On Forging ElGamal Signature and Other Attacks

By

Chan Hing Che

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Master of Philosophy

Division of Information Engineering

The Chinese University of Hong Kong

June 2000

Acknowledgement

It is really a challenge for an engineering student to choose cryptography as the M. Phil thesis. I would like to say thanks to my supervisor, Prof. Victor Wei, guiding and helping me to complete this challenge. His knowledge in the cryptography field helps me a lot in the research. I would waste much time in reading those cryptographic papers without his kindly guidance and details explanation. Discussion with Prof. Wei always gives me much inspiration. I have learnt a lot, during these two years of studying, especially in the thinking method.

I would also like to say thanks to Prof. Kwok-wai Cheung and Prof. Kit-ming Yeung for spending their precious time to listen my oral examination.

Many thanks to Jimmy Yeung, Chan Yiu Tong, and Rosanna Chan and the colleagues in the Information Integrity Laboratory. Also thanks to my friends in the Chinese University of Hong Kong. They all give me many unforgettable memories and enjoyable moments in these two years. My family support is very important for me to finish this master study. Thanks to my mother, my father, my brother, my sister and my respectful grandmother.

Abstract

A digital signature is a reliable electronic method of signing electronic documents that provides the recipient with a way to verify the sender, determine that the content of the document has not been altered since it was signed and prevent the sender from repudiating the fact that he or she signed and sent the electronic document.

This thesis mainly discusses the ElGamal signature scheme, especially on the forgery of this signature scheme and its variations. There are some ways to forge an ElGamal signature, without knowing the private key of the signer, if the parameters used in the signature scheme are not carefully chosen. One of this forgery is done by the Bleichenbacher's attack.

The other way of forging signature is to break the discrete logarithm. There are some algorithms to solve the discrete logarithm problem, such as baby-step giant-step, Pollard's ρ, Pohlig Hellman, index-calculus and the number field sieve. This thesis chooses quadratic field in the number field sieve to solve the discrete logarithm problem.

撮要

在日常生活中，我們經常會在文件中，支票上簽名，以証明文件及支票的有效性·但是在電子化世界上，要達到同樣的效果，我們不能單單把簽名掃描到電腦中，然後附於文件上· 因爲這些電腦檔案，我們可以作任意改動·所以在這個電子化的世界上，我們需要一個電子簽署的方法·

現在的電子簽署系統，都是利用公開鑰匙的系統·在公開鑰匙的系統中，每個人都會擁有兩支鑰匙，一支鑰匙是公開鑰匙，另一支是私人鑰匙·公開鑰匙是公開給人知道的，而私人鑰匙就要保密，不能給人知道的·在電子簽署的系統中，我們會用自己的私人鑰匙，利用一些數學程式，來附於文件上·當收件人收到這文件及其電子簽署，會利用寄件人的公開鑰匙，來驗証文件的真確性·

日常生活的簽署，會有被人僞冒的情況出現·電子簽署的情況也是一樣·在這篇論文中，主要是討論 ElGamal 這個電子簽署及其延伸方案·在 Bleichenbacher 一文中，有提及一個僞冒 ElGamal 這個電子簽署的方法，我們會利用這個攻擊方法，來對 ElGamal 的延伸進行類似的攻擊·而且，我們更會討論一些破解 discrete logarithm 的方法·在論文中，我們會提出利用 Quadratic Field NFS 來作破解·

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Every day, people sign their names to contracts, cheques, credit card receipts and other documents, showing that they are the originator of these documents. The signature allows other people to verify that a particular document did indeed originate from the signer. This is the use of the handwritten signature.

In general, a signature should have the following properties:

1. The signature is authentic. The signature convinces the document's recipient that the signer deliberately signed the document.

2. The signature is unforgeable. The signature is proof that the signer, and no one else, deliberately signed the document.

3. The signed document is unalterable. After the document is signed, it cannot be altered without generating new signature.

4. The signature cannot be repudiated. Once the document is signed, the signer cannot later claim that he or she didn't sign it.

How can we implement the handwritten signature in the digital world? Can we just scan the handwritten signature and append it with the document? Of course no! Since the scanned signature is just a computer file, it is trivial to duplicate. Second, it would be easy to cut and paste a scanned image from one document to another document. Third, computer files can be modified after they are 'signed'. Therefore, it is not a good solution.

So, what should we do? Thanks to the advance of cryptography. Before explaining how we can imitate handwritten signature in the digital world, let's have some introduction to cryptography first. In traditional cryptography, if two parties want to communicate in a secure way, both the sender and the receiver agree on a 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 eavesdropper, without knowing the secret key, cannot decrypt the message.

Can we use this symmetric cryptography to imitate handwritten signature, that is, the signer uses the secret key to 'sign' the message and the receiver uses the same key to 'verify' the signature? The answer is no. Since the receiver must know the secret key in order to 'verify' the signature, it means that the receiver can also use this secret key to 'sign' the signature. Therefore, forgery can be easily implemented if we use this scheme. It is still acceptable if this scheme is only used between two trusted parties. However, in the digital world, where we have to sign document to many people, we must prepare different secret keys for different people - it is unacceptable.

Such problem cannot be solved until 1976, Whitfield Diffie and Martin Hellman [8] introduced the concept of public-key cryptography. In the public-key cryptosystem, each person has a pair of keys, one key is called the public key and the other is called the private key. The public key is published, while the private key is kept secret.

But someone will ask, can we easily deduce the private key from the public key? No! Why? In the public key cryptosystem, we can easily compute the public key from the private key (since it is very likely that these two keys are the same), but it is infeasible to compute the private key from the public key. It is something like a one-way function: A one-way function is a mathematical function that it is significantly easier to compute in one direction (the forward direction) than in the opposite direction (the inverse direction). It might be possible, for example, to compute the function in the forward direction in seconds but to compute its inverse could take years, millennia, if at all possible.

How can it be done? It can be done by the well known hard problems in cryptography: integer factorisation, discrete logarithm and elliptic curve discrete logarithm. Here is an example, it is easy to compute 101×103, right? It is 10403. However, if I say, factor the number 10403, can we easily find its factors? Still yes, since 10403 is still a small number, we can find its factors by trial division. However, if I ask you to factor a number that is 100-, 200-digit long, is it still easy to find its factors? No! That is one of the hard problems, called integer factorisation.

With the help of the public key cryptosystem, we can implement a signature scheme in the digital world, called digital signature. The sender uses his private to 'sign' the document, and the receiver uses the corresponding public key to 'verify' the signature.

This thesis will discuss one of the signature schemes, call ElGamal signature scheme, which is a signature scheme based on the hard problem of discrete logarithm. What is this signature scheme about? Let's refer to the figure 1.1. First the signer, say, Alice, applies a hash function to the message, creating a so-called message digest. What is a hash function? A hash function is a mathematical function that takes an input m and returns a fixed-size string, called the hashed value. Often, the input string m is much longer than the hashed value. Given the hashed value, it is infeasible to find another m' such that they have the same hashed value. MD2, MD5 and Secure Hashing Algorithm (SHA) are some well-known hash functions.

So why should we use the hash function? It is because of two reasons, one is the message digest is generally much smaller than the original message; therefore, we can save some bandwidth in the transmission and save some computing power in handling the message afterwards. The other reason is because of the security issues, which will be discussed in chapter 3. Alice then uses her private key, together with the message digest, generates two values r and s by the discrete logarithm. We will call these two values a signature pair afterwards. This is the signing process of the signature scheme. Alice will then send this signature pair, together with the message to the receiver, say, Bob.

When Bob obtains the signature pair and the message, he does the following to

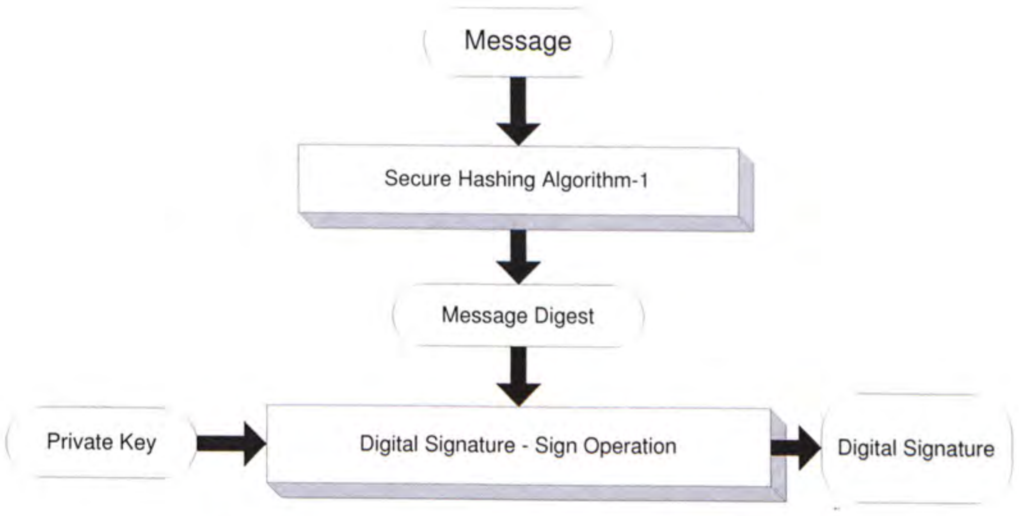
Message
Secure Hashing Algorithm-1
Message Digest
Private Key
Digital Signature - Sign Operation
Digital Signature

Figure 1.1: digital signature signing process

verify the signature. First, he also applies the hash function to the message and gets the value of the message digest. Then, he uses the public key of Alice, the message digest and the signature pair r and s, to compute some values. If these values are valid, then he will accept the signature; otherwise, he will reject the signature. The flow chart is shown in figure 1.2.

Is this signature scheme authentic? Yes, since only Alice has the private key, only she can sign the document with her private key. Deducing the private from the public key is hard, so once we received this signature, we can assure that it is come from Alice. With the same reason, once the message is signed and the signature is generated, Alice cannot deny signing the document. Since the hash function is used in the signing process, one cannot alter the content of the document (since it is infeasible to find another document which can generate the same hash value).

