INDIAN STATISTICAL INSTITUTE

KOLKATA

4th SEMESTER DESERTATION

MIDTERM PROGRESS REPORT

Symmetric Key Management & Distribution

Submitted by:
Sohan Das
Roll No. - CrS2119

SUBMITTED TO:
DEPARTMENT OF CRYPTOLOGY & SECURITY

Apr 13 2023

SYMMETRIC KEY MANAGEMENT AND DISTRIBUTION

MIDTERM PROGRESS REPORT

EXTERNAL SUPERVISOR
Capt. Ritesh Wahi

INTERNAL SUPERVISOR
Dr. Subhamoy Maitra

"PROJECT PROPOSAL AND ABSTRACT SUBMITTED FOR PARTIAL FULLFILLMENT OF THE DEGREE OF M.TECH IN CRYPTOLOGY & SECURITY"

"PROJECT IS BEING CARRIED OUT AT WEAPONS AND ELECTRONICS SYSTEMS ENGG. ESTB.(WESEE)"

New Delhi

April 13, 2023

Abstract

Motivation of this project comes from the below described problem :

"There are several users in an office, all are connected with a common server. They want to communicate with each other in a secure manner whilst ensuring confidentiality of data being shared. This can be achieved by using symmetric key encryption. However how do they share the keys for the same?"

Key management in a symmetric key encryption remains a challenge. Whilst this can be resolved by using PKI (public key infrastructure) we will endeavour to provide same or better security by using an alternate methodology.

In this project we will build an application through which all the users will remain connected with the server and communicate with each other in a secure fashion. Our server has access of *QRNG*, so the server can generate many more random numbers as per requirement using that *QRNG* and those random numbers can be used as (symmetric session)keys to encrypt any data or document, users wanna share through the server. Initially when a new user comes to join this network it has to register itself on the server and server will maintain a database for every users. At this time of registration, a master key for that user will be generated in both end and here the key distribution is happening. Then server will store that master key corresponding to that particular user in it's database in a secure fashion. At the user end master key will not be stored. We will manage to generate the master key using some *"master key id & password"* whenever it will be required.

Later time when a user A wants to communicate with another user B connected to the same server, firstly A will send a request to the server and then the server will generate a random number using *QRNG* and after that using A and B's master keys server will send that random number in encrypted form to both of them using some *symmetric key AEAD* algorithom and that random number will be used as a (symmetric)session key using which A and B can encrypt data and share between each other through the server securely.

In this way we are doing symmetric key management and distributing symmetric keys among users and server. We will prefer python programming language for implementing this project.

Contents

List of Tables

List of Figures

1 Problem Statement

“There are several users in an office, all connected to a common server. They want to communicate with each other in a secure manner while ensuring the confidentiality of the data being shared. This can be achieved by using symmetric key encryption. However, how do they share the keys for the same?"

2 Background of the problem

In our office, there are several users connected to a common offline server. If one user wants to communicate with another user securely through that server, they can do as follows : Encrypt the data and then send that encrypted data through the server; in this way, a secure communication between the users connected to the same server can be established. In this situation, we can choose any one of the following two options: the symmetric key encryption algorithm or the public key encryption algorithm. Using PKI (public key infrastructure), we can easily solve the above mentioned problem, but the PKI system has several disadvantages, some of which are as follows :

It mainly depends on some computationally hard problem. Therefore public key encryption algorithms are quite complicated in design and a lots of computation are required to encrypt plaintext. Similarly decryption algorithms are also complicated. If we use public key encryption algorithm to encrypt a large quantity of data and if we use it on regular basis then our system may become slow down.

Nowadays as computing power of our systems increases and as quantum computing becomes reality, it’s quite easy to decrypt the encrypted data just by using brute force.

For these above disadvantages I have tried to find out some alternative methodology where we can use symmetric key encryption algorithm with same or better security. If we encrypt data using symmetric key encryption algorithm then our problem reduces to how we can then send the symmetric key corresponding to the encrypted data through the server. Basically we have to share symmetric key through the server or have to find out or design some protocol to solve this problem.

3 Details of work done

There is a server in our office and several users connected with that common server and those users want to communicate securely among them through the server. Also we are trying to use symmetric key encryption algorithm to maintain confidentiality with same or better security than PKI system. Keeping all this requirements in mind we have to find out some protocol for sharing symmetric keys through network. As a solution we have found out "*Needham-Schroeder Symmetric Key Protocol*" based on symmetric key encryption algorithm, which forms the basis of well known "*Kerberos*" protocol. This protocol aims to establish a session key between two parties on a network, typically to protect further communication between them.

Let's first look into the "*Needham-Schroeder Symmetric Key Protocol*"[2], then will make decision how we can use this protocol to solve our purpose and also will do modifications according to our requirements and possible attacks :

3.1 *Needham-Schroeder Symmetric Key Protocol*[4]

3.1.1 *Algorithm*

Here, Alice (A) initiates the communication to Bob (B). S is a server trusted by both parties. In the communication :

- A and B are identities of Alice and Bob respectively
- KAS is a symmetric key known only to A and S
- KBS is a symmetric key known only to B and S
- NA and NB are nonces generated by A and B respectively

- KAB is a symmetric, generated key, which will be the session key of the session between A and B

The protocol can be specified as follows in security protocol notation :

1. $A \longrightarrow S : A, B, NA$
   Alice sends a message to the server identifying herself and Bob, telling the server she wants to communicate with Bob.

