AN END-TO-END IDENTITY-BASED EMAIL ENCRYPTION SCHEME

by

Fiona Yan Lee

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

October 2014

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Fiona Yan Lee

Thesis Title: AN END-TO-END IDENTITY-BASED EMAIL ENCRYPTION SCHEME

Date of Final Oral Examination: 17 October 2014

The following individuals read and discussed the thesis submitted by student Fiona Yan Lee, and they evaluated their presentation and response to questions during the final oral examination. They found that the student passed the final oral examination.

Jyh-haw Yeh, Ph.D.	Chair, Supervisory Committee
Dianxiang Xu, Ph.D.	Member, Supervisory Committee
Jim Buffenbarger, Ph.D.	Member, Supervisory Committee

The final reading approval of the thesis was granted by Jyh-haw Yeh, Ph.D., Chair, Supervisory Committee. The thesis was approved for the Graduate College by John R. Pelton, Ph.D., Dean of the Graduate College.

dedicated to my husband

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr.Yeh Jyh-haw, for the patient guidance, encouragement and advice he has provided throughout the times was I working on my thesis. Many thanks to Dr. Amit Jain who encouraged me to apply, and accepted me into the master program after I finished my undergraduate degree in computer science at Boise State University. I also would like to thank my committee members Dr. Buffenbarger, Dr. Alark Joshi, and Dr. xu, who have approved my thesis proposal and took time from their busy schedules to read my thesis.

I must express my deepest gratitude to Joey, my husband, for his unconditional love, continuous support and encouragements. There were times that I thought it is impossible to continue, and he has made it possible. In the past years, I have spent countless weekends, evenings doing my schoolwork. To my kids, Justin and Flovia Lee, I owe them lots and lots of mom-kids times.

Completing my undergraduate and graduate coursework would have been much more difficult if it was not for the support and friendship provided by my neighbor Rose Spires. I thank her for taking care of the kids while I was in classes, and being supportive while I faced other life difficulties.

Finally, I would like to thank my manager at Hewlett Packard company, for giving me the flexibility to attend classes and meeting with my advisor. I could not have finished my thesis in time without this flexibility.

ABSTRACT

Email has evolved into one of the most important methods of communication for any individual and organization. It's amazing how email has transformed our professional and social life. However, current industry standards do not place emphasis on email security; most emails are currently transmitted in plain text over the Internet or other networks. Emails can be intercepted easily by others. Potentially, every non-encrypted email sent over a network or stored at an email server can be read, copied or altered. There is a strong need for secure email delivery.

Some email service providers such as Google's gmail did take some actions to improve privacy protection based upon https protocol. The main motivation for https is to prevent wiretapping and man-in-the-middle attacks. It provides authentication of the gmail website and associated web server that one is communicating with, and it provides bidirectional encryption of communications between a client computer and the gmail server. In practice, this is a reasonable guarantee that one is communicating with precisely the gmail server that one is intended to communicate with, as well as ensuring that the contents of communications between the user and the gmail server cannot be read or forged by any third party. However, https only prevents emails from being sniffed during networking transmission. It does not prevent email server administrators, or anyone else who can gain access to various email servers to read the email messages because https is not an end-to-end encryption. There are end-to-end encryptions available such as PGP(pretty good privacy), it relies on public-key cryptography, in which users can each publish a public key associated

with a certificate that others can use it to encrypt messages to them, while keeping a private key as secret that they can use to decrypt such messages. Set-up, maintaining, publishing own public key and obtaining others' public key are essential for PGP to work properly. These tasks make PGP encryption not so easy to use for ordinary users who do not have a technical background.

This thesis represents an implementation of an end-to-end, identity-based encryption that can be used to encrypt email massages and attachments. It is end-to-end which means the originating party encrypting data to be readable only by the intended recipient. It is identity-based which means the public key of a user is some unique information about the identity of the user, for instance, the user's email address. Because users' public keys are derived from identifiers, identity-based encryption eliminates the need for a public key distribution infrastructure.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

HTTPS – Hyper Text Transfer Protocol Secure

PGP – Pretty Good Privacy, an Encryption Software

SMTP – Simple Mail Transfer Protocol

ECC – Elliptic Curve Cryptography

IDE – Identity-based Encryption

AES – Advanced Encryption Standard

KGC – Key Generation Center

ID-PKC – Identity-based Public Key Cryptography

CHAPTER 1

INTRODUCTION

1.1 Problem Context

For government agencies or companies with sensitive information being sent or received via emails, email security is very important to them. For most of us, the email messages we send wouldn't be classified as sensitive but once in a while, sending sensitive information via emails are necessary such as Social Security or credit card numbers, bank accounts or earning statements, etc. At such points, we all want to make sure the content of a message is kept confidential between sender and receiver. Unfortunately, the standard email messages are sent in plain text which makes it very attainable for eavesdropping, whether during transmission over networks or when they are stored on the email servers. There are some approaches have been taken by email service providers to protect emails from network traffic sniffers such as supporting https. There also is email encryption software available such as PGP. The downside of https is it is not end-to-end encryption. Emails are stored in plain text on various servers, and therefore anyone has gained access to the servers can access your emails. PGP uses end-to-end encryption, but it requires public key management. In this case, normal users are often struggling with sharing their own key and obtaining other people's public key.

1.1.1 Https Approach

Email messages are vulnerable as they travel over the Internet after leaving the email provider's server. Bad guys can intercept a message as it bounces from server to server on the Internet. This hazard typically arises when you use a public network but it can also be pose problems on your work or private network. In order to prevent network traffic being sniffed during transmission, emails service providers such as Google's gmail supports SSL/TLS on top of http to protect logins credientials, and encrypt the connection between your computer and the email server. The encryption also takes place between individual SMTP relays. While https protects network traffic from being sniffed during transmission, it is technically not encryption of emails because the content of messages is revealed to, and can therefore be altered by intermediate email relays. In other words, the encryption is not between the sender and the recipient. Below steps show how emails travel the Internet, from sender to server, then bounce between server to server, and finally from server to recipient. Clearly, https is not end-to-end encryption:

1. Sender to email server: this communication is encrypted based on SSL or TLS over http. Therefore, it ensures the sender is talking to gmail server and not anyone else. In addition, the https protocol prevents network sniffing here.

2. Store on email server: once email leaves sender's computer and arrives on email server, email server decrypts the communication and stores email in plain text format.

3. Email bounds between server to server: there can be many other SMTP servers in between. Again, the communication between server to server might be encrypted with https but each server can store a plain text copy of email.

4. Email server to recipient: finally, when the recipient logs in to read the email, the email is sent from one of the email servers to recipient using https to encrypt the communication.

From above illustration, we can see, because emails are stored in plain text on various servers, server administrators or anyone who can get access to one of the servers can read private email messages and collect any sensitive, valuable information that it contains.

1.1.2 PGP: An End-to-end Encryption Software

PGP is an encryption software that does end-to-end encryption. You can use it to encrypt your emails rendering them unreadable from the point at which they start on their Internet journey, to the point at which the intended recipient opens them. Your emails are still copied and saved at various servers along the way but they are in cipher text format so no one can read them unless the intended recipient decrypts them. PGP uses a variation of the public key system, also known as asymmetric cryptography. This requires two separate keys: one of which is secret (or private) that is known only to the owner, and one of which is public and can be known by everyone. The sender encrypts a message to someone else using their public key. When the receiver obtains the message, they decrypt it using their private key. The term "asymmetric" stems from the use of different keys to perform encryption and decryption. In contrast, symmetric-key algorithms variations of which have been used for thousands of years use a single secret key for both encryption and decryption. Since the public key system is very time-consuming, and symmetric key algorithms are nearly always much less computationally intensive than asymmetric ones, PGP combines the convenience of a public-key cryptosystem with the efficiency of a symmetric-key cryptosystem.

PGP uses a faster encryption algorithm (session key) to encrypt the message and then uses the public key to encrypt the session key that was used to encrypt the message. Both the encrypted message and the encrypted session key are sent to the recipient. The recipient who first uses his private key to decrypt the session key and then uses that key to decrypt the message [1].

To encrypt a message to Alice using PGP, Bob does the following:

1. Obtains Alice's public key.
2. Generates a fresh, small symmetric key for the data encapsulation scheme.
3. Uses the symmetric key just generated to encrypt the message under the data encapsulation scheme.
4. Uses Alice's public key to encrypt the symmetric key under the key encapsulation scheme.
5. Sends both of these encrypted symmetric key and message to Alice.

To decrypt the cipher text, Alice does the following:

1. Uses her private key to decrypt the symmetric key contained in the key encapsulation segment.
2. Uses this symmetric key to decrypt the message contained in the data encapsulation segment.

The downside of PGP includes:

1. All participants need to have public keys and have made them available to public.

2. It is difficult to find a third party as a certificate authority that everyone trusts.

3. It requires some extra work to keep track of all the valid and revoked certificates within the certificate authority.

4. It is computational expensive to use public key encryption.

While PGP can protect messages, it can also be hard to use in the correct way. Researchers at Carnegie Mellon University published a paper in 1999 showing that most people couldnt figure out how to sign and encrypt messages using the current version of PGP [4]. Eight years later, another group of Carnegie Mellon researchers published a follow-up paper saying that although a newer version of PGP made it easy to decrypt messages, most people still struggled with encrypting and signing messages, finding and verifying other people's public encryption keys, and sharing their own keys [2].

1.2 Thesis Statement

The main objective of this thesis is to design and implement an identity-based cryptography scheme that supports email messages and attachments encryption. It is end-to-end which means encryption happens at sender's end and decryption happens at receiver's end. Emails are in encrypted form whether during network transmission or stored at email servers. It is identity-based because one's public key is based on their email address so it eliminates the public key publishing and obtaining compared to the convention public key cryptosystems.

The thesis is organized in the following sections. Chapter 2 gives a brief introduction to the current cryptography systems such as asymmetric and symmetric

encryption. It also describes the mathematics background these systems are based on. Chapter 3 describes the related work and research done in the fields of elliptic curves, bilinear pairings, and identity-based cryptography. In Chapter 4, a pairing-based encryption key agreement protocol is proposed. Based on such key agreement protocol, we further define the algorithms and workflow for our identity-based encryption system. Chapter 5 gives the implementation details about our identity-based encryption system. Chapter 6 provides the efficiency and security analysis. Finally, Chapter 7 concludes the thesis.

CHAPTER 2

ASYMMETRIC AND SYMMETRIC CRYPTOGRAPHY

Cryptography is the science of using mathematics to encrypt and decrypt data. In the past decades, cryptography has became a cornerstone of computer and communications security. It is in widespread use today, and you are likely to have used it even if you dont realize it. For instance, When you shop on the Internet , cryptography is used to ensure privacy of your credit card number as it travels from you to the online store's server. In electronic banking, cryptography is used to ensure that your checks cannot be forged. Cryptography addresses a wide range of problems, but the most basic problem remains: the classical one of ensuring security of communication across an insecure medium. Its study touches on branches of mathematics that may have been considered esoteric, and it brings together fields like number theory, computational-complexity theory, and probability theory.

2.1 Asymmetric Cryptography

Public-key cryptography, also known as asymmetric cryptography, is a class of cryptographic algorithms which requires two separate keys, one is secret and one is public. Although different, the two parts of this key pair are mathematically linked. The public key can be known by everyone and is used for encrypting messages. Messages encrypted with the public key can only be decrypted in a reasonable amount of time

using the private key. RSA stands for Ron Rivest, Adi Shamir and Leonard Adleman, who first publicly described the asymmetric encryption scheme in 1977 [4]. Public-key cryptography is based on the intractability of certain mathematical problems. Its security is based on assuming that it is difficult to factor a large integer composed of two or more large prime factors. A typical asymmetric encryption scheme consists of three algorithms [4]:

Key generation

The randomized key generation algorithm takes no inputs and returns a pair of keys, the public key and matching private key

- Choose two distinct prime numbers p and q.

  The integers p and q should be chosen at random, and should be of similar bit-length. For 512 bit p and q, it will end up with a 1024-bit key size. And for 1024 bit p and q, it will end up with a 2048-bit key size. A 1024 bit key was considered secure 5 years ago. It is not true today due to increases in computation power. In 2014, RSA key sizes are required to grow from 1024 to 2048 bits.

- Compute n = pq

  n is used as the modulus for both the public and private keys.

- Compute $\varphi(n) = \varphi(p)\varphi(q) = (p-1)(q-1) = n - (p + q - 1)$, where φ is Euler's totient function.

- Choose an integer e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$

  e is known as the public key exponent.

- Calculate d as d $\equiv e^{-1}$ (mod φ(n))

  d is the multiplicative inverse of e (modulo φ(n)) and it is kept as the private key exponent.

The public key consists of the modulus n and the public exponent e which are publicly known. The private key consists of the modulus n and the private exponent d, which must be kept secret. p, q, and φ(n) must also be kept secret because they can be used to calculate d.

Encryption

If Alice wishes to send message M to Bob, Alice does the following:

- Turn M into an integer m, such that 0 <m <n by using a reversible protocol known as a padding scheme.
- Calculate C≡ m^e (mod n) where C is the cipher text.
- Alice then sends the cipher text C to Bob.

Decryption

Bob can recover m from C by using his private key exponent d to compute m≡ c^d (mod n)

The security of public key cryptography is based on the fact that, given the public known n, it appears to be quite hard to recover its 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. For instance, it has been estimated in the 1970s where recovering the prime factors of a 1024-bit number would take a year on a machine costing 10 million US dollar. A 2048-bit number would require several billion times

more work [5]. These estimates are much less today than it have been expected in the 1970s due to the discovery of faster factoring methods as well as steady advances in computing power. The recommended key sizes have accordingly increased over the years. No one knows whether even faster factoring methods might be discovered in the coming years. On the other hand, no one has proved that these cant be discovered. Both aspects remain important research areas in mathematics and public key cryptography.

2.2 Symmetric Cryptography

Symmetric cryptography differs from asymmetric cryptography by using the same key to encrypt and decrypt message. This means that the party encrypting the data and the party decrypting it need to share the same secret key. The distribution of the secret keys requires prior communications or secure channels. In practice, for symmetric cryptography, a secure channel is very difficult to achieve in the absence of an key distribution center (KDC) or a key translation center.

2.2.1 Discrete Logarithm Problem

The discrete logarithm problem is defined as:
given a group G, a generator g of the group and an element h of G, to find the discrete logarithm to the base g of h in the group G.
If G is a multiplicative cyclic group and g is a generator of G, then from the definition of cyclic groups, every element h in G can be written as g^x for some x. The discrete logarithm to the base g of h in the group G is defined to be x . For example, if the group is $(Z_5)^*$, and the generator is 2, then the discrete logarithm of 1 is 4 because 2^4

$\equiv 1 \bmod 5$. The hardness of finding discrete logarithms depends on the groups. For discrete logarithm-based cryptosystems, a popular choice of groups is $(Z_p)^*$ where p is a safe prime number. A safe prime is a prime number which equals 2q+1 where q is a large prime number. This guarantees that p-1 = 2q has a large prime factor so the discrete logarithm problem cannot be solved easily [11].

