

**A CERTIFICATELESS ONE-WAY GROUP KEY
AGREEMENT PROTOCOL FOR POINT-TO-POINT
EMAIL ENCRYPTION**

by

Srisarguru Sridhar

A thesis

submitted in partial fulfillment

of the requirements for the degree of

Master of Science in Computer Science

Boise State University

December 2016

© 2016
Srisarguru Sridhar
ALL RIGHTS RESERVED

BOISE STATE UNIVERSITY GRADUATE COLLEGE

DEFENSE COMMITTEE AND FINAL READING APPROVALS

of the thesis submitted by

Srisarguru Sridhar

Thesis Title: A Certificateless One-Way Group Key Agreement Protocol for Point-to-Point Email Encryption

Date of Final Oral Examination: 13 October 2016

The following individuals read and discussed the thesis submitted by student Srisarguru Sridhar, and they evaluated his 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

Gabe Dagher, Ph.D. Member, Supervisory Committee

Dianxiang Xu, Ph.D. Member, Supervisory Committee

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

dedicated to my beloved grandmother

ACKNOWLEDGMENTS

Firstly, I would like to thank my advisor, Dr.Jyh-haw Yeh, for the patient guidance, encouragement and advise he has provided for the completion of my thesis. I would also like to thank him for giving me the opportunity and encouragement to work in the field of cyber-security. I would like to thank my committee member Dr.Gaby whose guidance and encouragement played a vital role in completion of my thesis. I would also like to thank my committee member Dr.Xu for his constant support throughout my time at Boise State University.

I wish to express my gratitude to Dr.Amit Jain for his guidance during my initial time at Boise State University. I would like to thank all the faculty members with whom I have taken courses at Boise State University for their guidance and knowledge. I also would like to thank the Computer Science Department and College of Engineering for the scholarship provided to me for completing my Master's degree at Boise State University.

I want to thank my parents, Sridhar and Renuka, to whom I am always grateful for their unconditional support and for always believing in me. I also want to thank my family and close friends for their constant support and encouragement. Most importantly I would like to thank my grandmother Kanagambal for making me the person I am today and to whom I am eternally grateful to.

ABSTRACT

Over the years, email has evolved and grown to one of the most widely used form of communication between individuals and organizations. Nonetheless, the current information technology standards do not value the significance of email security in today's technologically advanced world. Not until recently, email services such as Yahoo and Google started to encrypt emails for privacy protection. Despite that, the encrypted emails will be decrypted and stored in the email service provider's servers as backup. If the server is hacked or compromised, it can lead to leakage and modification of one's email. Therefore, there is a strong need for point-to-point (P2P) email encryption to protect email user's privacy. P2P email encryption schemes strongly rely on the underlying Public Key Cryptosystems (PKC). The evolution of the public key cryptography from the traditional PKC to the Identity-based PKC (ID-PKC) and then to the Certificateless PKC (CL-PKC) provides a better and more suitable cryptosystem to implement P2P email encryption. Many current public-key based cryptographic protocols either suffer from the expensive public-key certificate infrastructure (in traditional PKC) or the key escrow problem (in ID-PKC). CL-PKC is a relatively new cryptosystem that was designed to overcome both problems. In this thesis, we present a CL-PKC group key agreement protocol, which is, as the author's knowledge, the first one with all the following features in one protocol: (1) certificateless and thus there is no key escrow problem and no public key certificate infrastructure is required. (2) one-way group key agreement and thus no back-and-forth message exchange is required; (3) n -party group key agreement (not just 2- or

3-party); and (4) no secret channel is required for key distribution. With the above features, P2P email encryption can be implemented securely and efficiently. This thesis provides a security proof for the proposed protocol using “proof by simulation”. Efficiency analysis of the protocol is also presented in this thesis. In addition, we have implemented the prototypes (email encryption systems) in two different scenarios in this thesis.

TABLE OF CONTENTS

ABSTRACT	vi
LIST OF TABLES	xi
LIST OF FIGURES	xii
LIST OF ABBREVIATIONS	xiii
LIST OF SYMBOLS	xv
1 Introduction	1
1.1 Introduction	1
1.2 Thesis Statement	4
2 Related Work	7
2.1 Previous work	7
2.2 Functionality comparison with different protocols	10
2.3 Existing email encryption softwares	11
3 Preliminary	14
3.1 Elliptic curves	14
3.2 Bilinear pairings	15
4 The CLOW-GKA protocol for P2P email encryption	17

4.1	KGC servers	17
4.2	CLOW-GKA for P2P email encryption	18
4.3	Email account registration	19
4.4	Sending a P2P encrypted email	21
4.5	Receiving a P2P encrypted email	22
4.6	Example	24
5	Security analysis	28
5.1	Adversarial model	28
5.2	Privacy by simulation	28
5.2.1	User Registration Phase	30
5.2.2	Email Sending and Receiving Phase	33
6	Efficiency analysis	37
6.1	Computational cost of the protocol	38
6.1.1	CLOW-GKA protocol: P2P email encryption	38
6.1.2	Sub-protocol 1	38
6.1.3	Sub-protocol 2	38
6.1.4	Sub-protocol 3	39
6.2	Email size analysis	40
6.3	Efficiency comparison with different protocols	40
7	Implementation	42
7.1	Scenarios of CLOW-GKA implementation	42
7.2	Common steps in our implementation	45
7.2.1	KGC parameters and system setup	45

7.2.2	User registration	46
7.2.3	Connecting with mail server	50
7.2.4	Sending, receiving, encrypting and decrypting emails	52
7.3	Future Implementations	55
8	Conclusions	58
8.1	Thesis Conclusion	58
8.2	Future directions	59
	REFERENCES	60
A	Software Application Interface	63
A.1	User Email Login and Register Interface	63
A.2	Inbox Interface	63
A.3	Compose and Sending Interface	64
A.4	Decrypt Email Interface	66
B	Web Application Interface	68
B.1	Organisation Login Interface	68
B.2	User Email Login and Register Interface	69
B.3	Inbox Interface	70
B.4	Compose and Sending Email Interface	71
B.5	Decrypt Email Interface	72

LIST OF TABLES

2.1	Functionality comparison between CLOW-GKA, PGP, Yeh, Zeng and Long's ID-PKC and Al-Riyami et al's CL-PKC, assuming there are n recipients in an email	11
6.1	Efficiency comparison between CLOW-GKA, PGP, Yeh, Zeng and Long's ID-PKC and Al-Riyami et al's CL-PKC, assuming there are n recipients in an email	41

LIST OF FIGURES

7.1	Flowchart of user registration steps in software application	48
7.2	Flowchart of user registration steps in web application	49
7.3	Javamail message parts	51
7.4	Logging into mail and retrieving emails for software application	51
7.5	Logging into mail and retrieving emails for web application	52
7.6	Flow of emails in software Application	54
7.7	Flow of emails in web Application	56

LIST OF ABBREVIATIONS

CLOW-GKA – Certificateless One-Way Group Key Agreement

PGP – Pretty Good Privacy

IDEA – International Data Encryption Standard

ID-PKC – Identity-based Public Key Cryptography

IBE – Identity-based Encryption

KGC – Key Generation Center

AES – Advanced Encryption Standard

P2P – Point-to-Point

KDK – Key Derivation Key

ECC – Elliptic Curve Cryptosystem

ECDLP – Elliptic Curve Discrete Logarithm Problem

CBDHP – Computational Bilinear Diffie-Hellman Problem

DBDHP – Decision Bilinear Diffie-Hellman Problem

OWB – One Way Bilinearity

CL-PKC – Certificateless Public Key Cryptosystem

PM – Point multiplication in group G_1

BP – Bilinear pairing

HASH – Map-to-point hash algorithm [5] [8][$H : \{0, 1\}^* \rightarrow G_1.$]

HASH2 – [10] defined this hash function as [$H : G_2 \rightarrow \{0, 1\}^n$ where n is bit length of plaintexts]

PBC – Pairing Based Cryptography

JPBC – Java Pairing Based Cryptography

HSQLDB – Hyperthreaded structured Query Language Database

GUI – Graphical User Interface

LIST OF SYMBOLS

- \in belongs to or an element of
- \forall for all
- \oplus ex-or
- $\stackrel{c}{\equiv}$ computationally indistinguishable

CHAPTER 1

INTRODUCTION

1.1 Introduction

This paper proposes a Certificateless One-way Group Key Agreement (CLOW-GKA) protocol for Point-to-Point (P2P) email encryption. The main motivations are that (1) email is the most used form of communication between individuals and organizations; (2) most organizations do not value the significance of email security; and (3) most email servers will backup messages the user sends to ensure delivery and thus the possibility of exposing backup messages to public is a real threat if the messages are not P2P encrypted.

P2P email encryption schemes are built on top of Public Key Cryptosystems (PKC). The evolution of the public key cryptography from the traditional PKC to the Identity-based PKC (ID-PKC) and then to the Certificateless PKC (CL-PKC) provides a better and more suitable cryptosystem now for researchers to design secure protocols for P2P email encryption.

The traditional PKC is still used mostly around the world till now, though its certificate infrastructure is not easy to manage. In traditional PKC, each participant assigns their own public and private keys. Thus, it requires a trusted Certificate Authority (CA) to authenticate each participant's public key. The additional server for CA and the management (creation, revocation and storage) of all the certificates

is a major issue the traditional PKC suffers.

PGP (Pretty Good Privacy) [1, 2] probably is the most popular P2P email encryption scheme used in the world now, which was built and implemented based on the traditional PKC. If a person wishes to use PGP to send a secure email, he/she needs to:

1. encrypt the email using the International Data Encryption Algorithm (IDEA) [3],
2. find and confirm the email receivers' public keys by verifying the corresponding public key certificates, and
3. encrypt the IDEA encryption key using the email receivers' public keys.

The encrypted email along with the encrypted IDEA key can then be sent over a regular emailing system. Upon receiving a PGP-encrypted email, each of the email receivers needs to

1. use his/her private key to decrypt the IDEA encryption key, and
2. use the IDEA key to decrypt the email.

The emailing processes listed above are all fairly easy to implement with the exception of finding and verifying email receivers' public keys. This is the main disadvantage of the traditional PKC and is even more of a problem when an email has multiple receivers. In that case, the sender needs to perform the public key verification process for each email receiver. Furthermore, the worse part of the traditional PKC is the requirement of a certificate infrastructure to issue, store, and revoke all the public key certificates.

Shamir [4] was the first one to propose the idea of Identity-based Public Key Cryptosystem ID-PKC to eliminate the certificate infrastructure. In ID-PKC, the participants' public keys are generated directly from their identities through some public known functions (usually hash functions). Thus, every participant can directly derive other participants' public keys from their identities without having to verify the validity of public keys. Since Shamir's paper [4], many ID-PKC schemes such as those in [5, 6, 7, 8, 9] have been proposed in literature. However, ID-PKC schemes require a Key Generation Center (KGC) to generate and distribute private keys for participants. This approach inevitably introduces the key escrow problem, in which if the KGC is malicious or compromised, the whole system will be compromised (since the KGC knows participants' private keys).

In 2003, Al-Riyami et al. [10] introduced a first Certificateless Public Key Cryptosystem (CL-PKC) to solve the key escrow problem. Following his paper, many certificateless protocols [11, 12, 13, 14] were proposed for various applications, including secure encryption, signature, or key agreement. For application like email encryption, certificateless signature schemes are not suitable. Certificateless encryption schemes are not suitable for email encryption either since for an email with multiple receivers the message or the encryption key may need to be encrypted multiple times, once by each email receiver's public key so that each receiver can use his/her private key to decrypt the message (or decrypt the encryption key first and then decrypt the message). Therefore, a secure key agreement protocol should be the best choice to implement the P2P email encryption if the protocol is a one-way and is an n -party protocol. Unfortunately, none of the certificateless key agreement protocols proposed in literature were designed for P2P email encryption and thus are not one-way and n -party.

1.2 Thesis Statement

The proposed CLOW-GKA protocol in this thesis, to the author's knowledge, is the first key agreement protocol with all the following features in one protocol that are required to implement a good P2P email encryption scheme.

- **Certificateless:** Certificateless public key cryptosystem CL-PKC is a relatively new crypto-paradigm. The proposed CLOW-GKA is a key agreement protocol based on the CL-PKC.

Though CL-PKC eliminates the need of the certificate infrastructure (suffered in traditional PKC) and solves the key escrow problem (suffered in ID-PKC), it introduces a milder problem that, unlike ID-PKC, participants in the CL-PKC schemes cannot derive other participants' public keys directly from their identities. Instead, Alice needs to access a public directory to obtain Bob's public key and any well-designed CL-PKC scheme needs to ensure that Bob's public key can be verified by Alice without using certificates. In the proposed CLOW-GKA protocol, all public keys can be certificatelessly verified and no intervention from any server is required during the verification process. In PKI we need an infrastructure which would be complex and expensive, because it requires trusted authorities to obey strict security policies. We also need additional server management and memory for creation, revocation and storage of the certificates. In CLOW-GKA there is no need for certificates or Certificate Authority. Hence there is no need for an additional server and management because the KGC does not store any information. Even though the KGC has to be online for user registration, it is a one time process for each user and the user never contacts the KGC again. But in PKI the user has to contact

the Certificate authority everytime public key authentication is needed. There exists a public directory in CLOW-GKA which contains the public keys where there no need for expensive management as required in the PKI cryptosystem.

- **One-way:** Most existing group key agreement protocols require several rounds of message exchanges before all participants can reach a group key. This approach obviously does not work for P2P email encryption since not all participants are online when the email sender sends emails. The proposed CLOW-GKA protocol requires no back-and-forth message exchange. The sender only needs to encrypt the email using a special constructed group key and then attach a set of Key Derivation Keys (KDKs) in the email, one KDK for each email receiver. Upon receiving the email, each receiver will based on his/her own private key, the KDK in the email and the sender's public key to derive the group key (i.e., the encryption key).
- **n -party:** Most ID-based or certificateless-based group key agreement protocols in literature were designed for 2-party or 3-party only, probably because they are not one-way protocols and thus the amount of back-and-forth broadcast messages may become un-manageable if the group size is n (especially when n is big). The proposed CLOW-GKA has no such limitation and can support arbitrary n parties.
- **Public channels for key distribution:** In order to eliminate the key escrow problem, most CL-PKC protocols require the KGC to assign and distribute a partial private key to each participant through some secret channels. The proposed CLOW-GKA uses the binding-blinding technique [15] to eliminate this secret channel requirement. That is, the KGC can send a quantity, in

which the partial private key is embedded within it, to the participant using any public channel. Only the participant receiving the quantity can recover the partial private key.

Whether the proposed P2P email encryption scheme is secure relies on the security of the underlying CLOW-GKA protocol. In Section 5 of this thesis, we provide a security proof for the CLOW-GKA protocol using a relatively new methodology “proof by simulation”.

This thesis is organized in seven chapters. Chapter 2 briefly describes some related work in literature. Chapter 3 provides some preliminaries, including the elliptic curve cryptography and bilinear pairings. Chapter 4 presents the CLOW-GKA protocol for P2P email encryption. Chapter 5 proves the security of the protocol, followed by analyzing the efficiency in Chapter 6. Chapter 7 presents our implementation of the CLOW-GKA protocol. The thesis is concluded in Chapter 8.

CHAPTER 2

RELATED WORK

2.1 Previous work

Other than PGP, the author did not find any paper in literature aimed at designing protocols specific for P2P email encryption. That is, we cannot identify any existing protocol which has all the four properties we described earlier in the previous section. However, many protocols, which were not designed for email encryption, have some of these four properties and thus are related to our work. In this section, we will briefly describe these related work.

In 2003, Al-Riyami et al. [10] designed the first CL-PKC (certificateless public key cryptosystem) to eliminate the key escrow problem by making the KGC to generate only a partial private key for each participant. Upon receiving a partial private key from KGC, each participant can then use it along with his/her own private key to construct his/her public key. In this way, even a compromised or malicious KGC, with only the knowledge of the partial private key, will not be able to derive any participant's private key and thus the key escrow problem can be prevented. However, in their scheme the partial private keys have to be distributed to participants using some secret channels. In addition, the paper, based on their CL-PKC setting, has only sketched a 2-party (not n -party) key agreement protocol. Furthermore, the protocol is not one-way. That is, it requires back-and-forth message exchanges for both parties

to agree on a key. For P2P email encryption, this key agreement protocol obviously is not suitable.