2. $S \longrightarrow A : Enc_{KAS}(NA, KAB, B, Enc_{KBS}[KAB, A])$
   The server generates KAB and sends back to Alice a copy encrypted under KBS for Alice to forward to Bob and also a copy for Alice. Since Alice may be requesting keys for several different people, the nonce assures Alice that the message is fresh and that the server is replying to that particular message and the inclusion of Bob's name tells Alice who she is to share this key with.

3. $A \longrightarrow B : Enc_{KBS}(KAB, A)$
   Alice forwards the key to Bob who can decrypt it with the key he shares with the server, thus authenticating the data.

4. $B \longrightarrow A : Enc_{KAB}(NB)$
   Bob sends Alice a nonce encrypted under KAB to show that he has the key.

5. $A \longrightarrow B : Enc_{KAB}(NB - 1)$
   Alice performs a simple operation on the nonce, re-encrypts it and sends it back verifying that she is still alive and that she holds the key.

3.1.2 *Communication Diagram*[5]

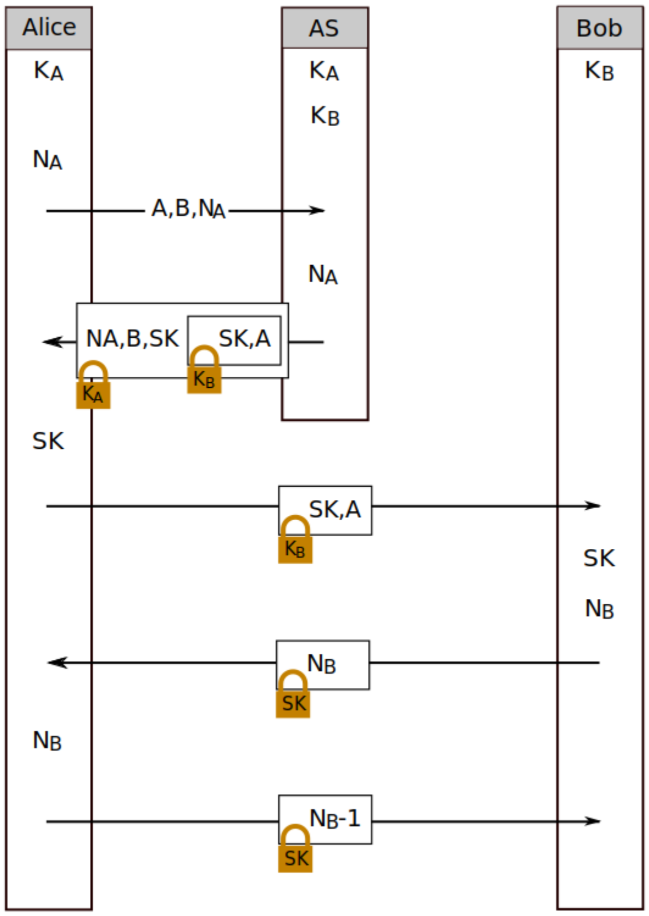


Figure 1: NS Symmetric Key Protocol

3.2 Protocol Designing :

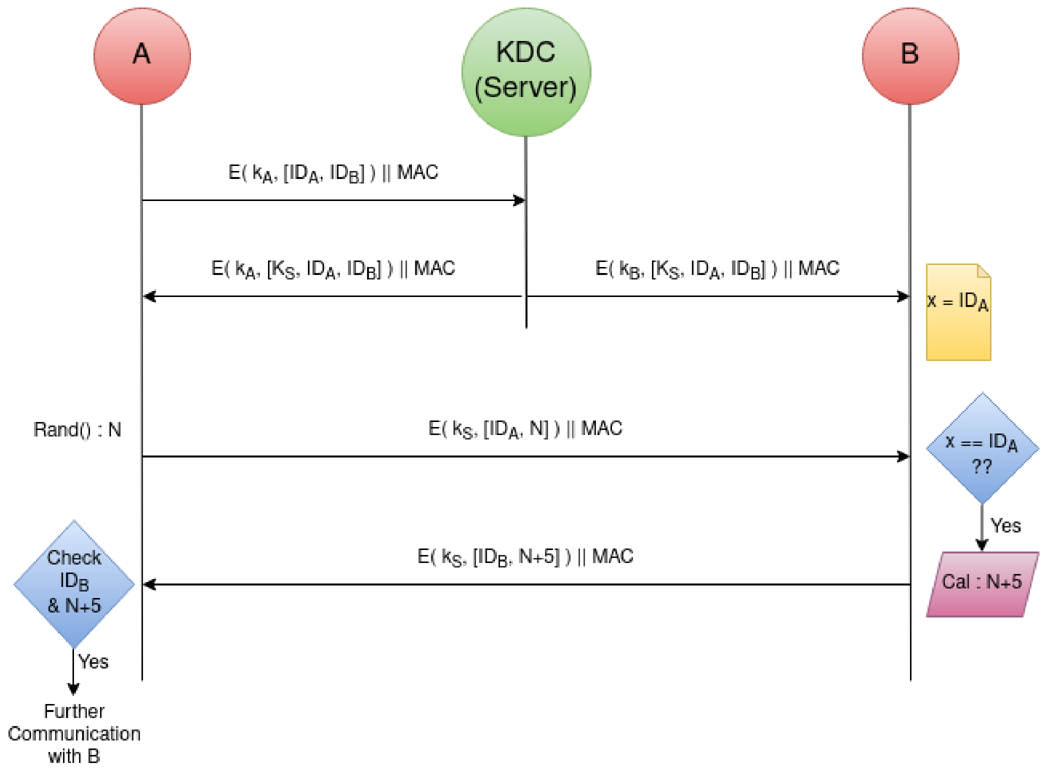
Using the above mentioned protocol, between client/user and server, whatever symmetric keys are distributed, are going to be called *session keys* as these keys will be thrown away just after encryption and decryption is done. “*Needham-Schroeder Symmetric Key Protocol*" is modified[1] to generate and distribute symmetric *session keys* as follows[3] :