2.2.2 Diffie-Hellman Key Exchange

When two parties want to use the symmetric encryption scheme, it is assumed they are in possession of a shared secret key. The class of protocols whereby a shared secret becomes available to two or more parties for subsequent cryptographic use are known as key establishment protocols. Key establishment is further subdivided into key transport and key agreement. In key transport, one party creates or obtains a secret value and securely transfers it to the other parties. A key agreement protocol is a key establishment technique in which a shared secret is derived by two (or more) parties as a function of information contributed by, or associated with, each of the parties, ideally, such that no third party can compute the resulting value.

DiffieHellman key exchange is a method to exchange cryptography keys. It allows two parties that have no prior knowledge of each other to establish a shared secret key over an insecure communications channel such as Internet. This shared secret key can then be used to encrypt subsequent communications using a symmetric encryption scheme. The goal of the protocol is to make it impossible to determine the shared secret key for some third party, despite the fact that any of the messages sent between the two end user might be intercepted.

To perform the Diffie-Hellman protocol, let G be a cyclic group with generator g. First Alice chooses a secret number a such that a is a group element of G, and Bob

chooses a secret b such that b is a group element of G. Then Alice sends g^a mod p to Bob, and Bob sends g^b mod p to Alice. Now, Alice computes K_A=$(g^b)^a$=g^{ab} mod p , and Bob computes K_B=$(g^a)^b$=g^{ab} mod p . Thus, the two of them have agreed on a shared secret group element g^{ab} mod p of group G. Alice and Bob can therefore, in theory, communicate privately over a public medium with an encryption method of their choice using the shared secret key $K_{AB} = g^{ab}$ mod p.
protocol handshake

1. Alice $\rightarrow$ Bob: g^a mod p

2. Bob $\rightarrow$ Alice: g^b mod p

The question then is: is it possible for the evesdropper let's say Eve to determine g^{ab} mod p from the communications sent between Alice and Bob? Note that there are only two messages sent between Alice and Bob, namely the two group elements g^a and g^b. Thus the problem becomes: given g^a, g^b and compute g^{ab}. This is the Diffie-Hellman problem, and the assumption that it is hard, even with the help of a powerful computer to conduct millions of trials [16]. One of the reasons for this assumption has to do with the relationship of the Diffie-Hellman problem to the problem of computing discrete logarithms in a cyclic group G. Note that if it were possible to efficiently compute the discrete logarithm a of g^a, then an attacker could easily solve the Diffie-Hellman problem by first computing a from g^a, and then calculating $(g^b)^a$=g^{ab}. Thus, the Diffie-Hellman problem is at least as hard as Discrete Log problem [16].

The Diffie-Hellman protocol can be extended to three parties. For three parties it takes two rounds and six broadcasts to establish a key. The first three message broadcasts are transmitted in the first round and the rest of the protocol broadcasts are transmitted in the next round [10, 11]. As in the two party case, we assume

all participants here agree on suitable parameters g and p in advance. The message flows of this protocol are given [10, 11]:

1. A $\rightarrow$ B, C: g^a mod p
2. B $\rightarrow$ A, C: g^b mod p
3. C $\rightarrow$ A, B: g^c mod p
4. A $\rightarrow$ B, C: g^{ba} mod p and g^{ca} mod p
5. B $\rightarrow$ A, C: g^{ab} mod p and g^{cb} mod p
6. C $\rightarrow$ A, B: g^{ac} mod p and g^{bc} mod p

After the first three broadcasts of above steps, entity A computes g^{ba} mod p and g^{ca} mod p, B computes g^{ab} mod p and g^{cb} mod p, C computes g^{ac} mod p and g^{bc} mod p. Once the protocol is complete, K_A, K_B and K_C are computed by A, B and C respectively. Where K_A, K_B and K_C are all equal to K_{ABC}=g^{abc} mod p . This value can serve as the secret key shared by A, B and C.

2.2.3 Symmetric Encryption Scheme

Symmetric cryptography scheme consists of three algorithms such as asymmetric cryptography- key generation, encryption, and decryption. The key generation algorithm is randomized. It takes no inputs. When it is run, it flips coins internally and uses these to select a key K. Typically, the key is just a random string of some length, in which case this length is called the key length of the scheme. Symmetric cryptography uses much smaller key size compared to asymmetric cryptography. 128 bits is considered sufficient length for symmetric algorithm keys. The Advanced

Encryption Standard published in 2001 uses a key size of 128 bits. It can use keys up to 256 bits but usually only for highly sensitive data.

Once in possession of a shared key based upon some key exchange methods, for example, the DiffieHellman key exchange method. The sender can run the encryption algorithm with key K and input message M to get back a string we call it cipher text. The cipher text can then be transmitted to the receiver. The encryption algorithm may be either randomized or stateful [6]. If randomized, it flips coins and uses those to compute its output on a given input K, M. Each time the algorithm is invoked, it flips coins anew. In particular, invoking the encryption algorithm twice on the same inputs may not yield the same response both times. If the encryption algorithm is stateful, its operation depends on a quantity called the state that is initialized in some pre-specified way. When the encryption algorithm is invoked on inputs K, M, it computes a cipher text based on K, M and the current state. Upon decryption, the receiver, after receiving a cipher text C, will run the decryption algorithm with the same key used to create the cipher text. The decryption algorithm is neither randomized nor stateful.

2.2.4 Mode of Operation

The following schemes rely either on a family of permutation or a family of functions to map input of bit strings of a fixed length to output of the same length. Effectively, the mechanisms spell out how to use the blockcipher to encrypt. We call such a mechanism a mode of operation of the blockcipher [8]. A blockcipher works on units of a fixed size known as a block size, but messages come in a variety of lengths. In practice, one could pad the message appropriately so that the padded message always had length a positive multiple of the block length, and then apply the encryption

algorithm to it. Several padding schemes exist. The simplest one is to add null bytes to the plain text to bring its length up to a multiple of the block size, but care must be taken that the original length of the plain text can be recovered; This is so, for example, if the plain text is a C style string which contains no null bytes except at the end. Slightly more complex padding scheme is the original DES method, which is to add a single one bit, followed by enough zero bits to fill out the block; if the message ends on a block boundary, a whole padding block will be added. Most sophisticated are CBC-specific schemes such as cipher text stealing or residual block termination, which do not cause any extra cipher text, at the expense of some additional complexity [6, 7, 8]. For the below schemes, it is convenient to assume that the length of the message to be encrypted is a positive multiple of a block length associated to the family.

ECB mode

Electronic Code Book (ECB) is a mode of operation for a blockcipher. The message is divided into blocks, and each block is encrypted separately. With the characteristic that each possible block of plain text has a defined corresponding cipher text value and vice versa. It is the simplest of the all encryption modes.

Key generation

Operating it in ECB mode, the key-generation algorithm simply picks a random string of certain length and returns it as a stateless symmetric key, and this key is used for encrypting and decrypting each block.

Encryption Algorithm using Key k operating on message M

E_k(M)

M[1]....M[m] $\leftarrow$ M

for i← 1 to m do

C[i]← $E_k(M[i])$

C← C[1]...C[m]

return C

Decryption algorithm using key k operating on cipher text C

$D_k(C)$

C[1]....C[m]← C

for i← 1 to m do

m[i]← $E_k(C[i])$

M← M[1]...M[m]

return M

CBC mode

In CBC mode, each block of plain text is XORed with the previous cipher text block before being encrypted. This way, each cipher text block depends on all plain text blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.

Encryption algorithm using key k operating on message M

E_k(M)

M[1]....M[m] ← M

C[0] ← $(0,1)^n$

for i← 1 to m do

C[i]← E_k(M[i]⊕C[i-1])

return C

Decryption algorithm using key k operating on cipher text C

$D_k(C)$

C[0]...C[m]$\leftarrow$ C

for i$\leftarrow$ 1 to m do

m[i]$\leftarrow$ $E_k(C[i] \oplus C[i-1])$

return M

There are other modes such as OFB and CTR modes that do not require any special measures to handle messages whose lengths are not multiples of the block size, since these modes work by XORing the plain text with the output of the block cipher. The last partial block of plain text is XORed with the first few bytes of the last keystream block, producing a final cipher text block that is the same size as the final partial plain text block. This characteristic of stream ciphers makes them suitable for applications that require the encrypted cipher text data to be the same size as the original plain text data, and for applications that transmit data in streaming form where it is inconvenient to add padding bytes [7].

CHAPTER 3

ELLIPTIC CURVE AND PAIRING-BASED ENCRYPTION

This chapter gives an introduction to the related works done in the fields of elliptic curves, bilinear pairings, and identity-based encryption.

3.1 Prior Research

This thesis is based on the research and implementation done in the fields of identity-based encryption and elliptic curve. Elliptic curves have been a subject of research for a long time. Initially, researchers were mainly interested in finding points on elliptic curves over infinite fields such as the field of rational or real numbers. The study of curves over finite fields, which at first sight seem to form rather boring abelian groups, aided in finding such points. In 1985, however, elliptic curves over finite fields found an application of their own in cryptography. Koblitz and Miller independently realized that discrete logarithm-based cryptosystems might provide better security when defined on the group of points on an elliptic curve rather than the conventional multiplicative group of a finite field [4]. Or alternatively, they figured, elliptic curves could enable shorter keys, while providing a similar level of security. Since then, a lot of research effort has been put into elliptic curve cryptography and numerous cryptosystems have been proposed.

This is where pairings first come into play. A pairing in this context is a function that takes as input two points on an elliptic curve and outputs an element of some multiplicative abelian group. Furthermore, a pairing satisfies some special properties, the most important of which is bilinearity. Due to these special properties, pairings are hard to construct. The two pairings that are known at present are the Weil pairing and the Tate pairing. In 1993, Menezes et al. discovered that the Weil pairing can be used to attack discrete logarithm-based systems on a certain class of elliptic curves; the so-called MOV-reduction [3]. One year later, Frey used the Tate pairing to describe a similar attack, called the FR-reduction [3]. This crypt-analytic use was the only known application of pairings for a long time. In 2000, however, Joux discovered that pairings can be used as cryptographic building blocks as well. The bilinearity of the pairings enables many cryptosystems with interesting properties. Jouxs discovery spurred an extensive research into new applications based on pairings [3]. The large number of articles on pairing-based cryptography that have appeared since 2000 indicates the tremendous amount of research effort put into this subject.

Undoubtedly the most striking application of pairings is the realization of identity-based cryptography. The fact that public keys are linked to the users identity guarantees the authenticity and therefore takes away the need for certificates as in conventional public-key cryptosystems. However, the implementation of an identity-based encryption scheme remained an open problem until Jouxs discovery of the constructive use of pairings. In 2001, Boneh and Franklin were able to design an efficient identity-based encryption scheme through pairings. Besides identity-based systems, numerous other pairing-based schemes with interesting properties have appeared, such as an efficient key agreement protocol and a signature scheme with short

signature. Identity-Based Encryption takes a breakthrough approach to the problem of encryption key management.

3.2 Additive and Multiplicative Group

Some mathematic groups have an interesting property: all the elements in the group can be obtained by repeatedly applying the group operation to a particular group element. If a group has such a property, it is called a cyclic group and the particular group element is called a generator [15]. A trivial example is the group Z_n, the additive group of integers modulo n. In Z_n, 1 is always a generator:

$1 \equiv 1 \bmod n$

$1+1 \equiv 2 \bmod n$

$1+1+1 \equiv 3 \bmod n$

...

$1+1+1+...+1 \equiv n \equiv 0 \bmod n$

If a group is cyclic, then there may exist multiple generators. For example, we know Z_5 is a cyclic group. The element 1 is a generator for sure. And if we take a look at 2, we can find:

$2 \equiv 2 \bmod 5$

$2+2 \equiv 4 \bmod 5$

$2+2+2 \equiv 6 \equiv 1 \bmod 5$

$2+2+2+2 \equiv 8 \equiv 3 \bmod 5$

$2+2+2+2+2 \equiv 10 \equiv 0 \bmod 5$

So all the group elements 0,1,2,3,4 in Z_5 can also be generated by 2. That is to say, 2 is also a generator for the group Z_5.

For multiplicative group modulo n, Z_n* is cyclic if and only if n is 1 or 2 or 4 or p^k or 2*p^k for an odd prime number p and k $\geq$ 1. So Z_5* is a cyclic group because 5 is a prime number. Actually all the elements in Z_5*, 1,2,3,4 can be generated by 2:

$2^1 \equiv 2 \bmod 5$

$2^2 \equiv 4 \bmod 5$

$2^3 \equiv 8 \equiv 3 \bmod 5$

$2^4 \equiv 16 \equiv 1 \bmod 5$

Not every element in a group is a generator and not every group is cyclic. For example, Z_{12}* is not a cyclic group. The elements in Z_{12}* are: 1,5,7,11. None of the elements can generate the whole group. If Z_n* is cyclic and g is a generator of Z_n*, then g is also called a primitive root modulo n [15].

3.3 Overview of Elliptic Curve

Consider a polynomial Curve1 in two variables X, Y. We are interested in the solutions to Curve1= 0 which describe a curve on a two-dimensional plane. We first observe that if Curve2 is another curve that is an affine transformation of Curve1, that is, if we can linearly transform (e.g. rotate, scale, shear) and then translate Curve1 to obtain Curve2 then a correspondence exists between the solutions to Curve1= 0 and the solutions to Curve2= 0. Knowing the solutions of one allows us to easily compute the solutions of the other, For this reason we consider such curves Curve1 and Curve2 to be equivalent. If every term in Curve1 has combined degree of at most 1, that is, if Curve1= aX + bY + c then Curve1 describes a line. The geometry of lines is too simple to yield anything cryptographically useful. If every term in Curve1 has combined degree at most 2, then Curve1 describes a single line, a pair of lines, a

parabola or a hyperbola. The first two possibilities can be viewed as special cases, occurring when Curve1 is reducible or degenerate in some sense. By adding points of infinity to the plane, we can find affine transformations that change any ellipse, parabola or hyperbola into the unit circle centered at the origin. Intuitively, the two ends of the parabola can be thought of as meeting at a point at infinity, forming a circle, and similarly opposite ends of hyperbolas connect at infinite points. Thus to study degree 2 curves is essentially to study the unit circle, whose geometry is again is too simple for our purposes. However, degree 3 curves, called elliptic curves, are nontrivial. For instance, unlike the previous two cases we cannot transform any elliptic curve in to any other, and elliptic curves have a rich structure well-suited for cryptography [23].

3.3.1 Number of Points on Elliptic Curve

Let F_q be a field for some prime q >3, unless otherwise specified we shall always define curves over a field of prime order and of characteristic greater than three. An elliptic curve E over such a field F_q is an equation of the form

$$\text{E: } Y^2 = X^3 + \text{aX} + \text{b where a, b} \in F_q.$$

Let

$$\delta = 4a^3 + 27b^2, \text{ the discriminant of the cubic in x .}$$

Then

E is singular if $\delta = 0$. The cubic has a repeated root, and nonsingular otherwise. we always consider nonsingular elliptic curves where the cubic has distinct roots.

For any field F_qk define E(F_qk) to be the set of all solutions of E over F_qk , called the finite points along with a special point denoted O, that is called the point at infinity. We write ♯E(F_qk) for the number of elements of E(F_qk). Intuitively, the point O can be thought of as the point where all lines parallel to the Y-axis meet. Mathematically, we solve the curve equation using projective coordinates [18] and one can show that O= (0,1,0) is always a unique infinite solution to the equation.

we quote two well-known theorems here [47]

Theorem(Hasse)

Let

$$\text{t}=q^k+\ 1\ -\ \sharp\text{E}(F_q\text{k}).$$

Then

$$\mid t \mid \leq 2\sqrt{q^k}.$$

Thus the number of points on an elliptic curve in a given field is on the same order as the size of the field. The quantity t is called the trace of Frobenius.