Xiong, Li and Qin proposed a certificateless threshold signature scheme in [11], which uses Al-Riyami et al.'s [10] CL-PKC to eliminate the key escrow problem. In this paper, threshold signature distributes the signing power among multiple signers in order to share responsibility and authority. Tso, Huang and Susilo proposed another certificateless signature scheme in [12], which eliminates the need for certificate management. Signatures are used for validating users' identities in this scheme. Though these two papers are certificateless and solved the key escrow problem, they are signature schemes and cannot be applied to implement P2P email encryption.

Baek's [13] Certificateless public key encryption without pairing is a scheme where they construct a new CLPKE scheme based on Schnorr signature [27] without bilinear pairings and claim that their computational cost is more efficient than the traditional CLPKE schemes which use pairings. Yang and Tan's [14] paper presented a new security model for certificateless key exchange, and a strongly secure CLKE protocol without pairing. It also presented a formal study on certificateless key exchange. In both these papers, based on their CLPKE setting, has only sketched a 2-party (not n -party) key agreement protocol.

Das's paper in [15] is a ID-PKC scheme which does not require a secure channel to pass the private key from the KGC. In this paper they use a binding-blinding techniques with the use of many parameters over the public channel to the KGC, and thus is less efficient. Unlike their scheme, in our scheme we just send one parameter over the public channel (please see Section 4.3 for details). Furthermore, the paper described a signature scheme which cannot be applied to P2P email encryption.

Boneh and Franklin's paper in [5] proposed an ID-PKC scheme which uses

multiple KGCs to eliminate the key escrow problem. The KGC infrastructure in this scheme is distributed and the master private key is jointly computed by these multiple KGC's in a way that no particular KGC has knowledge about the master private key. This proposed use of multiple KGCs to solve the key escrow problem is not feasible since the system infrastructure would be more complex to set up and harder to maintain.

Lin and Zhou [16] also proposed a P2P scheme based on ID-PKC but solves the key escrow problem based on CL-PKC. It is a P2P scheme designed for two-party communication only, unlike our scheme which can support unlimited n parties. The paper also proposed a scheme for sharing trust data through the concept of each party having a trust value. The trust value of one party is randomly assigned to another party in the network. The KGC is responsible for equally distributing the trust values within the parties in the network. Any party A wanting to know the trust value of another party B broadcasts a trust query and the party C which holds the trust value of B replies back the value. Based on the received trust value, the party A can decide to communicate with B further or not.

Zhang, Wu and Qin proposed an ID-PKC group key agreement protocol in [17], where group members merely negotiate a common encryption key which is accessible to any group member, but they hold their respective secret decryption keys. It is a one-way protocol. This protocol needs to use multi-signatures to verify identities and in addition it suffers from the key escrow problem since the KGC generates the private keys for the participants. This paper uses the random oracle method for their security proof. In our thesis, we use a relatively new method “proof by simulation” to prove the security.

Yeh, Zeng and Long's paper in [18] proposed an ID-PKC protocol for email

encryption. Like most of other ID-PKC protocols, this protocol suffers from the key escrow problem. Thus, the email encryption scheme proposed in their paper is not a truly P2P email encryption since the KGC is capable of deriving the encryption key to decrypt emails. In addition, their protocol requires secret channels for private key distribution.

Yuen, Susilo and Mu's paper in [19] proposed an ID-PKC scheme to solve the key escrow problem, in which each user has his/her own public key and secret key. The KGC generates the identity-based secret key for the user with respect to user's public key. Now the user uses both secret keys (his own secret key and ID-based secret key) to sign a message. In this way, the key escrow problem can be solved since the KGC only knows one of the two secret keys and therefore cannot forge any user's signatures.

There has been new schemes coming in to improve security, management, complexity, certificateless, lesser cost and higher efficiency. Signatures schemes are in the limelight in recent years and have been extensively used in many applications. Though this paper presents a key agreement protocol, the proposed cryptosystem setting can be easily adapted to develop efficient digital signature schemes. Unlike our CLOW-GKA protocol, all the related papers described here do not have all those four desired features and thus are not suitable for P2P email encryption.

2.2 Functionality comparison with different protocols

Table 2 provides a brief functionality comparison between the proposed CLOW-GKA protocol: P2P email encryption, PGP (Pretty Good Privacy) [1, 2] P2P email encryption, Yeh, Zeng and Long's [18] ID-PKC protocol for email encryption and

Al-Riyami et al. [10] CL-PKC (certificateless public key cryptosystem).

Table 2.1: Functionality comparison between CLOW-GKA, PGP, Yeh, Zeng and Long’s ID-PKC and Al-Riyami et al’s CL-PKC, assuming there are n recipients in an email

	CLOW-GKA	Yeh, Zeng and Long’s ID-PKC	Al-Riyami et al’s CL-PKC	PGP
Certificateless	• YES	• YES	• YES	• NO
One-way	• YES	• YES	• YES	• YES
n-party	• YES	• YES	• NO	• YES
No private channels	• YES	• NO	• NO	• NO
Solves Key escrow problem	• YES	• NO	• YES	• Not applicable
Digital signature	• NO	• NO	• NO	• YES
Message replay attack prevention	• NO	• NO	• NO	• NO

2.3 Existing email encryption softwares

Currently, there are some commercial secure email encryption softwares available in the market today. HPE Secure email [33], Trend Micro email [34], Proofpoint [36] and DataMotion SecureMail [35] are some of the most used email encryption software which use Identity-Based Encryption (IBE). LuxSci SecureLine [37], ProtonMail [38] and Mailvelope [39] are other email encryption software based on OpenPGP. Most of these commercial software can be integrated with a variety of email provider services.

HPE Secure email is offered by Hewlett-Packard Enterprises and it is one of the top reviewed products in the market today. HPE secure email is based on the Identity-Based Encryption which works as follows. Let us consider the situation where Alice wants to send an email to Bob. Alice encrypts the email using Bobs email address,

as the public key. The encrypted email is then sent to the Bob. When bob receives the email, bob contacts the key server. The key server then contacts a directory or an external authentication source to authenticate the identity of Bob. Once Bob is authenticated, the key server sends Bob's private key to Bob, with which Bob can decrypt the received message. Bob can use this private key to decrypt any following messages too. Trend Micro Email also follows the same setup and their key server generates private keys based on the Identity-Based Cryptosystem. Proofpoint and DataMotion SecureMail also claim they use the Identity-Based Encryption.

Since all the above software products use IBE, the key server will generate the private keys for all the users. This leads to the key escrow problem, in which if the key server is malicious or compromised, the whole system will be compromised (since the key server knows participants private keys). This key escrow problem will be overcome in our proposed email encryption system based on CLOW-GKA protocol.

ProtonMail is an email service provider like google which uses OpenPGP. Mailvelope is a chrome extension which can be integrated within the email service provider's interface. It is very useful since it does not have to develop/create any separate interface. Mailvelope also has a firefox addon. LuxSci SecureLine also uses OpenPGP and claim to provide point-to-point email encryption. The disadvantage of PGP, as discussed in Section 1.1, is the required certificate infrastructure for verifying email receivers' public keys. In our CLOW-GKA implementation all the public keys can be verified by the sender without the need of public key certificates or any other mechanism.

Most software products currently in the market still use the traditional public key infrastructure or PGP/OpenPGP for key management which is very expensive to maintain. Some newer email encryption software use the Identity-Based Encryption.

Though they eliminate the burden of key management but unfortunately it introduces the key escrow problem. Our protocol eliminates both the public key infrastructure requirement and the key escrow problem. In addition, our approach is based on a key agreement protocol, where the email is encrypted by a key which can be agreed (derived) by all the email participants. Thus, no need for multiple encryptions for a group email with multiple email receivers.

CHAPTER 3

PRELIMINARY

In this section, we describe the cryptographic basics of elliptic curves and bilinear pairings that are used frequently in ID-based and certificateless-based cryptosystems.

3.1 Elliptic curves

Elliptic curve cryptosystem (ECC) [23, 24] is a popular cryptosystem used in many different cryptographic protocols.

To set up an ECC, based on the National Security Agency's recommendation [25], an appropriate prime curve is $E_p(a, b) : y^2 = x^3 + ax + b$ over a prime p and a base point (generator) P with an order q (i.e., $qP = O$), where $(4a^3 + 27b^2) \not\equiv 0 \pmod{p}$ and O is the infinity point. An ECC $E_p(a, b)$ is an additive group with the following operation rules. For all points $A, B, C \in E_p(a, b)$,

1. $A + O = A$.
2. If $A = (x_A, y_A)$, then $-A = (x_A, -y_A)$ and thus $A + (x_A, -y_A) = O$.
3. Given $A = (x_A, y_A)$ and $B = (x_B, y_B)$ with $A \neq -B$, another point $C = A + B = (x_C, y_C)$ can be calculated by the rules below:

$$x_C = (\lambda^2 - x_A - x_B) \bmod p$$

$$y_C = (\lambda(x_A - x_C) - y_A) \bmod p$$

where

$$\lambda = \begin{cases} \frac{y_B - y_A}{x_B - x_A} \bmod p & \text{if } A \neq B \\ \frac{3x_A^2 + a}{2y_A} \bmod p & \text{if } A = B \end{cases}$$

4. Multiplication is defined as repeated additions. For example, for some positive integer k , the multiplication $k \cdot A = A + A + \dots + A$, i.e., A adds itself $k - 1$ times.

There are many ECC encryption and signature algorithms available in literature. All these algorithms are based on the hardness assumption of the Elliptic Curve Discrete Logarithm Problem that states:

- Elliptic Curve Discrete Logarithm Problem (ECDLP): Given two points A and B on a curve $E_p(a, b)$ with an order q , if $B = r \cdot A$ for some $r \in \mathbb{Z}_q^*$, it is computational hard to find r .

3.2 Bilinear pairings

Bilinear pairing has found recent use in various efficient encryption and signature schemes [5, 6, 7, 9, 20, 21, 22]. A symmetric bilinear pairings cryptosystem is described briefly in this section.

Let $(G_1, +)$ and (G_2, \times) be two cyclic groups of the same prime order q , and let P be the generator of the additive group G_1 , and $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear mapping if it has the following properties:

Bilinearity:

$\forall A, B, C \in G_1$, and $\forall r_1, r_2 \in Z_q^*$,

$$e(A, B) = e(B, A)$$

$$e(A, B + C) = e(A, B)e(A, C)$$

and thus

$$e(r_1A, r_2B) = e(A, B)^{r_1r_2} = e(r_2A, r_1B)$$

Non-degeneracy:

If P is a generator of G_1 , then $e(P, P)$ is a generator of G_2 .

Computability:

$\forall A, B \in G_1$, there exists a polynomial-time algorithm to efficiently compute the bilinear mapping $e(A, B)$.

An elliptic curve is a typical example that can be used as the G_1 group. The security of most bilinear mapping based cryptographic algorithms is related to the hardness assumption of the bilinear variants of the Diffie-Hellman problems, which are:

- Computational Bilinear Diffie-Hellman Problem (CBDHP): Given A, r_1A, r_2A for $r_1, r_2 \in Z_q^*$, compute r_1r_2A .
- Decision Bilinear Diffie-Hellman Problem (DBDHP): Given A, r_1A, r_2A, r_3A for $r_1, r_2, r_3 \in Z_q^*$, and an element $g \in G_2$, decide if $g = e(A, A)^{r_1r_2r_3}$.
- One Way Bilinearity (OWB): Given a random element $g \in G_2$, find a pair $(A_1, A_2) \in G_1$ such that $e(A_1, A_2) = g$.

There is no algorithm available currently which is able to efficiently solve these hard problems.

CHAPTER 4

THE CLOW-GKA PROTOCOL FOR P2P EMAIL ENCRYPTION

This section proposes the CLOW-GKA (certificateless one-way group key agreement) protocol and describes its application to P2P email encryption.

4.1 KGC servers

For the proposed CLOW-GKA protocol to allow an email sender and a group of n (≥ 1) email receivers to agree on a key for P2P email encryption, all of the email participants must register through a same KGC (key generation center). This can be easily achieved by letting the email service provider (such as Google) act as the KGC. For some security sensitive organizations, for example, government agencies may want to establish their own internal email system for their employees only, the agency can also have a dedicated server serves as the KGC.

However, no matter in which cases (public or internal), the KGC will not be able to know any participant's private key, and thus not able to derive any agreed group key (i.e., email encryption key) and subsequently decrypt any of the emails if the KGC is compromised. Thus, the proposed CLOW-GKA protocol can implement a truly P2P email encryption scheme, i.e., no other entity (except the email sender and receivers) is able to decrypt the emails.

4.2 CLOW-GKA for P2P email encryption

In this section, we formally present the CLOW-GKA protocol for P2P email encryption. Within this protocol, several sub-protocols will be called and executed. The entities participating in the protocol are the KGC and the email users, where each email user can play different roles either as an email sender or as an email receiver.

CLOW-GKA protocol: P2P email encryption

Assume an email sender with an identity ID_0 would like to send a P2P encrypted email to $n \geq 1$ email receivers with identities $ID_i, \forall 1 \leq i \leq n$. This protocol is based on a cryptosystem with a set of parameters set up by the KGC. The KGC, ID_0 , and all ID_i need to execute the steps described in this protocol.

1. The KGC server defines a cryptosystem and publishes a set of parameters $\langle G_1, G_2, e, P, P_{pub}, q, H \rangle$, where parameters G_1, G_2, e, P and q were defined in Section 3.2. The KGC also chooses a master secret $s \in Z_q^*$ and then computes $P_{pub} = sP$ as the system's public key. Finally, the KGC chooses a hash function $H : \{0, 1\}^* \rightarrow G_1$. The map-to-curve and map-to-point algorithms from Weil pairings in [5, 8] are such functions that can map users' identities to points on an elliptic curve (the group G_1).
2. Each email user, including ID_0 and all $ID_i, \forall 1 \leq i \leq n$, needs to have an email account before sending/receiving emails. Each email user executes the **Sub-protocol 1** to register an email account through the KGC. In Sub-protocol 1, each email user will generate his/her private and public keys.

3. Each email user, including ID_0 and all ID_i , publishes his/her public key along with his/her identity in a public directory.
4. The email sender ID_0 executes the **Sub-protocol 2** to send an encrypted email to all email receivers ID_i .
5. Each email receiver ID_i executes the **Sub-protocol 3** to receive and decrypt the email.

4.3 Email account registration

All email users need to register an email account through the KGC first before they can send or receive any P2P encrypted email.

Sub-protocol 1: Email account registration

Input: An email user with an identity ID_j , $\forall 0 \leq j \leq n$, and a random number r_j .

Output: ID_j 's private key s_j and public key $\langle P_j, R_j \rangle$.

A user ID_j registers an email account and generates his/her private and public keys through the KGC by the following steps:

1. ID_j computes and sends $r_j Q_j$ to KGC, where r_j is a random secret for ID_j and $Q_j = H(ID_j)$.
2. KGC computes $D_j = sr_j Q_j$ and sends it back to ID_j through any public channel, in which a partial private key sQ_j is embedded in D_j .

3. ID_j chooses his/her private key s_j and then computes his/her public key $\langle P_j, R_j \rangle$, where

$$P_j = s_j P \quad (4.1)$$

$$R_j = s_j^{-1} r_j^{-1} D_j = s_j^{-1} r_j^{-1} r_j s Q_j = s_j^{-1} s Q_j \quad (4.2)$$

Note that in the step 2 of the Sub-protocol 1, unlike other ID-based or certificateless-based protocols, the proposed CLOW-GKA protocol does not require secret channels for the KGC to distribute the partial private key sQ_j . We use the binding-blinding technique to avoid the requirement of secret channels. Rather than sending the partial private key sQ_j directly, the KGC sends $D_j = sr_jQ_j$ to ID_j through any public channel, where only the ID_j knows the random secret r_j . Based on the hardness assumption of the Elliptic Curve Discrete Logarithm problem, it is hard for everyone (except the KGC and the user ID_j) to derive the partial private key sQ_j even he/she eavesdrops D_j from the public channel. The secrecy of the partial private key sQ_j is important to the security of the protocol since if a third party is able to eavesdrop sQ_j , then this party can create a fake public key for ID_j . This malicious party just needs to choose his/her own random secret s'_j , and based on the eavesdropped sQ_j , to generate $\langle P'_j, R'_j \rangle$ and then claims that this fake public key is ID_j 's public key. Unfortunately the system cannot detect this illegal fabrication. That is, the fake public key can pass the system provided public key verification check (see Equation 4.3 below).

