Press, ISBN 978-1-58488-508-5

[8] Wendy Chou, Elliptic Curve Cryptography and Its Applications to Mobile Devices. University of Maryland, College Park.

[9] RSA Laboratories' Frequently Asked Questions About Today's Cryptography, v4.0

[10] Menezes, Alfred; van Oorschot, Paul C.; Vanstone, Scott A. (October 1996). Handbook of Applied Cryptography. CRC Press. ISBN 0-8493-8523-7.

[11] Burt Kaliski,The Mathematics of the RSA Public-Key Cryptosystem

[12] Melissa Helgeson, "Security and Applications of ElGamal's Encryption Algorithm", University of Minnesota, Morris 600 E. 4 Th St. Morris, MN 56267 001 (507) 581-3327

[13] Wendy Chou,Elliptic Curve Cryptography and Its Applications to Mobile Devices.University of Maryland, College Park.

[14] Sattar J Aboud,Public Key Cryptography for mobile payment information Technology Advisor Iraqi Council of Representatives Iraq,Baghdad

[15] Tata elxsi ltd, Elliptic curve Cryptography an impelementation tutuorila ,india

[16] Mihnea Rădulescu,PUBLIC-KEY CRYPTOGRAPHY : THE RSA AND THE RABIN CRYPTOSYSTEMS,2008

[17] Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman. NTRU: A Ring Based Public Key Cryptosystem. In Algorithmic Number Theory (ANTS III), Portland, OR, June 1998, J.P. Buhler (ed.), Lecture Notes in Computer Science 1423, Springer-Verlag, Berlin, 1998, 267-288.

[18] J. Hoffstein, J. Silverman, "Optimizations for NTRU. Public-Key Cryptography and Computational Number Theory," DeGruyter, pp. 11–15, 2000.

[19] http://www.developer.android.com, http://www.eclipse.org/downloads/ 19/05/2013

[20] J.F. DiMarzio, "Android™ A Programmer's Guide", Copyright © 2008 by The McGraw-Hill Companies, pages 6, 7.

[21] Stefan Tillich and Johann Großsch¨adl, A Survey of Public-Key Cryptography on J2ME-Enabled Mobile Devices, Graz University of Technology Institute for Applied Information Processing and Communications Inffeldgasse 16a, A–8010 Graz, Austria

[22] Eric Neidhardt, Asymmetric Cryptography for Mobile Devices Telekom Innovation Laboratories and TU Berlin Berlin, Germany.

[23] Johann Großsch¨adl, Dan Page, and Stefan Tillich, Efficient Java Implementation of Elliptic Curve Cryptography for J2ME-Enabled Mobile Devices, University of Luxembourg, CSC Research Unit, LACS,rue Richard Coudenhove-Kalergi, L–1359 Luxembourg, Luxembourg.

[24] http://www.flexiprovider.de/overview.html 24,05,2013

[25] STANDARDS FOR EFFICIENT CRYPTOGRAPHY, SEC 2: Recommended Elliptic Curve Domain Parameters, Certicom Research, http://www.secg.org/collateral/sec2_final.pdf.

[26] JeF Hoffstein, Nicholas Howgrave-Graham, Jill Pipher, Joseph H. Silverman, William Whyte, Performance Improvements and a Baseline Parameter Generation Algorithm for NTRUSign, 5 Burlington Woods, MA 01803.
https://www.securityinnovation.com/uploads/Crypto/NTRUSignParams-2005-08.pdf

الملخص

ان الهدف من هذه المذكرة هو إيجاد افضل أنظمة التشفير فعالية وملائمة لبيئة الهواتف الذكية، من حيث السرعة ودرجة الأمان مع الاخذ بعين الاعتبار الإمكانيات المادية المتوفرة لدى الهواتف الذكية، لذا قمنا بدراسة بعض انظمة التشفير المتوفرة، واستخلاص افضل الأنظمة وتطبيقها على الهواتف التي تستخدم الاندرويد كنظام للتشغيل، علماً ان بعض هذه الانظمة لم تتطبق من قبل في مجال الهواتف النقالة و قمنا بتطبيقها لأول مرة كطريقة (NTRU , El-Gamal) ومن ثم قمنا بعمل دراسة تحليلية لمختلف العمليات كالتشفير وفك التشفير بمختلف الخصائص وبعد مقارنة النتائج تحصلنا على مستوى عال من الحماية و الفعالية.

كلمات دلالييه: الاندرويد، هاتف، أداء، NTRU

Abstract

The objective of this dissertation is to find the most efficient and suitable cryptosystems for intelligent mobiles environments, from the speed and safety aspects, by taking in consideration the limited resources of these mobiles. We studied some cryptosystems and we took out the best of them. Then we implemented them on some android mobile phones. After that, we have made an analytical study for the different operations as encryption and decryption…etc., and after the comparison of the obtained results gave us a high level of security and firmness.

Keys words: NTRU;ECC; Android mobile; performance

Résumé

L'objectif de ce mémoire est de trouver les crypto systèmes les plus efficaces et adaptés pour les environnements mobiles intelligents, en fonction de la vitesse et l'aspect de sécurité, en prenant en considération les ressources limitées de ces mobiles. Nous avons étudié certains crypto systèmes et nous avons pris le meilleur d'entre eux, que nous les avons mis en place sur certains téléphones mobiles Android, Après cela, nous avons fait une étude analytique sur les différentes opérations telles que le chiffrement et le déchiffrement...La comparaison des résultats obtenus nous a permis d'obtenir un niveau élevé de sécurité et de fermeté.

Mottes clés : NTRU ; ECC ; Android mobile ; performance