Theorem(Weil)

Let t=q+1 - ♯E(F_q) where q is a prime power. Factor the polynomial x^2 - tx+q as (x - α)(x - β)over C[x]. Then

$$\sharp E(F_q k) = q^k + 1 - (\alpha^k + \beta^k).$$

This last theorem is more practical in the following form. Let t0= 2, Let t1=q+ 1 - ♯E(F_q). Define tn recursively by

$$t_n = t1t_{n-1} - qt_{n-2}.$$

then

$$\sharp \mathrm{E}(F_q\mathrm{k}) = q^k + 1 - t_k$$

For example, consider the curve E given by $Y^2 = X^3 + \mathrm{X} + 6$ over F_{19} , an example used by Balasubramanian and Koblitz [19]. There are 18 points

(0,5),(4,6),(2,4),(3,6),(14,3),(12,13),
(18,2),(10,3),(6,0),(10,16),(18,17),(12,6),
(14,16),(3,13),(2,15),(4,13),(0,14), O

thus the trace of Frobenius t= 2. According to the Theorem(Weil), over F_{19}, we have

$$\sharp \mathrm{E}(F_{19}2) = 19^2 + 1 - t_2$$

where t_2= 2*2-19*2 =-34, thus $\sharp \mathrm{E}(F_{19}2) = 396$

3.3.2 Group Operations over Elliptic Groups

Some special addition operations are defined over elliptic curves, and this with the inclusion of a point O, called point at infinity . If three points are on a line intersect an elliptic curve, the their sum is equal to this point at infinity O which acts as the identity element for this addition operation. For special addition operations, check Figure 3.1

Let the points P=(x_1,y_1) and Q=(x_2,y_2) be in the elliptic group E_p(a,b) and O is the point at infinity. the rules for addition over the elliptic group E_p(a,b) are:

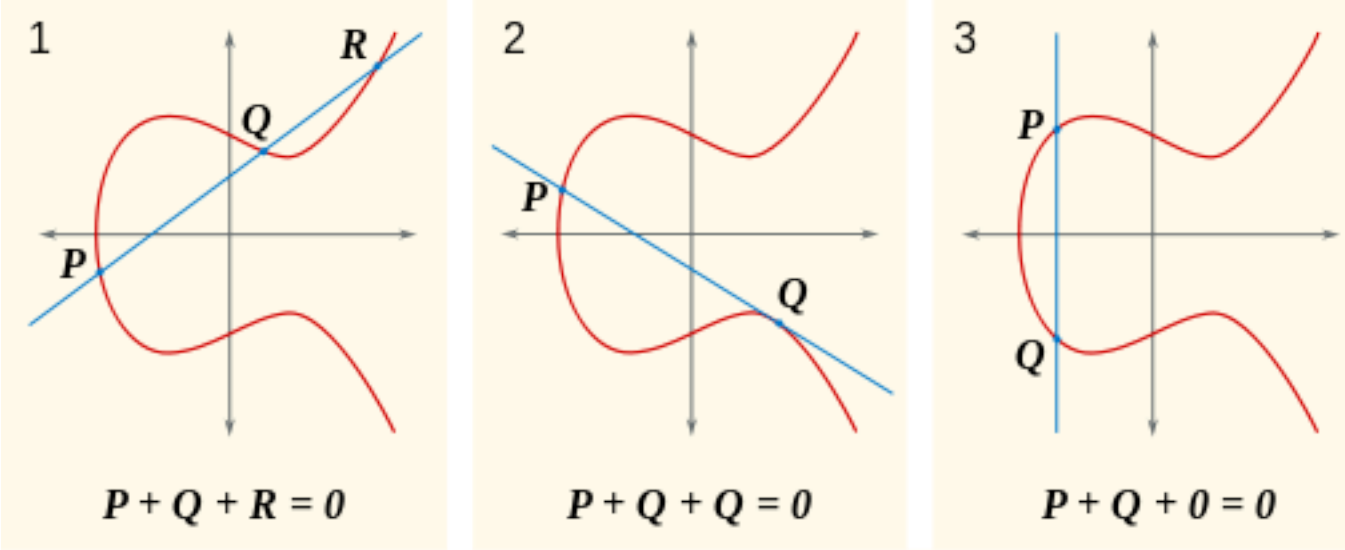


Figure 3.1: Special Group Addition Operations

1. P + O=O + P=P

2. if x_2=x_1 and y_2= -y_1, that is P=(x_1,y_1)and Q=(x_2,y_2)=(x_1,-y_1)= -P, then P+Q=O

3. If Q $\neq$ P, then the sum P+Q=(x_3,y_3) is givenby

   x_3=λ^2- x_1 -x_2 mod p

   y_3=$\lambda(x_1 - x_3) - y_1$ mod p

   where

   λ= $\frac{y_2-y_1}{x_2-x_1}$ if P $\neq$ Q

   and

   λ= $\frac{3x_1^2+a}{2y_1}$ if P = Q

Let's see a simple example:

Let P=(3,10)$\in$ E_{23}(1,1). Then 2P=(x_3,y_3) is equal to:

2P=P+P=(x_1,y_1)+(x_1,y_1)

Since P=Q and x_2=x_1, then values of λ,x_3 and y_3 are given by:

$\lambda = \frac{3x_1^2+a}{2y_1}$ mod p

$=\frac{3(3^2)+1}{2X10}$ mod 23

$=\frac{5}{20}$ mod 23

= 41 mod 23

= 6

$x_3 = \lambda^2 - x_1 - x_2$ mod p

$=6^2$-3-3 mod 23

= 30 mod 23

= 7

$y_3 = \lambda(x_1 - x_3) - y_1$ mod p

=6(37)10 mod 23

=34 mod 23

= 12

Therefore 2P=(x_3, y_3)=(7,12). The most expensive step is the division in the computation of λ.

The multiplication kP is obtained by doing the elliptic curve addition operation k times by following the same additive rules.

3.4 Overview of Bilinear Pairing

Let G_1, G_2, and G_t be cyclic groups of the same order, more specific, let G_1, G_2 be additive groups and G_t be a multiplicative group, all of prime order r. Let g_1 be a

generator of G_1, and g_2 be a generator of G_2, A bilinear map or a bilinear pairing is an efficiently computable function $\hat{e} : G_1 \times G_2 \rightarrow G_t$ such that satisfies the following conditions:

1. **bilinearity**

   it holds that

   $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$ for all a, $b \in Zr$(the ring of integers modulo r).

2. **non-degeneracy**

   $\hat{e}(g_1, g_2) \neq 1$.

3. **computability**

   $\hat{e}$ can be efficiently computed.

the tuple $(r, g_1, g_2, G_1, G_2, G_t)$ is called asymmetric bilinear pairing if G_1 and G_2 are different groups. On the other hand, if $G_1 = G_2 = G$ and g is a generator of G then the tuple (r, g, G, G_t) is called symmetric bilinear pairing. If G is cyclic, the map e will be commutative since we have $\hat{e}$(S,T)=$\hat{e}$(T,S) for any S,T in G. This is because for a generator g in G , there exist integers a and b such that S= g^a and T=g^b. Therefore $\hat{e}$(S,T) = $\hat{e}$(g^a,g^b) = $\hat{e}$(g,g)^{ab}= $\hat{e}$(g^b,g^a) = $\hat{e}$(T,S). In the symmetric pairing the order of G and G_t need not to be prime. A composite group order is useful for some cryptosystems [21], but we must be aware that in this case even if g_1 and g_2 have order r, $\hat{e}$(g_1, g_2) may not be a generator of G_t, but rather a generator of some subgroup of G_t whose order is a factor of r. When r is composite, it is incorrect to say all nondegenerate pairings are equivalent up to a constant. Indeed, if d greater than 1 is a divisor of r and g_1, g_2 are generators of G_1, G_2 respectively, then a bilinear map that takes (g_1, g_2) to a dth root of unity is still nondegenerate [21]. Since there

is more than one choice for d for a composite r, these maps cannot be equivalent up to a constant. They are however still completely determined by $\hat{e}(g_1, g_2)$ where g_1, g_2 are generators. This also means that $\hat{e}(g_1, g_2) = 1$ does not imply that at least one of g_1, g_2 is the identity element, in contrast to the case when r is prime, if $\hat{e}(g_1, g_2) = 1$, it implies at least one of g_1, g_2 is the identity element. Therefore, some care is needed when dealing with composite r. Facts about the pairing that are true for prime r do not always carry over.

3.4.1 Tate Pairing

The most commonly used pairings for cryptography are Weil pairing and Tate pairing. In this section, we will show how to compute pairing using the Miller algorithm. Since we are using Tate pairing for our implementation, we will skip Weil pairing and focus on Tate pairing. The definitions and algorithms presented in following subsections can be found on many pairing-based cryptography articles.

Tate Pairing Definition

Let E be an elliptic curve containing n points over a field F_q. Let G be a cyclic subgroup of E(F_q) of order r with r, q coprime. Let k be the smallest positive integer such that r divides $q^k - 1$. F_qk is the smallest extension of F_q containing the rth roots of unity. The Tate pairing [23]

$$\hat{e} : E[r] \cap E(F_qk) \text{ X } E(F_qk)/rE(F_qk) \rightarrow F_qk^*/F_qk^{*r}$$

is defined as follows

Let f_P be a rational function with divisor $(f_P) = (P)^r$. Choose an R $\in E(F_qk)$ such

that R $\neq$ P, P - Q, O, -Q. The define

$$\text{f(P, Q)} = \frac{f_P(Q+R)}{f_P(R)}$$

It can be shown that the above value is independent of the choice of R and:

- f(aP, bQ) = $\hat{e}(P,Q)^{ab}$ for all P, Q, a, b.

- f(P, Q) = 1 for all P if and only if Q=O.

- f(P, Q) = 1 for all Q if and only if P=O.

- $f(\phi(P), \phi(Q)) = f(P,Q)^q$ for all P, Q $\in E[r]$, where ϕ denotes the Frobenius map.

3.4.2 Using Miller's Algorithm to Calculate Tate Pairing

For any points U, V, let $L_{U,V}$(X, Y) be an equation for a line through U and V, let T_U(X, Y) be an equation for the tangent through U. let V_U(X, Y) be an equation for a vertical line through U(e.g.X -x, where U= (x, y)). We now consider the Tate pairing. We wish to compute f_P(Q) where f_P has divisor $(P)^r$. Thus define the intermediate functions f_k by [23]

$$(f_k) = (P)^k/(\text{kP})$$

we have $(f_r) = (f_P)$. we can show

$$(f_k) = \frac{L_{iP}}{V_{(i+1)P}} \text{ for i=1 to i=k-1}$$

since

$$(f_k) = \frac{(P)(P)}{(2P)} \cdot \frac{(P)(2P)}{(3P)} \ldots \frac{(P)((k-1)P)}{(kP)}$$

We find

$$(f_{2k} = (f_k^2 T_{kP} / V_{2kP})$$

which can be shown with direct calculation:

$$\frac{(P)^{2k}}{(2kP)} = \frac{(P)^{2k}}{(kP)^2} \cdot \frac{(kP)^2(-2kP)}{(2kP)(-2kP)}$$

and similarly we can show

$$(f_{k+1} = (f_k L_{kP,P} / V_{(k+1)P})$$

leading to the following algorithm that computes f_P(Q) given points P, Q(where P has order r). Let the binary representation of $r_t ... r_0$

Millers algorithm for Tate pairing: $f_P(Q)$ [25]

$x \leftarrow 1$

$Z \leftarrow P$

for i $\leftarrow$ *t-1,...,0 do*

  $x \leftarrow x^2 T_z(Q) / V_{2z}(Q)$

  $Z \leftarrow 2Z$

  if r_i*=1 then*

    $x \leftarrow x L_{Z,P}(Q) / V_{Z+P}(Q)$

    $Z \leftarrow Z+P$

end if

end for

When the algorithm finishes we have x=f_r(Q) (and Z= rP=O).

3.5 Identity-based Encryption Using Pairing

In 1984, Shamir[29] asked for a public key encryption scheme in which the public key can be an arbitrary string. Shamir's original motivation for identity-based encryption was to simplify certificate management in public key encryption systems. The concept was named identity-based public key cryptography (ID-PKC) by Shamir and has subsequently also became known as identifier-based public key cryptography in some articles.

There is a new role needed in identity-based public key cryptography, and is named as Private Key Generator (PKG) or Key Generation Center (KGC)to reflect this. The role of the PKG is to issue the private key corresponding to the public key (derived from the identifier IDA) to entity A. This issuing only occurs after entity A is authenticated by the PKG. To generate private keys, the PKG makes use of a master-key which must be kept secret. The requirement to have an authentic CA public key is replaced by the requirement to have authentic PKG parameters.
In his 1984 paper[29], Shamir was only able to construct a concrete identifier-based public key signature scheme. Developing a concrete satisfactory identifier-based public key encryption scheme remained an open problem. There have been several proposals for IBE schemes [30, 31]. However, none of these are fully satisfactory . Some solutions require that users not collude [31]. Other solutions require the PKG to spend a long time for each private key generation request. Some solutions require

tamper resistant hardware [30]. It is fair to say that until the Boneh/Franklin scheme is proposed by Dan Boneh and Matthew K. Franklin in 2001 [32], constructing a usable IBE system was an open problem.

CHAPTER 4

AN IDENTITY-BASED ENCRYPTION SYSTEM DESIGN

The following two chapters are the main contributions to the thesis. The first chapter introduces a fully functional identity-based encryption scheme design. The second chapter describes the implementation of the project.

In our IDE system, when Alice sends email to Bob, she simply encrypts the message and attachment using a key let's call it keyAB, which is derived from Bob's public key and Alice's private key. Bob's public key is generated from Bob's email address so there is no need for Alice to obtain Bob's public key certificate. When Bob receives the message, he derives a decryption key let's call it keyBA, which can be calculated from Bob's private key and Alice's public key. Based on some pairing properties, keyAB actually equals keyBA. Theoritically, such IBE system can be built based on any bilinear map $\hat{e} : G_1 X G_2$ as long as variant of the computational Diffie-Hellman problem in G_1 is hard.

4.1 Elliptic Curve and Pairing Construction

There are two mathematical settings to choose from when implementing cyclic groups: finite fields and elliptic curves. For finite fields, one picks a prime n and uses a subgroup G of Zn of prime order r, so the group operation is field multiplication. Note an RSA cryptosystem arises when n is instead chosen to be the product of two

large primes p, q, in which case computations are still possible in Zn even if its order is unknown [22]. For elliptic curves, one takes an elliptic curve E over some finite field K and takes some subgroup G of the group of points E(K) with prime order r, so the group operation is point addition.