4.4 Sending a P2P encrypted email

The email sender ID_0 needs to execute the Sub-protocol 2 below to send a P2P encrypted email to a group of n email receivers $ID_i, \forall 1 \leq i \leq n$.

Sub-protocol 2: Sending a P2P encrypted email

To send a P2P encrypted email to n email receivers $ID_i, \forall 1 \leq i \leq n$, the sender ID_0 needs to perform the following steps:

1. ID_0 obtains each email receiver ID_i 's public key $\langle P_i, R_i \rangle$ from the public directory.
2. ID_0 verifies each ID_i 's public key by checking the equality of the Equation 4.3 below.

$$e(P_i, R_i) \stackrel{?}{=} e(P_{pub}, Q_i) \quad (4.3)$$

since

$$e(P_i, R_i) = e(s_i P, s_i^{-1} s Q_i) = e(s P, s_i s_i^{-1} Q_i) = e(P_{pub}, Q_i).$$

If the equality checking in Equation 4.3 returns true, the validity of the public key $\langle P_i, R_i \rangle$ is verified.

3. ID_0 chooses a random number $r \in Z_q^*$ and computes

$$x_j = e(r Q_0, s_0 P_j), \quad \forall 0 \leq j \leq n \quad (4.4)$$

4. ID_0 computes n key derivation keys y_i 's, one for each email receiver ID_i , where

$$y_i = \oplus_{j=0,1,\dots,i-1,i+1,\dots,n}(x_j), \quad \forall 1 \leq i \leq n \quad (4.5)$$

5. ID_0 computes the group key (will be used to encrypt the email message)

$$K = x_0 \oplus x_1 \oplus \dots \oplus x_n \quad (4.6)$$

6. ID_0 uses the group key K to encrypt the email message and then sends the encrypted email along with the key derivation keys and the random number $(y_1, y_2, \dots, y_n, r)$ as an attachment to all n email receivers ID_i 's.

For the Equation 4.3, the email sender only needs to use the KGC's public information P_{pub} and the email receiver's identity ID_i (or $Q_i = H(ID_i)$) to verify the validity of the ID_i 's public key $\langle P_i, R_i \rangle$. This certificateless public-key verification capability (i.e., Any user can verify another user's public key without the need to have a third party to serve as a certificate authority) is an important feature for a well-designed certificateless protocol.

4.5 Receiving a P2P encrypted email

Upon receiving the P2P encrypted email from the email sender ID_0 , each email receiver ID_i , $\forall 1 \leq i \leq n$, needs to execute the Sub-protocol 3 to derive the group key K and then use it to decrypt the message.

Sub-protocol 3: Receiving a P2P encrypted email

Each email receiver $ID_i, \forall 1 \leq i \leq n$, performs the following three steps:

1. ID_i first obtains the email sender ID_0 's public key $\langle P_0, R_0 \rangle$ from the public directory.
2. ID_i checks the validity of ID_0 's public key $\langle P_0, R_0 \rangle$ based on the Equation 4.3.
3. ID_i derives the group key K using his/her own private key s_i , the corresponding key derivation key y_i , and the email sender ID_0 's identity Q_0 ($= H(ID_0)$) and public key P_0 . Equation 4.7 below describes this key derivation process.

$$K = y_i \oplus e(s_i Q_0, r P_0) \quad (4.7)$$

since

$$\begin{aligned}
 y_i \oplus e(s_i Q_0, r P_0) &= y_i \oplus e(r Q_0, s_i P_0) \\
 &= y_i \oplus e(r Q_0, s_i s_0 P) \\
 &= y_i \oplus e(r Q_0, s_0 P_i) \\
 &= y_i \oplus x_i \\
 &= (x_0 \oplus \dots \oplus x_{i-1} \oplus x_{i+1} \oplus \dots \oplus x_n) \oplus x_i \\
 &= K
 \end{aligned}$$

4. Finally, ID_i can decrypt the email using the just derived group key K .

4.6 Example

For demonstration purpose, this section will give a walk through example. In this example, we assume the email sender ID_0 is sending an email to $n = 3$ receivers ID_1, ID_2 , and ID_3 .

1. **Cryptosystem setup:** The KGC sets up a cryptosystem with all the parameters as described in the step 1 of CLOW-GKA protocol in Section 4.2.
2. **Email account registration:** Each of the users ID_0, ID_1, ID_2 and ID_3 registers an email account through the KGC and during the registration, their public keys will be generated.
3. **Public key directory:** All the users ID_0, ID_1, ID_2 and ID_3 publish their public keys in a public directory.
4. **Sending an encrypted email:**

(1) The email sender ID_0 accesses the public directory to obtain all email receiver ID_1 's, ID_2 's and ID_3 's public keys $\langle P_1, R_1 \rangle, \langle P_2, R_2 \rangle, \langle P_3, R_3 \rangle$.

(2) The email sender ID_0 , based on Equation 4.3, verifies the validity of all three public keys by checking

$$e(P_1, R_1) \stackrel{?}{=} e(P_{pub}, Q_1);$$

$$e(P_2, R_2) \stackrel{?}{=} e(P_{pub}, Q_2);$$

$$e(P_3, R_3) \stackrel{?}{=} e(P_{pub}, Q_3)$$

since

$$e(P_1, R_1) = e(s_1P, s_1^{-1}sQ_1) = e(sP, s_1s_1^{-1}Q_1) = e(P_{pub}, Q_1)$$

$$e(P_2, R_2) = e(s_2P, s_2^{-1}sQ_2) = e(sP, s_2s_2^{-1}Q_2) = e(P_{pub}, Q_2)$$

$$e(P_3, R_3) = e(s_3P, s_3^{-1}sQ_3) = e(sP, s_3s_3^{-1}Q_3) = e(P_{pub}, Q_3)$$

(3) The sender ID_0 picks a random number r and computes

$$\begin{cases} x_0 = e(rQ_0, s_0P_0) \\ x_1 = e(rQ_0, s_0P_1) \\ x_2 = e(rQ_0, s_0P_2) \\ x_3 = e(rQ_0, s_0P_3) \end{cases}$$

(4) The sender ID_0 computes three key derivation keys as follows.

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

(5) The sender ID_0 generates the group (encryption) key

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

(6) The sender encrypts the email using the group key K and sends (y_1, y_2, y_3, r) along with the email.

5. Receiving an encrypted email:

(1) Each email receiver ID_1, ID_2 or ID_3 needs to access the public directory to obtain ID_0 's public key $\langle P_0, R_0 \rangle$.

(2) Based on Equation 4.3, each email receiver ID_1, ID_2 or ID_3 verifies the sender ID_0 's public key $\langle P_0, R_0 \rangle$ by checking $e(P_0, R_0) \stackrel{?}{=} e(P_{pub}, Q_0)$.

(3) The email receiver ID_1 computes

$$\begin{aligned}
 y_1 \oplus e(s_1 Q_0, r P_0) &= x_0 \oplus x_2 \oplus x_3 \oplus e(r Q_0, s_1 P_0) \\
 &= x_0 \oplus x_2 \oplus x_3 \oplus e(r Q_0, s_1 s_0 P) \\
 &= x_0 \oplus x_2 \oplus x_3 \oplus e(r Q_0, s_0 P_1) \\
 &= x_0 \oplus x_2 \oplus x_3 \oplus x_1 \\
 &= K
 \end{aligned}$$

The email receiver ID_2 computes

$$\begin{aligned}
 y_2 \oplus e(s_2 Q_0, r P_0) &= x_0 \oplus x_1 \oplus x_3 \oplus e(r Q_0, s_2 P_0) \\
 &= x_0 \oplus x_1 \oplus x_3 \oplus e(r Q_0, s_2 s_0 P) \\
 &= x_0 \oplus x_1 \oplus x_3 \oplus e(r Q_0, s_0 P_2) \\
 &= x_0 \oplus x_1 \oplus x_3 \oplus x_2 \\
 &= K
 \end{aligned}$$

The email receiver ID_3 computes

$$\begin{aligned}
 y_3 \oplus e(s_3 Q_0, r P_0) &= x_0 \oplus x_1 \oplus x_2 \oplus e(r Q_0, s_3 P_0) \\
 &= x_0 \oplus x_1 \oplus x_2 \oplus e(r Q_0, s_3 s_0 P) \\
 &= x_0 \oplus x_1 \oplus x_2 \oplus e(r Q_0, s_0 P_3) \\
 &= x_0 \oplus x_1 \oplus x_2 \oplus x_3 \\
 &= K
 \end{aligned}$$

Thus, all three email receivers are able to derive the same group key K which was generated and used by the email sender to encrypt the email.

- (4) Finally all three email receivers can decrypt the encrypted email using the group key K .

CHAPTER 5

SECURITY ANALYSIS

5.1 Adversarial model

CLOW-GKA protocol is secure against static and non-colluding adversaries in the semi-honest adversarial model, also called *honest-but-curious*. The adversary obtains the internal state of the corrupted party, including the transcripts of all messages received. It correctly follows the protocol specification while attempting to learn private information from other parties. This model may also be of use in settings where the use of the correct software running the correct protocol can be enforced. Semi-honest adversaries are also called passive. Semi-honest adversaries can also be static (the party is corrupted before the execution of the protocol) In the rest of this section, we prove the security of our protocol in the setting of semi-honest adversarial model.

5.2 Privacy by simulation

According to the simulation paradigm [26], a protocol is secure against semi-honest adversaries if whatever can be computed by a party participating in the protocol can be computed based merely on its input and output. If the view of a party is simulatable based only on the party's input and output, then whatever the adversary

can learn from the input and output of the corrupted party, can be learned without the execution of the protocol. If the view each party in the protocol is simulatable based only on the it's input and output, then we can conclude that the protocol is privacy-preserving, and consequently secure.

Definition of security

- Let $f = (f_1, \dots, f_m)$ be a m -ary probabilistic polynomial-time functionality and let Π be a multi-party protocol for computing f .
- The view of the i th party ($i \in 1, 2, \dots, m$) during an execution of Π on $x = (x_1, \dots, x_m)$ and security parameter n is denoted by $\text{view}_i^\Pi(x, n)$ and equals $(w, r_i, m_{i,1}, \dots, m_{i,t})$ where $w \in x = (x_1, \dots, x_m)$ (its value depending on the value of i), r_i equals the contents of the i th party's internal random tape, and $m_{i,j}$ represents the j th message that it received.
- The output of the i th party during an execution of Π on $x = (x_1, \dots, x_m)$ and security parameter n is denoted by $\text{output}_i^\Pi(x, n)$ and can be computed from its own view of the execution. We denote the joint output of all parties by $\text{output}^\Pi(x, n) = (\text{output}_1^\Pi(x, n), \dots, \text{output}_m^\Pi(x, n))$

In the case where the functionality f is deterministic (our protocol is deterministic too), a simpler definition can be used. Specifically, we do not need to consider the joint distribution of the simulators output with the protocol output. Rather we separately require correctness, meaning that

$$\{\text{output}^\Pi(x, n)\}_{x \in (\{0,1\}^*); n \in \mathbb{N}} \stackrel{c}{\equiv} \{f(x, n)\}_{x \in (\{0,1\}^*); n \in \mathbb{N}}$$

and in addition, that there exist probabilistic polynomial-time algorithm (simulator) S such that for every $I \subseteq \{1, \dots, m\}$:

$$\{S_I(1^n, x_I, f_I(x))\}_{x \in (\{0,1\}^*) ; n \in N} \stackrel{c}{\equiv} \{\text{view}_I^\Pi(x, n)\}_{x \in (\{0,1\}^*) ; n \in N}$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability. \square

According to the security definition above, it is sufficient to show that we can effectively simulate the view of each party in the two phases of CLOW-GKA protocol given only the input and output of that party, in order to prove the protocol is secure.

1. User Registration Phase:

- **Scenario 1:** We will simulate the view of the user trying to register with the KGC. Here the user would interact with a simulator (the simulator will simulate KGC in the ideal world).
- **Scenario 2:** We will simulate the view of the KGC. Here the KGC would interact with the simulator (the simulator will simulate the user).

2. Email Sending and Receiving Phase:

- **Scenario 3:** We will simulate the view of one receiver. Here the receiver will only interact with the simulator (the simulator will simulate the sender as well as other receivers).

5.2.1 User Registration Phase

Theorem 1: *Let f be a polynomial-time two-party single-output functionality that takes as input two private keys s_j from user ID_j and D_j from KGC, and*

outputs a public key pair $\langle P_j, R_j \rangle$. Then Sub-protocol 1 securely computes f in the presence of static semi-honest adversaries.

Proof. In Sub-protocol 1, a user with an identity ID_j first sends a parameter to KGC. Then the KGC computes the partial private key and sends it back to ID_j , which uses it to compute its public key pairs. The interaction between KGC and ID_j is secure due to the binding-blinding technique which has the hardness assumption of the Elliptic Curve Discrete Logarithm problem [29]. According to the simulation paradigm [26], A protocol is secure against static semi-honest adversaries if whatever can be computed by a party participating in the protocol can be computed based on its input and output only.

Case 1 - User with identity ID_j is corrupted. We construct a simulator \mathcal{S}_1 that is given private key s_j and public key $\langle P_j, R_j \rangle$ as inputs, and it generates the view of ID_j in Sub-protocol 1. We need to prove that:

$$\{\mathcal{S}_1(s_j, \langle P_j, R_j \rangle)\}_{s_j, P_j, R_j \in \{0,1\}^*} \stackrel{c}{\equiv} \{\text{view}_{ID_j}^{\Pi}(s_j, s)\}_{s_j, s \in \{0,1\}^*} \quad (5.1)$$

where $\mathcal{S}_1(s_j, \langle P_j, R_j \rangle)$ is the simulator as defined above and Π denotes Sub-protocol 1, by proving that each simulated message received by ID_j in Sub-protocol 1 is indistinguishable from a message received in the real world. In Step 2 of Sub-protocol 1, ID_j receives $D_j = sr_j Q_j$. Therefore, \mathcal{S}_1 must generate D_j and send it to ID_j such that ID_j is able to obtain $\langle P_j, R_j \rangle$. However, \mathcal{S}_1 cannot honestly generate D_j as it does not know KGC's master key s . The crux of this proof is in showing that D'_j generated by \mathcal{S}_1 is computationally indistinguishable from D_j that ID_j receives in the real protocol execution. \mathcal{S}_1

uniformly chooses random number r'_j that ID_j would use to generate $r_j Q_j$ (in Step 1). Next, we know that

$$R_j = s_j^{-1} r_j'^{-1} D_j \quad (5.2)$$

So rewriting Equation 5.2 we get

$$D_j = R_j s_j r'_j$$

Since \mathcal{S}_1 knows R_j , s_j and r'_j , \mathcal{S}_1 is able to compute D_j without knowing s and sends it to ID_j . ID_j can now generate $\langle P_j, R_j \rangle$ as it does in the real execution.