Is this signature scheme unforgeable? In reality, handwritten signature can be

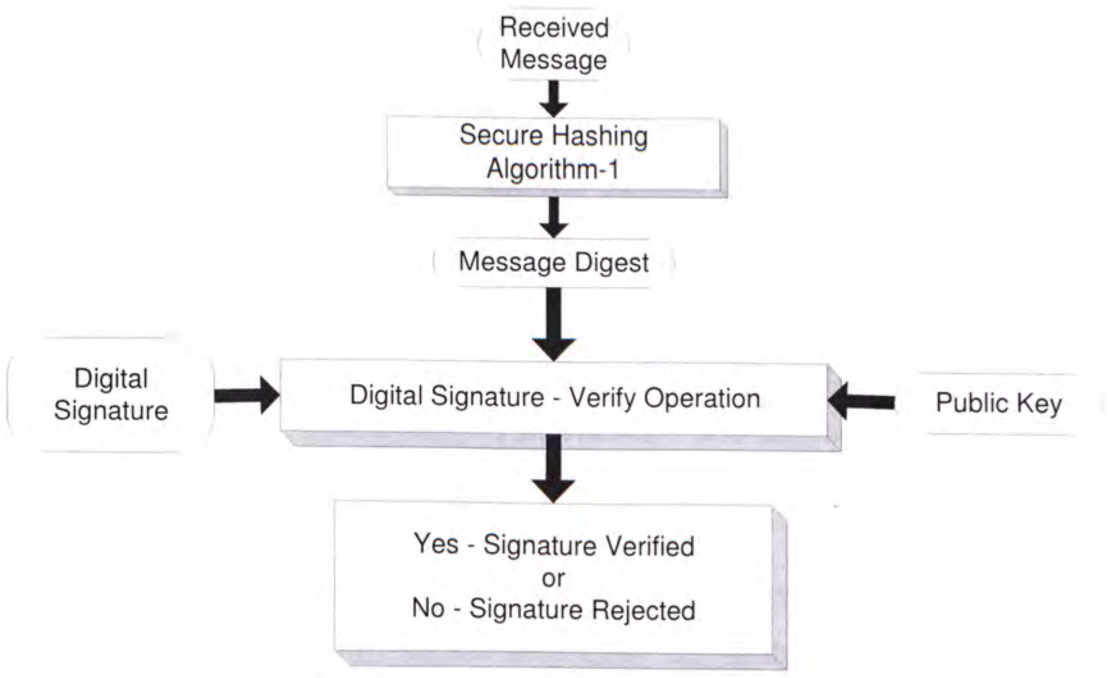

Figure 1.2: digital signature verifying process

forged. Signatures can be lifted from one piece of paper and moved to another, and documents can be altered after signing. In digital signature, forgery can also be possible. One method is to compute the private key from the public key. Yet I have said that the ElGamal signature scheme is based on the hard problem called discrete logarithm, there are still some algorithms to speed up the cracking. However, until now, there are no algorithm can solve this hard problem in a reasonable time. These algorithms will be discussed in chapter 2 and 4. The other method is to forge the signature without knowing the private key. In chapter 3, we will discuss that if the parameters used in the ElGamal signature scheme are not carefully chosen, the signature scheme will be vulnerable to several attacks, one of them is the topic of this thesis - Bleichenbacher's attack. We will extend this Bleichenbacher's attack to other variations of ElGamal signature scheme.

If we choose the parameters carefully, this signature scheme is not likely to be forged. One of the secure signature schemes is Digital Signature Standard. The National Institute of Standards and Technology (NIST), in cooperation with the National Security Agency, proposes DSS which is the digital authentication standard of the U.S. government. This standard will also be discussed in the later part of chapter 3.

What is the use of digital signature? With the digital signature, we can sign the document with our private key and others can easily verify the document with our public key. One of the uses is in email system. Since the 'name' and 'email address' of an email are not authentic, we cannot assure that an email from "Alice" is really come from Alice. With the help of digital signature, once we have received the signature pair and the original message of Alice, we believe that the sender is Alice.

Digital timestamps may be used in connection with digital signatures to bind a document to a particular time of origin. It is not sufficient to just note the date in the message, since dates on computers can be easily manipulated. It is better that timestamping is done by someone everyone trusts, such as a certifying authority.

Chapter 2

Background

This chapter will give the background information necessary for further discussion. Since most of the material discussed in this thesis is about cryptography, and many cryptographic algorithms are based on the theory of abstract algebra, such as group, ring, field. Therefore, the first section will be about abstract algebra.

2.1 Abstract Algebra

Before the introduction of abstract algebra, some terms must be defined (*binary operation, commutativity, associativity*).

Definition 2.1.1 *A binary operation $*$ on a set S is a mapping from $S \times S$ to S.*

Definition 2.1.2 *A binary operation $*$ is commutative if it satisfies*

$$a * b = b * a$$

Definition 2.1.3 *A binary operation $*$ is associative if it satisfies*

$$(a * b) * b = a * (b * c)$$

2.1.1 Group

Group is the basic of the cryptography. Many cryptographic algorithms are based on the group theory. The definition of group and some examples will be given here.

Definition 2.1.4 *A group G is a nonempty set together with a binary operation $*$ which satisfies the following properties:*

- *$*$ is associative (associativity).*
- *There is an element e in G such that $a * e = a$ and $e * a = a$ for every element a in G (the element e is called the identity element of G).*
- *For every element a in G, there is an element x in G such that $a * x = e$ and $x * a = e$ (The element x is called the inverse of a).*

The group just defined may be represented by the symbol $(G, *)$. This notation makes it explicit that the group consists of the set G and the binary operation $*$. If there is no danger of confusion, we shall denote the group simply with the letter G.

Here is an example of a group - the group of integers modulo 6. This group consists of six elements,

$$\{0, 1, 2, 3, 4, 5\}$$

and a binary operation called *addition modulo 6*, +. Table 2.1 shows the operations.

It means that $0+1=1$, $2+3=5$, $4+3=1$. We often use $\mathbb{Z}_6$ to denote this

+	0	1	2	3	4	5
0	0	1	2	3	4	5
1	1	2	3	4	5	0
2	2	3	4	5	0	1
3	3	4	5	0	1	2
4	4	5	0	1	2	3
5	5	0	1	2	3	4

Table 2.1: Operation table for integers modulo 6

group. In general, the set $\mathbb{Z}_n$, with the binary operation of addition modulo n, forms a group. However, the set $\mathbb{Z}_n$, with the binary operation of multiplication modulo n, is not a group.

Definition 2.1.5 *A group G is called an Abelian group if the commutative law holds in the group.*

Definition 2.1.6 *A subgroup H of a group G is a group such that the set H is a subset of G.*

Definition 2.1.7 *A finite group is a group G which contains a finite number of elements.*

2.1.2 Ring

A ring R is a nonempty set together with two binary operations, namely, addition + and multiplication $*$ which satisfies the following properties:

1. R with addition alone is an Abelian group.

2. Multiplication is associative.

3. Multiplication is distributive over addition. That is, for all a,b, and c in R,

$$a * (b + c) = a * b + a * c$$

and

$$(b + c) * a = b * a + c * a$$

Since R with addition alone is an Abelian group, there is in R a neutral element for addition: it is called the zero element and is written 0. Also, every element has an additive inverse called its negative; the negative of a is denoted by $-a$. Subtraction is defined by

$$a - b = a + (-b)$$

2.1.3 Field

Definition 2.1.8 *A field is a nonempty set F together with two binary operations, namely, addition $+$ and multiplication $*$ which satisfies the following properties:*

1. *F with addition alone is an Abelian group*

2. *$F - \{0\}$ with multiplication is an Abelian group, where 0 is the identity element for addition.*

3. *Multiplication is distributive over addition. That is, for all a,b, and c in R,*

$$a * (b + c) = a * b + a * c$$

and

$$(b+c)*a = b*a + c*a$$

The set $\mathbb{Z}_p$ under the usual operations of modulo addition and modulo multiplication forms a field, provided that p is a prime number .

Definition 2.1.9 *A finite field is a field F which contains a finite number of elements.*

2.1.4 Useful Theorems in Number Theory

There are two useful theorems which are frequently used in number theory: they are the Chinese Remainder Theorem and Euler's Theorem.

Chinese Remainder Theorem Let $n_1, n_2, \ldots, n_k$ be pairwise relatively prime integers, then the system of simultaneous congruences

$$\begin{aligned} x &= a_1 \pmod{n_1} \\ x &= a_2 \pmod{n_2} \\ &\vdots \\ x &= a_k \pmod{n_k} \end{aligned} \tag{2.1}$$

has a unique solution modulo $n = n_1 n_2 \ldots n_k$. The solution x to the simultaneous congruences can be computed as

$$x = \sum_{i=1}^{k} a_i N_i M_i \pmod{n} \tag{2.2}$$

where

$$N_i = n/n_i \tag{2.3}$$

and

$$M_i = N_i^{-1} \pmod{n_i} \tag{2.4}$$

Euler's Theorem

Theorem 2.1.1 *If a and m are integers such that $\gcd(a, m) = 1$, then $a^{\phi(m)} = 1$ (mod m), where $\phi(m)$ is the Euler's ϕ-function.*

Corollary 2.1.1 *If p is prime, then the number of invertible elements in the complete residue system modulo p is $p-1$, i.e., $\phi(p) = p-1$. Therefore, for any $a \neq 0$ (mod p), $a^{p-1} = 1$ (mod p).*

2.2 Discrete Logarithm

Discrete logarithm is one of the hard problems in the cryptography field. It is difficult to solve because until now, there is no polynomial running time algorithm to solve the discrete logarithm problem. Many algorithms (will be discussed later) runs sub-exponential time. They are not as fast as polynomial time algorithms, yet they are considerably faster than exponential time methods.

Definition 2.2.1 *If G is a finite group, the order of G is the number of elements in G.*

Definition 2.2.2 *A cyclic group G of order n is a group defined by an element α (the generator), where the powers of the generator (α, α^2, ..., α^n) are unique.*

Definition 2.2.3 *Let G be a finite cyclic group of order n, α be the generator of G, and $\beta \in G$, the discrete logarithm problem is to find an integer x, $0 \leq x \leq n-1$, such that $\alpha^x = \beta$.*

In general, a discrete logarithm problem involves a prime number p and a generator α. The group used is a multiplication group $\mathbb{Z}_p^*$. Then the discrete logarithm problem becomes:

> Given p, α and β, find an x such that $\beta = \alpha^x \pmod{p}$ or solve the discrete logarithm $\log_\alpha \beta$.