4.1.1 Elliptic Curve Construction

Let E be an elliptic curve over a field K . The group operation means that every point on E generates a cyclic group G . Then we can use cyclic group cryptography provided that its order is prime, that the basic operations, namely group operation, inversion, hashing, are efficient, and that problems such as discrete log are difficult. On elliptic curves, the construction of a point of order r, or a factor of r, from some given point P ∈E(K)can be accomplished similarly by multiplying P by n/r where n = ♯E(K). This is because, let n= ♯E(K), then from Abelian group theory for any prime r dividing n, there exists a point P∈E(K) of order r. and furthermore, if r^2 does not divide n then there is exactly one subgroup G of E(K) of order r. This suggests the following procedure for implementing any cryptographic scheme based on cyclic groups of prime order [26]:

1. Choose any curve E(K) and somehow work out n= ♯E(K).

2. Find a prime r divides n, such that r^2 does not divide n. We shall work in the unique cyclic subgroup G is subgroup of E(K) of points of order r.

3. When a random group element of G is required, first choose a random point of E(K) and then multiply by n/r. Similarly, when hashing to a point of G, first hash to a point in E(K) and then multiply by n/r .

4. Other operations are straightforward: every time a group operation is required, we perform a point addition. To find an inverse of a group element, we negate the y-coordinate of a point. When an exponentiation is called for we carry out a point multiplication.

If a pairing is desired, we must seek out elliptic curves whose orders satisfy various conditions. As a result, instead of choosing a curve first and counting the number of points and hoping for a large prime factor r of n, we must use families of curves where the size of the group is known in advance and has the requisite properties where ♯E(K) is always easy to determine and furthermore, E(K) is always cyclic. The plane curve over a finite field (rather than the real numbers) which consists of the points satisfying the equation $y^2 = x^3$ + ax + b, along with a distinguished point at ∞ is suitable for pairing-based cryptography.

4.1.2 Finding a Random Point on an Elliptic Curve

Let E: $Y^2 = X^3$+aX+b be an elliptic curve over a field K . There always exists an unique infinite solution, namely O. We describe a simple method for finding the finite points of E. For any x ∈ K, we may attempt to solve $Y^2 = X^3$+aX+b for Y by finding a square root of the right-hand side. We momentarily postpone describing the details of square root algorithms. For now, assume we can find square roots. When solutions for Y do exist for a given x, we have found exactly two points, one for each square root, except in the rare case when the point lies on the X-axis, which can happen in at most three places. Also, recall from an above theorem that the size of K is roughly the same as the number of points on E(K). Combining these two facts shows that for approximately half of the choices for x ∈ K ,a square root exists and we can solve E to find a point. Thus we have a fast method of finding random points on E:

1. Choose x $\in$ K at random.

2. Solve $Y^2 = X^3$+aX+b for Y. If there are no solutions then go to the previous step.

3. Flip a coin to decide which solution of Y to use.

Of course, it is impossible to choose the point at infinity with this method, and points that lie on the X-axis have a slightly higher probability of been picked than other points. For cryptography this is of no concern since the point of infinity is usually unwanted, and the probability of finishing at a point with zero Y-coordinate is negligible since there are at most three of them. Moreover, it is often unimportant which square root is chosen. If one insists on choosing all points of E(K) uniformly, one could simply add a step before choosing x . Let n= ♯E(K). Then with 1/n probability, choose O or one of the points lying on the X-axis, otherwise proceed with the above algorithm, except in the second step, we also go back to the first step if the only solution is Y= 0.

Before attempting to find a square root of a given element x ∈K, we can check that one actually exists first. When K has prime order, one can compute the Legendre symbol before attempting to square root x. More generally it can be checked that X^2 - x is reducible. Alternatively, one can omit the check, proceed with a square root algorithm, and compare the square of the output with x: if there is a mismatch then x is not a square after all. It remains to describe how to take square roots. For a field of prime order one can use the Tonelli-Shanks algorithm to compute square roots [18, 20]. For a general finite field, one must use a more complex algorithm. Perhaps the simplest of these Legendres method which can be viewed as factoring X^2 -x. Faster algorithms exist, though sometimes require precomputation.

4.1.3 Hashing to Points

Finding points by choosing an X-coordinate and solving for Y suggests an efficient algorithm for hashing to a point in E . The input is hashed to some x ∈ K, and then a corresponding y is sought. On failure, a new x -coordinate is deterministically generated from x, and again we attempt to solve E for y. Repeating this process as many times as necessary eventually yields a valid point (x, y)∈E(K).

4.1.4 Pairing Construction

Constructing a pairing is a balancing act. F_q must be large enough so that E(F_q) can foil generic discrete log attacks, while F_qk must be large enough to resist finite field discrete log attacks. At the same time, F_q and F_qk should be as small as possible to minimize time and space usage [23]. More precisely:

1. r must be a large enough prime so that generic discrete logarithm attacks in a group of order r are ineffective. Since q is about the size of ♯E(F_q), this places a similar lower bound on q

2. q ought to be as small as possible, so that computations in F_q are as fast as possible.

3. q^k must be large enough so that finite field discrete logarithm attacks in F_qk are ineffective. Also q should not have low Hamming weight [33, 34], nor be a power of a small prime [35].

4. q^k must be small enough so that operations in F_qk are efficient. All other things being equal, q^k should be small as possible so that operations are as fast as possible.

Observe the first three statements are true for any cryptographically using elliptic curve, not just for pairing-based cryptography. The last condition, requiring F_qk to be small enough to compute on, is responsible for much of the difficulty in pairing-based cryptography research, as finding curves with small k is nontrivial.

Currently it is acceptable to have an 160-bit r. As for q^k, 1024 bits is adequate for many applications, and calculations in fields of this size can certainly be performed.

4.2 Scheme Design

Our IBE scheme is specified by several algorithms:

System parameters

The curve generator takes two primes r and q, of length 160 and 512 bit respectively, returns a master key and a curve y^2=x^3+x over the field F_q for some prime q=3 mod 4. A pairings are constructed on top of the returned curve. Both G_1 and G_2 are the group of points E(F_q) so this pairing is symmetric. The order r is some prime factor of q+1. The system parameters such as the curve and pairing are public known.

User key generate

The algorithm takes the system paramater, master-key, and an arbitrary string ID $\in 0, 1^*$ as input, and returns a public key and private key pair to an user. The ID is an email address of the user in this case.

Encryption

Takes as input the encryption key KeyAB, plain text message M and returns a cipher text C.

Decryption

Takes as input the decryption key KeyBA, a cipher text C, and returns the plain text

message M.

4.3 Key Agreement Protocol Design

Prior to any secured communication, users must set up the details of the cryptography. Secret-key (symmetric) cryptography requires the initial exchange of a shared key in a manner that is private so it does not reveal to any eavesdropping party what key has been agreed upon. Exchanging of such a key was extremely troublesome until the Diffie-Hellman key exchange protocol was published in 1975. Diffie-Hellman key exchange protocol makes it possible to exchange a key over an insecure communications channel. If you recall from chapter 2, Diffie-Hellman does not specify any prior agreement or subsequent authentication between the participants. In other words, DiffieHellman key exchange protocol does not provide authentication of the parties, and is thus vulnerable to man-in-the-middle attacks. In order to defeat such attacks, a widely used mechanism is the use of digitally signed keys. When Alice and Bob have a public-key infrastructure, they may digitally sign an agreed Diffie-Hellman key, or sometimes, such Diffie-Hellman keys are signed by a certificate authority. The main goal of this thesis is to design a key agreement protocol and build an identity-based encryption system on top of the protocol. The key agreement protocol should enable two or more parties agreeing on a shared secret key, but any eavesdropping party can not possibly know what key has been agreed upon. In addition, if the agreed key is derived from user's identity then there is no need of public key infrastructure or certificate authority to sign such key.

4.3.1 Key Agreement for Emails with One Recipient

Encryption

When Alice sends email to Bob, she does the following:

1. Picks a random number r and then computes X_A=r*H(ID_A) where H(ID_A)is the hash of Alice's email address which also is Alice's public key.

2. Computes the encryption key

$$\text{KeyAB} = \hat{e}(S_A, \text{H}(ID_B))^r \text{ XOR } \hat{e}(S_A, \text{H}(ID_B))$$

   where S_A is Alice's private key, H(ID_B) is Bob's public key, and r is the random number from step1. $\hat{e}$ is the pairng function which calculated using Miller algorithm.

3. Encrypts message using encryption key KeyAB.

4. Alice sends the cypher text C along with X_A to Bob.

Decryption

When Bob receives the encrypted email from Alice, he does the following to get back the original message

1. Computes decryption key

$$\text{keyBA} = \text{e}(X_A, S_B) \text{ XOR } \text{e}(\text{H}(ID_A), S_B)$$

   where X_A is the multiplication of Alice's email address and the random number r that Alice included in the email, S_B is Bob's private key, and ID_A is Alice's public key.

The KeyBA that Alice used to encrypt the original message actually equals to KeyBA so Bob can use the derived KeyBA to decrypt the email:

KeyBA = e(X_A, S_B) XOR e(H(ID_A), S_B)

= e(rH(ID_A), SH(ID_B)) XOR e(H(ID_A), SH(ID_B))

= e(SH(ID_A), H(ID_B))^r XOR e(SH(ID_A), H(ID_B))

= e(S_A, H(ID_B))^r XOR e(S_A, H(ID_B))

= KeyAB

4.3.2 Key Agreement for Group Emails

I would like to thank my advisor, Dr. Yeh, for sharing his paper: "P2P email encryption by an identity-based one-way group key agreement protocol". The theories presented by Dr. Yeh are the basis to the key agreement protocol for group emails [46].

Group email encryption

In any email application, a sender should be able to email a message to a group of $n > 0$ receivers.

Let ID_0 be the email sender's identity and let ID_i, for $i = 1, 2, \ldots, n$, denote the identity for each email receiver in the group of n people. When the sender sends email to the group, he does the following:

1. Picking a random number $r \in Z_q^*$ and computes

$$x_i = e(S_0, rP_i) \in G_2, \forall i = 0, 1, 2, \ldots, n \tag{4.1}$$

where S_0 is the private key of the email sender ID_0 and $P_i = H(ID_i)$ is the public key of the email receiver ID_i.

2. The email sender generates the encryption key K by computing

$$K = \oplus_{\forall i=0,1,\ldots,n}(x_i) \tag{4.2}$$

3. The email sender also computes $y_i, \forall i = 1, 2, \ldots, n$, as follows.

$$y_i = \oplus_{\forall j \neq i}(x_j) \tag{4.3}$$

or in other words,

$$y_i = x_0 \oplus x_1 \oplus \ldots \oplus x_{i-1} \oplus x_{i+1} \oplus \ldots \oplus x_n \tag{4.4}$$

4. The email sender encrypts the email using the secret key K and then sends the encrypted email out along with $(r, y_1, y_2, \ldots, y_n)$.

Group email decryption

Upon receiving the email from ID_0, each recipient ID_i can compute the secret key K by using y_i (attached in the email) and the public key $P_0 = H(ID_0)$ of the email sender ID_0 as below.

$$K = y_i \oplus e(rP_0, S_i) \tag{4.5}$$

since

$$\begin{aligned} y_i \oplus e(rP_0, S_i) &= y_i \oplus e(rP_0, sP_i) \\ &= y_i \oplus e(sP_0, rP_i) \\ &= y_i \oplus e(S_0, rP_i) \\ &= y_i \oplus x_i \\ &= (\oplus_{\forall j \neq i}(x_j)) \oplus x_i \end{aligned}$$

$$= K$$

Group email examples

Two email receivers

Assume an email sender ID_0 would like to send an email to two email receivers ID_1 and ID_2.

1. ID_0 picks a random number r and computes

$$\begin{cases} x_0 = e(S_0, rP_0) \\ x_1 = e(S_0, rP_1) \\ x_2 = e(S_0, rP_2) \end{cases}$$

2. ID_0 generates the encryption key

$$K = x_0 \oplus x_1 \oplus x_2$$

3. ID_0 computes

$$\begin{cases} y_1 = x_0 \oplus x_2 \\ y_2 = x_0 \oplus x_1 \end{cases}$$

4. ID_0 encrypts the email using the key K and sends (r, y_1, y_2) along with the email.

5. For the two recipients, ID_1 computes

$$\begin{aligned} y_1 \oplus e(rP_0, S_1) &= x_0 \oplus x_2 \oplus e(rP_0, SP_1) \\ &= x_0 \oplus x_2 \oplus e(sP_0, rP_1) \end{aligned}$$

$$= x_0 \oplus x_2 \oplus e(S_0, rP_1)$$
$$= x_0 \oplus x_2 \oplus x_1$$
$$= K$$

and ID_2 computes

$$\begin{aligned} y_2 \oplus e(rP_0, S_2) &= x_0 \oplus x_1 \oplus e(rP_0, SP_2) \\ &= x_0 \oplus x_1 \oplus e(sP_0, rP_2) \\ &= x_0 \oplus x_1 \oplus e(S_0, rP_2) \\ &= x_0 \oplus x_1 \oplus x_2 \\ &= K \end{aligned}$$

Thus, each of the email recipients can derive the same key K that was originally generated by the email sender ID_0.

Three email receivers

Assume an email sender ID_0 would like to send an email to three email receivers ID_1, ID_2 and ID_3.

1. ID_0 picks a random number r and computes

$$\begin{cases} x_0 = e(S_0, rP_0) \\ x_1 = e(S_0, rP_1) \\ x_2 = e(S_0, rP_2) \\ x_3 = e(S_0, rP_3) \end{cases}$$

2. ID_0 generates the encryption key

$$K = x_0 \oplus x_1 \oplus x_2 \oplus x_3$$

3. ID_0 computes

$$\begin{cases} y_1 = x_0 \oplus x_2 \oplus x_3 \\ y_2 = x_0 \oplus x_1 \oplus x_3 \\ y_3 = x_0 \oplus x_1 \oplus x_2 \end{cases}$$

4. ID_0 encrypts the email using the key K and sends (r, y_1, y_2, y_3) along with the email.

5. For the three recipients, ID_1 computes

$$\begin{aligned} y_1 \oplus e(rP_0, S_1) &= x_0 \oplus x_2 \oplus x_3 \oplus e(rP_0, SP_1) \\ &= x_0 \oplus x_2 \oplus x_3 \oplus e(sP_0, rP_1) \\ &= x_0 \oplus x_2 \oplus x_3 \oplus e(S_0, rP_1) \\ &= x_0 \oplus x_2 \oplus x_3 \oplus x_1 \\ &= K \end{aligned}$$

and ID_2 computes

$$\begin{aligned} y_2 \oplus e(rP_0, S_2) &= x_0 \oplus x_1 \oplus x_3 \oplus e(rP_0, SP_2) \\ &= x_0 \oplus x_1 \oplus x_3 \oplus e(sP_0, rP_2) \\ &= x_0 \oplus x_1 \oplus x_3 \oplus e(S_0, rP_2) \\ &= x_0 \oplus x_1 \oplus x_3 \oplus x_2 \\ &= K \end{aligned}$$

and ID_3 computes

$$\begin{aligned} y_3 \oplus e(rP_0, S_3) &= x_0 \oplus x_1 \oplus x_2 \oplus e(rP_0, SP_3) \\ &= x_0 \oplus x_1 \oplus x_2 \oplus e(sP_0, rP_3) \end{aligned}$$

$$= x_0 \oplus x_1 \oplus x_2 \oplus e(S_0, rP_3)$$
$$= x_0 \oplus x_1 \oplus x_2 \oplus x_3$$
$$= K$$

Thus, all three email recipients can derive the same key K that was originally generated by the email sender ID_0.