3.2.1 *Generation and distribution of Session Keys*

Note

The server, in our office, has the access of ***QRNG***, so it can generate random numbers from that ***QRNG*** and these random numbers can be used as session keys

■ ***Communication Diagram :***



A
KDC (Server)
B
E(k_A, [ID_A, ID_B]) || MAC
E(k_A, [K_S, ID_A, ID_B]) || MAC
E(k_B, [K_S, ID_A, ID_B]) || MAC
x = ID_A
Rand() : N
E(k_S, [ID_A, N]) || MAC
x == ID_A ??
Yes
Cal : N+5
E(k_S, [ID_B, N+5]) || MAC
Check ID_B & N+5
Yes
Further Communication with B

Figure 2: Session (Symmetric) Key Generation

■ ***Algorithm :***

Here, A and B are two parties, KDC (key distribution center) is the server having access of QRNG, and trusted by both parties. In Communication :

- ID_A , ID_B are identities of A and B respectively
- K_A is a symmetric key known only to A and KDC. We call K_A ,the master key of A.
- K_B is a symmetric key known only to B and KDC. We call K_B ,the master key of B.
- N is nonce, may be time-stamp.
- MAC stands for Message Authentication Code which is used to check integrity of the encrypted data.

The protocol can be specified as follows :

1. $A \longrightarrow KDC \,:\, Enc(K_A; ID_A, ID_B)||MAC$
   A sends a message to the KDC(server) identifying herself and B encrypted under her master key K_A attached with MAC
2. $KDC \longrightarrow A \,:\, Enc(K_A; K_S, ID_A, ID_B)||MAC;$
   $KDC \longrightarrow B \,:\, Enc(K_B; K_S, ID_A, ID_B)||MAC$
   Server generates a session key, say KS and sends that session key KS to both A and B in encrypted form, encrypted under their corresponding master keys.
3. $A \longrightarrow B \,:\, Enc(K_S; ID_A, N)||MAC$
   A sends a nonce (may be time-stamp) encrypted under KS to show that she has the key.
4. $B \longrightarrow A \,:\, Enc(K_S; ID_B, N + 5)||MAC$
   B performs a simple operation on the nonce, re-encrypting it and sends it back verifying that he is still alive, he holds the key K_S .

3.2.2 *Generation and distribution of Master Key :*

The above described *session key* distribution protocol assumed that the server and every user/client is sharing a symmetric key, between them, which is somehow known to both of them. As an example, consider the user-A, then the symmetric key for user-A, K_A is shared (only) between himself and the server. Let's call it *Master Key* for user-A. Now it's really hectic to setup all these *Master Keys* for all users in the server. So here I am proposing an protocol, through which, the server and a user can distribute that *master key* (symmetric key known by both from prior) of that user between them. Since the security of the *session key distribution protocol* totally depends on two assumptions :
1. The server is a trusted server. It's not sharing the details of a user to another (the shared secret symmetric key, we called it *master key*) and also if a user A wants to make a secure connection to user B, then the server sends the *session key* to A & B not to another user C.
2. Every user keeps it's *master key* secret.
So, those *master keys* should be distributed securely and also those should be kept secret by both of them. Server can keep that database of *master keys* securely using an *MFA dongle*, but in client/user's system it's quite difficult to store the *master key* securely. So, the *master key* generation process will depend on some id & password method. Let's go to the protocol :

■ **Algorithm :**

- It will be a one time process and this algorithm will be executed at the of registration.
- Every user/client who wants to connect with the network/server, must have to register itself before using this service.
- Here C stands for client and S stands for server.
- All the computation to generate shared key k_1 will be performed in some prime field. We would like to choose some 256-bit large prime P and a generator g of that field
- This protocol depends on the "*Lagrange interpolation theorem over finite fields*", specifically the theorem says that n+1 points on a polynomial uniquely determines a polynomial of degree less than or equal to n . As an example

through 2 given points exactly one straight line can be drawn. Straight lines are nothing but a polynomial of degree less than or equal to 1 ($y = ax + b$ is the equation of a straight line where $ax + b$ is a polynomial of degree 1 and if $a = 0$ then $y = b$ type of straight lines are consisting of polynomials of degree 0). This theorem also holds on finite fields. We will use this theorem to compute master key over the prime field $(Z_P, \oplus, \odot)$, where $\oplus$ stands for addition modulo P and $\odot$ stands for multiplication modulo P . This concept is taken from SSS (Shamir's Secret Sharing) in MPC (Multi Party Computation).