2.3 Solving Discrete Logarithm

This section will discuss some methods to solve the discrete logarithm problem.

2.3.1 Exhaustive Search

It is the naive method to solve the discrete logarithm. It tries all the possibilities of x until we find one that $\alpha^x = \beta \pmod{p}$.

Algorithm Compute α^0, α^1, α^2, until we find an i such that $\alpha^i = \beta \pmod{p}$.

Example The element $\alpha = 19$ is a generator of the group G of $\mathbb{Z}_{191}^*$. Find x such that $19^x = 82 \pmod{191}$.

Solution We try $i = 0, 1 \ldots$ and finally we find that $19^{19} = 82 \pmod{191}$. Therefore, $x = 19$. (The intermediate steps are shown in table 2.2)

Since this method searches all the possibilities of x, it takes $O(n)$ multiplications. Owing to its complexity, it is only useful for a small n only.

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
19^i	1	19	170	174	59	166	98	143	43	53	52	33	54	71	12	37	130	178	135	82

Table 2.2: Intermediate steps of exhaustive search algorithm

2.3.2 Baby Step Giant Step

The baby-step giant-step algorithm makes use of a time-memory trade-off to search an interval of length n for a discrete logarithm using only $O(\sqrt{n})$ operations.

The idea of baby-step giant-step is: Let $m = \lceil \sqrt{n} \rceil$ where n is the order of α. We can write $x = im + j$, where $0 \leq i, j < m$. Hence, $\alpha^x = \alpha^{im}\alpha^j$, which implies $\beta(\alpha^{-m})^i = \alpha^j$.

Algorithm

1. Set $m = \sqrt{n}$.
2. Construct a table with entries (j, α^j) for $0 \leq j < m$.
3. For $i = 0$ to $m - 1$,
   (a) Set $\gamma = \beta(\alpha^{-m})^i$.
   (b) If $\gamma = \alpha^j$, then break and output $x = im + j$.
   (c) Else continue.

In step 3(b), we will compare the value of γ with α^j. To facilitate the comparison, it is better to sort the column α^j. Roughly speaking, it takes $O(\sqrt{n})$ storage and $O(\sqrt{n})$ operations.

Example Use the same example as section 2.3.1.

Solution

1. $m = \lceil \sqrt{n} \rceil = 14$

2. Construct the table as in table 2.3.

i	0	1	2	3	4	5	6	7
19^i	1	19	170	174	59	166	98	143
i	8	9	10	11	12	13	14	
19^i	43	53	52	33	54	71	12	

Table 2.3: Intermediate steps of baby-step giant-step algorithm

3. Then we compute $\alpha^{-14} = 16 \pmod{191}$. We try i=0,1,... and obtain $\gamma = 82 * 16^1 = 166 \pmod{191}$ where $166 = 19^5$. Therefore, we can calculate x by $1 * 14 + 5 = 19$. Thus x is solved.

2.3.3 Pollard's rho

Pollard's rho algorithm [20] is the best collision search algorithm. It gets its name because the algorithm produces a trail of numbers that when graphically represented with a line connecting successive elements, the trail looks like the Greek letter rho, ρ. The objective of this algorithm is to find where the tail meets the loop.

Let $f(x)$ be a polynomial with integer coefficients. Starting from $x_0 = 1$, define a sequence of group elements x_0, x_1, x_2, … where

$$x_{i+1} = f(x_i) = \begin{cases} \beta \cdot x_i, & \text{if } x_i \in S_1, \\ x_i^2, & \text{if } x_i \in S_2, \\ \alpha \cdot x_i, & \text{if } x_i \in S_3 \end{cases} \tag{2.5}$$

for $i \geq 0$. The set S_1, S_2 and S_3 are chosen such that they are of roughly equal size.

This sequence in turn defines two sequences of integers a_0, a_1, a_2, … and b_0, b_1, b_2, … satisfying $x_i = \alpha^{a_i}\beta^{b_i}$ for $i \geq 0$, where $a_0 = 0$ and $b_0 = 0$:

$$a_{i+1} = \begin{cases} a_i, & \text{if } x_i \in S_1, \\ 2a_i \bmod n, & \text{if } x_i \in S_2 \\ a_i + 1 \bmod n, & \text{if } x_i \in S_3 \end{cases} \tag{2.6}$$

and

$$b_{i+1} = \begin{cases} b_i + 1 \bmod n, & \text{if } x_i \in S_1 \\ 2b_i \bmod n, & \text{if } x_i \in S_2 \\ b_i, & \text{if } x_i \in S_3, \end{cases} \tag{2.7}$$

where n is the order of the group. When an x_i is computed, it is compared with the other sequence elements x_j (for $j = 0, 1, \ldots, i-1$). If a collision is found, i.e., there exists a j such that $x_i = x_j$, we can probably solve the discrete logarithm problem. Since if we find a pair i and j such that $x_i = x_j$, then $\alpha^{a_i}\beta^{b_i} = \alpha^{a_j}\beta^{b_j}$. Taking logarithm to base α, we have

$$\begin{aligned} a_i + b_i \log_\alpha \beta &= a_j + b_j \log_\alpha \beta \pmod n \\ (b_i - b_j)\log_\alpha \beta &= a_j - a_i \pmod n \\ \log_\alpha \beta &= \frac{a_j - a_i}{b_i - b_j} \pmod n \end{aligned} \tag{2.8}$$

It is not practical to keep track of all values of x_i. The Pollard's ρ method employs a clever procedure to detect a collision, called Floyd's cycle detecting algorithm. Instead of finding a collision of x_i and x_j, we find two elements x_i and x_{2i}. If we find $x_i = x_{2i}$, we can follow the similar argument as above and solve the discrete logarithm of $\log_\alpha \beta$, where

$$\log_\alpha \beta = \frac{a_{2i} - a_i}{b_i - b_{2i}} \pmod{n} \tag{2.9}$$

Example The element $\alpha = 19$ is a generator of the subgroup G of $\mathbb{Z}_{383}^*$ of order $n = 191$. The discrete logarithm problem is to find an x such that $\alpha^x = 336 \pmod{383}$. First, we have $\alpha = 19$, $\beta = 336$, $n = 191$. Then we partition the set G into three subsets S_1, S_2, S_3 according to this rule:

$$\begin{aligned} x \in S_1 &\quad \text{if} \quad x \equiv 1 \pmod{3} \\ x \in S_2 &\quad \text{if} \quad x \equiv 2 \pmod{3} \\ x \in S_3 &\quad \text{if} \quad x \equiv 3 \pmod{3} \end{aligned} \tag{2.10}$$

Using the Pollard's ρ algorithm, and setting $x_0 = 1$, $a_0 = 0$ and $b_0 = 0$, we obtain table (2.4):

Since we find a collision when $i = 11$, where $a_{11} = 4$, $b_{11} = 18$, $a_{22} = 89$, $b_{22} = 104$. Then

$$\begin{aligned} \log_{19} 336 &= \frac{89 - 4}{18 - 104} \pmod{191} \\ &= 19 \pmod{191} \end{aligned} \tag{2.11}$$

2.3.4 Pohlig-Hellman

Pohlig-Hellman algorithm [19] takes the advantage of the prime factorisation of $p - 1$. This algorithm is best understood by first considering the special case

i	x_i	a_i	b_i	x_{2i}	a_{2i}	b_{2i}
1	336	0	1	294	0	2
2	294	0	2	330	0	8
3	261	0	4	134	1	16
4	330	0	8	116	3	16
5	128	0	16	205	4	17
6	134	1	16	9	5	18
7	248	2	16	50	20	72
8	116	3	16	161	21	73
9	289	4	16	25	44	146
10	205	4	17	293	88	103
11	323	4	18	323	89	104

Table 2.4: Intermediate steps of Pollard's ρ method

when $p = 2^n + 1$.

For $p = 2^n + 1$

This algorithm is to find the binary expansion of x $(b_0, b_1, \ldots, b_{n-1})$

$$x = \sum_{i=0}^{n-1} b_i 2^i$$

The least significant bit b_0 of x is determined by raising β to the $(p-1)/2 = 2^{n-1}$ power and applying the rule

$$\beta^{(p-1)/2} = \begin{cases} +1, & b_0 = 0 \\ -1, & b_0 = 1 \end{cases} \pmod p \tag{2.12}$$

This fact is established by noting that, since α is primitive,

$$\alpha^{(p-1)/2} = -1 \pmod p \tag{2.13}$$

and therefore,

$$\beta^{(p-1)/2} = (\alpha^x)^{(p-1)/2} = (-1)^x \pmod p \tag{2.14}$$

If x is divisible by two, i.e., $b_0 = 0$, then (2.14) gives $+1$; otherwise, it gives -1. The next bit in the expansion of x is then determined by letting

$$\gamma = \beta\alpha^{-b_0} = \alpha^{x_1} \pmod{p} \tag{2.15}$$

where

$$x_1 = \sum_{i=1}^{n-1} b_i 2^i \tag{2.16}$$

To find the next bit b_1, we raise γ to $(p-1)/4 = 2^{n-2}$ power and applying the rule

$$\gamma^{(p-1)/4} = \begin{cases} +1, & b_1 = 0 \\ -1, & b_1 = 1 \end{cases} \pmod{p} \tag{2.17}$$

Reasoning as before, if x is divisible by four, i.e., $b_1 = 0$, then (2.17) will give $+1$; otherwise, it will give -1.

In general, to find the bit b_i, we first must have

$$\gamma = \alpha^{x_i} \pmod{p} \tag{2.18}$$

where x_i is

$$x_i = \sum_{j=i}^{n-1} b_j 2^j \tag{2.19}$$

Then we raise γ to the $m-th$ power where

$$m = \frac{p-1}{2^{i+1}} \tag{2.20}$$

and apply the rule

$$\gamma^m = \begin{cases} +1, & b_i = 0 \\ -1, & b_i = 1 \end{cases} \pmod{p} \tag{2.21}$$

For arbitrary primes

Let the prime factorisation of $p-1$ is

$$p-1 = p_1^{n_1} p_2^{n_2} \dots p_k^{n_k}, \quad p_i < p_{i+1} \tag{2.22}$$

The algorithm is to find the value of $x \pmod{p_i^{n_i}}$ for $i = 1, 2, \dots, k$ and compute x via the Chinese Remainder Theorem.