CHAPTER 5

AN IDENTITY-BASED ENCRYPTION SYSTEM IMPLEMENTATION

Our IBE system uses JavaMail API to communicate with an email server. It modifies the Java Pairing-Based Cryptography (JPBC) library for system parameter setup, and pairing calculation. The JPBC library is a wrapper for the PBC library. The PBC library is an open source C library that provides an abstract interface to a cyclic group with a bilinear pairing, insulating the programmer from mathematical detail. Some known example schemes that build on top of PBC library including BLS signatures [36], Joux tripartite Diffie-Hellman [37], and Boneh-Lynn-Shacham short signature [38].

5.1 Javamail

The JavaMail API includes the javax.mail package and other sub-packages. The javax.mail package defines classes that are common to all mail systems. The javax.mail.internet sub-package defines classes that are specific to mail systems based on Internet standards such as MIME, SMTP, POP3, and IMAP. Its message and MIMEType, multipart classes are used to read, send emails and attachments. See Figure 5.1 for

Javamail Message and Multipart class hierarchy.

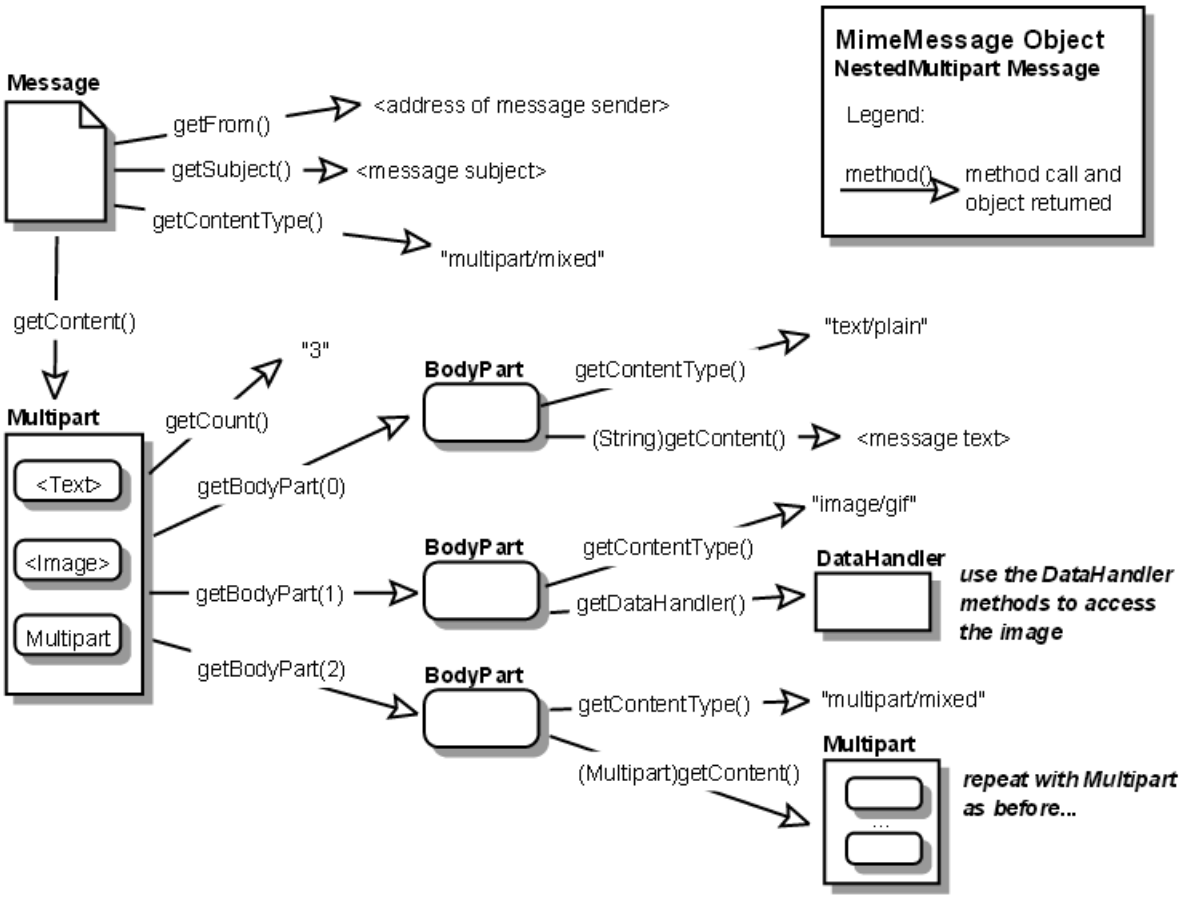


Figure 5.1: Javamail Message and Multipart Class Hierarchy

5.1.1 Session and Properties

A mail Session object manages the configuration options and user authentication information used to interact with messaging system. Typically the Properties object contains user-defined customization in addition to system-wide defaults. An application can use the system Properties object via the System.getProperties method. The

call to the getInstance method creates a new Session object. A mail-enabled client uses the Session object to retrieve a Store or Transport object in order to read or send mail. Typically, the client retrieves the default Store or Transport object based on properties loaded for that session.

5.1.2 Message Class

The Message class is an abstract class that defines a set of attributes and a content for a mail message. The Message class implements the Part interface. The Part interface defines attributes that are required to define and format data content carried by a Message object. The Message class adds From, To, Subject, Reply-To, and other attributes necessary for message routing via a message transport system. When contained in a folder, a Message object has a set of flags associated with it. JavaMail provides Message subclasses that support specific messaging implementation.

5.1.3 MIME Type and Multipart Class for Attachments

The MIME RFCs 2045, 2046 and 2047 define message content structure by defining structured body parts, a typing mechanism for identifying different media types, and a set of encoding schemes to encode data into mail-safe characters. Java Mail's Multipart class implements multipart messages. A multipart message is a Message object where the content-type specifier has been set to multipart. The Multipart class is a container class that contains objects of type Bodypart. A Bodypart object is an instantiation of the Part interface it contains either a new Multipart container object, or a DataHandler object. We use multipart class for email attachments.
See Appendix A for implementation of using JavaMail to send and read messages.

5.2 Date Type and Functions

The implemented data types and functions can be categorized as following:

- element

  elements of an algebraic structure, fields, groups, and rings. Element arithmetic functions including addition , multiplication and other operations in rings and fields. For groups of points on an ellitpic curve, such as the G_1 and G_2 groups associated with pairing, both addition and multiplication represent the group operation.

- pairingParameters

  the pairingParameters are constructed from curves. An interface of Generator class provides a generate method and returns an instance of pairing parameters. A PairingParameters instance that maps the pairing parameters to specific values that can be accessed by calling specialized methods.

- pairing

  pairing can be initialized from pairingParameters. It is a bilinear map that takes two elements as input, one from G_1 and one from G_2, and outputs an element of G_t. One examples of pairing functions including pairingissymmetric, returns true if G_1 and G_2 are the same group, returns false if G_1 and G_2 are different group. Other pairing functions including GetG1() which returns the G_1 group and GetGT() returns the G_t group which is the group of rth roots of unity.

5.3 Server and Client Algorithems

When a KGC sets up, it runs the server algorithms to generate an elliptic curve for the pairing, and a master key that can be used for users' public and private key generation. Please refer to Appendix A for detailed implementations.

Type A curves

Let q be a prime satisfying q= 3 (mod 4). Let E be the curve y^2=x^3+ax for any a. Then we have ♯E(F_q) =q+ 1, and ♯E(F_{q^2}) = $(q+1)^2$. For any odd r dividing q+ 1 we have that G=E(F_q)[r] is cyclic and has embedding degree k= 2 [39]. consider the distortion map [45]

$$\psi(\mathrm{x},\ \mathrm{y}) = (\mathrm{x},\ \mathrm{iy})$$

Then ψ maps points of E(F_q) to points of E(F_{q^2})/E(F_q). Thus if f denotes the Tate or Weil pairing, then defining e:G X G $\rightarrow$ F_{q^2} by

$$\mathrm{e}(\mathrm{P},\ \mathrm{Q}) = f(\mathrm{P},\psi(\mathrm{Q}))$$

gives a bilinear nondegenerate map.

The Setup for this type of pairing for a cryptosystem can be done as follows [23]

1. An order r is chosen, large enough to avoid generic discrete logarithm attacks

2. Recall we require finite field discrete logarithm attacks on F_{q^2} to be impractical. Thus we randomly generate h where h is a multiple of four and sufficiently large to guarantee $(hr)^2$ is big enough to resist finite field attacks. For example, if r is 160 bits long, and we want q^2 to be about 1024 bits long, then h must be about 352 bits long.

3. Next it is checked that q=hr - 1 is prime. We have q= 3 mod 4 by choice of h. If q is not prime, we go back to the previous step and choose another h.

If h is constrained to be a multiple of 3 as well, then cube roots are easy to compute in F_q: for all x $\in F_q$ we see $x^{-(q-2)/3}$ is the cube root of x. Observe cube roots are unique since each element is a cube.

Master key

A master key s is returned as part of the setup algorithm. It can be either 256 or 512 bit long. The master key is used to generate all users private keys. It should be kept secret and only known to the "Private Key Generator Server" (PKG).

Hash function

The hash function takes a string which is an email address as input. First, it performs the standard hash algorithm on it, and then maps the output byte array from the standard hash function to an element in the group G.

The hash algorithm finding points by choosing an X-coordinate and solving for Y in E. The input is hashed to some x $\in$ K, and then a corresponding y is sought. On failure, a new x -coordinate is deterministically generated from x, and again we attempt to solve E for y. Repeating this process as many times as necessary eventually yields a valid point (x, y)$\in$E(K)

Key pair generation

When a user registers with a PKG server, he gives the server his email address. The PKG server runs the key generation algorithm which takes input the email address and returns a private and public key pair to the user. The public key keyPublic is an element of G_1 that mapped to the hash value of user's email address. The private key keyPrivate is the result of pairing multiplication between the master key and the public key.

Two main client side algorithms are encryption and decryption. After an encryption key KeyAB is derived, Alice passes the key and original message to the Advanced Encryption Standard (AES) algorithm, and this algorithm returns Alice an encrypted message also called cipher text. Upon Bob received the cipher text, he computers the decryption key KeyBA and then pass KeyBA along with the cipher text to the AES algorithm, and this algorithm returns the original message back to Bob. see Appendix B for details of AES.

5.4 Java Projects and Packages

The implementation of our IDE system is organized into 3 main java projects and each project including several packages

- elliptic-curve project
- id-based-encryption-server project
- id-based-encryption-client project

The elliptic-curve project contains the functions to generate elliptic curves and construct pairing over the curve. It is a modification and extension of the JPBC library. It includes interfaces and classes to access the underlying algebraic structures such as fields, pairing, etc. It provides the addition and multiplication operations in rings and fields, group addition and group multiplication for groups elements, pairing operations to manipulate pairing. This project is compiled into a .jar file and shared by the server and client. The id-based-encryption-server project is the simplest of three. Its setParemeter() function will setup and publish a pairing based on the curve paramenters(rBits, qBits). All the server does is waiting for key generation request.

Its generateKeyPair() function takes input an user email address and returns a pair of public/private key to that user. The id-based-encryption-server project has a data directory to store the curve parameters, master key and records of user accounts. For security reason, the data directory should be encrypted itself or configured with proper access control. The id-based-encryption-server project also includes a serverLog directory upon installation. The id-based-encryption-client project consists of several packages as well. It includes a clientLog directory for logs, a data directory to store user's key pair and serve published curve parameters, and a download directory for storing email attachments. Its client.crypto package handles encryption/decryption key calculation and AES encryption and decryption operations. Its client.email package refers to JavaMail library, handles all email related operations and infrastructures. The client.gui package is a graphic interface that simplifies the user's interaction with the client program. It provides a familiar email environment to users and makes the encryption, decryption easy to use for less technique skilled users.

CHAPTER 6

PERFORMANCE EVALUATION AND SECURITY ANALYSIS

We evaluate the performance of our IDE system based on two factors: speed and storage. Speed is the computation time actual algorithms take to run. Storage is primarily a measure of memory needed for store and transfer cipher text. In addition to performance evaluation, we also show the security level our IDE system provides with smaller key size.

6.1 Performance

Since an IBE scheme has four algorithms, we focus on the performance of each algorithm:

- server parameter and master key setup
- private and public key pair generation
- encryption speed and cipher text length
- decryption speed

6.1.1 Testing Environment

OS: Windows 7 home premiun, 64-bit operating System

Model: Inspiron 1764

Processor: Intel(R) Core(TM)i3CPU M330@2.13GHz

RAM: 4.00GB(3.86GB Usable)

6.1.2 Server Parameter Setup and Master Key Generation

The elliptic curve and pairing setup algorithm should only be run once with chosen rBits, qBits, and masterKey length. For testing purposes, we ran it a few times with different rBits, qBits, and masterKey lengths and recorded the execution time under different sizes of each parameter (See Server Algorithms section for definitions of rBits, qBits).

Table 6.1: Server Setup Test Results

rBits(bits)	qBits(bits)	masterKey(bits)	Time(milliSecond)
160	256	256	907
160	256	512	923
256	512	256	978
256	512	512	1239

6.1.3 Key Pair Generation

After the server runs the setup algorithm, the curve parameters and master key are stored to different files. When the server receives a key generating request, it reads the curve parameters, and the master key from the files. Then it takes a user's email address as input, and returns a pair of public/private keys to that user. Again, for testing purposes, we ran the setup algorithm more than once to get different curve parameters and master keys.

Table 6.2: Key Pair Generation Test Results

rBits(bits)	qBits(bits)	masterKey(bits)	EmailAddress	Time(milliSecond)
256	512	256	fiona201301@gmail.com	109
256	512	512	fiona201301@gmail.com	156
256	512	256	fionazeng@u.boisestate.edu	125
256	512	512	fionazeng@u.boisestate.edu	167

6.1.4 Emails with Single Receiver without Attachment

This table shows the connection time (Conn.), key derivation (Der.), encryption (Enc.) and decryption (Dec.) time for emails with only one recipient. The connection time is the time it takes for the email client to connect to an email server using java mail API. We also take the cipher text size as part of our measurements.

Table 6.3: Test Results for Emails with Single Recipient without Attachment

	Cipher	Sender			Recipient		
Msg. (char)	size (char)	Conn. (ms)	Der. (ms)	Enc. (ms)	Conn. (ms)	Der. (ms)	Dec. (ms)
524	875	4493	157	198	4926	153	698
3009	5670	4953	168	224	4583	172	813
10658	12944	5614	153	229	4922	144	935

6.1.5 Emails with Single Receiver with Attachments

From above table, we can see that the connection time is pretty stable around 4 to 6 seconds, so we do not list the connection time here. The encryption algorithm was able to encrypt a randomly generated file with size of 1GB, but the file could not be attached to an email since it was so big.