Case 2 KGC is corrupted. We construct a simulator \mathcal{S}_2 that is given master key of KGC s and public key $\langle P_j, R_j \rangle$ as inputs, and generates the view of KGC in Sub-protocol 1. We need to prove that:

$$\{\mathcal{S}_2(s, \langle P_j, R_j \rangle)\}_{s, P_j, R_j \in \{0,1\}^*} \stackrel{c}{\equiv} \{\text{view}_{KGC}^\Pi(s_j, s)\}_{s_j, s \in \{0,1\}^*} \quad (5.3)$$

where $\mathcal{S}_2(s, \langle P_j, R_j \rangle)$ is the simulator as defined above and Π denotes Sub-protocol 1. In Step 1 of Sub-protocol 1, KGC receives $r_j Q_j$ from ID_j . \mathcal{S}_2 computes $Q_j = H(ID_j)$. Next, it uniformly chooses random number r'_j , computes $r'_j Q_j$ and send it to KGC. $r_j Q_j$ is computationally indistinguishable from $r'_j Q_j$ as both r_j and r'_j were randomly chosen from a uniform distribution. \square

5.2.2 Email Sending and Receiving Phase

Theorem 2: *Let $f: \{0, 1\}^* \times \{0, 1\}^* \times \dots \times \{0, 1\}^* \rightarrow \{0, 1\}^*$ be a polynomial-time multi-party single-output functionality, and let Sub-protocol 1 be a secure user registration protocol in the presence of static semi-honest adversaries. Sub-protocol 2 and Sub-protocol 3 securely computes f in the presence of static semi-honest adversaries.*

Proof: In Sub-protocol 2, a sender with identity ID_0 sends an encrypted email with some parameters to n email receivers $ID_i, \forall 1 \leq i \leq n$. In Sub-protocol 3, each receiver ID_i receives an encrypted email with some parameters, derives the decryption key, and then decrypts the email. In Step 2 of both Sub-protocol 2 and Sub-protocol 3, the user checks the validity of other user's public key, e.g., ID_0 checks for the validity of ID_i 's public key pair $\langle P_i, R_i \rangle$:

$$e(P_i, R_i) = e(P_{pub}, Q_i) \quad (5.4)$$

since: $e(P_i, R_i) = e(s_i P, s_i^{-1} s Q_i) = e(s P, s_i s_i^{-1} Q_i) = e(P_{pub}, Q_i)$

Although parameters P_i, R_i, P_{pub}, Q_i are all public, no secret information can be derived from these parameters due to the hardness assumption of the Elliptic Curve Discrete Logarithm problem [29]. Note that since in Sub-protocol 2 and Sub-protocol 3 the sender does not receive any message, there is no need to simulate the view of the sender. We only need to simulate the view of one receiver (let us assume it to be ID_1) in each sub-protocol to prove they are secure according to the simulation paradigm.

Case 3 - Receiver with identity ID_1 is corrupted:

In this case, we construct a simulator \mathcal{S}_3 that is given encryption key K and private key s_1 of ID_1 as inputs, and generates the view of ID_1 in Sub-protocol 2 and Sub-protocol 3. \mathcal{S}_3 will simulate sender ID_0 and other receivers $ID_i, \forall 2 \leq i \leq n$. We need to prove that:

$$\{\mathcal{S}_3(s_1, K)\}_{s_1, K \in \{0,1\}^*} \stackrel{c}{\equiv} \{\mathbf{view}_{ID_1}^{\Pi}(s_1, (x_i, \forall 0 \leq i \leq n))\}_{s_1, (x_i, \forall 0 \leq i \leq n) \in \{0,1\}^*} \quad (5.5)$$

where $\mathcal{S}_3(s_1, K)$ is the simulator as defined above and Π denotes Sub-protocol 2 and Sub-protocol 3 combined. \mathcal{S}_3 uniformly chooses random number r' that ID_0 would choose. Notice that ID_0 first computes x_0 and $x_i, \forall 1 \leq i \leq n$ using its private key s_0 in the real execution. \mathcal{S}_3 cannot honestly compute x_0 and $x_i, \forall 1 \leq i \leq n$ because it does not know private key s_0 of ID_0 . Next, ID_0 computes $y_i, \forall 1 \leq i \leq n$ and K and encrypts the email using K and sends $(y_1, y_2, \dots, y_n, r)$ along with the encrypted email in the real protocol execution. The crux of this proof is in showing that $(y_1, y'_2, \dots, y'_n, r')$ along with the encrypted email (generated by \mathcal{S}_3) is computationally indistinguishable from the real $(y_1, y_2, \dots, y_n, r)$ along with the encrypted email that ID_1 receives in the real protocol execution. We know that:

$$x_1 = e(r'Q_0, s_0P_1)$$

\mathcal{S}_3 computes x_1 with input (s_1, K) as follows:

$$\begin{aligned}
x_1 &= e(s_1 Q_0, r' P_0) \\
&= e(r' Q_0, s_1 s_0 P) \\
&= e(r' Q_0, s_0 P_1) \text{ (as computed by } ID_0)
\end{aligned}$$

\mathcal{S}_3 now knows s_1, Q_0, r', P_0 , and \mathcal{S}_3 . Since:

$$\begin{aligned}
y_1 &= x_0 \oplus x_2 \oplus \dots \oplus x_n \\
K &= x_0 \oplus x_1 \oplus x_2 \oplus \dots \oplus x_n
\end{aligned}$$

\mathcal{S}_3 computes y_1 as follows:

$$\begin{aligned}
y_1 &= K \oplus x_1 \\
&= x_0 \oplus x_1 \oplus x_2 \oplus \dots \oplus x_n \oplus x_1 \\
&= x_0 \oplus x_2 \oplus \dots \oplus x_n \text{ (as computed by } ID_0)
\end{aligned}$$

\mathcal{S}_3 now has K, r', y_1 . It can now encrypt email using K as in real execution. Since \mathcal{S}_3 uses the same K as in real execution the encrypted email generated by \mathcal{S}_3 will be computationally indistinguishable from the encrypted email generated in the real execution. Even though \mathcal{S}_3 has all the parameters that receiver ID_1 needs to derive back K and decrypt the email we still need to make sure the message $(y_1, y'_2, \dots, y'_n, r')$ along with the encrypted email is computationally indistinguishable from the real execution message. So \mathcal{S}_3 has to compute y'_2, \dots, y'_n . But \mathcal{S}_3 does not know x_0 and $x_i, \forall 2 \leq i \leq n$. Therefore, \mathcal{S}_3 randomly chooses $n-1$ elliptic curve points and applies bilinear mapping to generate x'_0 and $x'_i, \forall 2 \leq i \leq n$, from which it computes y'_2, \dots, y'_n as shown in Equation 4.5.

Next, \mathcal{S}_3 sends $(y_1, y'_2, \dots, y'_n, r')$ along with the encrypted email to receiver ID_1 . Since x'_0 and $x'_i, \forall 2 \leq i \leq n$ were randomly generated and based on the One-way Bilinearity problem [28], y'_2, \dots, y'_n are computationally indistinguishable from the real y_2, \dots, y_n . \square

Theorem 3: *Let Π be a polynomial-time multi-party protocol which is CLOW-GKA protocol: P2P email encryption and let Sub-protocol 1, Sub-protocol 2 and Sub-protocol 3 be secure in the presence of static semi-honest adversaries, then Π securely executes in the presence of static semi-honest adversaries.*

Proof: By Theorem 1 it is possible to securely execute Sub-protocol 1 and by Theorem 2 it is possible to securely execute Sub-protocol 2 and Sub-protocol 3 assuming the existence of static semi-honest adversaries. Furthermore, recall that CLOW-GKA protocol: P2P email encryption is made up of the sub-protocol blocks mentioned above. Combining these facts with Theorem 3 we can come to the conclusion that CLOW-GKA protocol: P2P email encryption is secure in the presence of static semi-honest adversaries.

CHAPTER 6

EFFICIENCY ANALYSIS

In this section we analyze our proposed CLOW-GKA protocol: P2P email encryption scheme in four subsections. First, we analyze the computational cost for each of our protocol blocks. Secondly, we analyze the efficiency with respect to email size. Next, we analyze the efficiency of our protocol with other related protocols with the use of a table. Finally, we do a functionality comparison. In our analysis we use the following notations for the operations associated with our CLOW-GKA scheme.

PM	Point multiplication in group G_1
BP	Bilinear pairing
HASH	Map-to-point hash algorithm [5] [8][$H : \{0, 1\}^* \rightarrow G_1$.]
HASH2	[10] defined this hash function as [$H : G_2 \rightarrow \{0, 1\}^n$ where n is bit length of plaintexts]

We can note that if a practical elliptic curve E/F_3163 is used to implement the group G_1 , then one BP operation requires $\approx 11,110$ modular multiplications in F_3163 [21]. Also, a PM operation of E/F_3163 requires only a few hundred modular multiplications in F_3163 .

6.1 Computational cost of the protocol

In this subsection we are going to analyse the computational cost of the main protocol block and its sub-protocol blocks.

6.1.1 CLOW-GKA protocol: P2P email encryption

In this protocol block, the KGC server defines a cryptosystem and publishes a set of parameters. The KGC here uses a PM operation to compute the system's public key. This operation is quite efficient and does not represent a significant cost.

6.1.2 Sub-protocol 1

Each email user needs to register an email account through the KGC. The user first uses a HASH and a PM operation to send a parameter to KGC. The KGC again uses a PM operation to compute the partial private key and sends it back to the user. The user computes its public key pair using 2 PM operations as shown in the equations 4.1 and 4.2. So the total computational cost for each user is (4 PM + 1 HASH). If there are n email users then the total cost would be $n \times (4 \text{ PM} + 1 \text{ HASH})$.

6.1.3 Sub-protocol 2

To send an email to n recipients, the sender is required to

1. verify n recipients public key by using the Equation 4.3. This requires 1 HASH and 2 BP operations for verifying each recipients public key.

2. calculate $x_0, x_1, x_2, \dots, x_n$: From Equation 4.4 and we see that each computation requires 1 HASH, 2 PM, and a BP operation. Since HASH is already done in **Sub-protocol 1** we exclude that from the cost.
3. calculate y_1, y_2, \dots, y_n : From Equation 4.5, the calculation of each y_i requires \oplus ing all x_j 's $\forall j \neq i$.
4. derive the encryption key K . From Equation 4.6, this derivation requires \oplus ing all x_i 's.
5. AES encrypt the email using the encryption key K .

The bit-wise \oplus operation is extremely efficient, making the costs for calculating K and y_i 's negligible. From the email sender's perspective, the main computational cost stems from the public key verification, AES encryption and the calculation of $X = \{x_0, x_1, \dots, x_n\}$. Since each x_i calculation requires a 2 PM, and a BP operation and each public key verification requires 1 HASH and 2 BP operations, the total cost for this block is (n) HASH, $(2n + 2)$ PM and $(3n + 1)$ BP operations.

6.1.4 Sub-protocol 3

To receive an email, a recipient needs to

1. verify the sender's public key by using the Equation 4.3. This requires 1 HASH and 2 BP operations.
2. re-construct the group key using Equation 4.7. This requires 1 HASH, 2 PM and 1 BP operation (the \oplus operation is again ignored).

3. AES decrypt the message using the re-constructed group key K .

So the total cost is 2 HASH, 2 PM and 3 BP operations. We see that the computational cost of sending an email is linearly proportional to the number of recipients while the cost of receiving an email is constant.

6.2 Email size analysis

Our proposed CLOW-GKA for P2P email encryption is a n -party protocol and so the size of an email will increase depending on the number of recipients. The most important factor for the increase in the email size is due to the inclusion of key agreement information $(r, y_1, y_2, \dots, y_n)$ with all emails. Each one of the key agreement information will have the same size as the key of the selected curve, since the bilinear pairing uses an elliptic curve to implement our scheme. An elliptic curve with a key size of $\approx 2n$ bits is required for the security of a symmetric encryption scheme with an n -bit key according to the National Institute of Standards and Technology. So the increase in the size of an email to n recipients would be a $256 \times (n + 1)$ bit increase when using an elliptic curve and a 128-bit security. Another factor which leads to increase in size is due to use of AES. AES uses block ciphers which if not full will fill the blocks with random bits. Since AES has a block size of 256 bits the added size is not that significant.

6.3 Efficiency comparison with different protocols

Table 1 provides a brief efficiency comparison between the proposed CLOW-GKA protocol: P2P email encryption, PGP (Pretty Good Privacy) [1, 2] P2P email encryp-

tion, Yeh, Zeng and Long's [18] ID-PKC protocol for email encryption and Al-Riyami et al. [10] CL-PKC (certificateless public key cryptosystem).

Table 6.1: Efficiency comparison between CLOW-GKA, PGP, Yeh, Zeng and Long's ID-PKC and Al-Riyami et al's CL-PKC, assuming there are n recipients in an email

	CLOW-GKA	Yeh, Zeng and Long's ID-PKC	Al-Riyami et al's CL-PKC	PGP
System Setup	• 1 PM;	• None	• 1 PM;	• PGP application installation
Key distribution and generation	• 1 HASH; • 4 PM;	• 1 HASH; • 1 PM;	• 1 HASH; • 2 BP; • 4 PM;	• Generate public/private key pair using the PGP application
Encryption	• $(2n + 2)$ PM; • $(3n + 1)$ BP; • (n) HASH; • Email encryption.	• $(n + 1)$ PM; • $(n + 1)$ BP; • $(n + 1)$ HASH; • Email encryption.	• 1 HASH; • 1 HASH2; • 3 BP; • 1 PM;	• Encrypt message using IDEA using one time session key • Encrypt session key using RSA with recipients public key and include with message • Signature by creating a hash code using SHA-1 and encrypting the message digest using RSA with sender's private key
Decryption	• 2 PM; • 3 BP; • 2 HASH; • Email decryption.	• 1 PM; • 1 BP; • 1 HASH; • Email decryption.	• 1 BP; • 1 HASH2;	• Decrypt session key using RSA • Decrypt message using session key • Verify signature
Message size	• Encrypted message along with all y_i 's (the key agreement info).	• Encrypted message along with all y_i 's (the key agreement info).	• Encrypted ciphertext	• Encrypted message along with digital signature and n encrypted session keys

In Al-Riyami et al. [10] CL-PKC (certificateless public key cryptosystem) the encryption cost is for just one recipient since it is not an n-party scheme.

CHAPTER 7

IMPLEMENTATION

In our implementations we have used the Java Pairing-Based Cryptography (JPBC) library for system parameter setup and pairing calculation. JPBC is a wrapper for the PBC library. The PBC library is a free C library that performs all the mathematical operations of pairing-based cryptosystems. BLS signatures [30], Joux tripartite Diffie-Hellman [31], and Boneh-Lynn-Shacham short signature [22] are some of the protocols demonstrated using the PBC library. We can implement our CLOW-GKA protocol for email encryption in three different scenarios, out of which we have implemented two. These two implementations will be discussed in detail in the following sections.

7.1 Scenarios of CLOW-GKA implementation

1. Software Application: This is an application and can be downloaded and installed on one's personal computer, through which the user can register with the CLOW-GKA cryptosystem, login to the user's registered email accounts, view emails, encrypt and decrypt emails. There is an online server acting as the KGC. Once the user has registered with the online KGC server all the user parameters will be stored in the user's local machine. The user now can use this application to send and receive encrypted emails to other users who have also

registered with the same KGC server. We used a simple java GUI to implement this scenario and a local server simulation that acts as the KGC.

Pros:

- Personal keys are stored locally and thus more safer.

Cons:

- Since all the user parameters (including public/private keys) are stored locally, the user can encrypt emails only on the devices with the application installed.
- Application must be downloaded and installed to each personal device the user usually uses (such as desktop, laptop, phone, etc).
- Due to having to connect to the email service provider's server and pull all of the information onto the local GUI, application might be slow.
- Needs a different version for each operating system which leads to more work in updating application across all platforms.

2. Web Application: This type of web application is more suitable for organizations with a central server. In this scenario, employees can just login into their organization accounts online on a web browser and can register with the system(KGC) which will also be present on the centralized organization server. Since this type of application is specific to an organization, only the employees of the organization will be able to register and use this cryptosystem. All the user parameters (including public/private keys) are stored in a database on the organization's server. We have used servlets and jsp to implement this scenario

and we used Tomcat for our server and HSQLDB as our database to store the user parameters.

Pros:

- Is accessible anytime and anywhere.
- Good for an organization having its own servers.
- No need for any software installation on any personal devices.

Cons:

- Not entirely point to point email encryption because it is first sent to the organization server.
- Not good for a public service as all the secret files are stored on the server and if the server is compromised the whole system is compromised.

3. Web-browser Plugin: In this scenario, the CLOW-GKA would be implemented as a plugin which can be installed on web-browsers and this would be an add on to the existing interface of the email service providers. The user will be able to use the plugin to register, send encrypted emails and decrypt received emails directly from the existing email provider's interface.

Pros:

- Is accessible anytime and anywhere.
- Versatile, fast, and easy to use.
- No need for special interface to send and receive emails.