Consider the following expansion of $x \pmod{p_i^{n_i}}$:

$$x = \sum_{j=0}^{n_i-1} b_j p_i^j \tag{2.23}$$

where $0 \le b_j \le p_i - 1$.

The least significant coefficient, b_0, is determined by raising β to the $(p-1)/p_i$ power,

$$\begin{aligned} \beta^{(p-1)/p_i} &= \alpha^{(p-1)x/p_i} \pmod{p} \\ &= \gamma_i^x \pmod{p} \\ &= \gamma_i^{b_0} \pmod{p} \end{aligned} \tag{2.24}$$

where

$$\gamma_i = \alpha^{(p-1)/p_i} \tag{2.25}$$

is a primitive p_i-th root of unity. There are therefore only p_i possible values for $\beta^{(p-1)/p_i} \pmod{p}$, and the resultant value uniquely determines b_0.

To determine the next digit b_1 in the base p_i expansion of $x \pmod{p_i^{n_i}}$, we must first have

$$\zeta = \beta\alpha^{-b_0} = \alpha^{x_1} \pmod{p} \tag{2.26}$$

where x_1 is

$$x_1 = \sum_{j=1}^{n_i-1} b_j p_i^j. \tag{2.27}$$

and then raise ζ to the $(p-1)/p_i^2$ power

$$\begin{aligned} \zeta^{(p-1)/p_i^2} &= \alpha^{(p-1)x_i/p_i^2} \pmod p \\ &= \gamma_i^{x_i/p_i} \pmod p \\ &= \gamma_i^{b_i} \pmod p \end{aligned} \tag{2.28}$$

Again, there are only p_i possible values of $\zeta^{(p-1)/p_i^2}$ and this valid determines b_1. This process is continued to determine all the coefficients b_j.

Example

1. The prime factorisation of $p - 1 = 190$ is $2 \cdot 5 \cdot 19$

2. (a) Compute $x_1 = x \pmod 2$

$$\begin{aligned} \alpha^{95} &= 19^{95} \pmod{191} \\ &= 190 \pmod{191} \\ \beta^{95} &= 82^{95} \pmod{191} \\ &= 190 \pmod{191} \end{aligned} \tag{2.29}$$

Therefore, the coefficient $b_0 = 1$. Then $x_1 = 1 \pmod 2$.

(b) Compute $x_2 = x \pmod 5$

$$\begin{aligned} \alpha^{38} &= 19^{38} \pmod{191} \\ &= 39 \pmod{191} \\ \beta^{38} &= 82^{38} \pmod{191} \\ &= 49 \pmod{191} \end{aligned} \tag{2.30}$$

We use extensive search method to find the coefficient $b_0 = 4$. Therefore, $x_2 = 4 \pmod 5$)

(c) Compute $x_3 = x \pmod{19}$

$$\begin{aligned} \alpha^{10} &= 19^{10} \pmod{191} \\ &= 52 \pmod{191} \\ \beta^{10} &= 82^{10} \pmod{191} \\ &= 1 \pmod{191} \end{aligned} \tag{2.31}$$

We use extensive search method to find the coefficient $b_0 = 0$. Therefore, $x_3 = 0 \pmod{19}$

(d) Now, we have

$$\begin{aligned} x &= 1 \pmod 2 \\ x &= 4 \pmod 5 \\ x &= 0 \pmod{19} \end{aligned} \tag{2.32}$$

Using Chinese Remainder Theorem, we find that $x = 19$.

2.3.5 Index Calculus

Algorithm

1. Choose a subset $S = \{p_1, p_2, \ldots, p_t\}$ of G

2. (a) Select a random number k, $0 \leq k \leq n-1$, and compute α^k

   (b) Try to write α^k as a product of elements in S:

$$\alpha^k = \prod_{i=1}^{t} p_i^{c_i}, \quad c_i \geq 0$$

If successful, take logarithms of both sides of equation to obtain a linear relation

$$k = \sum_{i=1}^{t} c_i \log_\alpha p_i \pmod{p-1}.$$

(c) Repeat these steps until there are enough relations.

3. Solve the above linear system and obtain $\log_\alpha p_i$.

4. (a) Select a random number k, $0 \leq k \leq p-1$, and compute $\beta \cdot \alpha^k$.

   (b) Try to write $\beta \cdot \alpha^k$ as a product of elements in S. So,

$$\beta \cdot \alpha^k = \prod_{i=1}^{t} p_i^{d_i}, \quad d_i \geq 0. \tag{2.33}$$

   Repeat 4a until the attempt is successful. Then

$$\log_\alpha \beta = \sum_{i=1}^{t} d_i \log_\alpha p_i - k \pmod{p} \tag{2.34}$$

Example Use the same example as in section 2.3.1,

1. First, the factor base chosen is $S = \{2, 3, 5, 7\}$

2. We randomly generate k and obtain the following relations

$$\begin{aligned}
19^{20} \bmod 191 &= 2 \cdot 3 \cdot 5 \\
19^{12} \bmod 191 &= 2 \cdot 3^3 \\
19^{92} \bmod 191 &= 3^2 \cdot 5 \\
19^{108} \bmod 191 &= 2^3 \cdot 3 \cdot 5 \\
19^{141} \bmod 191 &= 2 \cdot 3 \cdot 7
\end{aligned} \tag{2.35}$$

3. Taking the logarithms of both sides,

$$\begin{aligned}
20 &= \log 2 + \log 3 + \log 5 \pmod{190} \\
12 &= \log 2 + 3\log 3 \pmod{190} \\
92 &= 2\log 3 + \log 5 \pmod{190} \\
108 &= 3\log 2 + \log 3 + \log 5 \pmod{190} \\
141 &= \log 2 + \log 3 + \log 7 \pmod{190} \qquad (2.36)
\end{aligned}$$

4. Then solve the system of linear equations with four unknowns

$$\begin{aligned}
\log 2 &= 44 \\
\log 3 &= 116 \\
\log 5 &= 50 \\
\log 7 &= 171 \qquad (2.37)
\end{aligned}$$

5. Suppose that the integer $k = 65$ is selected. Since

$$\begin{aligned}
\beta \cdot \alpha^k &= 82 \cdot 19^{65} \pmod{191} \\
&= 81 \pmod{191} \\
&= 3^4 \pmod{191} \qquad (2.38)
\end{aligned}$$

6. It follows that

$$\begin{aligned}
\log_{19} 82 + 65 &= 4\log_{19} 3 \pmod{190} \\
\log_{19} 82 &= 4 \cdot 116 - 65 \pmod{190} \\
&= 19 \qquad (2.39)
\end{aligned}$$

Chapter 3

Forging ElGamal Signature

This chapter will first discuss the original ElGamal signature schemes and analyze the security of this signature scheme. In [3], it shows that there are some security risks in the ElGamal signature scheme if the parameters are not carefully chosen. In this case, forgery is possible even that the adversary doesn't know the private key of the signer. I will extend such type of forgery to the variations of the ElGamal signature scheme. The last part will discuss a variant of the ElGamal signature scheme that is the digital authentication standard adopted by the U.S. government - it is Digital Signature Standard (DSS)

3.1 ElGamal Signature Scheme

ElGamal signature scheme is based on the discrete logarithm problem (which is discussed in chapter 2). This scheme mainly involves four steps: 1. Choose the public parameters p and α, these parameters can be shared among several users. 2. Generate the key pair (private key and public key) of the signer. 3. Given a

message m, generate a pair (r, s), which is called the signature on message m.
4. Verify the signature pair on the message.

Public Parameters

1. Choose a large prime number p, and the multiplicative group $\mathbb{Z}_p^*$, where the group operation is multiplication modulo p.
2. Choose a generator α of the multiplicative group $\mathbb{Z}_p^*$.
3. Publish p and α.

Key Generation

1. Randomly generate an integer x, where $1 \leq x \leq p-2$.
2. Compute $y = \alpha^x \pmod{p}$.
3. Private key: x

   Public key: y.

Signature Generation

1. Select a random number k, $1 \leq k \leq p-2$, such that $\gcd(k, p-1) = 1$.
2. Compute $r = \alpha^k \pmod{p}$.
3. Compute $s = k^{-1}(m - rx) \pmod{p-1}$.
4. Signature pair (r, s).

Signature Verification

1. Verify that $1 \leq r \leq p-1$; if not, reject the signature.
2. Compute $v_1 = y^r r^s \pmod{p}$.

3. Compute $v_2 = \alpha^m \pmod{p}$.
4. If $v_1 = v_2$, accept the signature; otherwise, reject the signature.

In chapter 1, we have mentioned the model of digital signature. Suppose Alice wants to send a message to Bob. Alice uses her private key to sign the message (or message digest) and Bob uses Alice's public key to verify the signature pair.

The goal of an adversary is to forge signatures; that is, to produce signatures which will be accepted as those of some other entity. In general, we can categorize the type of forgery as follows:

1. *total break*. An adversary is either able to compute the private key information of the signer, or finds an efficient signing algorithm functionally equivalent to the valid signing algorithm.
2. *selective forgery*. An adversary is able to create a valid signature for a particular message or class of messages chosen a priori. Creating the signature does not directly involve the legitimate signer.
3. *existential forgery*. An adversary is able to forge a signature for at least one message. The adversary has little or no control over the message whose signature is obtained, and the legitimate signer may be involved in the deception.

Total break in ElGamal signature scheme equals to the solving of discrete logarithm problem. Some algorithms (baby-step giant step, Pollard's ρ, Pohlig-Hellman and index-calculus) have been covered in chapter 2. Chapter 4 will discuss another method, which can also solve the discrete logarithm.

Selective forgery is possible in the ElGamal signature scheme. These cases are now discussed.

3.2 ElGamal signature without hash function

The ElGamal signature scheme presented in the previous section doesn't employ any hash function on the message m. In practice, a hash function should be applied to the message m in the signature generation step. We will discuss why the hash function is needed in this section.