We use AES to encrypt email message and attachments. AES, as a block cipher, does not change the size of encrypted message. The output size should be the same as the input size. However, AES, being a block cipher, requires the input to be multiple of

Table 6.4: Test Results for Emails with Single Recipient with Attachments

File size (bytes)	Cipher size (bytes)	File type	Sender		Recipient	
			Der. (ms)	Enc. (ms)	Der. (ms)	Dec. (ms)
25,366	25,360	word doc	160	66	167	93
128,771	128,768	image(png)	189	132	156	398
140,297	140,288	PDF	160	164	160	401
3,303,581	3,303,568	image(jpg)	166	755	162	1993

block size (16 bytes). For inputs that are not multiple of 16 bytes, padding schemes are used to round up the input to the next module of 16. According to this formular [41]

long size = inputSizeInBytes;

long postAESSize = size + (16 - (size % 16));

The cipher text will be max 16 more bytes than the plain text. It may vary a little depending on the padding scheme. For an email message, because we append the random number r at the end of the message, the cipher text length is the length of original message, plus the padding, plus the length of r. For an attachment, the size of an encrypted file should be no more than 16 bytes than the size of original file. Interestingly, from our experiment an encrypted file is actually few bytes smaller than the original file in size.

6.1.6 Emails with Multiple Receivers

Emails were sent to multiple recipients to measure how the system scaled with the number of email recipients. The marginal increase in transmission time was consistent for larger numbers of recipients and so, for the sake of brevity, we show only the results of two and three recipient emails. The connection times were also omitted because of the lack of variance in the values.

Table 6.5: Test Results for Emails with Two Recipients

Msg. (char)	Sender		Recipient1		Recipient2	
	Der. (ms)	Enc. (ms)	Der. (ms)	Dec. (ms)	Der. (ms)	Dec. (ms)
524	335	202	103	681	98	662
3009	349	226	96	892	112	824
10658	389	276	101	1064	116	922

Table 6.6: Test Results for Emails with Three Recipients

Msg. (char)	Sender		Recipient1		Recipient2		Recipient3	
	Der. (ms)	Enc. (ms)	Der. (ms)	Dec. (ms)	Der. (ms)	Dec. (ms)	Der. (ms)	Dec. (ms)
524	477	192	105	668	109	662	102	676
3009	498	212	96	876	112	824	107	864
10658	481	296	101	998	116	922	112	972

6.1.7 Results Summary

We can summarize the test results as following:

- The system parameters and master key setup algorithm should only run once by the KGC. With rBits=256, qBits=512 and masterKey=512 bits, it takes less than two seconds to finish.

- It takes less than one second to generate a key pair for a typical user based on the user's email address.

- It takes 4 to 6 seconds to connect to an email server via Java Mail library. It only takes a fraction of the connection time to encrypt a message. The decryption operation takes 2 to 4 times longer than the encryption operation, but it is still very efficient compared to the connection time. The rate of encryption is about 384 bytes per ms.

- Cipher size is the same or slightly bigger than the original plain text size. There is no storage or space concern.

- For group emails, the encryption key derive time increases proportional to the number of receivers. This is because the sender needs to calculate xi from i=0, upon to i=n and needs to calculate yi for each receiver.

6.2 Security

The security of elliptic-curve-based cryptosystems are not well understood compared to RSA related public-key cryptosystems. Due in large part to the abstruse nature of elliptic curves, few cryptographers understand elliptic curves. Many cryptographic schemes treat elliptic curve and pairings as a *black box*, focus on purely cryptographic aspects and not mathematical and algorithmic subtleties. Since the details of elliptic curves and pairings, particularly their selection and implementation, can be quite complex, when we do the security analysis we are making some assumptions regarding the properties of elliptic curves and pairings.

Master key security

RSA public-key systems are secure assuming that it is difficult to factor a large integer composed of two or more large prime factors. For elliptic-curve-based cryptosystems, it is assumed that finding the discrete logarithm of a random elliptic curve element with respect to a publicly known base point is infeasible. If the elliptic curve groups is described using multiplicative notation, then the elliptic curve discrete logarithm problem is:

given points P and Q in the group, find a number that Pk = Q;

k is called the discrete logarithm of Q to the base P. The security of our master

key(s) depends on the ability to compute point multiplications and the inability to compute the multiplicand given the original and product points. Recall that for each user's key pair, one's public key is a point that mapped from the hash of one's email address and one's private key is a point equals to the multiplication of one's public key and master key. The point multiplication is efficiently computed so the private key generation is very efficient. But on the other hand, given the base point (one's public key), and the product point (one's private key), computing the master key requires solving the discrete logarithm in elliptic curve which is infeasible.

Encryption and decryption key security

The security of our encryption and decryption key is based on the bilinear Diffie-Hellman problem.

Let $\hat{e}$ be a bilinear pairing on (G_1, G_T). The bilinear Diffie-Hellman problem (BDHP)is the following [43]

- E: Elliptic curve defined over F_q.
- P: Point in E(F_q) of prime order r.
- k: Smallest positive integer such that r divides q^k-1 .
- μr: Group of rth roots of unity in Fqk.
- Q: Order r-point in E(Fqk) such that Q not in P
- e:$\hat{e}(P,Q) \rightarrow \mu r$: bilinear, non-degenerate map

BDHP: Given P, aP, bP, cP and Q, compute $\hat{e}(P,P)^{abc}$.

The hardness of the BDHP implies the hardness of the Diffie-Hellman problem in both G_1 and G_T groups. If the Diffie-Hellman problem in G_1 can be efficiently solved, then

one could solve an instance of the bilinear Diffie-Hellman problem by computing P^{ab} and then $\hat{e}(P^{ab}, P^c) = \hat{e}(P, P)^{abc}$. Also, if the Diffie-Hellman problem in G_T can be efficiently solved, then the bilinear Diffie-Hellman problem instance could be solved by computing g= $\hat{e}$(P, P), $g^{ab} = \hat{e}(P^a, P^b)$, $g^c = \hat{e}(P, P^c)$ and then g^{abc}. An eavesdropper who wishes to compute the shared secret key from some known public keys is faced with the task of solving an instance of the bilinear Diffie-Hellman problem. No one knows of any way to solve the BDHP except by finding discrete logs in both G_1 and G_T groups.

Security of AES

After a key establishment is made, this key is used as a symmetric key to encrypt and decrypt data via AES. One way to compute the shared secret key is solving the bilinear discrete logarithm problem. Another way is by using a brute-force attack. There is a physical argument that a 128-bit symmetric key using AES is computationally secure against brute-force attack: the faster supercomputer can do 10.51 x 1015 floating point operations per second, it will takes 1.02 x 1018 years to crack AES with 128-bit key [44]. There are records that AES has been cracked by National Security Agency(NSA). While NSA has all the computational resources and capabilities to crack AES, this was possibly also aided by the fact that the cryptosystems that use AES have implementation flaws.

CHAPTER 7

CONCLUSION

Nowadays almost everybody has one or more email accounts. Individuals rely on emails to communicate. Businesses can't conduct business today without emails. Email services are free and very convenient to use. However, for privacy-sensitive users, the privacy protection of emails provided by the email service providers is usually under their expectation. Most emails are currently transmitted in plain text over Internet or other networks. They can be intercepted easily by others. Between the sender and the recipients, emails pass through many servers and routers. In addition, mail servers regularly conduct unprotected backups of emails that passes through. Potentially, every unencrypted email sent over networks or stored on a mail server can be read, copied or altered. There is a strong need for secure email delivery.

Some email service providers such as Google's gmail did take some actions to protect users privacy by using https protocol that encrypts emails as they travel between web browsers and gmail servers. It helps protect data from being snooped by third parties during transmission. For highly privacy sensitive users, they may not be satisfied by https because this is not end-to-end encryption. There are other end-to-end encryptions available such as PGP. The downside of PGP is that most people struggle with finding and verifying other peoples public keys, and sharing their own keys.

This thesis presents the implementation of an identity-based encryption scheme based on elliptic curve and pairing. Our IDE system is end-to-end encryption so the emails are stored in encrypted format on the email servers. Such a system does require a key generation server, but it eliminates the need for certificates and some of the problems associated with them. If our IDE system is adopted by closed organizations such as large corporations or banks, they can have their own KGC. For open KGC, it is a little bit harder to establish a system that everyone trusts.

The benefits promised by elliptic curve and pairing-based cryptosystems is smaller key size, reducing storage and transmission requirements. Small keys are important, especially in a world where more and more cryptography is done on less powerful devices like mobile phones. While multiplying two prime numbers together is easier than factoring the product into its component parts, when the prime numbers start to get very long, even just the multiplication step can take some time on a low powered device. While we could likely continue to keep RSA secure by increasing the key length, that comes with a cost of slower cryptographic performance on the client. On another hand, elliptic-curve-based cryptosystem is not well established compared to RSA. RSA relies on the hardness of factorization, which has been studied for 2500 years. In comparison, discrete logarithm itself has been studied for long time but discrete logarithm on elliptic curves only sport about 30 years of research.

REFERENCES

[1] "How PGP works" http://searchsecurity.techtarget.com/definition/Pretty-Good-Privacy, 1991.

[2] eds. L. Cranor and G. Simson. O'Reilly, "In Security and Usability: Designing Secure Systems that People Can Use," 2005, pp. 679-702.

[3] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. Journal of Cryptology, 17(1):297319, 2004.

[4] http://doctrina.org/How-RSA-Works-With-Examples.html

[5] Burt Kaliski, "The Mathematics of the RSA Public-Key Cryptosystem," http://www.mathaware.org/mam/06/Kaliski.pdf.

[6] M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. "A Concrete Security Treatment of Symmetric Encryption: Analysis of the DES Modes of Operatio n". Proceedings of the 38th Symposium on Foundations of Computer Science, IEEE, 1997.

[7] "Symmetric Encryption" http://cr.yp.to/bib/2004/bellare-chap4.pdf

[8] Al-Riyami, Sattam S., and Kenneth G. Paterson. "CBE from CL-PKE: A generic construction and efficient schemes." Public Key Cryptography-PKC 2005. Springer Berlin Heidelberg, 2005. 398-415.

[9] Sattam S. Al-Riyami "Cryptographic Schemes based on Elliptic Curve Pairings" http://www.isg.rhul.ac.uk

[10] S.S. Al-Riyami and K.G. Paterson. "Authenticated three party key agreement protocols". International Conference on Cryptography and Coding, pages 332359

[11] American National Standards Institute ANSI X9.42. "Public key cryptography for the financial services industry: Agreement of symmetric keys using discrete logarithm cryptography", 2001.

[12] American National Standards Institute ANSI X9.63. "Public key cryptography for the financial services industry: Key agreement and key transport using elliptic curve cryptography," 2001.

[13] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, and M. Scott. "Efficient algorithms for pairing-based cryptosystems." Advances in Cryptology CRYPTO 2002, volume 2442 of Lecture Notes in Computer Science , pages 354368.

[14] R. Barua, R. Dutta, and P. Sarkar. "An n-party key agreement scheme using bilinear map". Cryptology ePrint Archive, Report 2003/062, 2003. http://eprint.iacr.org/

[15] Changyu Dong "Math in Network Security: A Crash Course" www.doc.ic.ac.uk

[16] L. Adleman and M. Huang, Function field sieve methods for d iscrete logarithms over finite fields, Information and Computation, 151 (1999)

[17] "Elliptic Curves," http://www-math.ucdenver.edu

[18] I.F.Blake, G, Seroussi and N.P.Smart, "Elliptic Curves in Cryptograhy" Cambrige University Press, 1999

[19] R. Balasubramanian and N. Koblitz. "The improbability th at an elliptic curve has subexponential discrete log problem under the Menezes-Oka moto-Vanstone algorithm." Journal of Cryptology, 11(2):141145, Spring 1998

[20] A. J. Menezes, S. A. Vanstone, and P. C. V. Oorschot. "Handbook of Applied Cryptography." CRC Press, Inc., Boca Raton, FL, USA, 1996

[21] J. H. Silverman. "The arithmetic of elliptic curves." Springer-Verlag, Berlin, 1995

[22] R. Sakai, K. Ohgishi, and M. Kasahara. "Cryptosystems based on pairing. In The 2000 Symposium on Cryptography and Information Security" , Okinawa, Japan, 2000.

[23] Ben Lynn. "On the implementation of pairing-based cryptosystems", Ph.D. Disertation, Stanford University, 2007

[24] D. Page, N. P. Smart, and F. Vercauteren. " A comparison of MNT curves and supersingular curves." Cryptology ePrint Archive, Report 2004/165, 2004. http://eprint.iacr.org

[25] V. Miller. "The Weil pairing, and its efficient calculation." Journal of Cryptology,17(4):235262, 2004

[26] A. Menezes, T. Okamoto, and S. Vanstone. "Reducing elliptic curve logarithms to logarithms in a finite field." Proceedings of the twenty-third annual ACM symposium on Theory of computing , New York, NY, USA, 1991.

[27] A. K. Lenstra and E. R. Verheul. “Selecting cryptographic key sizes.” Journal of Cryptology, 2001.

[28] R. L. Rivest, A. Shamir, and L. Adleman. “A method for obtaining digital signatures and public-key cryptosystems.” Communications of the ACM, 1978.

[29] A. Shamir, “Identity-based cryptosystems and signature schemes,” Advances in Cryptology-Crypto’84,Lecture Notes in Computer Science,Vol.196,Springer-Verlag,pp. 47-53,1984

[30] S.Tsuji and T.Itoh, “An ID-based cryptosystem based on the discrete logarithm problem”, IEEE Journal on Selected Areas in Communication ,vol.7,no.4,pp.467-473,1989

[31] U.Maurer and Y.Yacobi,“Non-interactive public-key cryptography”, in Advances in Cryptology-Crypto’91,Lecture Notes in Computer Science,Vol.547,Springer-Verlag, pp.498-507,1991

[32] Dan Boneh, Matthew K. Franklin, “Identity-Based Encryption from the Weil Pairing,” Advances in Cryptology - Proceedings of CRYPTO 2001 (2001)

[33] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. “Efficien t algorithms for pairing- based cryptosystems.” In CRYPTO 02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology , pages 354368, London, UK, 2002.

[34] D. R. Stinson. “Some baby-step giant-step algorithms for the low hamming weight discrete logarithm problem.” In Math, pages 379391, 2002

[35] D. Coppersmith. “Fast evaluation of logarithms in fields of characteristics two.” In IEEE Transactions on Information Theory , volume 30, pages 587594, 1984.

[36] BLS signatures https://crypto.stanford.edu/pbc/manual

[37] Antoine Joux “A One Round Protocol for Tripartite Diffie-Hellman” http://cgi.di.uoa.gr/ aggelos/crypto/page4/assets/joux-tripartite.pdf

[38] Dan Boneh, Ben Lynn, and Hovav Shacham “Short signatures from the Weil pairing” https://www.iacr.org/archiveasiacrypt2001/22480516.pdf

[39] D. Freeman, M. Scott, and E. Teske. “A taxonomy of pairing -friendly elliptic curves.” preprint, 2006.

[40] A. Menezes, T. Okamoto, and S. Vanstone. “Reducing ellip tic curve logarithms to logarithms in a finite field.” Proceedings of the twenty-third annual ACM symposium on Theory of computing, pages 8089, New York, NY, USA, 1991.