The protocol can be described as follows :

1. $C \longrightarrow S$: Ping the server to be signed up.
2. $S \longrightarrow C$: *STS* Protocol or *DHKE* or *ECDHKE*
   Basically use some key exchange protocol to generate a shared key k_1 (say). Here in figure I have chose *DHKE* protocol.
3. $C \longrightarrow S$: $c_1 = Enc_{k_1}(p_1), where p_1 = (x_1, y_1), x_1 = SHA-256(Mk\_Id)$ $\&y1 = SHA-256(Mk\_Pass)$
   Here we have used SHA-256 as a hash function, we did the hash of Mk_Id and Mk_Pass for these reasons, one is that it outputs 256-bit long hashed value, which can be used directly to compute 256-bit long Master-key and also the confidentiality remains maintained as it's a OWF (one way function). Client/user sends the p_1 encrypted under the shared key k1 to the KDC (server). Server will store this $p1 = (x_1, y_1)$securely.
4. $S \longrightarrow C$: $c_2, c_3, c_4, c_5, where c_j = Enc_{k_1}(p_j), p_j = (x_j, y_j), and x_j, y_j$ are 256-bit long random numbers for all j = 2,3,4,5 Server chose (4 $*$ 2 =)8 random numbers and server can easily get these random numbers from it's *QRNG*. Then server encrypts these 4 order pairs or co-ordinates using the shared key k_1 and sends those ciphertext to the client/user.

■ **Communication Diagram of *Master Key* Generation**

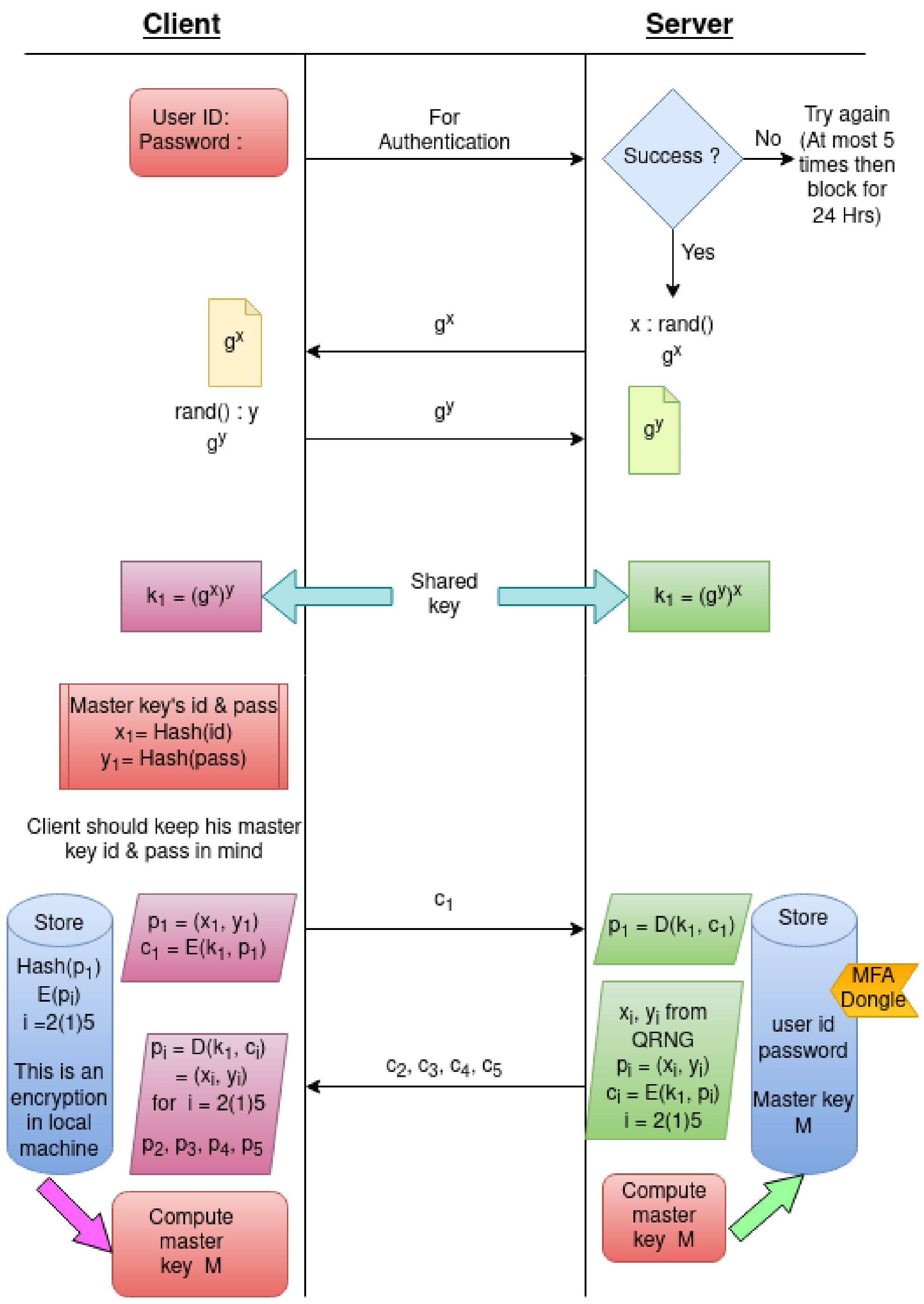


Figure 3: Master (Symmetric) Key Generation

■ **Computation of Master Key :**

Client/user can decrypt those and keep those $p_j s(j = 2, 3, 4, 5)$ securely in it's local database using some encryption algorithm. Now both the client and server have 5 points p_1, p_2, p_3, p_4, p_5 . Using "*Lagrange Interpolation over finite fields*", both of them can easily compute the unique polynomial of degree less than or equal to 4 by their own. Let's say this uniquely generated polynomial from those 5 points is $f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + a_4 x^4$, where a_j 's are numbers in the prime (finite) field $(Z_P, \oplus, \odot)$. Now both of them can set $f(0) = a_0$ or both can select some number say b by their mutual discussion and can evaluate $f(b)$ and set this value as master key for that user.