Suppose we already have a valid signature pair (r, s) on the message m, we can make use of this signature pair to reproduce another valid signature pair on some messages as follows:

Select integers A, B, and C arbitrarily such that $(Ar - Cs)$ is relatively prime to $p - 1$. Set

$$\begin{aligned} r' &= r^A \alpha^B y^C \pmod{p}, && (3.1) \\ s' &= r's/(Ar - Cs) \pmod{p-1}, && (3.2) \\ m' &= r'(Am + Bs)/(Ar - Cs) \pmod{p-1}. && (3.3) \end{aligned}$$

Then the (r', s') are a valid signature pair on message m'.

To see how we can get this r', s' and m', here is my derivation. First, we assume that r' is controlled by three variables, r, α and y, so we set

$$r' = r^A \alpha^B y^C \pmod{p} \tag{3.4}$$

If (r', s') is a valid signature pair on message m', then it should satisfy

$$\begin{aligned} \alpha^{m'} &= y^{r'} r'^{s'} \pmod p \\ &= y^{r'} (r^A \alpha^B y^C)^{s'} \pmod p \\ &= y^{r'+Cs'} r^{As'} \alpha^{Bs'} \pmod p \end{aligned} \tag{3.5}$$

If we can express $y^{r'+Cs'} r^{As'}$ in terms of α powers, then we can found the value of m'. To achieve this, we can get some hints from the equation

$$y^r r^s = \alpha^m \tag{3.6}$$

So the key is to find an t such that

$$y^{r'+Cs'} r^{As'} = (y^r r^s)^t \pmod p \tag{3.7}$$

To obtain the value of t, first by comparing the coefficients in equation (3.7), we have

$$\begin{aligned} r' + Cs' &= rt \pmod{p-1} \\ As' &= st \pmod{p-1} \end{aligned} \tag{3.8}$$

Therefore,

$$\begin{aligned} r's + Css' &= Ars' \pmod{p-1} \\ (Ar - Cs)s' &= r's \pmod{p-1} \\ s' &= \frac{r's}{Ar - Cs} \pmod{p-1} \end{aligned} \tag{3.9}$$

Once s' is found, we can find t from (3.8)

$$t = \frac{Ar'}{Ar - Cs} \pmod{p-1} \tag{3.10}$$

Therefore, the value of m' can be found in equation (3.5)

$$\begin{aligned}\alpha^{m'} &= y^{r'+Cs'}r^{As'}\alpha^{Bs'} \pmod p \\ &= (y^r r^s)^t \alpha^{Bs'} \pmod p \\ &= (\alpha^m)^t \alpha^{Bs'} \pmod p \qquad (3.11)\end{aligned}$$

Therefore, m' is

$$\begin{aligned}m' &= mt + Bs' \pmod{p-1} \\ &= m\frac{Ar'}{Ar-Cs} + B\frac{r's}{Ar-Cs} \pmod{p-1} \\ &= \frac{r'(Am+Bs)}{Ar-Cs} \pmod{p-1} \qquad (3.12)\end{aligned}$$

Here the (r', s') is a valid signature pair on message m'. Note that in this forgery, we can only sign a particular type of message, which is specified in equation (3.3). So it is a kind of *selective forgery*. There is one interesting thing, if we set $A = 0$ in equation (3.1), (3.2) and (3.3), we have

$$\begin{aligned}r' &= \alpha^B y^C \pmod p \\ s' &= -r'C \pmod{p-1} \\ m' &= -r'B/C \pmod{p-1} \qquad (3.13)\end{aligned}$$

It means that we can generate legitimate signatures without knowing any signatures in prior.

In order to prevent such selective forgery, we can apply a hash function on the message in the signature generation step. That is, the signature generation step will become

$$s = k^{-1}[h(m) - rx] \pmod{p-1} \qquad (3.14)$$

instead of

$$s = k^{-1}[m - rx] \pmod{p-1} \tag{3.15}$$

If no hash function is used, such forgery is possible for some particular messages m. If hash function is used, such forgery is possible for some particular $h(m)$. Owing to the nature of hash function, it is difficult to find the message m from $h(m)$. Therefore, such type of forgery can be avoided.

3.3 Security of ElGamal signature scheme

In this section, some security issues of ElGamal signature scheme will be discussed.

1. Same k cannot be used twice; otherwise, we can probably calculate the private key x of the signer. Suppose the signer generate two signature pairs (r, s_1) and (r, s_2) with the same k, so we have

   $$s_1 = k^{-1}\{h(m_1) - rx\} \pmod{p-1} \tag{3.16}$$

   and

   $$s_2 = k^{-1}\{h(m_2) - rx\} \pmod{p-1} \tag{3.17}$$

   Then,

   $$(s_1 - s_2)k = h(m_1) - h(m_2) \pmod{p-1} \tag{3.18}$$

   If $\gcd(s_1 - s_2, p-1) = 1$, then

   $$k = \frac{1}{(s_1 - s_2)}[h(m_1) - h(m_2)] \pmod{p-1} \tag{3.19}$$

Once k is known, we can solve for x by substituting it into either equation (3.16) or (3.17). Therefore,

$$x = \frac{h(m_1) - s_1 k}{r} \pmod{p-1} \tag{3.20}$$

or

$$x = \frac{h(m_2) - s_2 k}{r} \pmod{p-1} \tag{3.21}$$

2. It is important that the verifier checks whether $1 \leq r < p$ is satisfied. If this check is not done, we can produce another signature pair (r_2, s_2) on message m_2 if we have a valid signature pair (r_1, s_1) on message m_1 at hand.

   Proof. If $h(m_1)^{-1} \pmod{p-1}$ exists, set

$$u = h(m_2)h(m_1)^{-1} \pmod{p-1} \tag{3.22}$$

   Now (r_2, s_2) can be found by setting

$$s_2 = s_1 u \pmod{p-1} \tag{3.23}$$

   and by computing r_2 satisfying

$$\begin{aligned} r_2 &= r_1 u \pmod{p-1} && (3.24) \\ r_2 &= r_1 \pmod{p} && (3.25) \end{aligned}$$

   r_2 can be found by using the Chinese Remainder Theorem. This (r_2, s_2) is a valid signature pair on message m_2 because

$$\begin{aligned} y^{r_2} r_2^{s_2} &= y^{r_1 u} r_1^{s_1 u} \pmod{p} \\ &= (y^{r_1} r_1^{s_1})^u \pmod{p} \\ &= \alpha^{h(m_1)u} \pmod{p} \\ &= \alpha^{h(m_2)} \pmod{p} && (3.26) \end{aligned}$$

3. This is the case for discrete logarithm at $GF(2^n)$. If the extension polynomial is x^n+x+1 and α is a root of this polynomial. Suppose the signature $r=(1,1,\ldots,1)$, we can solve for k since

$$\begin{aligned} 1+\alpha+\cdots+\alpha^{n-1} &= \frac{\alpha^n+1}{\alpha+1} \\ &= \frac{\alpha}{\alpha^n} \\ &= \alpha^{1-n} \end{aligned} \tag{3.27}$$

Generally, if the public key $(1,1,\ldots,1)$, then we can solve the private key easily.

3.4 Bleichenbacher's Attack

In [3], it shows that forgery is possible if the prime number p or the generator α are not chosen carefully. First, we will present the Bleichenbacher's attack on ElGamal signature scheme and show that such attack can be extended to variations of the ElGamal signature scheme.

Theorem 3.4.1 *Let $p-1=bw$ where b is smooth and let y be the public key of user A. If a generator $\beta=cw$ with $0<c<b$ and an integer t are known such that $\beta^t=\alpha$ (mod p), then a valid ElGamal signature (r,s) on a given h can be found.*

Proof. The equation

$$\alpha^{wz}\equiv y^w \pmod p \tag{3.28}$$

can be solved for z. Since α is the generator of the group $\mathbb{Z}_p^*$ with order $p-1$, then the subgroup H generated by α^w has a smooth order b. Therefore, we can use the algorithm of Pohlig and Hellman to solve the above equation.

Now let

$$r = \beta \pmod p \tag{3.29}$$

and

$$s = t(h - cwz) \pmod{p-1} \tag{3.30}$$

This (r, s) is a valid signature pair on message h since

$$\begin{aligned} r^s y^r &= (\beta^t)^{h-vwz}(y)^{cw} \pmod p \\ &= \alpha^{h-cwz}\alpha^{cwz} \pmod p \\ &= \alpha^h \pmod p \end{aligned} \tag{3.31}$$

Corollary 3.4.1 *If α is smooth and divides $p-1$, then it is possible to generate a valid ElGamal signature.*

Proof. Let $\beta = (p-1)/\alpha$ and $t = (p-3)/2$. Then $\beta^t = (-1)\beta^{-1} = \alpha (\text{mod } p)$. Thus it follows by Theorem 3.4.1 that signatures can be forged.

We can see that forgery can be possible even if we have applied the hash function to the message. The probability of finding a generator β depends on the value of b. If b is small, then it is unlikely that we will find a generator. Moreover, the generator α should be chosen carefully such that it does not divide $p-1$. Since p and α are shared among several users. These parameters are usually generated from an authority. With the Bleichenbacher's attack, an authority can generate a trapdoor prime in which the β and t can be found easily. Two different methods to generate this trapdoor are shown here.

3.4.1 Constructing trapdoor

Method A When p is fixed and $p-1 = bw$ with b smooth, then we can find α, β and t in the following way (provided that b is not too small).