Cons:

- Updating the plugin is not easy.
- Due to needing a different version for each email service provider and web browser, we are once again looking at a high number of versions of this software.
- Would need to re-register one's email account for each web browser on each device one would be using this program on (desktop, laptop, phone, tablet, etc).

7.2 Common steps in our implementation

- KGC parameters and system setup
- User registration
- Connecting with Gmail server
- Sending, receiving, encrypting and decrypting emails

7.2.1 KGC parameters and system setup

The algorithm defined in the protocol has a set of parameters $\langle G_1, G_2, e, P, P_{pub}, q, H \rangle$, where parameters G_1, G_2, e, P and q were defined in Section 3.2. The KGC also chooses a master secret $s \in Z_q^*$ and then computes $P_{pub} = sP$ as the system's public key. Finally, the KGC chooses a hash function $H : \{0, 1\}^* \rightarrow G_1$. The map-to-curve and map-to-point algorithms from Weil pairings in [5, 8] are such functions that can map users' identities to points on an elliptic curve (the group G_1).

Initially when the KGC is setup, our implementation will generate (1) an elliptic curve for the pairing as the group G_1 ; (2) A master key s which is chosen randomly

in Z_q^* ; (3) A random generator P is chosen; (4) Hash function takes in a string which is the email address and it first performs a standard hash function on it and then the output array is mapped to an element in the cyclic group as $H : \{0, 1\}^* \rightarrow G_1$ (5) The system's public key P_{pub} is calculated. Finally all the system parameters are made public except the master key s which should be known only to the KGC itself.

Implementation in Software Application

Here the KGC is an online server. At first when the KGC server is setup it undergoes the same setup as defined above and then the KGC is made available for user registration. The KGC server here will be used only for user registration and none of the user's private parameters are stored here.

Implementation in Web Application

In this implementation the KGC will be on the organization's central server. The same setup as defined above are done and KGC is made available for user registration. Unlike the KGC in the software application the KGC server will store all of the user's parameters.

7.2.2 User registration

After the KGC parameters are setup, the KGC server is made available for user registration. In the user registration part user ID_j registers an email account and generates his/her private and public keys through the KGC. ID_j computes and sends $r_j Q_j$ to KGC. KGC computes $D_j = sr_j Q_j$ and sends it back to ID_j through any public channel. ID_j chooses his/her private key s_j then computes his/her public key $\langle P_j, R_j \rangle$ as shown in Section 4.3. The public keys generated are published to a public

directory. In the two subsections below we will see how they are implemented for each scenario.

Implementation in Software Application

In the software application the user registration process is described using a flowchart in Figure 7.1. When the application is opened the user can login into their email account. The user's email account should have been registered with the KGC in order to login. If not they should register their email account with the KGC server. Once the user's email account has been registered, all the user's private parameters are stored locally on the machine in which the software application has been installed. Now the user publishes their public keys to a public directory. In this scenario the KGC server is only used for user registration part and the server is never used again for any other process.

Implementation in Web Application

After the KGC setup, the user can open any browser on any machine that is connected to the internet and login to their employee account provided by the organization. Only users with valid employee accounts will be able to register with the system to use CLOW-GKA cryptosystem. Once they have logged into their employee profile, they can register with the system in the web browser. The organization server will get the email address and will register with the system and generate/store the user parameters in a database on the server. The user parameters are generated in the same way as defined in the protocol.

Note that all the user's parameters are stored on the organization's server and not on any local machine. Once registered the user can now login to their email account

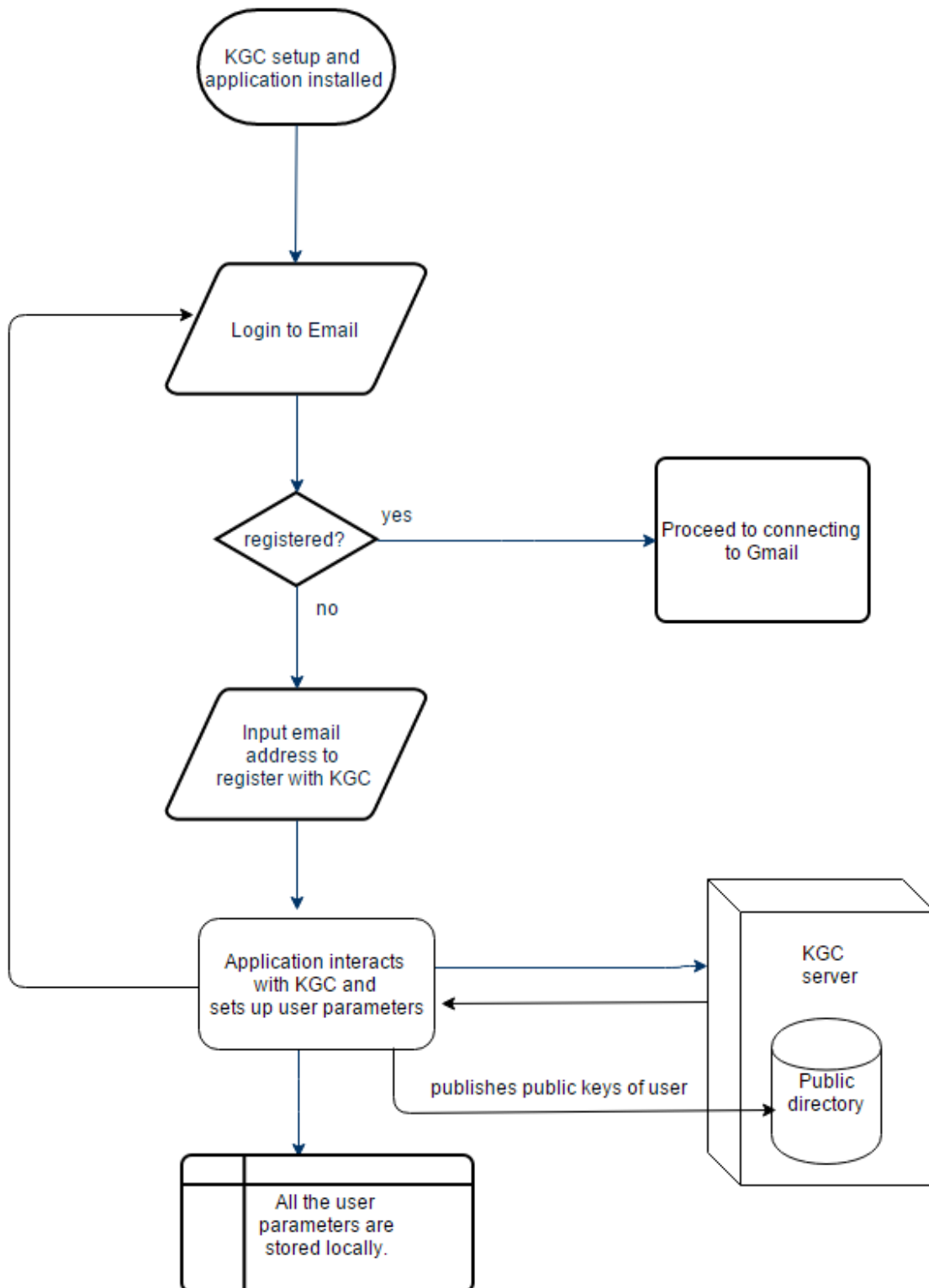


Figure 7.1: Flowchart of user registration steps in software application

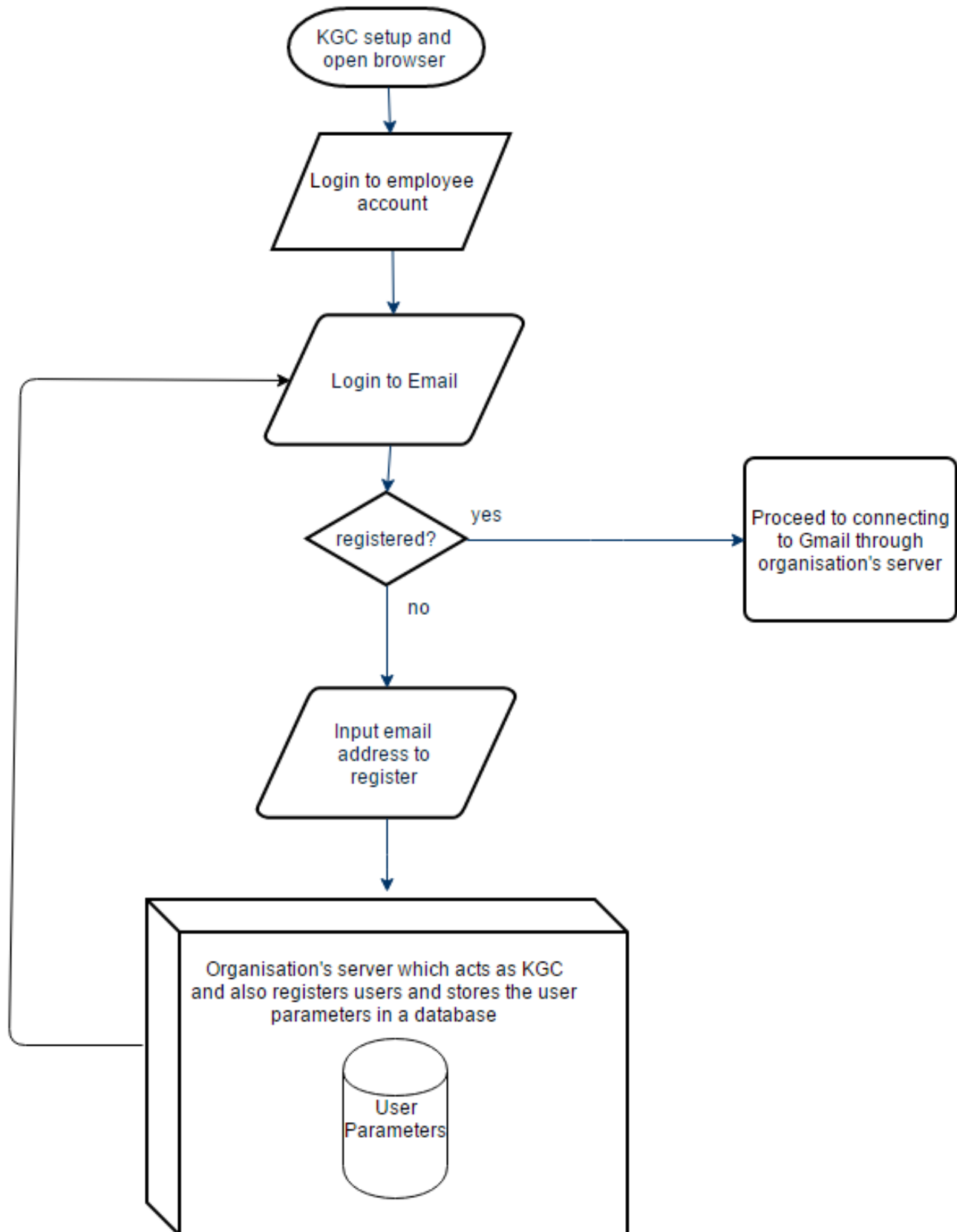


Figure 7.2: Flowchart of user registration steps in web application

using the web browser. Since all the user's private parameters are stored in the organization's server the user will be able to login to their email account from any browser and on any device with an internet connection and use the CLOW-GKA protocol via their employee account. The flow of steps in this implementation scenario is shown in Figure 7.2.

7.2.3 Connecting with mail server

After user registration the user will be able to login to their email accounts. After validating the credentials of the user's email account, the messages can be retrieved from the mail server. Figure 7.3 shows the process of retrieving emails and how to use methods to retrieve specific parts of the email like subject. This process is common for both implementations.

Implementation in Software Application

In this scenario the application directly connects to the mail server and retrieves the messages and displays them in the inbox interface of the application. A simple Figure 7.4 illustrating the flow of events in our software implementation.

Implementation in Web Application

In this scenario the credentials are first sent to the organization's server for verification, then the server connects to the email provider's server to retrieve messages and sends them to the browser for display. Figure 7.5 shows the flow of events in our web implementation.

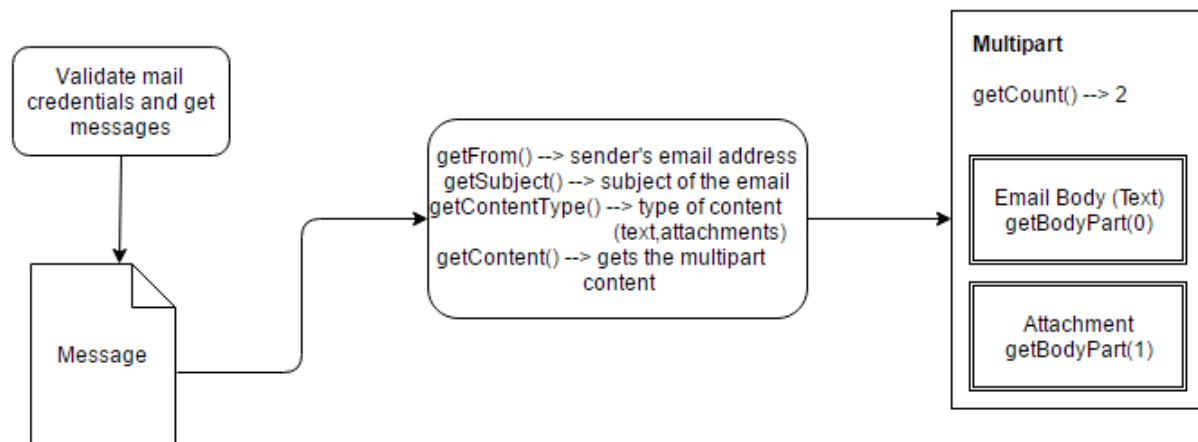


Figure 7.3: Javamail message parts

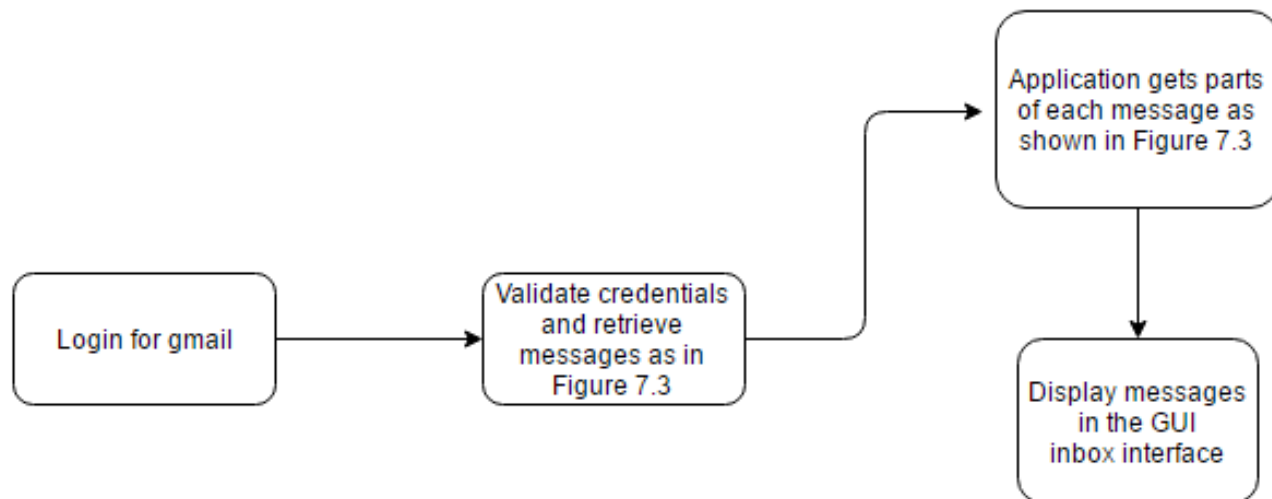


Figure 7.4: Logging into mail and retrieving emails for software application

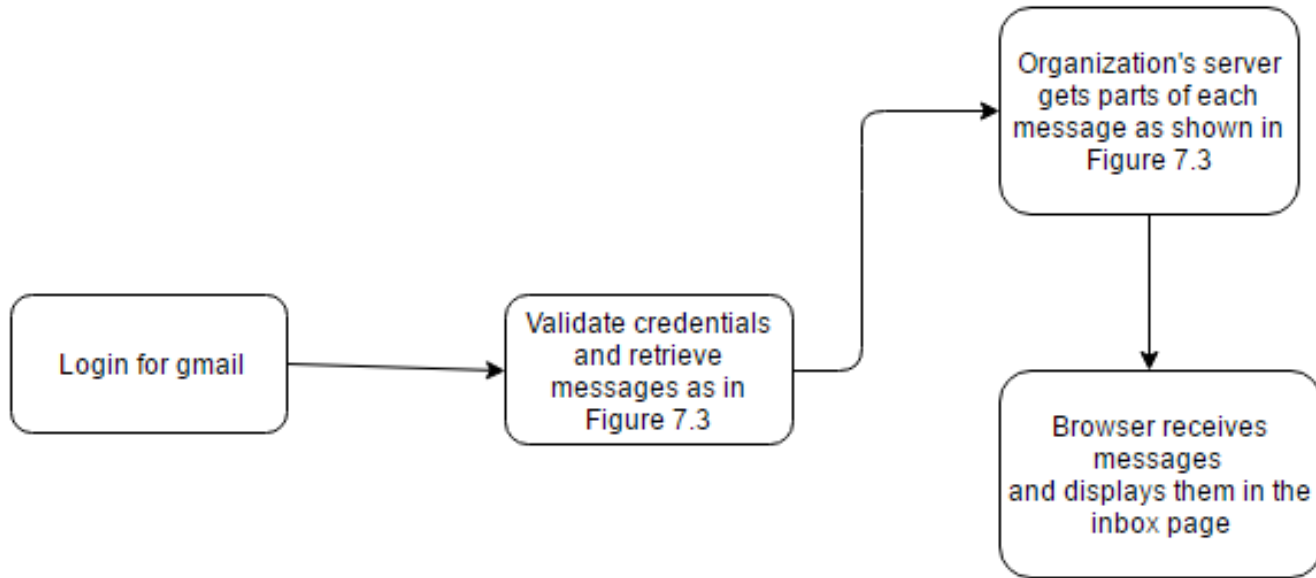


Figure 7.5: Logging into mail and retrieving emails for web application

7.2.4 Sending, receiving, encrypting and decrypting emails

After the user is logged into his email account and his emails have been retrieved, the user now can view the emails in his inbox, send emails as either encrypted or in plaintext, decrypt received emails that have been encrypted. Note that in order to encrypt and send an email to another user, that user must have registered with the same CLOW-GKA cryptosystem.