Server will compute the master key corresponding to the user/client and keep store that master key in it's database securely. At the user end user will keep store only those 4 points p_2, p_3, p_4, p_5 securely in it's local machine's database using some encryption algorithm. At the end we can add two communications extra just for checking whether the master key is generated correctly or not. This is the whole algorithm for generating master key of a user/client or we can say it the registration process of a user for joining network.

3.3 Implementation Part :

Let's first focus on implementation of client/user side application. After completing it server side application will be built and after that secure connection between server and client/user side application will be established.

■ **Description of Client/User side application :**

I have started to build up a client-server application through which that required secure communication via server can be done. I have preferred *PyQt* version 5 to make the *GUI* of the client side application, *Python3* to write all the codes, should perform in background and *Sqlite3* to maintain the local database for authentication locally like *SAM*. Basically we need an application through which a valid user can connect with server and can send or receive data securely through the local offline server. Below I am first describing works as per requirements in short and latter those will be described in detail.

■ Process going to be happened in this client/user side application :

1. At first the *LogIn* page will be opened and user can be logged in by putting his/her correct username and password. If the user is not an existing user then by clicking the *SignUp* button (s)he can go to the *SignUp* page and register himself by filling up the required fields.

2. After logging in, the Main Application widget will be opened. Here user can write message or attach file and send it to other user connected with the same network.

3. By clicking send button, firstly the *MK_id_pwd_PopUp* form will be opened, and ask for master-key-id and master-key-password. In background our application do hashes these two values using SHA-256 to get $p_1 = (x_1, y_1)$, co-ordinates of 256-bit long numbers in the finite field $(Z_P, \oplus, \odot)$. Then it will fetch pre-generated $p_j = (x_j, y_j)s(forj = 2, 3, 4, 5)$ which were kept secret at time of registration in local machine or in local database using some local system's encryption algorithm. These five points will be sent to the *Lagrange_Interpolation.py* as input. This function will return the *master-key* of that user. The above python code does the Lagrange interpolation over the pre-specified finite field. Now this computed *master key* will be hashed with *SHA-256* and this hash value will be compared with the pre-stored hash value of the master-key which was stored at the time of registration process.

4. If this verification becomes successful then that *master-key* will be used as encryption key of our chosen encryption algorithm *AES-256*, otherwise it will raise a popup window which will show that either "*master-key-id*" or "*master-key-password*" is wrong.

5. Now user has generated his *master-key*. Using this *master-key* and following the "*session-key generation and distribution*" algorithm, user/client will communicate securely with the server over the network. Then the server will provide the *session-key* to both the sender User-A and the receiver User-B.

6. After receiving the *session-key* from the server, the client/user side application will call the "*aes256.py*" with the session key as encryption key and

will encrypt the message and attached file.

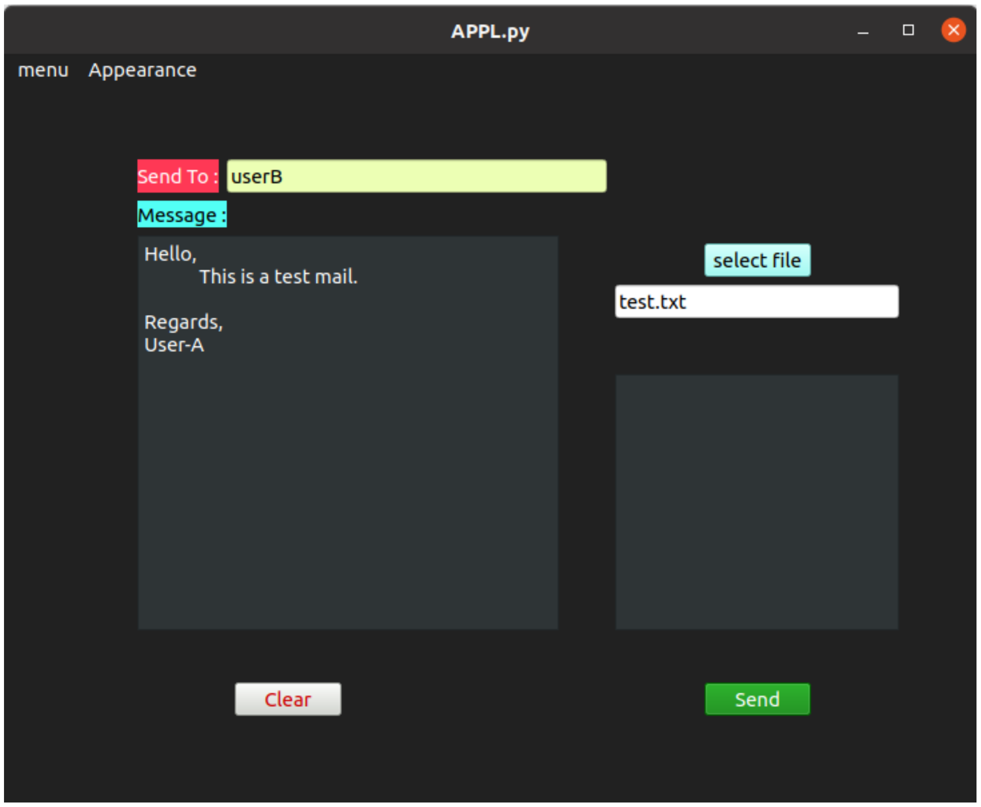
7. As per *session-key* generation algorithm User-B also will get the *session-key*. After getting it User-B will remain in listen mode, expecting to get encrypted data from User-A.