1. Choose $c \in \{1, \dots, b-1\}$ randomly until $\beta = cw$ is a generator of $\mathbb{Z}_p^*$
2. Choose t with $\gcd(t, p-1) = 1$
3. Compute $\alpha = \beta^t$

Method B When the generator α is fixed, then p, β and t can be generated as follows

1. Select three positive integers u, v and c such that v is odd and $c^v\alpha^u$ has approximately the size of the prime to construct.
2. Compute the smooth divisors of $c^v\alpha^u - 1$
3. If there exists a smooth divisor $d > c$ of $c^v\alpha^u - 1$ such that $p = c^v\alpha^u - d^v$ is prime, $\frac{p-1}{2} - u$ is relatively prime to $p-1$ and α is a generator of $\mathbb{Z}_p^*$, then compute

$$\begin{aligned} \beta &= c\frac{p-1}{d} \\ t &= v(\frac{p-1}{2} - u^{-1}) \pmod{p-1} \end{aligned} \tag{3.32}$$

Since d divides $c^v\alpha^u - 1$ and $p - 1 = c^v\alpha^u - d^v - 1$, so d also divides $p-1$. Thus β satisfies the precondition of Theorem 3.4.1. Since

$$\alpha^u c^v = d^v \pmod{p} \tag{3.33}$$

we have

$$\alpha^{-u} = \left(cd^{-1}\right)^{v} \pmod{p} \tag{3.34}$$

Therefore,

$$\begin{aligned}\beta^{v} &= \left(c\frac{p-1}{d}\right)^{v} \pmod{p} \\ &= \left[cd^{-1}(p-1)\right]^{v} \pmod{p} \\ &= \left(-cd^{-1}\right)^{v} \pmod{p} \\ &= (-1)\alpha^{-u} \pmod{p} \\ &= \alpha^{\frac{p-1}{2}-u} \pmod{p} \end{aligned} \tag{3.35}$$

Hence

$$\beta^{t} = \alpha \pmod{p} \tag{3.36}$$

3.5 Extension to Bleichenbacher's attack

Many variations of the basic ElGamal signature scheme have been proposed and some of these variations are also vulnerable by Bleichenbacher's attack. Here is my extension of Bleichenbacher's attack to these variations.

In the basic ElGamal signature scheme, after suitable rearrangement, the *signing equation* can be written as

$$u = vx + kw \pmod{p-1}$$

where $u = h(m)$, $v = r$, and $w = s$ in the original ElGamal signature scheme.

These variations generally involve in permutating the terms u, v, and w in the signing equation. Table (3.1) shows the variations of the ElGamal signature scheme.

3.5.1 Attack on variation 3

In the variation 3, the signing equation is

$$s = rx + kh(m) \pmod{p-1} \tag{3.37}$$

and the verification equation is

$$y^r r^{h(m)} = \alpha^s \pmod{p} \tag{3.38}$$

To forge the signature, we follow the steps as in Bleichenbacher's attack, except that

$$s = cwz + \frac{1}{t}h(m) \pmod{p-1} \tag{3.39}$$

Proof

$$\begin{aligned} \alpha^s &= \alpha^{cwz+\frac{1}{t}h(m)} \pmod{p} \\ &= \alpha^{cwz}\alpha^{\frac{1}{t}h(m)} \pmod{p} \\ &= y^{cw}r^{h(m)} \pmod{p} \end{aligned} \tag{3.40}$$

	u	v	w	Signing equation	Verification
1	h	r	s	$h = rx + ks$	$y^r r^s = \alpha^h$
2	h	s	r	$h = sx + kr$	$y^s r^r = \alpha^h$
3	s	r	$h(m)$	$s = rx + kh(m)$	$y^r r^{h(m)} = \alpha^s$
4	s	$h(m)$	r	$s = xh(m) + kr$	$y^{h(m)} r^r = \alpha^s$
5	r	s	$h(m)$	$r = sx + kh(m)$	$y^s r^{h(m)} = \alpha^r$
6	r	$h(m)$	s	$r = xh(m) + ks$	$y^{h(m)} r^s = \alpha^r$

Table 3.1: Variations of the ElGamal signature scheme

3.5.2 Attack on variation 5

In variation 5, the signing equation is

$$r = sx + kh(m) \pmod{p-1} \tag{3.41}$$

and the verification equation is

$$y^s r^{h(m)} = \alpha^r \pmod{p} \tag{3.42}$$

Again, to forge the signature, we follow the steps as in Bleichenbacher's attack, except that

$$s = \frac{rt - h(m)}{tz} \pmod{p-1} \tag{3.43}$$

Proof

$$\begin{aligned} y^s r^{h(m)} &= y^{\frac{rt-h(m)}{tz}} r^{h(m)} \pmod{p} \\ \text{(with some prob.)} &= \alpha^{\frac{z[rt-h(m)]}{tz}} r^{h(m)} \pmod{p} \\ &= \alpha^{\frac{rt-h(m)}{t}} r^{h(m)} \pmod{p} \\ &= \alpha^r r^{-h(m)} r^{h(m)} \pmod{p} \\ &= \alpha^r \pmod{p} \end{aligned} \tag{3.44}$$

3.5.3 Attack on variation 6

In variation 6, the signing equation is

$$r = xh(m) + ks \pmod{p-1} \tag{3.45}$$

and the verification equation is

$$y^{h(m)} r^s = \alpha^r \pmod{p} \tag{3.46}$$

Similarly, we follow the steps as in Bleichenbacher's attack, except that

$$s = t[r - zh(m)] \pmod{p-1} \tag{3.47}$$

Proof

$$\begin{aligned} y^{h(m)} r^s &= y^{h(m)} r^{t[r-zh(m)]} \pmod{p} \\ &= y^{h(m)} \alpha^{r-zh(m)} \pmod{p} \\ &= y^{h(m)} \alpha^r \alpha^{-zh(m)} \pmod{p} \\ \text{(with some prob.)} &= y^{h(m)} \alpha^r y^{-h(m)} \pmod{p} \\ &= \alpha^r \pmod{p} \end{aligned} \tag{3.48}$$

3.6 Digital Signature Standard(DSS)

The National Institute of Standards and Technology (NIST) published the Digital Signature Algorithm (DSA) in the Digital Signature Standard (DSS), which is a part of the U.S. government's Capstone project. DSA is based on the discrete logarithm problem and a variant of the ElGamal signature scheme. It is more secure than the ElGamal signature scheme since it chooses the parameters carefully such that the attacks described in previous sections do not exist.

Public Parameters

1. Select a prime number q, where $2^{159} < q < 2^{160}$.
2. Select a prime number p, where $2^{l-1} < p < 2^l$ for $512 \leq l \leq 1024$ and l is a multiple of 64 and with the property that q divides $(p-1)$
3. Select an integer g, where $1 < g < p-1$ such that $g^{(p-1)/q} > 1 \pmod{p}$

4. Compute $\alpha = g^{(p-1)/q} \pmod{p}$, so α is the generator of the cyclic group of order q in $\mathbb{Z}_p^*$

5. Publish p, α

Key Generation

1. Randomly generate an integer x, where $0 < x < q$

2. Compute $y = \alpha^x \pmod{p}$

3. Private key: x

   Public key: y

Signature Generation

1. Randomly generate an integer k, where $0 < k < q$

2. Compute $r = \{\alpha^k \pmod{p}\} \pmod{q}$

3. Compute $k^{-1} \pmod{q}$

4. Compute $s = k^{-1}\{h(m) + rx\} \pmod{q}$

5. Signature pair (r, s)

Signature Verification

1. Check whether $0 < r' < q$ and $0 < s' < q$; if either condition is violated, reject the signature

2. Compute $w = s'^{-1} \pmod{q}$ and $h(m')$

3. Compute $u_1 = wh(m') \pmod{q}$ and $u_2 = r'w \pmod{q}$

4. Compute $v = [\alpha^{u_1} y^{u_2} \pmod{p}](\bmod\ q)$

5. If $v = r'$, accept the signature

Now, we want to prove that the above signature verification works, i.e., if $m = m'$, $r = r'$ and $s = s'$, then $v = r'$. We need the following result to proceed.

Lemma 3.6.1 *Let p and q be primes such that q divides $p-1$, g is a positive integer less than p, and $\alpha = g^{(p-1)/q}$ (mod p). Then $\alpha^q = 1$ (mod p), and if $m = n$ (mod q), then $\alpha^m = \alpha^n$ (mod p)*

Proof.

$$\begin{aligned} \alpha^q &= \{g^{(p-1)/q}\}^q \pmod p \\ &= g^{p-1} \pmod p \\ &= 1 \pmod p \end{aligned} \tag{3.49}$$

If $m = n \pmod q$, then $m = n + kq$ for some integer k.

$$\begin{aligned} \alpha^m &= \alpha^{n+kq} \pmod p \\ &= \alpha^n \alpha^{kq} \pmod p \\ &= \alpha^n (\alpha^q)^k \pmod p \\ &= \alpha^n \pmod p \end{aligned} \tag{3.50}$$

Theorem 3.6.1 *If $m = m'$, $r = r'$ and $s = s'$ in the signature verification, then $v = r'$.*

Proof. We have

$$\begin{aligned} w &= s'^{-1} \pmod q \\ &= s^{-1} \pmod q \end{aligned} \tag{3.51}$$

$$\begin{aligned} u_1 &= wh(m') \pmod q \\ &= wh(m) \pmod q \end{aligned} \tag{3.52}$$

$$\begin{aligned} u_2 &= r'w \pmod q \\ &= rw \pmod q \end{aligned} \tag{3.53}$$

Now,

$$\begin{aligned} v &= [\alpha^{u_1} y^{u_2} (\text{mod } p)](\text{mod q}) \\ &= [\alpha^{wh(m)} y^{rw} (\text{mod } p)](\text{mod q}) \\ &= [\alpha^{wh(m)} \alpha^{rwx} (\text{mod } p)](\text{mod q}) \\ &= [\alpha^{(h(m)+rx)w} (\text{mod } p)](\text{mod q}) \end{aligned} \tag{3.54}$$

and

$$s = k^{-1}[h(m) + rx] \pmod q \tag{3.55}$$

Hence

$$\begin{aligned} w &= k[h(m) + rx]^{-1} \pmod q \\ w[h(m) + rx] &= k \pmod q \end{aligned} \tag{3.56}$$

Thus by the lemma

$$\begin{aligned} v &= \alpha^k (\text{mod } p) \pmod q \\ &= r \pmod q \\ &= r' \pmod q \end{aligned} \tag{3.57}$$

Here is an example of typical DSS signature generation [7]. The prime modulus p chosen is

$p =$ 7 434 410 770 759 874 867 539 421 675 728 577 177 024 889 699 586 189 000 788 950 934 679 315 164 676 852 047 058 354 758 883 833 299 702 695 428 196 962 057 871 264 685 291 775 577 130 504 050 839 126 673

and the prime divisor of $p-1$, q is

$q =$ 1 138 656 671 590 261 728 308 283 492 178 581 223 478 058 193 247

Therefore, $(p-1)/q$ is

$(p-1)/q =$ 6 529 106 583 441 773 144 705 959 127 165 952 066 964 259 152 507 339 624 395 270 343 160 295 123 275 238 619 980 855 555 928 611 777 361 776