[41] "Announcing the ADVANCED ENCRYPTION STANDARD (AES)." Federal Information Processing Standards Publication 197. United States National Institute of Standards and Technology (NIST). November 26, 2001

[42] Avi Kak. "AES: The Advanced Encryption Standard," Lecture Notes on Computer and Network Security, https://engineering.purdue.edu/kak/compsec

[43] John Bethencourt. "Intro to Bilinear Maps," http://www.upl.cs.wisc.edu

[44] Mohit Arora. "How secure is AES against brute force attacks," http://www.eetimes.com

[45] D. Page, N. P. Smart, and F. Vercauteren." A comparison of MNT curves and supersingular curves." Cryptology ePrint Archive, 2004. http://eprint.iacr.org/

[46] Jyh-haw Yeh," P2P email encryption by an identity-based one-way group key agreement protocol", Boise State University, 2014

[47] E. Verheul. "Evidence that XTR is more secure than supers ingular elliptic curve cryptosystems." Journal of Cryptology, 17(4):277296, 2004

[48] K. Rubin and A. Silverberg. "Supersingular abelian vari eties in cryptology." In Advances in Cryptology Crypto 2002 , volume 2442 of Lecture Notes on Computer Science, 2002

APPENDIX A

JAVA SOURCE CODE OF SOME OF THE MAIN ALGORITHMS

The following are the java implementations of various algorithms we introduced earlier.

A.1 Curve and Pairing Functions

hash to a point on the curve

```
public CurveElement setFromHash(byte[] source, int offset, int length) {
    infFlag = 0;
    x.setFromHash(source, offset, length);
    IElement t = field.getTargetField().newElement();
    for (;;) {
        t.set(x).square().add(curveField.a).mul(x).add(curveField.b);
        if (t.isSqr())
            break;
        x.square().add(t.setToOne());
        }
    y.set(t).sqrt();
    if (y.sign() <0)
```

```
        y.negate();
    if (curveField.cofac != null)
        mul(curveField.cofac);
    return this;
    }
```

twice a group point element

```
public IElement[] twice(IElement[] elements) {
    int i;
    int n = elements.length;
    IElement[] table = new IElement[n];
    IElement e0, e1, e2;
    CurveElement q, r;

    q = (CurveElement) elements[0];
    e0 = q.getX().getField().newElement();
    e1 = e0.duplicate();
    e2 = e0.duplicate();

    for (i = 0; i <n; i++) {
        q = (CurveElement) elements[i];
        table[i] = q.getY().getField().newElement();

        if (q.infFlag != 0) {
            q.infFlag = 1;
```

```
            continue;
        }

        if (q.getY().isZero()) {
            q.infFlag = 1;
            continue;
        }
    }
    for (i = 0; i <n; i++) {
        q = (CurveElement) elements[i];
        table[i].set(q.getY()).twice();
        if (i>0)
            table[i].mul(table[i - 1]);
        }
    e2.set(table[n - 1]).invert();
    for (i = n - 1; i >0; i–) {
        q = (CurveElement) elements[i];
        table[i].set(table[i - 1]).mul(e2);
        e2.mul(q.getY()).twice();
    }
    table[0].set(e2);
    for (i = 0; i <n; i++) {
        q = (CurveElement) elements[i];
        if (q.infFlag != 0)
            continue;
```

```
        e2.set(q.getX()).square().mul(3).add(a).mul(table[i]);
        e1.set(q.getX()).twice();
        e0.set(e2).square().sub(e1);
        e1.set(q.getX()).sub(e0).mul(e2).sub(q.getY());
        q.getX().set(e0);
        q.getY().set(e1);
        q.infFlag = 0;
    }
    return elements;
}
```

choose rbits as prime for efficiency

```
public static BigInteger generateSolinasPrime(int bits, Random random) {
    BigInteger r, q;
    int exp2, sign1;
    while (true) {
        r = BigInteger.ZERO;
        if (random.nextInt(Integer.MAX_ VALUE) % 2 != 0) {
            exp2 = bits - 1;
            sign1 = 1;
        } else {
            exp2 = bits;
            sign1 = -1;
        }
        r = r.setBit(exp2);
```

```
        q = BigInteger.ZERO
        .setBit((random.nextInt(Integer.MAX_ VALUE) % (exp2 - 1)) + 1);
        if (sign1 >0) {
            r = r.add(q);
        } else {
            r = r.subtract(q);
        }
        if (random.nextInt(Integer.MAX_ VALUE) % 2 != 0) {
            r = r.add(BigInteger.ONE);
        } else {
            r = r.subtract(BigInteger.ONE);
        }
        if (r.isProbablePrime(10))
            return r;
        }
    }
```

A.2 Server Algorithms

server side curve and pairing parameters

```
public CurveParameters generate() {
    boolean found = false;
    BigInteger q;
    BigInteger r;
```

```
BigInteger h = null;
int exp1 = 0;      int exp2 = 0;
int sign0 = 0;      int sign1 = 0;
do {
    r = BigInteger.ZERO;
    if (random.nextInt(Integer.MAX_VALUE) % 2 != 0) {
        exp2 = rbits - 1;
        sign1 = 1;
    } else {
        exp2 = rbits;
        sign1 = -1;
    }
    r = r.setBit(exp2);
    q = BigInteger.ZERO;
    exp1 = (random.nextInt(Integer.MAX_VALUE) % (exp2 - 1)) + 1;
    q = q.setBit(exp1);
    if (sign1 >0) {
        r = r.add(q);
    } else {
        r = r.subtract(q);
    }
    if (random.nextInt(Integer.MAX_VALUE) % 2 != 0) {
        sign0 = 1;
        r = r.add(BigInteger.ONE);
    } else {
```

```
        sign0 = -1;
        r = r.subtract(BigInteger.ONE);
    }
    if (!r.isProbablePrime(10))
        continue;
    for (int i = 0; i <10; i++){
        q = BigInteger.ZERO;
        int bit = qbits - rbits - 4 + 1;
        if (bit <3)
            bit = 3;
        q = q.setBit(bit);
        h = BigIntegerUtils.getRandom(q, random).multiply( BigIntegerUtils.TWELVE);
        q = h.multiply(r).subtract(BigInteger.ONE);
        if (q.isProbablePrime(10)) {
            found = true;
            break;
        }
    }
} while (!found);
DefaultCurveParameters params = new DefaultCurveParameters();
params.put("type", "a");
params.put("q", q.toString());
params.put("r", r.toString());
params.put("h", h.toString());
params.put("exp1", String.valueOf(exp1));
```

```
params.put("exp2", String.valueOf(exp2));
params.put("sign0", String.valueOf(sign0));
params.put("sign1", String.valueOf(sign1));
if (generateCurveFieldGen) {
    Field Fq = new ZrField(random, q);
    CurveField curveField = new CurveField¡Field¿(random, Fq.newOneElement(),
        Fq.newZeroElement(), r, h);
    params.put("genNoCofac", Base64.encodeBytes(curveField.getGenNoCofac().toBytes()));
    }
return params;
}
```

system parameter setup

1. TypeACurveGenerator cg = new TypeACurveGenerator(rBits, qBits);
2. CurveParameters param = cg.generate();
3. Pairing pair = PairingFactory.getPairing(curveFileName);

Our pairings are constructed on the curve $y^2 = x^3 + x$ over the field F_q for some prime q = 3 mod 4. Both G1 and G2 are the group of points E(F_q), so this pairing is symmetric. The order r is some prime factor of q + 1. To be secure, generic discrete log algorithms must be infeasible in groups of order r, and finite field discrete log algorithms must be infeasible in finite fields of order q^2, so we have rbits = 256 and qbits = 512.

The CurveGenerator is initialized with the input r and q, after it is initialized, we call its generate() method to generate the pairing paramsters and store them in a file

for later use. A pairing parameter include properties of curve type (Type A), r, and q values. The group order r is rbits(256 bit) long, and the order of the base field q is qbits(512 bit) long. Calling the pairing parameters' toString() method will return a string presentation of the curve in the form of:

type a

q 13878449027838634308485324894656547831531975681771763457230535958189

902783863430848532489465654783153197568177176345723053595818966261819 8

485571313053465219264169094659253058991193288334905567053374793379337 41

3504974113504991091

r 57896051520404444502349473890026478309277103328569307

279939099816203660444783929254190115345789605152040444

502349279939099816203660444783929254190115347166480788

619263

h 23971322159935931350831141021728819933102752251554391242318242437710000917108 4

exp1 232

exp2 255

sign0 -1

sign1 1

Once the pairingParamaters is generated, an instance of Pairing interface can be obtained from the pairingParamaters. the Pairing interface provides methods to access the algebraic structure involved in the pairing computation :

```
/* Return Zr */
Field Zr = pairing.getZr();
/* Return G1 */
Field G1 = pairing.getG1();
```

/* Return G2 */

Field G2 = pairing.getG2();

/* Return GT */

Field GT = pairing.getGT();

generate client key pair

public String generateKeyPair(String email){

.....

byte[] hash = Hash(account.emailAddress);

IElement elePubKey = pair.getG1().newElement() .setFromHash(hash, 0, hash.length);

account.publicKey = elePublicKey.toBytes();

account.privateKey = elePublicKey.getImmutable().mul(masterKey).toBytes();

.....

return account.toString();

}

A.3 Client Algorithms

derive encryption key

public BigInteger getEncryptKeyAB(String receiverEmail, BigInteger r) throws NoSuchAlgorithmException {

byte[] hash = Hash(receiverEmail);

IElement receiverPubElement = clientPairing.getG1().newElement();

receiverPubElement.setFromHash(hash, 0, hash.length);

BigInteger keyAB = clientPairing.pairing(clientPriElement, receiverPubElement)

```
        .pow(r)
        .toBigInteger()
        .xor(clientPairing.pairing(clientPriElement, receiverPubElement)
        .toBigInteger());
    return keyAB;      }
```

derive decryption key

```
public BigInteger getDecryptKeyBA(String senderEmail, BigInteger r) throws NoSuchAlgorithmException {
    byte[] hash = Hash(senderEmail);
    IElement senderPubElement = clientPairing.getG1().newElement();
    senderPubElement.setFromHash(hash, 0, hash.length);
    BigInteger keyBA = clientPairing
        .pairing(senderPubElement.getImmutable().mul(r), clientPriElement)
        .toBigInteger()
        .xor(clientPairing.pairing(senderPubElement, clientPriElement)
        .toBigInteger());
    return keyBA;
    }
```

A.4 Group Email Algorithms

derive encryption group key

```
public BigInteger getEncryptGroupKey(String senderEmail, ArrayList<String >receiversEmails, BigInteger r) throws NoSuchAlgorithmException {
```

```
ArrayList<BigInteger >xList = new ArrayList<BigInteger >();
byte[] hashSender = Hash(senderEmail);
IElement senderPubElement = clientPairing.getG1().newElement();
senderPubElement.setFromHash(hashSender, 0, hashSender.length);
BigInteger xSender = clientPairing.pairing(clientPriElement, senderPubElement
        .getImmutable()
        .mul(r)).toBigInteger();
for (int i = 0; i < receiversEmails.size(); i++) {
    byte[] ithReceiverHash = Hash(receiversEmails.get(i));
    IElement ithReceiverPubElement = clientPairing.getG1().newElement();
    ithReceiverPubElement.setFromHash(ithReceiverHash, 0, ithReceiverHash.length);
    BigInteger xi = clientPairing.pairing(clientPriElement, ithReceiverPubElement
            .getImmutable()
            .mul(r)).toBigInteger();
    xList.add(xi);
    }
BigInteger groupKey = xSender;
for (int j = 1; j < xList.size(); j++) {
    groupKey = groupKey.xor(xList.get(j));
}
BigInteger yValue=null;
while (yList.size() < xList.size()) {
    yValue = xSender;
    for (int k = 0; k < xList.size(); k++) {
        if (k != yList.size()) {
```

```
                yValue = yValue.xor(xList.get(k));
            }
        }
        yList.add(yValue);
    }
    return groupKey;
}
```

derive decryption group key

```
public BigInteger getDecryptGroupKey(String senderEmail, BigInteger r, BigInteger
yi) throws NoSuchAlgorithmException {
    byte[] hash = Hash(senderEmail);
    IElement senderPubElement = clientPairing.getG1().newElement();
    senderPubElement.setFromHash(hash, 0, hash.length);
    BigInteger key = clientPairing.pairing(senderPubElement
        .getImmutable().mul(r), clientPriElement)
        .toBigInteger();
    return yi.xor(key);
    }
```

A.5 Using Java Mail

connect to email server

```
connectToServer {
```

```
    Properties props = System.getProperties();
    props.setProperty("mail.store.protocol", "imaps");
    Session session = Session.getInstance(props, null);
    store = session.getStore("imaps");
    store.connect("imap.gmail.com", address, password);
}
```

send email using java mail message class

```
SendEmail{
    ...
    msg.setFrom(new InternetAddress(sender));
    msg.addRecipient(Message.RecipientType.TO, new InternetAddress( receiver));
    msg.setSubject(subject);
    Transport.send(msg);
    ...
    }
```

read email using java mail message class

```
ReadEmail{
    ...
    for (Message msg : messages) {
        ArrayList<String>list = new ArrayList<String>();
        MimeMessage m = (MimeMessage) msg;
        mailId = m.getMessageID();
        from = InternetAddress.toString(msg.getFrom());
```

recipient = InternetAddress.toString(msg .getRecipients(Message.RecipientType.TO));

String replyTo = InternetAddress.toString(msg.getReplyTo());

sendDate = msg.getSentDate().toString();

subject = msg.getSubject();

...

}

}

send email with attachment use mimemessage and multipart classes

sendEmailWithAttachments{

....

Message msg = new MimeMessage(session);

Multipart multipart = new MimeMultipart();

MimeBodyPart messageBodyPart = new MimeBodyPart();

messageBodyPart.setContent(msgBody, "text/plain");

multipart.addBodyPart(messageBodyPart);

if (fileNamesWithPath != null) {

if (fileNamesWithPath.size() >0) {

MimeBodyPart[] attachParts = new MimeBodyPart[fileNamesWithPath .size()];

for (int i = 0; i ¡ fileNamesWithPath.size(); i++) {

attachParts[i] = new MimeBodyPart();

attachParts[i].attachFile(fileNamesWithPath.get(i));

multipart.addBodyPart(attachParts[i]);

}

}

```
}
msg.setContent(multipart);
msg.setFrom(new InternetAddress(sender));
msg.addRecipient(Message.RecipientType.TO, new InternetAddress( receiver));
msg.setSubject(subject);
Transport.send(msg);
}
```

read email with attachment use mimemessage and multipart classes

```
readEmailWithAttachments {
    ...
    for (Message msg : messages) {
        ArrayList<String>list = new ArrayList<String>();
        MimeMessage m = (MimeMessage) msg;
        mailId = m.getMessageID();
        from = InternetAddress.toString(msg.getFrom());
        recipient = InternetAddress.toString(msg
            .getRecipients(Message.RecipientType.TO));
        String replyTo = InternetAddress.toString(msg.getReplyTo());
        sendDate = msg.getSentDate().toString();
        subject = msg.getSubject();
        Object content = msg.getContent();
        if (content instanceof Multipart) // has attachment
        {
            int parts = ((Multipart) content).getCount();
```

```
            count = parts - 1;
            contentString = ((Multipart) content).getBodyPart(0)
                .getContent().toString();
            for (int i = 1; i <parts; i++) {
                Part part = ((Multipart) content).getBodyPart(i);
                String fileName = part.getFileName();
                list.add(fileName);
                }
                ...
            }
        }
    }
```

download email attachments

```
downloadAttachment{
    .....
    for (Message msg : messages) {
        ....
        Object content = msg.getContent();
        if (content instanceof Multipart) // has attachment
        {
            int parts = ((Multipart) content).getCount();
            count = parts - 1;
            contentString = ((Multipart) content).getBodyPart(0)
                .getContent().toString();
```

```
            for (int i = 1; i <parts; i++) {
                Part part = ((Multipart) content).getBodyPart(i);
                String fileName = part.getFileName();
                list.add(fileName);
                }
            } else{
            ...
        }
    }
}
```

APPENDIX B

ADVANCED ENCRYPTION STANDARD ALGORITHM

The Advanced Encryption Standard (AES) was an encryption algorithm for securing sensitive but unclassified material by U.S. Government agencies, and it eventually become the encryption standard for commercial transactions in the private sector [41]. AES calls for a symmetric algorithm using block encryption of 128 bits in size (block cipher of 16 bytes). Unlike public-key cryptography, which use a pair of keys, symmetric-key cryptography use the same key (KeyAB=KeyBA in our IBE system) to encrypt and decrypt data. The encrypted data returned by block ciphers have the same number of bits that the input data had. Iterative ciphers use a loop structure that repeatedly performs permutations and substitutions of the input data.