8. After receiving the encrypted data and decrypting it , User-B will throw the session key away. Also after sending the encrypted data to User-B, sender User-A will throw that away.

■ Description of the client/user side application :

1. ***Main Application widget :***

    - We need a widget where a user can specify to whom he/she wants to send data.
    - A message box is needed to write down messages
    - An option is required to choose a file, user may want to send other user
    - Also there should be options for Log In, Sign Up, and Log Out.
    - A send button is required



userB

Hello,
This is a test mail.

Regards,
User-A

test.txt

Figure 4: Main Application Widget

2. ***Log In Widget :***

   For authentication purpose a Log In page is required. This should be connected with local *Sqlite3* database.

   It has two fields 1. Username and 2. Password. Also there should be a submit button for Logging In , a button for Signing Up for those users who are not registered and also there will be a option of Forget Password .

   If a user puts correct credentials then he/she will be connected to the server and there those credentials will be verified again. If verification from the server's database becomes successful then he/she will be redirected to the Main Application widget and will be connected to the server so that he/she can send messages or files to others and also will be able to view his/her inbox.

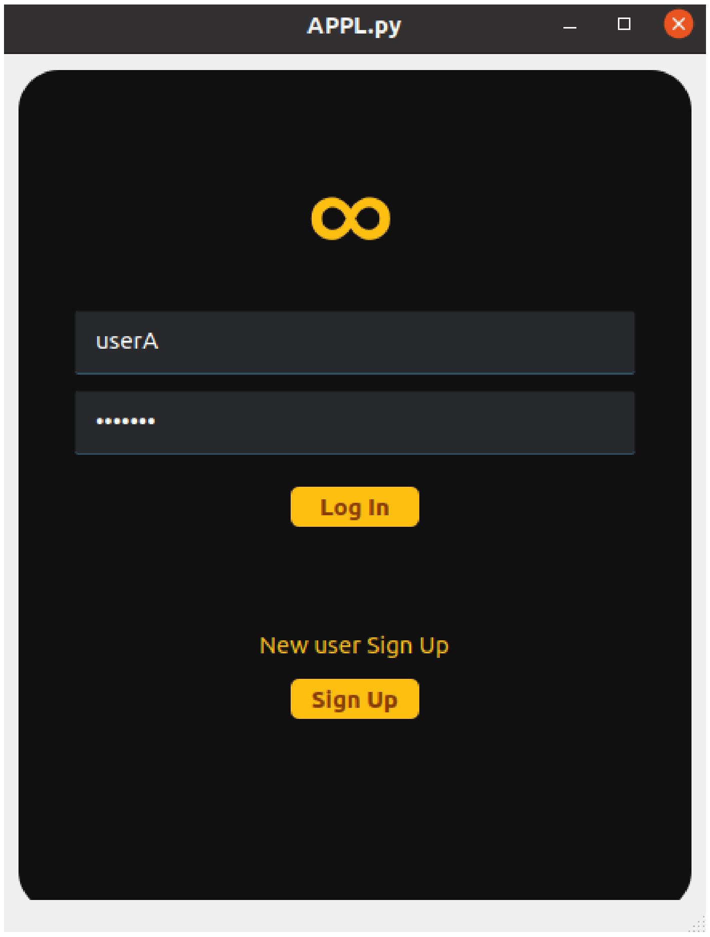


Figure 5: Log In Widget

3. ***Sign Up Widget***

   For registration process of a user this widget is required. This will remain connected with server and it's database. While a new user comes to join the network he/she has to register first to the network, has to set unique *Username*, *Password*, *Master-Key ID*, and *Master-Key Password*. This unique Username and Hash(Password) will be loaded to the local machines's database for authentication purpose at the time of *Log_In* , Also these four details i.e. *Username* and hash values of *Password*, *MK_ID*, *MK_Pass* with added salt corresponding to a user will be stored in server's database securely.

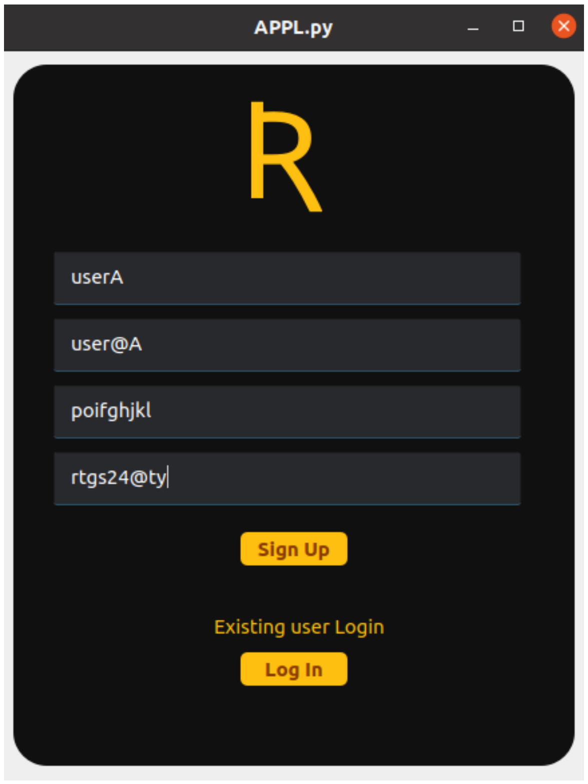


Figure 6: Sign Up Widget

■ **Functionalities :**

Sl No.	**Button/Action/Field**	**Description**
1.	Enter the username	User has to set his unique username
2.	Enter the password	User has to set his password also
3.	Enter the MK_ID	User will choose his Master-Key ID and will write here
4.	Enter the MK_Pass	User will choose his Master-Key Password and will write here
5.	Sign Up	This button connects this application with local database and also with the server. Master key generation and distribution algorithm will run in behind and registration of a user will be done.
6.	Log In	This button will redirect to the Log In page