To send a P2P encrypted email to n email receivers $ID_i, \forall 1 \leq i \leq n$, the sender ID_0 first obtains each email receiver ID_i 's public key $\langle P_i, R_i \rangle$ from the public directory. Then ID_0 verifies each ID_i 's public key by checking $e(P_i, R_i) \stackrel{?}{=} e(P_{pub}, Q_i)$. ID_0 chooses a random number $r \in Z_q^*$ and computes $x_j \quad \forall 0 \leq j \leq n$. Now ID_0 computes n key derivation keys y_i 's, one for each email receiver ID_i and computes the group key K as shown in Equations 4.5 and 4.6 respectively in Section 4.4. Finally, ID_0 uses the group key K to encrypt the email message and then sends the encrypted

email along with the key derivation keys and the random number $(y_1, y_2, \dots, y_n, r)$ as an attachment to all n email receivers ID_i 's.

To decrypt a received encrypted email, each email receiver $ID_i, \forall 1 \leq i \leq n$ first obtains the email sender ID_0 's public key $\langle P_0, R_0 \rangle$ from the public directory. ID_i checks the validity of ID_0 's public key based on the Equation 4.3. ID_i derives the group key K using his/her own private key s_i , the corresponding key derivation key y_i , and the email sender ID_0 's identity $Q_0 (= H(ID_0))$ and public key P_0 as shown in Equation 4.7 in Section 4.5. Finally, ID_i can decrypt the email using the just derived group key K .

Implementation in Software Application

Figure 7.6 flowchart shows the scenario of this implementation. When the user wants to send an encrypted group email, the user composes a new email and fills in email addresses of all the recipients the user wishes to send the email to. When the user presses "send encrypted email" button, the application will (1) retrieve the public keys of the all the recipients from the public directory and verifies them; (2) derive the group key with which it will encrypt the email; and then (3) send the encrypted email with the key derivation keys (as attachment) to the mail service provider's server, which in turn will send the email to all the intended recipients using its own service. The recipient now can decrypt the email using this application. When the recipient wants to decrypt the email, the application will retrieve the required parameters, including the private key of the recipient which is stored locally on the device, and then derive the group key to decrypt the email.

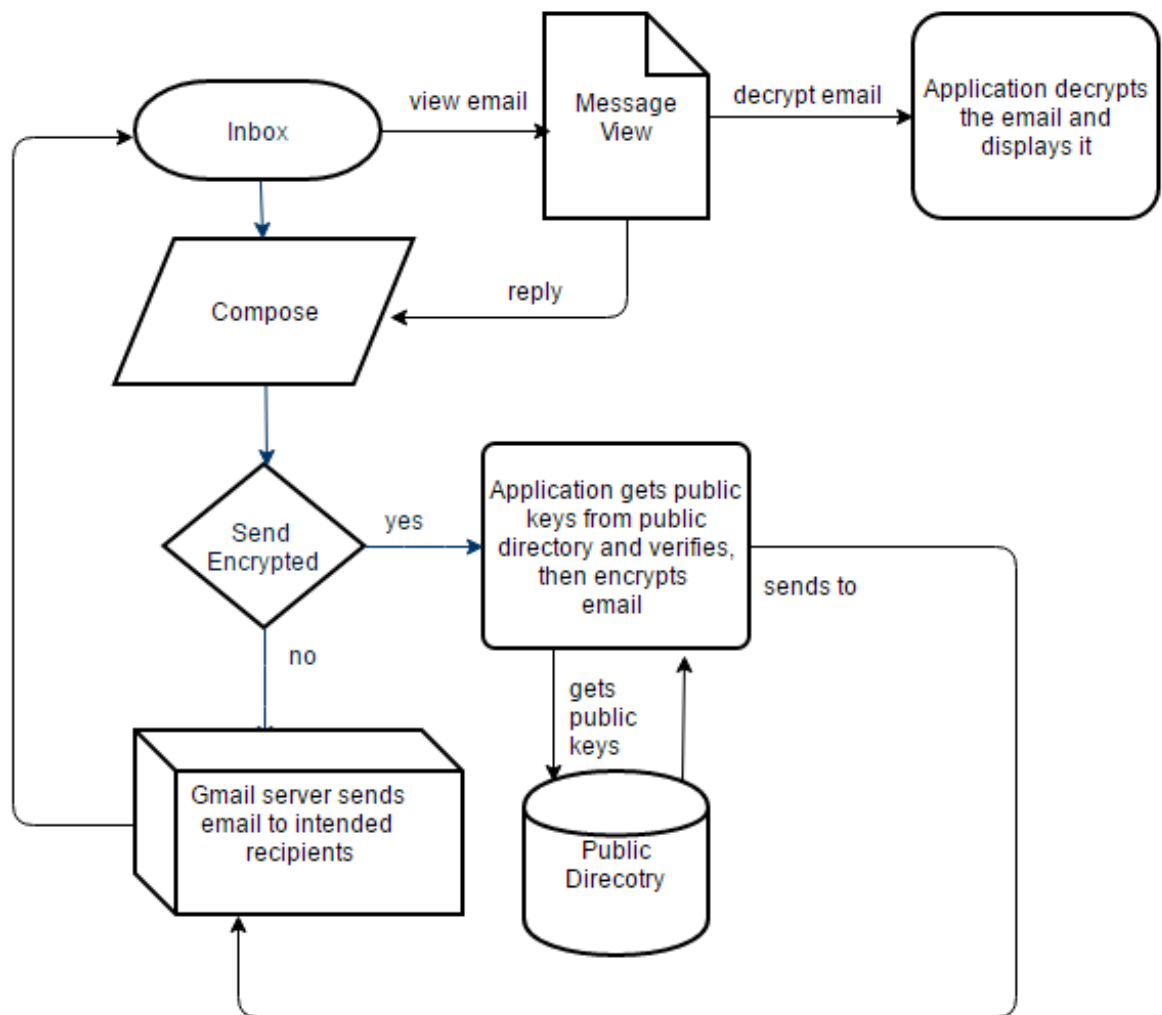


Figure 7.6: Flow of emails in software Application

Implementation in Web Application

Figure 7.7 flowchart shows the scenario of this implementation. When the user wants to send an encrypted group email, the user composes a new email on the browser and fills in the email addresses of all the recipients the email is intended to. When the user clicks "send encrypted email" button, the browser sends all the information to the organization's server. The organization's server will now derive the group key because all the user parameters (including public and private keys) are stored in a database on the same server. After the email has been encrypted using the group key, the organization's server sends the encrypted email to the email provider's server which will send the email to all the recipients using its own service. When the recipient wants to decrypt the email, the encrypted email is sent to the organization's server where it will derive the group key to decrypt the email and then send it back to the browser. The organization's server is able to derive the group key because as mentioned above all the user's private parameters are stored in a database on the organization's server.

Screenshots of our implementation of software and web application's interface can be found on Appendix A and Appendix B respectively.

7.3 Future Implementations

In the future, the third implementation scenario for a web browser plugin can be developed and implemented. The web browser plugin would be a good addition since the user can just use his email service provider's interface on the browser with the added point-to-point email encryption feature. Moreover, different open source email applications can be used to implement the protocol using the implemented software

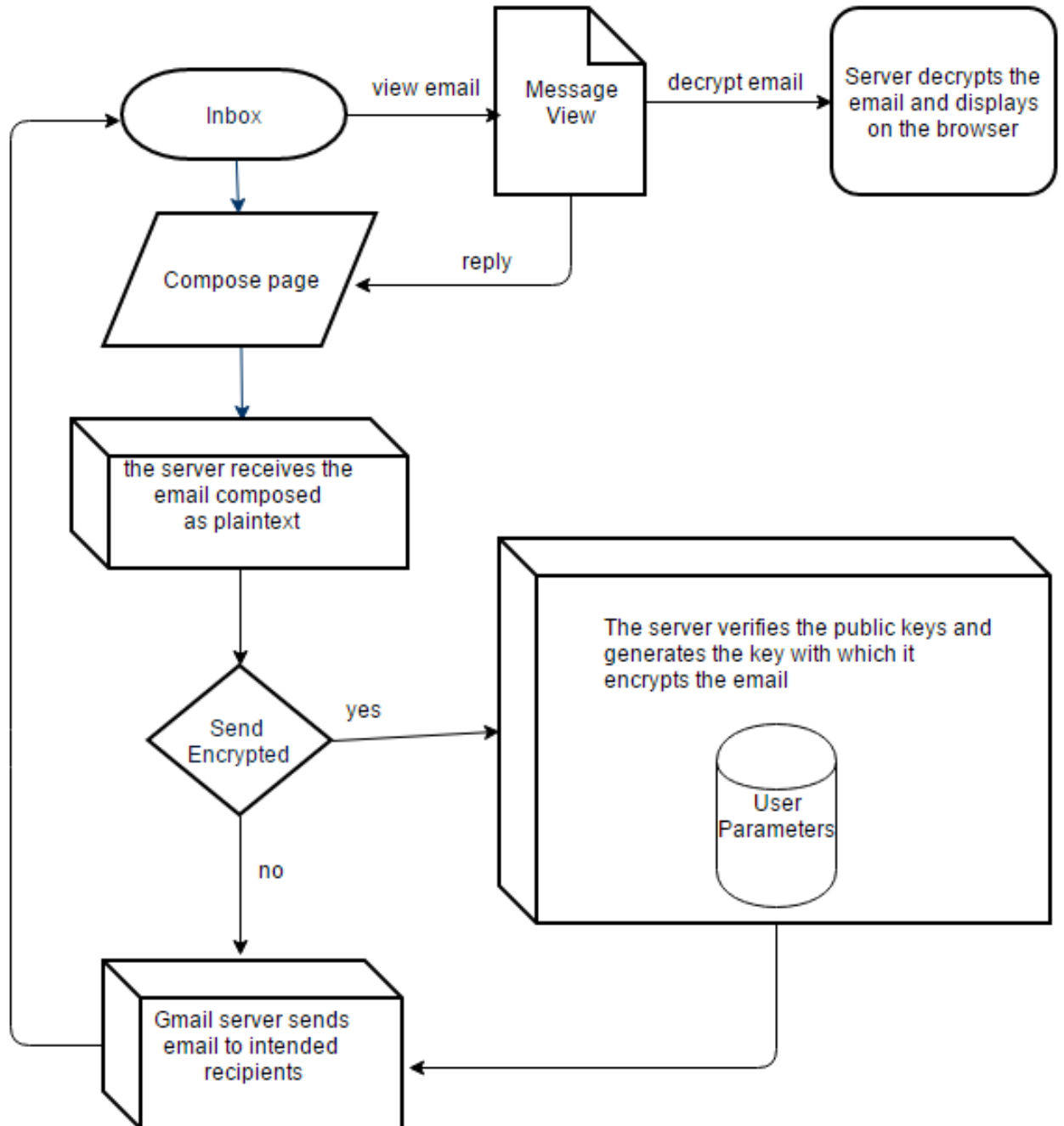


Figure 7.7: Flow of emails in web Application

application as a model. These open source email applications will have more user friendly features that our basic implementation does not have. The same can be said for our web application where more features can be implemented to make the user experience more satisfying and convenient. Our implementations can only connect with gmail for now. In the future it can be extended to as many leading email service providers as possible.

CHAPTER 8

CONCLUSIONS

8.1 Thesis Conclusion

To protect email privacy, a point-to-point email encryption scheme is required, where only the email sender and receivers can decrypt the emails. In this thesis, we proposed a certificateless one-way group key agreement protocol with four features which are either necessary or suitable to implement point-to-point email encryption schemes. These four features are (1) certificateless, (2) one-way, (3) n -party and (4) no secret channels. To the author's best knowledge, no other existing group key agreement protocol has these four features simultaneously. This thesis also gives a security proof of the proposed protocol using a methodology "proof by simulation". This methodology is relatively new and thus very few examples (or case studies) can be found in literature. The proof presented in this thesis can serve as a good example for future researchers who would like to use this methodology to prove other protocols' security. In addition, the efficiency of the protocol is also analyzed. Finally, three implementation scenarios for our CLOW-GKA protocol are discussed, in which two scenarios have been implemented in this thesis.

8.2 Future directions

In Chapter 4, Section 4.3 we know that each user ID_j computes and sends r_jQ_j to KGC, where r_j is a random secret for ID_j and $Q_j = H(ID_j)$. KGC computes $D_j = sr_jQ_j$ and sends it back to ID_j through any public channel, in which a partial private key sQ_j is embedded in D_j . Here, the KGC does not verify whether the r_jQ_j received is from the actual ID_j . So if an attacker ID_a impersonates ID_j sends his own r_aQ_j then the KGC will compute $D_a = sr_aQ_j$ for the attacker and send it back. This illegal fabrication cannot be detected in our system. One of the future direction would be to solve this problem. As explained in the future implementation section of Chapter 7, further features can be added to make the implementations more user friendly. The third scenario of using web-browser plugins will be the major task of our future work.

REFERENCES

- [1] P. Zimmermann, “Why I wrote PGP?”
<https://www.philzimmermann.com/EN/essays/WhyIWrotePGP.html>, 1991.
- [2] The International PGP Homepage, “PGP Documentation,”
<http://www.pgpi.org/doc/>
- [3] X. Lai and J. Massey, “A Proposal for a New Block Encryption Standard,” EUROCRYPT, pp. 389-404, 1990.
- [4] A. Shamir, “Identity-based Cryptosystems and Signature Schemes,” CRYPTO’84, LNCS 196, pp. 47-53, 1985.
- [5] D. Boneh, M. Franklin, “Identity-based encryption from the Weil pairing,” CRYPTO’01, pp. 213-229, 2001.
- [6] C. Gentry, A. Silverberg, “Hierarchical ID-based cryptography,” Advances in Cryptography - ASIACRYPT’02, Springer-Verlag, pp. 548-566, 2002.
- [7] F. Zhang, K. Kim, “ID-based blind signature and ring signature from pairings,” Advances in Cryptography - ASIACRYPT’02, Springer-Verlag, pp. 533-547, 2002.
- [8] X. Yi, “An Identity-based Signature Scheme from the Weil Pairing,” IEEE Communications Letter, Vol. 7, No. 2, pp. 76-78, 2002.
- [9] H. Elkamchouchi, Y. Abouelseoud, “A new proxy identity-based signcryption scheme for partial delegation of signing rights,” Cryptology ePrint Archive, Report 2008/041, 2008.
- [10] S. Al-Riyami and K. Paterson, “Certificateless public key cryptography,” Advances in Cryptology - ASIACRYPT’03, pp. 1-40, 2003.
- [11] H. Xiong, F. Li, and Z. Qin, “Certificateless threshold signature secure in the standard model,” Information Sciences, Vol. 237, pp. 73-81, 2013.
- [12] R. Tso, X. Huang, and W. Susilo, “Strongly secure certificateless short signatures,” Journal of Systems and Software, vol. 85, no. 6, pp. 1409-1417, 2012.