The generator for the cyclic subgroup of order q in $\mathbb{Z}_p^*$, α is

$\alpha =$ 5 154 978 420 348 751 798 752 390 524 304 908 179 080 782 918 903 280 816 528 537 868 887 210 221 705 817 467 399 501 053 627 846 983 235 883 800 157 945 206 652 168 013 881 208 773 209 963 452 245 182 466

The private key, x, is

$x =$ 185 200 992 879 430 349 246 928 890 763 396 348 557 950 060 052

Then the public key, $\beta = \alpha^x$ is

$\beta = $ 1 313 262 929 912 132 563 149 647 614 777 216 522 615 273 453 570 746 151 675 467 137 191 353 050 812 753 072 254 631 435 649 494 300 072 321 220 858 192 262 592 093 219 893 426 623 085 296 074 200 376 115

Signature generation Assume the hashed message, $h(m)$ is

$h(m) = $ 968 236 873 715 988 614 170 569 073 515 315 707 566 766 479 517

We randomly generate an integer, k

$k = $ 305 736 015 983 784 127 425 204 148 486 426 464 792 703 852 479

Then r=[α^k (mod p)] (mod q) is

$r = $ 797 387 772 302 493 209 021 291 765 790 742 058 535 532 839 360

The inverse of k

$k^{-1} = $ 76 032 227 071 633 570 089 031 757 830 396 511 732 987 263 255

So

$s = $ 376 128 930 725 767 930 183 372 771 997 677 399 746 658 752 712

signature verification Calculate $w = s'^{-1} \pmod{q}$

w = 901 773 569 445 286 146 281 696 254 684 032 552 931 964 781 595

$u_1 = w \cdot h(m') \pmod{q}$ and

u_1 = 1 092 677 610 583 452 800 814 993 273 503 516 509 004 891 674 269

$u_2 = r'w \pmod{q}$

u_2 = 742 761 371 689 099 697 708 866 970 523 451 752 290 280 404 144

Compute $v = \{\alpha^{u_1} y^{u_2} \pmod{p}\} \pmod{q}$

v = 797 387 772 302 493 209 021 291 765 790 742 058 535 532 839 360

which is equal to r, so we accept this signature.

Chapter 4

Quadratic Field Sieve

4.1 Quadratic Field

In this chapter, integers in quadratic field will be used in the number field sieve to solve the discrete logarithm problem. First we will have some introduction of the quadratic field.

Theorem 4.1.1 *An algebraic number of the form $a + b\sqrt{D}$ where a and b are rational numbers and D is a squarefree integer forms a quadratic field and it is denoted as $\mathbb{Q}(\sqrt{D})$.*

Let $a+b\sqrt{D}$ and $c+d\sqrt{D}$ are the algebraic numbers in the quadratic field. The four arithmetic operations in quadratic field are defined as

$$\begin{aligned}(a+b\sqrt{D}) \pm (c+d\sqrt{D}) &= (a \pm c) + (b \pm d)\sqrt{D} \\ (a+b\sqrt{D})(c+d\sqrt{D}) &= (ac+bdD) + (ad+bc)\sqrt{D} \\ \frac{a+b\sqrt{D}}{c+d\sqrt{D}} &= \frac{ac-bdD}{c^2-d^2D} + \frac{bc-ad}{c^2-d^2D}\sqrt{D} \qquad (4.1)\end{aligned}$$

As D is a square-free integer, $c^2 - d^2D$ will always $\neq 0$, except when $c = d = 0$. Thus all divisions by $c + d\sqrt{D}$ can be carried out, except the division by $0 + 0\sqrt{D} = 0$, the zero element of the field. Since the result of any arithmetic operation can always be reduced to $p + q\sqrt{D}$, with p and q rational numbers, the domain is closed under arithmetic operations, and it is thus a field.

4.1.1 Integers of Quadratic Field

The integers in the quadratic field is defined as

Definition 4.1.1 *The number z is said to be an integer of the quadratic field $\mathbb{Q}(\sqrt{D})$ if it satisfies a quadratic equation of the form*

$$z^2 + pz + q = 0 \tag{4.2}$$

where coefficients p and q are rational integers[1].

Theorem 4.1.2 *The integers of $\mathbb{Q}(\sqrt{D})$ are of the form $x = r + s\rho$ for r and s being arbitrary rational integers, and where*

$$\rho = \begin{cases} \sqrt{D} & \textit{if } D = 2 \textit{ or } = 3 \quad (\textit{mod } 4) \\ \frac{-1+\sqrt{D}}{2} & \textit{if } D = 1 \quad (\textit{mod } 4) \end{cases} \tag{4.3}$$

Proof. If the number $z = r + s\sqrt{D}$ is a solution of the quadratic equation in (4.2), then

$$(z - r)^2 = Ds^2 \tag{4.4}$$

That is

$$z^2 - 2rz + r^2 - Ds^2 = 0 \tag{4.5}$$

[1] *To avoid imbecility, we shall call the ordinary integers* rational integers

By comparing equation in (4.2) and (4.5), $-2r$ and $r^2 - Ds^2$ should be integers. Putting $r = a/2$ for integer a, $2r$ is always a rational integer and

$$r^2 - Ds^2 = \frac{a^2}{4} - Ds^2 \tag{4.6}$$

is a rational integer if either

1. a is even, in this case r and s are rational integers, or
2. a is odd, in this case, $a^2/4 = 1/4 \pmod 1$, and $Ds^2 = 1/4 \pmod 1$. This condition is satisfied if $s = 1/2 \pmod 1$ and $D = 1 \pmod 4$.

Therefore, combining these cases, we have the above theorem.

Some examples of the integers in quadratic field are: the integers in $\mathbb{Q}(\sqrt{1})$ are simply called rational integers, and the integers in $\mathbb{Q}(\sqrt{-1})$ are called Gaussian integers.

4.1.2 Primes in Quadratic Field

Definition 4.1.2 *The number $x = a + b\sqrt{D}$ and $\bar{x} = a - b\sqrt{D}$ are called conjugate numbers in $\mathbb{Q}(\sqrt{D})$*

Definition 4.1.3 *The norm of x in $\mathbb{Q}(\sqrt{D})$ is*

$$N(x) = x\bar{x} \tag{4.7}$$

Definition 4.1.4 *If x is an integer of $\mathbb{Q}(\sqrt{D})$ and*

$$N(x) = \pm 1 \tag{4.8}$$

then x is called a unit of $\mathbb{Q}(\sqrt{D})$.

Definition 4.1.5 *If the integer α of $\mathbb{Q}(\sqrt{D})$ cannot be decomposed as $\alpha = \beta\gamma$, with β and γ integers of $\mathbb{Q}(\sqrt{D})$, unless one of β or γ is a unit, then α is said to be a prime of $\mathbb{Q}(\sqrt{D})$. Otherwise α is said to be composite in $\mathbb{Q}(\sqrt{D})$*

Definition 4.1.6 *An integral domain D in which factorisation into irreducible elements is possible, and all such factorisations are unique, is called a unique factorisation domain.*

The algebraic integers in an arbitrary quadratic field do not necessary have unique factorisations. There are exactly 25 quadratic fields in which there exists unique factorisations: they are -163, -67, -43, -19, -11, -7, -3, -2, -1, 2, 3, 5, 6, 7, 11, 13, 17, 19, 21, 29, 33, 37, 41, 57, and 73 [10].

4.2 Number Field Sieve

The number field sieve runs sub-exponential time in solving discrete logarithm problem. This method is similar to the index-calculus in which they both have the factor base. First, we will give a sketch of the algorithm. The number field sieve is based on the fact that:

Theorem 4.2.1 *Suppose the discrete logarithm problem is $\log_a b$. Let q be a prime divisor of $p-1$. If we are able to find $s, t \in \mathbb{Z}$, with t coprime to q satisfying the property*

$$a^s b^t = w^q \pmod{p} \tag{4.9}$$

for some w, then we can compute x mod q.

Proof. Since if (4.9) holds, then

$$a^{s+tx} = w^q \pmod{p} \tag{4.10}$$

Since a is the generator, we can let $w = a^i$, then (4.10) can be written as

$$a^{s+tx} = a^{iq} \pmod p \tag{4.11}$$

Therefore,

$$s + tx = iq \pmod{p-1} \tag{4.12}$$

Since q is a factor of $p-1$,

$$s + tx = 0 \pmod q \tag{4.13}$$

Therefore, we can compute $x \bmod q$ where

$$x = -st^{-1} \pmod q \tag{4.14}$$

Construction of the number field The number field sieve is based on the observation that it is possible to construct a number field $K = \mathbb{Q}(\alpha)$ and a ring homomorphism φ from $\mathbb{Z}[\alpha]$ of K to $\mathbb{Z}/n\mathbb{Z}$ such that $\varphi(\alpha) = m \pmod p$. To generate the number field, we need an irreducible polynomial

$$f(X) = X^n + a_{n-1}X^{n-1} + \ldots + a_1X + a_0 \tag{4.15}$$

We restrict the degree of $f(X)$ to be two, i.e., we use the integers in quadratic field. The reason is that: The algorithm runs better if the number field is a unique factorisation domain. There are only 25 quadratic fields which have unique factorisation domain. We can save some time in searching a field that meets this requirement.

In order to construct the ring homomorphism, we have to find a m such that

$$f(m) = 0 \pmod p \tag{4.16}$$

Choose the factor base In the second stage, we have to choose two factor bases. The first factor base B_1 consists of small primes, including a and b. The second factor base B_2 consists of those prime with small norm in the quadratic field.

Sieving step The sieving step involves the concept of smoothness, here are the definitions of smoothness for both rational integers and algebraic integers.