Table 1: Functionalities

4. ***MK_id_pw_PopUP Widget :***
   This widget is nothing but a form with just two fields mentioned below. By pressing Send button in the Main Application widget, this widget will be popped up

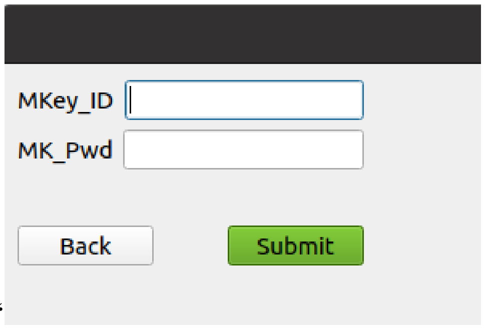


Figure 7: MK ID & Pass Widget

■ **Functionalities :**

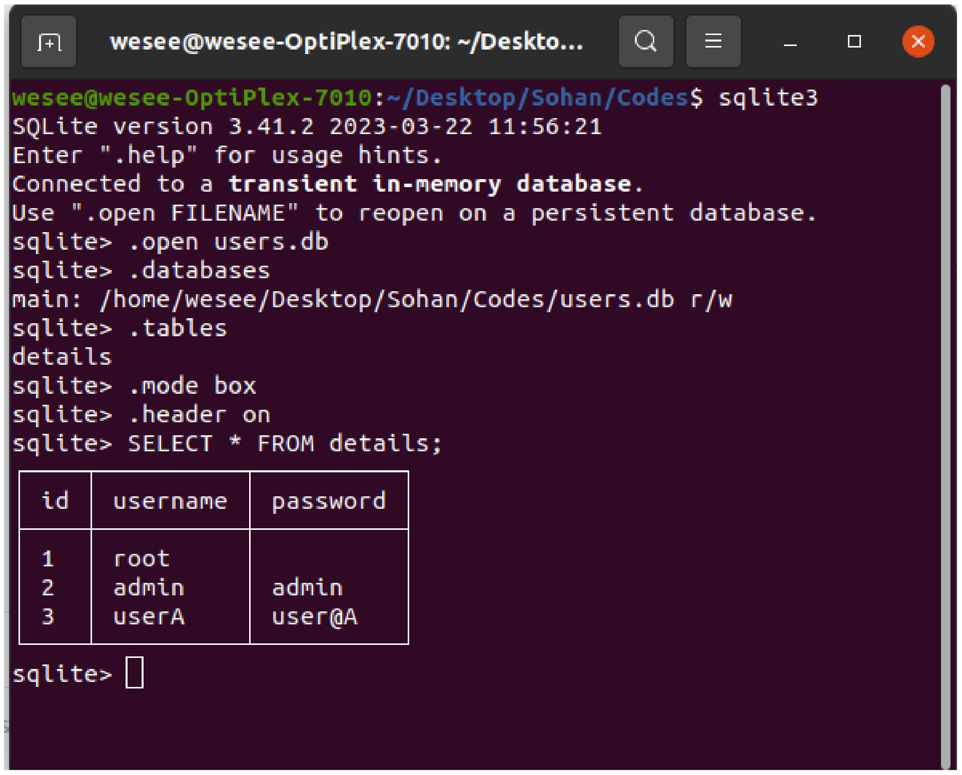
Sl No.	**Button/Action/Field**	**Description**
1.	MKey_ID	In this placeholder user has to put his secret master key id
2.	MKey_Pwd	In this placeholder user has to put his secret master key password
3.	Submit	This button sends message and selected file to the recipient's address
4.	Back	This button simply set the previous Main Application widget

Table 2: Functionalities

In behind after pressing Submit button our application will be connected with local database and fetch those 4 points $p_i = (x_i, y_i)(i = 2, 3, 4, 5)$ which are kept in user's database and then using these 4 points and the point $p_i = (x_i, y_i)$, we will use Lagrange interpolation over finite field to generate the master key, where x1 = SHA-256(MK_ID) and $y_1 = SHA - 256($ MK_Pwd). This master key will be used to encrypt the message, user wrote in message box and also to encrypt the file, user selected in Main Application widget. Here we have used AES-256 as encryption algorithm. After encryption both the message and the selected file will be sent to the recipient.

5. ***Local SQLite3 Database :***

   I have used SQLite3 to create user-side local database. “users.db” is created and thereis currently only one table named “details” This table has currently three columns named “id”, “username”, “password” . We can change this according to our requirements .



```
wesee@wesee-OptiPlex-7010:~/Desktop/Sohan/Codes$ sqlite3
SQLite version 3.41.2 2023-03-22 11:56:21
Enter ".help" for usage hints.
Connected to a transient in-memory database.
Use ".open FILENAME" to reopen on a persistent database.
sqlite> .open users.db
sqlite> .databases
main: /home/wesee/Desktop/Sohan/Codes/users.db r/w
sqlite> .tables
details
sqlite> .mode box
sqlite> .header on
sqlite> SELECT * FROM details;
┌────┬──────────┬──────────┐
│ id │ username │ password │
├────┼──────────┼──────────┤
│ 1  │ root     │          │
│ 2  │ admin    │ admin    │
│ 3  │ userA    │ user@A   │
└────┴──────────┴──────────┘
sqlite> 
```

Figure 8: SQLite3 local database

4 Future plans :

1. Try to find out some better key exchange protocol than Diffie-Hellman key exchange protocol
2. Sort out all the attacks of my designed session key generation algorithm and then will try to modify accordingly
3. I have setup the server and have to make connection with this application and finally have to deploy it.