- [13] J. Baek, R. Safavi-Naini, and W. Susilo, "Certificateless public key encryption without pairing," 8th International Information Security Conference (ISC 2005), LNCS 3650, pp. 134-148, 2005.
- [14] G. Yang and C.H. Tan, "Strongly secure certificateless key exchange without pairing," ASIACCS'11, pp.71-79, 2011.
- [15] M.L. Das, "A key escrow-free identity-based signature scheme without using secure channel," Cryptologia, Vol. 35, No. 1, pp. 58-72, 2011.
- [16] H. Lin and Y. Zhou, "A secure protocol for sharing trust data in hybrid P2P network," Journal of Networks, Vol. 6, No. 4, pp. 607-614, 2011.
- [17] L. Zhang, Q. Wu, B. Qin, and J. Domingo-Ferrer, "Identity-based authenticated asymmetric group key agreement protocol," 16th International Conference on Computing and Combinatorics, COCOON'10, LNCS 6196, pp. 510-519, 2010.
- [18] J.H. Yeh, F. Zeng, and T. Long, "P2P email encryption by an identity-based one-way group keyagreement protocol," 20th IEEE International Conference on Parallel and Distributed Systems (ICPADS 2014), pp. 760-767, 2014.
- [19] T.H. Yuen, W. Susilo, and Y. Mu, "How to construct identity-based signatures without the key escrow problem," International Journal of Information Security, Springer-verlag, Vol. 9, No. 4, pp. 297-311, 2010.
- [20] H.Y. Lin, T.S. Wu, S.K. Huang, Y.S. Yeh, "Efficient proxy signcryption scheme with provable CCA and CMA security," Computers and Mathematics with Applications Vol. 60, No. 7, pp. 1850-1858, 2010.
- [21] P.S.L.M. Barreto, H.Y. Kim, B. Lynn, M. Scott, "Efficient algorithms for pairing-based cryptosystems," CRYPTO'02, pp. 354-368, 2002.
- [22] D. Boneh, B. Lynn, H. Shacham, "Short signature from the Weil pairing," Advances in Cryptography - ASIACRYPT'01, Springer-Verlag, pp. 514-532, 2001.
- [23] N. Koblitz, "Elliptic Curve Cryptosystems," Mathematics of Computation, Vol. 48, pp. 203-209, 1987.
- [24] V. Miller, "Use of Elliptic Curves in Cryptography," CRYPTO'85, pp. 417-426, 1985.
- [25] National Security Agency, "Recommended Set of Advanced Cryptography Algorithms - Suite B", 2005.

- [26] C. Hazay, Y. Lindell “Efficient Secure Two-Party Protocols,” Springer-Verlag Berlin Heidelberg, Vol. 2, 2010.
- [27] C. P. Schnorr “Efficient Identification and Signatures for Smart Cards,” Advances in Cryptology — CRYPTO’ 89 Proceedings, Springer New York, New York, NY, Vol. 2, pp. 239-252, 1990.
- [28] M. B. Barbosa “Identity Based Cryptography From Bilinear Pairings,” Universidade do Minho Campus de Gualtar Braga Portugal, 2005.
- [29] D. Hankerson, A. Menezes “Elliptic Curve Discrete Logarithm Problem,” Encyclopedia of Cryptography and Security Springer US, Boston, MA, pp. 397-400, 2011.
- [30] “BLS signatures” <https://crypto.stanford.edu/abc/manual>
- [31] Antoine Joux “A One Round Protocol for Tripartite Diffie-Hellman,” <http://cgi.di.uoa.gr/~aggelos/crypto/page4/assets/joux-tripartite.pdf>
- [32] S.S. Al-Riyami and K.G. Paterson. “Authenticated three party key agreement protocols,” International Conference on Cryptography and Coding, pages 332359
- [33] HPE Secure Email, 2015, <https://www.voltage.com/products/email-security/hpe-securemail/>
- [34] Trend Micro email, 2008, <http://www.trendmicro.com/us/enterprise/network-web-messaging-security/email-encryption/>
- [35] Data-Motion SecureMail, 2010, <https://www.datamotion.com/products/securemail/>
- [36] Proofpoint Email Protection, 2005, <https://www.proofpoint.com/us/products/email-protection>
- [37] LuxSci SecureLine, 2011, <https://luxsci.com/extranet/secure-email.html>
- [38] ProtonMail, 2013, <https://protonmail.com/security-details>
- [39] Mailvelope, 2014, <https://www.mailvelope.com/en/>

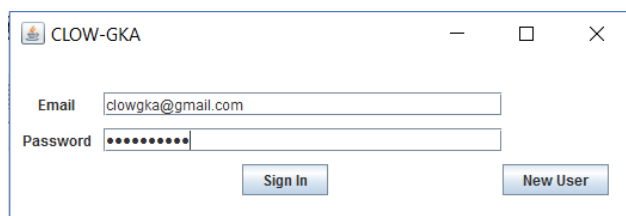
APPENDIX A

SOFTWARE APPLICATION INTERFACE

This Appendix contains the screenshots of our implemented graphical user interface of our software application.

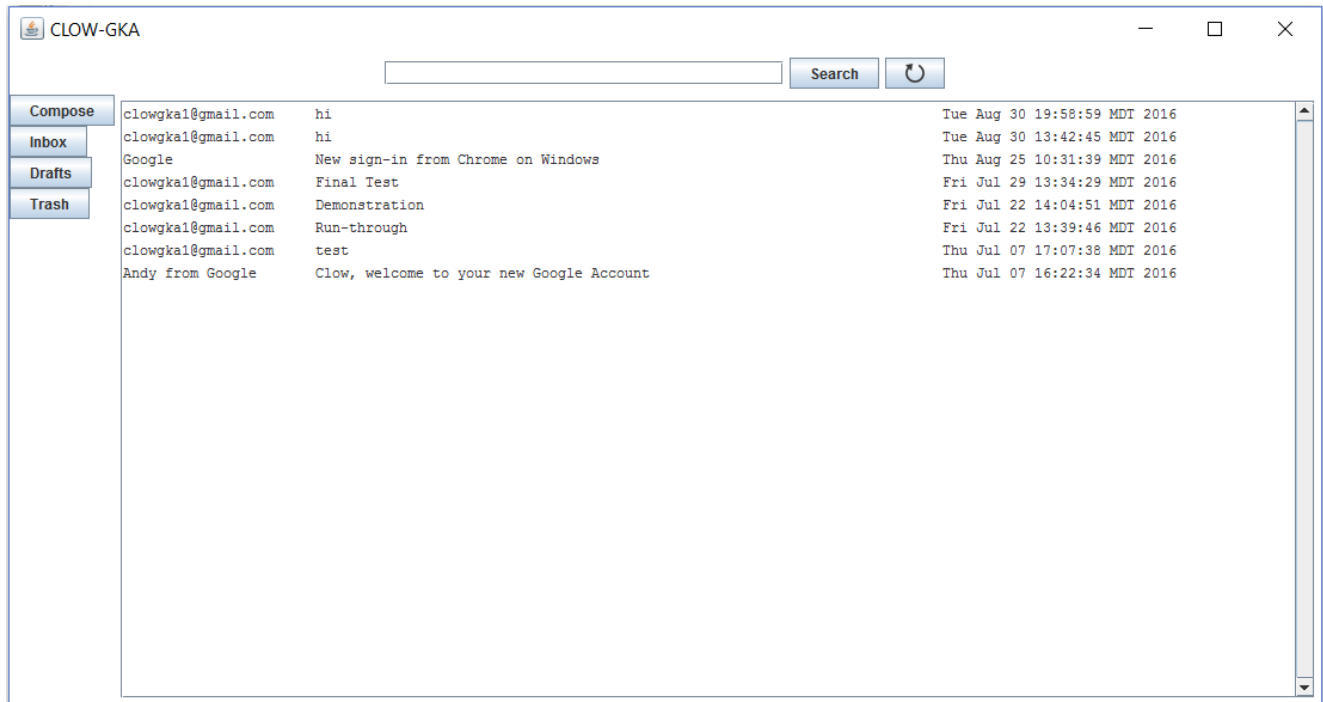
A.1 User Email Login and Register Interface

This is the first interface the user comes across when the application is opened. If the user is new, they can register with the system by clicking on the new user button.



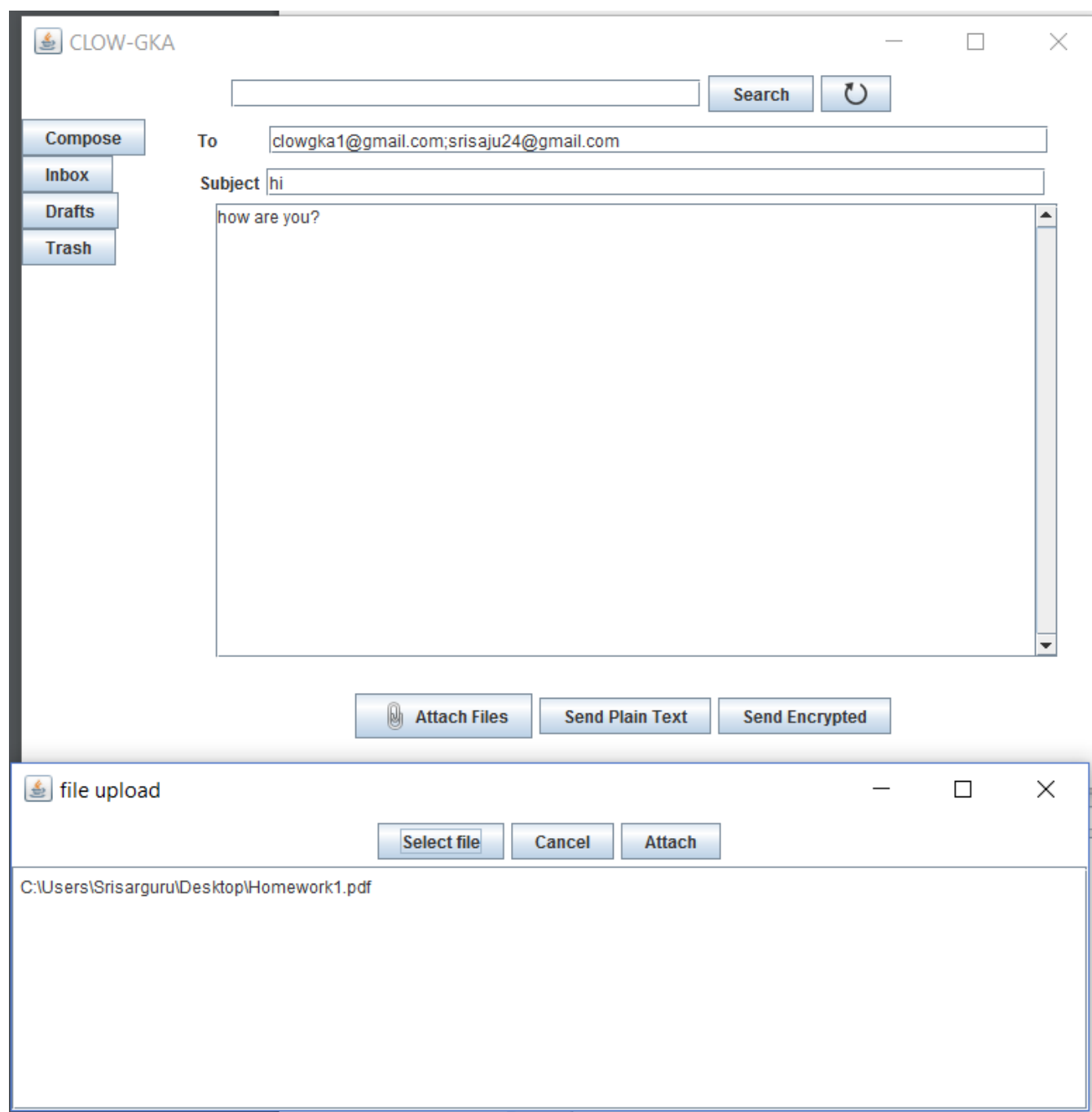
A.2 Inbox Interface

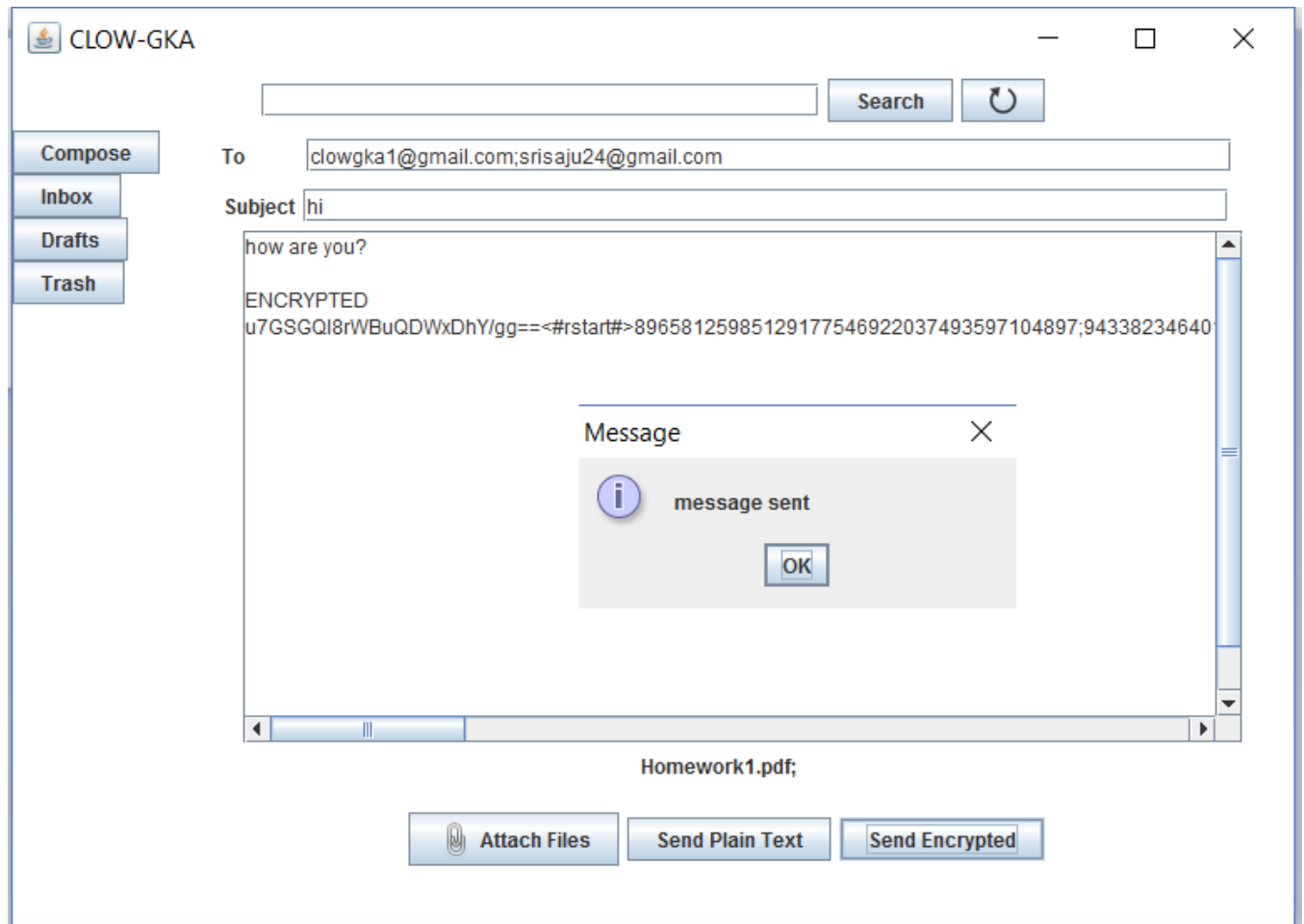
Once successfully logged in, the inbox interface would have all the emails pulled up from the email provider's server and displays it. The user can view each email by clicking on it.