The AES algorithm was required to offer security of a sufficient level to protect data for the next 20 to 30 years. It was to be easy to implement in hardware and software, as well as in restricted environments such as smart card and offer good defenses against various attacks [41]. AES supporting key sizes of 128, 192 and 256 bits. In our IBE system, we use the first 192 bits of KeyAB to encrypt the email and use the first 192 bits of KeyBA to decrypt the email. The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the plain text into the final output-cipher text. The number of cycles of repetition are as follows:

10 cycles of repetition for 128-bit keys.

12 cycles of repetition for 192-bit keys.

14 cycles of repetition for 256-Tbit keys.

Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself. A set of reverse rounds are applied to transform cipher text back into the original plain text using the same encryption key. The AES encryption routine begins by copying the 16-byte (128 bit block size) input array into a 4 X 4 byte matrix named State and the AES encryption algorithm is operated on State[] and can be described in pseudocode [42].

Cipher Algorithm Pseudocode

```
Cipher(byte[] input, byte[] output)
{
    byte[4,4] State;
    copy input[] into State[]
    AddRoundKey for (round = 1; round <Nr-1; ++round)
    {
        SubBytes
        ShiftRows
        MixColumns
        AddRoundKey
    }
    SubBytes
    ShiftRows
    AddRoundKey
```

copy State[] to output[]

}

The encryption algorithm performs a preliminary processing step that's called AddRoundKey in the specification. AddRoundKey performs a byte-by-byte XOR operation on the State matrix using the first four rows of the key schedule, and XORs input State[r,c] with round keys table w[c,r]. The main loop of the AES encryption algorithm performs four different operations on the State matrix, called SubBytes, ShiftRows, MixColumns, and AddRoundKey in the specification. The AddRoundKey operation is the same as the preliminary AddRoundKey except that each time AddRoundKey is called, the next four rows of the key schedule are used. The SubBytes routine is a substitution operation that takes each byte in the State matrix and substitutes a new byte determined by the Sbox table. ShiftRows is a permutation operation that rotates bytes in the State matrix to the left. The MixColumns operation is a substitution operation that is the trickiest part of the AES algorithm to understand. It replaces each byte with the result of mathematical field additions and multiplications of values in the byte's column. The addition and multiplication are special mathematical field operations, not the usual addition and multiplication on integers. The four operations SubBytes, ShiftRows, MixColumns, and AddRoundKey are called inside a loop that executes Nr times (the number of rounds for a given key size) less 1. The number of rounds that the encryption algorithm uses is either 10, 12, or 14 and depends on whether the seed key size is 128, 192, or 256 bits. For example, our key size is 192 bits so Nr equals 12, the four operations are called 11 times. After this iteration completes, the encryption algorithm finishes by calling SubBytes, ShiftRows, and AddRoundKey before copying the State matrix to the output parameter.

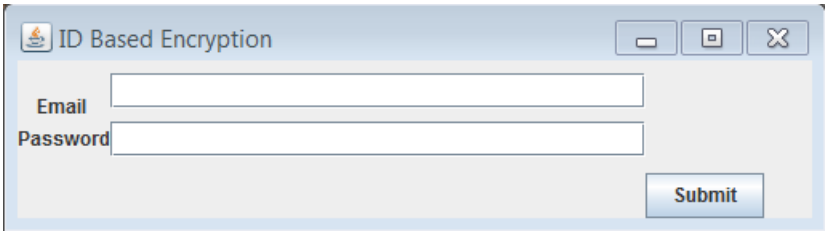
APPENDIX C

GUI

The following are the graphic user interfaces that most email users will feel familiar with since they are similar to most email web interface.

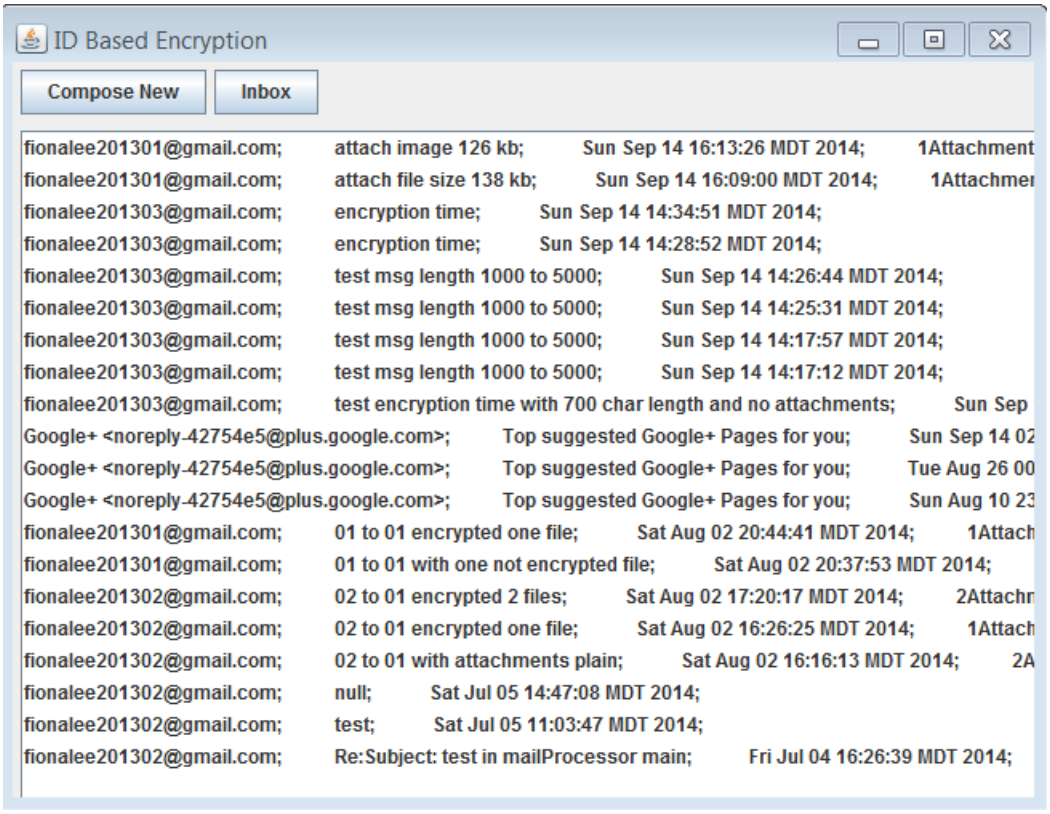
C.0.1 User Login Interface

The entry point of the client program. it is similar to most of the website email log-in page.



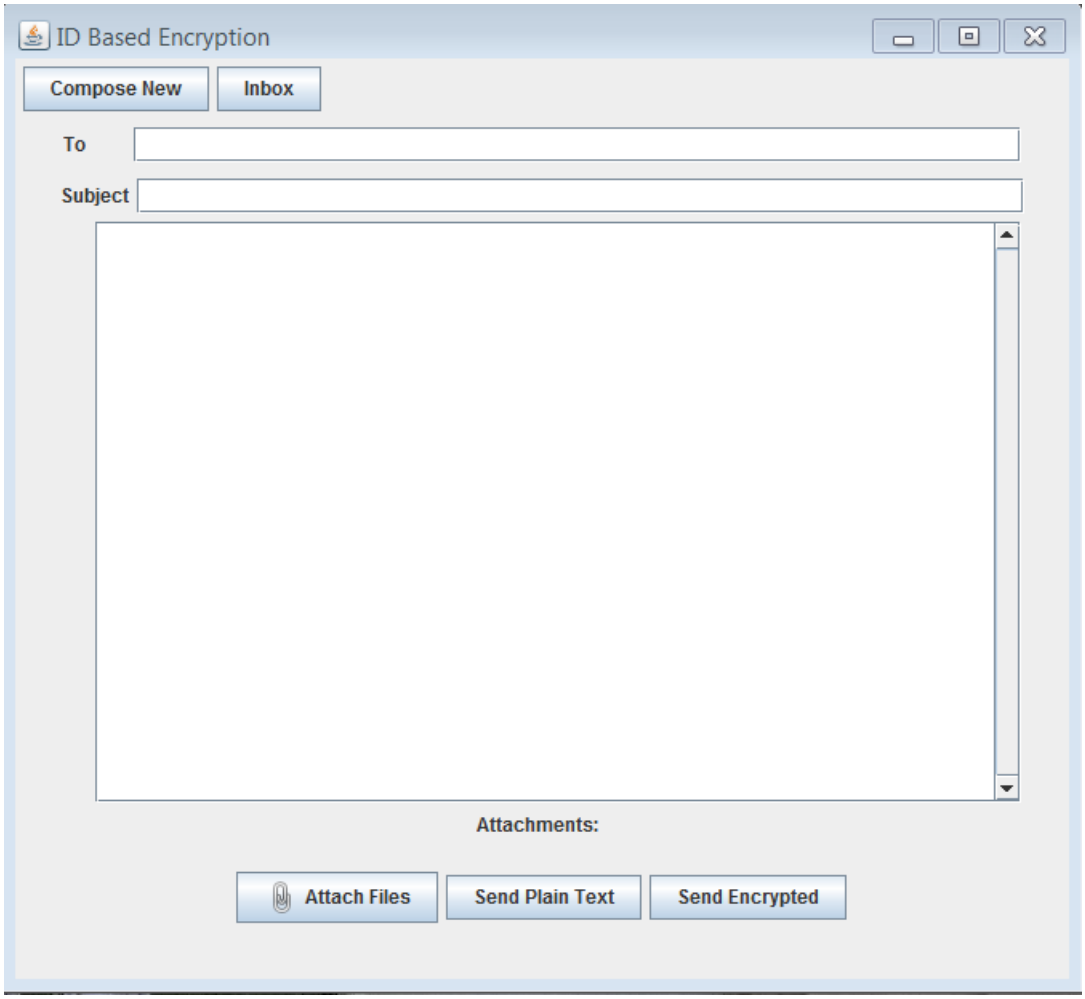
C.0.2 Inbox Interface

After successful login, the inbox panel reads emails from inbox folder from the email server.



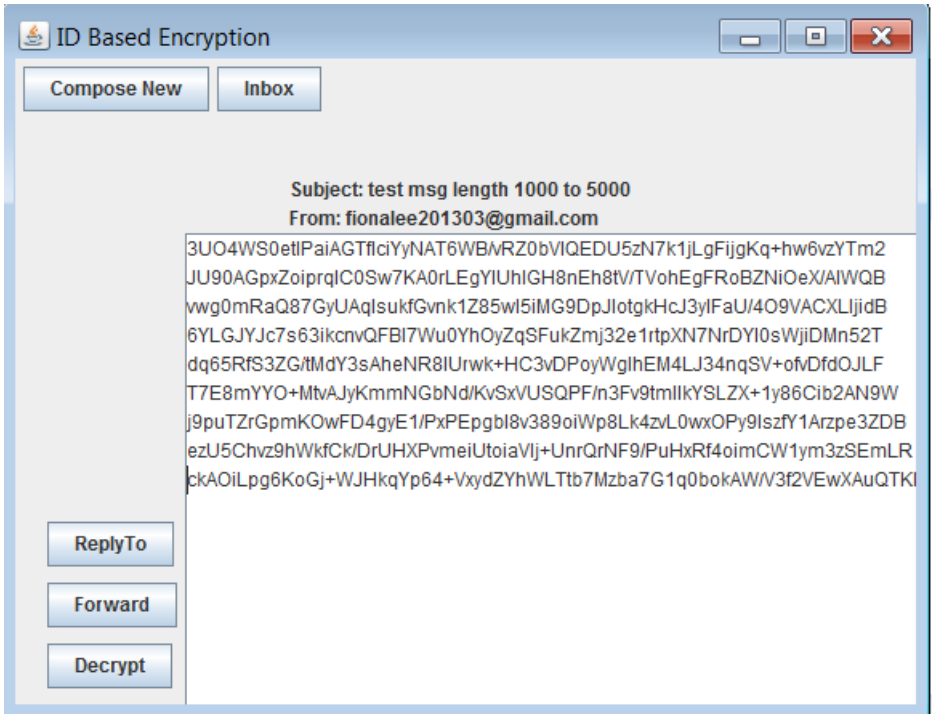
C.0.3 Email Compose Interface

The compose panel letting an user to compose an email. the "Attach Files" button enables an user to select files to attach from file system. An user has options to send email and attachments in plain text or encrypted format.



C.0.4 Individual Email Message Interface

The email message panel opens a selected email. the Forward and ReplyTo buttons work the same way as they are in most email website interface.


ID Based Encryption
Compose New
Inbox
Subject: test msg length 1000 to 5000
From: fionalee201303@gmail.com
3UO4WS0etlPaiAGTflciYyNAT6WBvRZ0bVlQEDU5zN7k1jLgFijgKq+hw6vzYTm2
JU90AGpxZoiprqlC0Sw7KA0rLEgYlUhlGH8nEh8tV/TVohEgFRoBZNiOeX/AlWQB
vwg0mRaQ87GyUAqlsukfGvnk1Z85wl5iMG9DpJlotgkHcJ3ylFaU/4O9VACXLljidB
6YLGJYJc7s63ikcnvQFBl7Wu0YhOyZqSFukZmj32e1rtpXN7NrDYl0sWjiDMn52T
dq65RfS3ZG/tMdY3sAheNR8lUrwk+HC3vDPoyWglhEM4LJ34nqSV+ofvDfdOJLF
T7E8mYYO+MtvAJyKmmNGbNd/KvSxVUSQPF/n3Fv9tmlIkYSLZX+1y86Cib2AN9W
j9puTZrGpmKOwFD4gyE1/PxPEpgbl8v389oiWp8Lk4zvL0wxOPy9lszfY1Arzpe3ZDB
ezU5Chvz9hWkfCk/DrUHXPvmeiUtoiaVlj+UnrQrNF9/PuHxRf4oimCW1ym3zSEmLR
ckAOiLpg6KoGj+WJHkqYp64+VxydZYhWLTtb7Mzba7G1q0bokAW/V3f2VEwXAuQTKl
ReplyTo
Forward
Decrypt