■ **Server Configuration :**

Mainly two types of server is required, one is a mail server and other is storage server. Also a database is required to maintain.

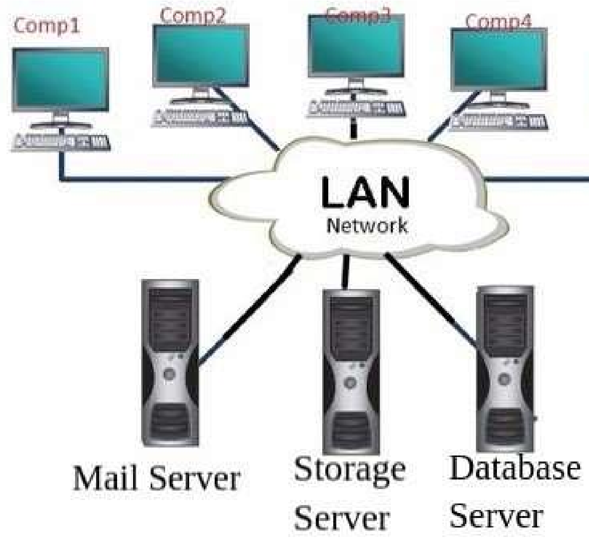


Figure 9: Required Servers

(a) ***Mail Server :***
This mail server will be used to send encrypted data through the network. It will have two categories: outgoing mail server and incoming mail server. Outgoing mail servers uses SMTP (simple mail transfer protocol). We use POP3 (post office protocol) for incoming mail server. Since we are not going to use internet so we will not go with IMAP.

(b) ***Storage Server :***
This server will keep store key-banks generated from QRNG. Also it will be used to keep backups. We have configure it

(c) ***Database / Authentication Server :***
This server will maintain databases of user's "master-keys". Also authentication service will be maintained from this server.

We will configure all of the above servers. Also for maintenance we have to build up a server side application which will helps to take backups, helps to maintain database and other related things. Also we have to distribute loads to provide users better services.

■ Load Balancing :

It's a process or technique of distributing workloads across multiple resources to optimize efficiency, reliability, and capacity. This in turn will result in high performance with less resource utilization. Load balancers are usually referred to as Proxied Devices and are placed between Clients and Servers, which means they act as servers for clients and vice versa. They distribute the incoming requested services across multiple and the best available servers.

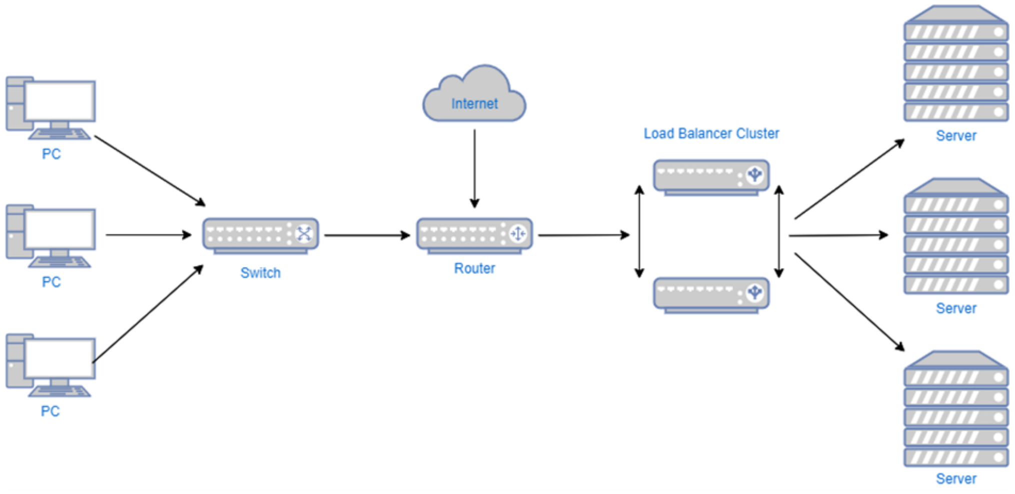


Figure 10: Load Balancing

4. Finally, we have to do the security analysis, have to look into the time complexities of various algorithms and overall performance after deployment.

5 Conclusions :

.

Session Key distribution protocol is modified.

Master Key generation and distribution protocol is designed.

The client/user side application is built .

Now I have to configure the server, have to build the server side application or software for maintenance and finally make connection between client/user side application with the server.

References

[1] Giovanni Maria Sacco Dorothy E. Denning. Timestamps in key distribution protocols. 24, August 1981. https://dl.acm.org/doi/pdf/10.1145/358722.358740.

[2] Michael D. Schroeder Roger M. Needham. Using encryption for authentication in large networks of computers. 21, December 1978. https://dl.acm.org/doi/pdf/10.1145/359657.359659.

[3] Michael D. Schroeder Roger M. Needham. Acm sigops operating systems review. 21, January 1987. https://dl.acm.org/doi/pdf/10.1145/24592.24593.

[4] Michael Schroeder Roger Needham. Needham schroeder symmetric key.

[5] the free encyclopedia Wikipedia. Needham–schroeder protocol, January.