Definition 4.2.1 *A rational integer is called B-smooth if every prime number dividing it is at most B*

Definition 4.2.2 *An algebraic integer is called B-smooth if every prime number dividing its norm is at most B*

The sieving step determines a set T of pairs (c, d) such that

1. the rational integer $(c + dm)$ is B_1-smooth
2. the algebraic integer $(c + d\alpha)$ is B_2-smooth

Linear algebra Use the Lanczos algorithm to determine a subset $S \subset T$ such that

1. $\prod_{(c,d)\in S}(c + dm)$ is only divisible by a and b
2. $\prod_{(c,d)\in S}(c + d\alpha)$ is a q-th power in $\mathbb{Z}[\alpha]$

Compute result After having accomplished this, we achieve congruence (4.9) via

$$a^s b^t = \prod_{(c,d)\in S}(c + dm)$$

$$\begin{aligned} &= \prod_{(c,d)\in S} \varphi(c+d\alpha) \\ &= \varphi(\delta^q) \\ &= w^q \pmod p \end{aligned} \tag{4.17}$$

4.3 Solving Sparse Linear Equations Over Finite Fields

The system of linear equations collected in number field sieve are generally very large. These systems cannot be solved using ordinary linear algebra techniques, due to both time and space constraints. However, the equations generated are extremely sparse, there are only a few non-zero coefficients per equation. Here are some methods to solve the sparse matrix.

4.3.1 Lanczos and conjugate gradient methods

Suppose that we have to solve the system

$$Ax = b \tag{4.18}$$

for a column n-vector x, where A is a symmetric $n \times n$ matrix, and b is a given column n-vector. Let

$$b_0 = b, \tag{4.19}$$

$$c_1 = Ab_0, \tag{4.20}$$

$$b_1 = c_1 - \frac{< c_1, c_1 >}{< b_0, c_1 >} b_0, \tag{4.21}$$

and then, for $i \geq 1$, define

$$\begin{aligned} c_{i+1} &= Ab_i, && (4.22) \\ b_{i+1} &= c_{i+1} - \frac{< c_{i+1}, c_{i+1} >}{< b_i, c_{i+1} >} b_i - \frac{< c_{i+1}, c_i >}{< b_{i-1}, c_i >} b_{i-1} && (4.23) \end{aligned}$$

The algorithm stops when it finds a b_j that is conjugate to itself, i.e., such that $< b_j, Ab_j >= 0$. This happens for some $j \leq n$. If $w_j = 0$, then

$$x = \sum_{i=0}^{j-1} d_i b_i \tag{4.24}$$

is a solution, where

$$d_i = \frac{< b_i, b >}{< b_i, c_{i+1} >} \tag{4.25}$$

(If $w_j \neq 0$, the algorithm fails)

The Lanczos algorithm was invented to solve systems with real coefficients. To solve systems over finite fields, we can just apply the above equations to a finite field situation. [12] discusses some solutions when the matrix is not square.

4.3.2 Structured Gaussian Elimination

The basic idea of the structured Gaussian elimination is to declare some columns with many non-zero coefficients as heavy, and those with few non-zero coefficients as light.

1. Delete all columns which have a single nonzero coefficient and the rows in which those columns have nonzero coefficients. Repeat this step until there are no more columns with a single nonzero entry.

2. Select those αM columns which have the largest number of nonzero elements for some $\alpha > 0$. Call these columns "heavy", the others "light".

Typical value of α might be $1/32$. The entries in the "heavy" columns for every given row might be stored on a disk, with a pointer attached to the row list indicating the storage location. These pointers would have coefficients attached to them, which are set to 1 initially. The weight of a row is then defined as the number of nonzero coefficients in its "light" columns.

3. Eliminate variables corresponding to rows of weight 1 by subtracting appropriate multiples of those rows from other rows that have nonzero coefficients corresponding to those variables. If u times row i is to be subtracted from row j, the pointers attached to row j are to have added to them the pointers of row i, with their coefficients multiplied by u. Repeat this step until there are no more rows of weight 1. At the end of this process there are likely to be many more equations than unknowns.

4. If r rows are excess, drop the r rows with highest weight.

4.3.3 Wiedemann Algorithm

Let the linear equations to be solved is

$$Ax = b,$$

where A is a sparse non-singular $n \times n$ matrix. Let the minimal polynomial of A be $f(z)$, where

$$f(z) = \sum_{j=0}^{d} c_j z^j \tag{4.26}$$

for $d \leq n, c_0 \neq 0$. By the definition of the minimal polynomial,

$$\sum_{j=0}^{d} c_j A^j = 0, \tag{4.27}$$

Thus,

$$\begin{aligned}
\sum_{j=0}^{d} c_j A^j b &= 0 \\
\sum_{j=1}^{d} c_j A^j b + c_0 b &= 0 \\
b &= -c_0^{-1} \sum_{j=1}^{d} c_j A^j b \\
&= A \left[-c_0^{-1} \sum_{j=1}^{d} c_j A^{j-1} b \right] \qquad (4.28)
\end{aligned}$$

Then we can obtain the solution

$$x = -c_0^{-1} \sum_{j=1}^{d} c_j A^{j-1} \tag{4.29}$$

In general, we use the Structured Gaussian Elimination to reduce the size of matrix. And then we can apply either the Wiedemann or Lanczos algorithm to solve the system of equations.

Chapter 5

Conclusion

Cryptography is a strict discipline, in the sense that the parameters used in the cryptographic algorithms should be carefully chosen. Otherwise, it may pose some security holes for the attacker to break in the system. For example, we have discussed the ElGamal signature scheme in chapter 3.

1. A hash function should be applied to the message first in the signature generation, otherwise, we can generate a valid signature pair without knowing the private key of signer.

2. When we receive the signature, we should check that r should be less than p.

3. A different k should be selected for each message to be signed. Otherwise, the private key of the signer can be probably recovered.

4. The prime number should be large enough to prevent the index-calculus attack.

5. $p-1$ should contain a large prime factor q, to prevent the Pohlig-Hellman attack.

6. From the Bleichenbacher's attack, the generator α chosen should not divide $p-1$.

This thesis extends the Bleichenbacher's attack to the variations of the ElGamal signature scheme. However, such types of attack often impose a stricter condition.

The Digital Signature Standard is a more secure version of digital signature than ElGamal signature scheme. It is because the prime number p contains at least a large prime q, which can prevent some attacks like index-calculus, Pohlig-Hellman and Bleichenbacher. But with the advance of the computing technology, the prime should go beyond one thousand to two thousand bits.

In chapter 4, we have discussed a special kind of number field sieve - the integers in quadratic field. The use of quadratic integers facilitates the searching for unique factorisation domain (since in quadratic field, there are only 25 unique factorisation domain). In using the number field sieve, we have to deal with a large and sparse matrix. It will be time-consuming if we just use ordinary Gaussian elimination to solve such matrix. Therefore, some techniques like Lanczos Algorithm, Structured Gaussian Elimination and Wiedemann Algorithm are used.

Bibliography

[1] L.M. Adleman, *A subexponential algorithm for the discrete logarithm problem with applications to cryptography*, Proceedings of the IEEE 20th Annual Symposium on Foundations of Computer Science, 55-60, 1979.

[2] Daniel J. Bernstein, A. K. Lenstra, *A general number field sieve implementation*, Springer-Verlag, 1993.

[3] Daniel Bleichenbacher, *Generating ElGamal signature without knowing the secret key*, Advances in Cryptology - EUROCRYPT 1996

[4] J. Buchmann, J. Loho, J. Zayer *An implementation of the general number field sieve*, Advances in Cryptology - CRYPTO 1993.

[5] J. P. Buhler, H. W. Lenstra, Jr., Carl Pomerance, *Factoring integers with the number field sieve*, Springer-Verlag, 1993.

[6] Jean-Marc Couveignes, *Computing a square root for the number field sieve*, Springer-Verlag, 1993.

[7] Wei Dai, *Crypto++ 3.2*, http://www.eskimo.com/ weidai/cryptlib.html.

[8] W. Diffie and M.E. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory 22 (1976), 644-654.

[9] Taher ElGamal, *A public key cryptosystem and a signature scheme based on discrete logarithms*, IEEE Trans. Information Theory, IT-31, 1985.

[10] G. H. Hardy and E. M. Wright, *An Introduction to the Theory of Numbers, Fifth edition*, Oxford, 1979.

[11] P. Horster, M. Michels, H. Petersen, *Meta ElGamal signature schemes*, Proc. 2nd ACM conference on Computer and Communications security.

[12] B. A. LaMacchia, A. M. Odlyzko, *Solving large sparse linear systems over finite fields*, Advances in Cryptology - CRYPTO 1990.

[13] B. A. LaMacchia, A. M. Odlyzko, *Computation of discrete logarithms in prime fields*,

[14] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The number field sieve*, Springer-Verlag, 1993.

[15] Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone, *Handbook of applied cryptography*, CRC Press, 1999.

[16] A. M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in Cryptology - EUROCRYPT 1984.

[17] National Institute of Standards and Technology, *The digital signature standard*

[18] Charles C. Pinter, *A book of abstract algebra, Second Edition*, McGraw-Hill

[19] Stephen C. Pohlig, Martin E. Hellman, *An improved algorithm for computing logarithms over GF(p) and its cryptographic significance*, IEEE Trans. Information Theory, IT-24, 1978.

[20] J. M. Pollard, *Monte Carlo methods for index computation (mod p)*, Mathematics of Computation, Vol. 32, No. 143, 1978.

[21] Hans Riesel, *Prime numbers and computer methods for factorization*, Progress in Mathematics, Vol 57.

[22] RSA Security Inc., *RSA Laboratories' Frequently Asked Questions About Today's Cryptography*, http://www.rsasecurity.com/rsalabs/faq/.

[23] Oliver Schirokauer, Damiam Weber, Thomas Denny, *Discrete logarithms: the effectiveness of the index calculus method*, Proc. 2nd Algorithmic Number Theory Symp. (ANTS '96).

[24] Bruce Schneier, *Applied cryptography, protocols, algorithms, and source code in C*, John Wiley & Sons, Inc.

[25] Damiam Weber, *An implementation of the general number field sieve to compute discrete logarithms mod p*, Advances in Cryptology - EUROCRYPT 1995.

[26] Damiam Weber, *Computing discrete logarithms with the general number field sieve*, Algorithmic Number Theory Symp. (ANTS '96).

[27] D. H. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Information Theory, IT-32, 1986.