A.3 Compose and Sending Interface

The user can compose a new email and attach files. The user will have an option to either send the email encrypted or as plaintext.





A.4 Decrypt Email Interface

When the user is viewing a decrypted email, they can decrypt it by clicking the decrypt button which would decrypt the email and display it. The attachments are also decrypted and downloaded. The path of the downloaded file is displayed after downloading.

CLOW-GKA

Search [Refresh]

Compose
Inbox
Drafts
Trash

Subject: hi
From: clowgka@gmail.com
Attachments: Homework1.pdf;
Download Attachment

u7GSGQI8rWBuQDWxDhY/gg==<#start#>89658125985129177546922037493597104897;943382346401735633197105439001561876747879540727530867677257096510144
DECRYPTED
how are you?

Reply
Forward
Decrypt

Message
file decrypted to D:\Documents\Java workspace\ClientIDEncryption\download
OK

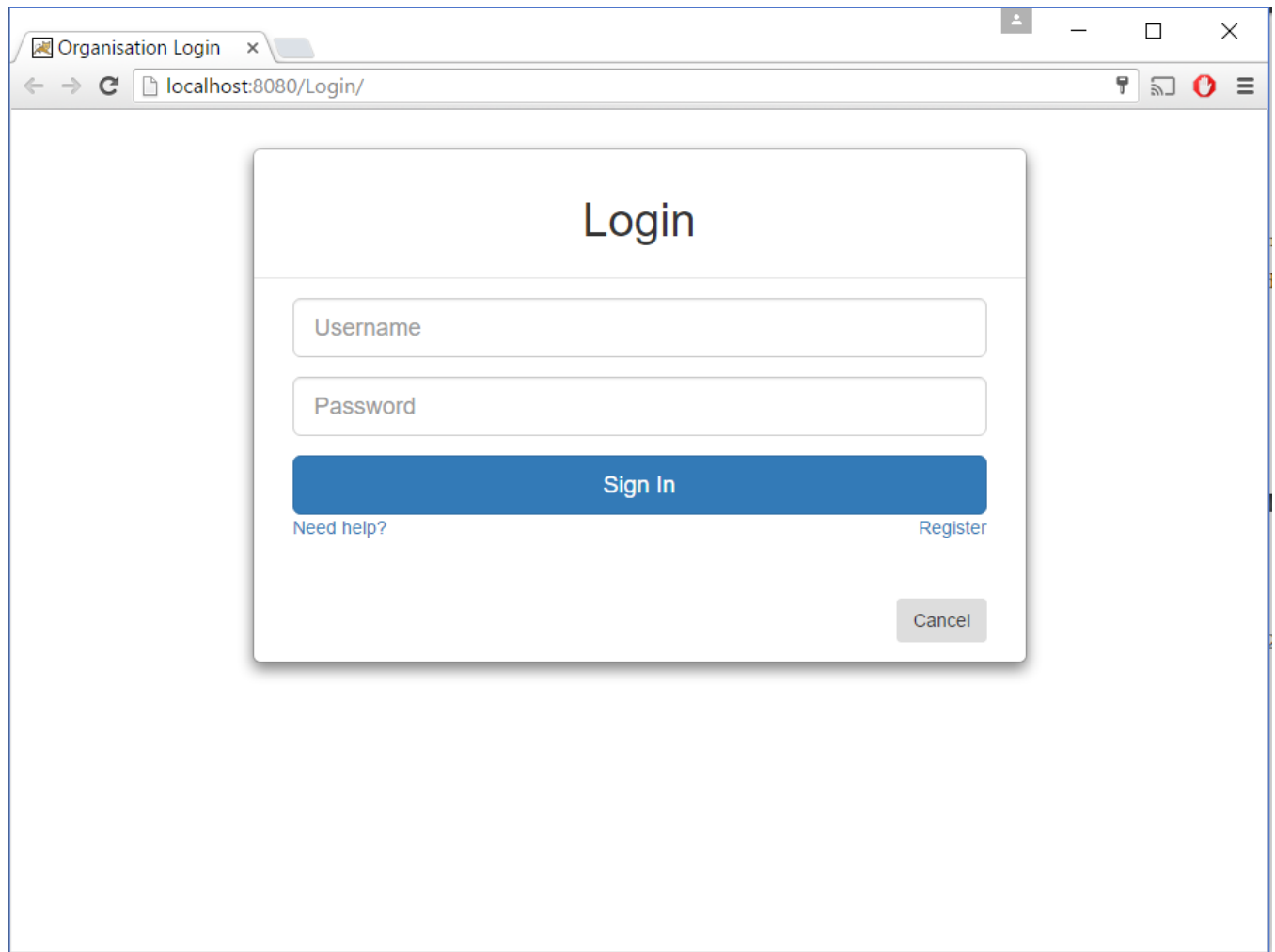
APPENDIX B

WEB APPLICATION INTERFACE

This Appendix contains the screenshots of our implemented graphical user interface of our web application.

B.1 Organisation Login Interface

The first page the user would encounter would be the login to their employee account in an organisation. Only after logging into the user's employee account will he be able to register with the CLOW-GKA system

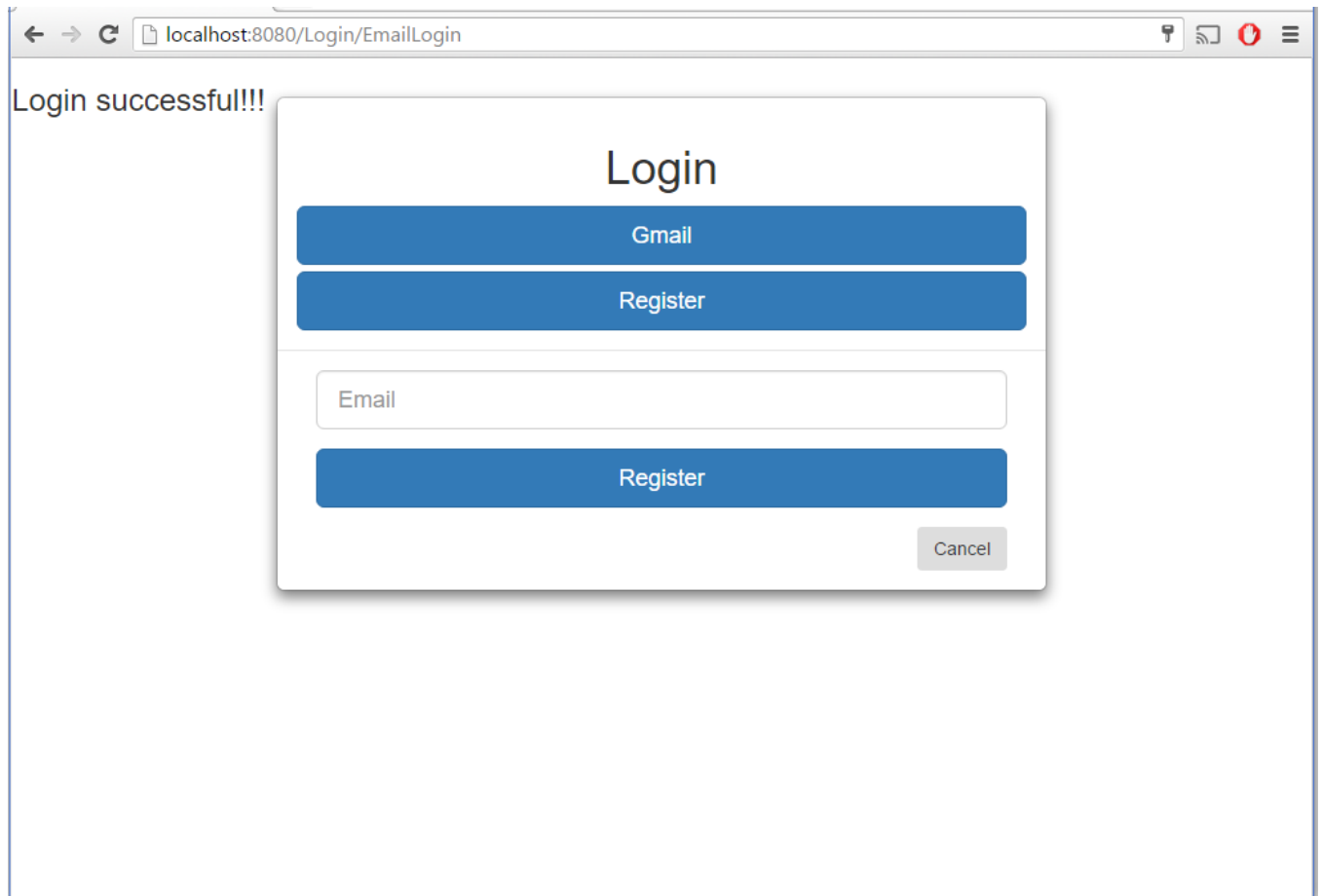


The image shows a web browser window with the title "Organisation Login" and the address bar displaying "localhost:8080/Login/". The main content is a login form with the following elements:

- A heading "Login" centered at the top.
- An input field labeled "Username".
- An input field labeled "Password".
- A prominent blue button labeled "Sign In".
- Two links: "Need help?" on the left and "Register" on the right, both in blue text.
- A "Cancel" button in a light gray box at the bottom right.

B.2 User Email Login and Register Interface

After logging into his employee account, the user can now register and login to his email account.



B.3 Inbox Interface

Once the user has successfully logged into his email account, the inbox interface would have all the emails pulled up from the email provider's server and displays it. The user can view each email by clicking on it.

localhost:8080/Login/Gmail

clowgka@gmail.com

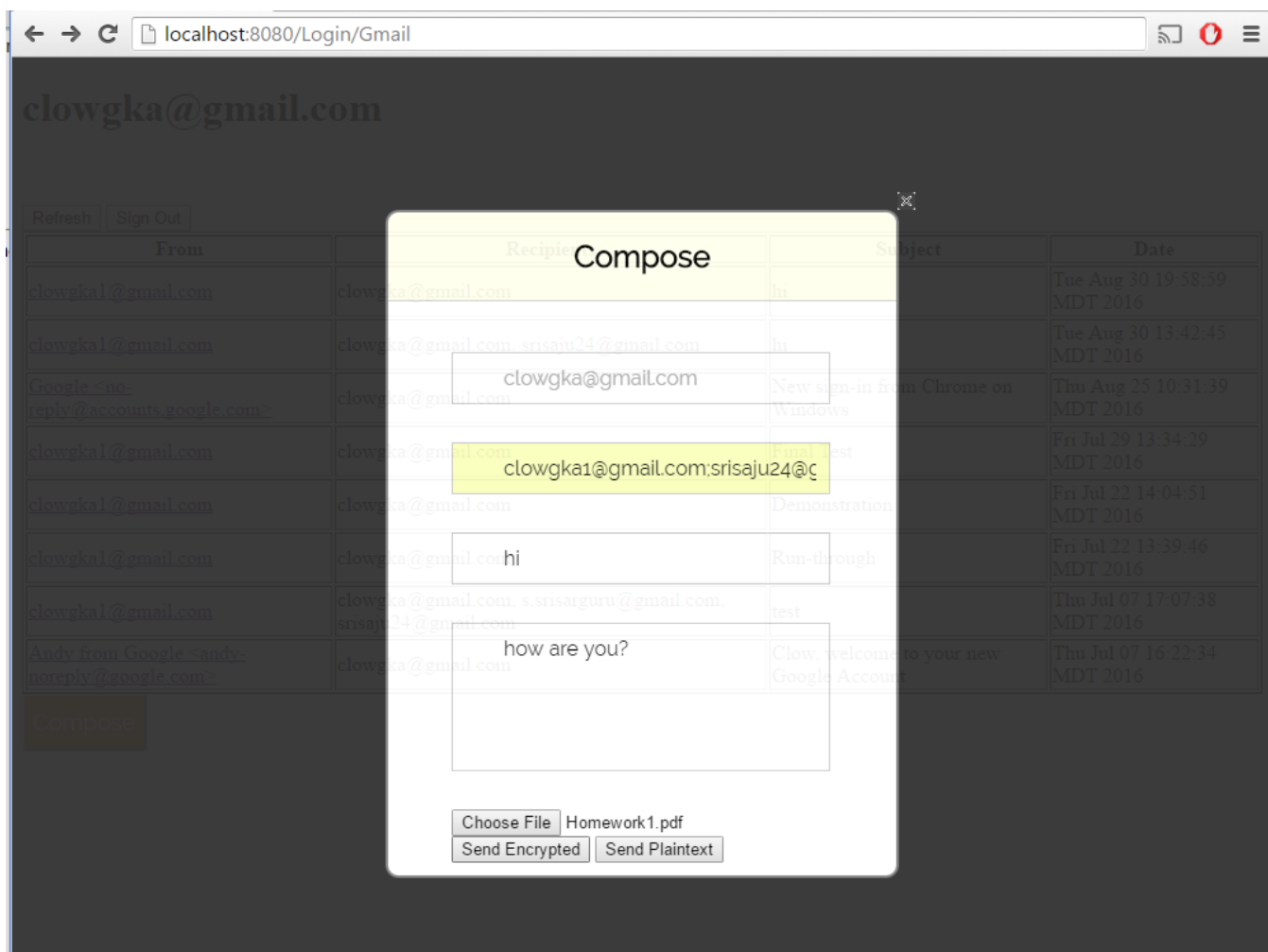
Refresh Sign Out

From	Recipients	Subject	Date
clowgka1@gmail.com	clowgka@gmail.com	hi	Tue Aug 30 19:58:59 MDT 2016
clowgka1@gmail.com	clowgka@gmail.com, srisaju24@gmail.com	hi	Tue Aug 30 13:42:45 MDT 2016
Google <no-reply@accounts.google.com>	clowgka@gmail.com	New sign-in from Chrome on Windows	Thu Aug 25 10:31:39 MDT 2016
clowgka1@gmail.com	clowgka@gmail.com	Final Test	Fri Jul 29 13:34:29 MDT 2016
clowgka1@gmail.com	clowgka@gmail.com	Demonstration	Fri Jul 22 14:04:51 MDT 2016
clowgka1@gmail.com	clowgka@gmail.com	Run-through	Fri Jul 22 13:39:46 MDT 2016
clowgka1@gmail.com	clowgka@gmail.com, s.srisarguru@gmail.com, srisaju24@gmail.com	test	Thu Jul 07 17:07:38 MDT 2016
Andy from Google <andy-noreply@google.com>	clowgka@gmail.com	Clow, welcome to your new Google Account	Thu Jul 07 16:22:34 MDT 2016

Compose

B.4 Compose and Sending Email Interface

The user can compose and email and attach files from their device. The user will have the option of choosing whether to send the email encrypted or not.



B.5 Decrypt Email Interface

The following contains the screenshot when a user views an email and decrypts it.

localhost:8080/Login/viewandreplymail.jsp?name=0

From: clogka@gmail.com

To: clogka1@gmail.com, srisaju24@gmail.com

Subject: hi

Email Body:
82+PP3vBDMmZ4QwbhYGkTA==
<#rstart#>326316268965770255588520885119439649238:13685072144706095770815972166097838461131505507639612716644129593928769
1927241516243396112386426:2498888189822917179021001289540801918461682471216172524038162763878835256334519433159360821369
845<#rend#>

Optional Message: null

Decrypt

Reply

localhost:8080/Login/DecryptEmail

From: clogka@gmail.com

To: clogka1@gmail.com, srisaju24@gmail.com

Subject: hi

Email Body: how are you?

Optional Message: D:\Documents\Java Workspace\SampleLogin\download

Decrypt

Reply

