Comp. 5610 Computer and Network Security 1
Project Title: Encrypted messaging application for secure communication.

Team Members:
Rohan Girase.
Manoj Reddy Dumpa.
Omkar Salunke.
Omkar Gadgil.

Abstract:
Inspired by all the latest messaging applications our objective is to create a secure messaging application of our own. Just like a typical application following are the components which will be the core of the application.

Description:
Our Goal was to create a secure end-to-end encrypted messaging application with various features that will allow two users to securely communicate with each other without any security breach such as man in the middle attack, Denial of Service and any other kind of attacks.
For creating this application we require basic building blocks of cryptosystem, encryption techniques and fundamentals of SDLC (Software Development life cycle).

Technology Stack:

Front-End:
There was a need to develop an intuitive and Responsive GUI so that users do not face any difficulties for communication therefore, the front end of the application was Developed using Angular 2+ with Typescript and Ionic framework that will allow application to be viewed on native mobile browsers. Angular 2 and Ionic has built in UI components such as Navbar Component and Ionic Badges that makes application responsive and can be viewed on all Display screens.

We used HTTP get request and REST API to make calls to database server and securely store the encrypted messages.

Back-End:
The backend was needed to store encrypted messages and end establish a secure communication with front end. This was completely achieved using Python. Authentication services and Cryptosystem was developed in backend.

Cryptosystem and Encryption:

We used following encryption algorithm and Cryptosystems:

1. Diffie-Hellman.
2. AES.
3. CBC.

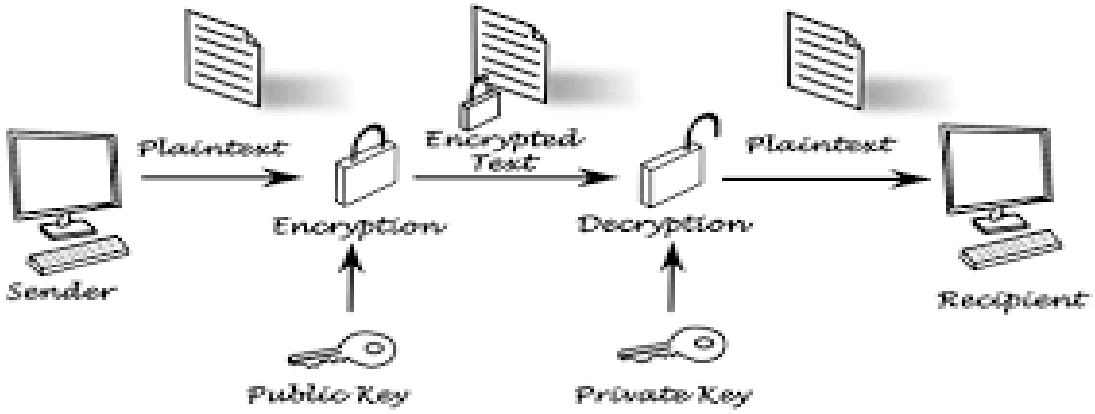

Figure 1: System Architecture

System Architecture: There are 2 parties involved namely Sender and Receiver. The Sender sends message to receiver through a secured communication channel. The Receiver receives the Decrypted message which is encrypted by sender using Hybrid encryption i.e. symmetric and public key encryption.
Various algorithms are implemented during the message processing such as AES, Diffie-Hellman key exchange.

1. Diffie-Hellman

Diffie–Hellman Key Exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network.
Private key was generated using the random number generator and Math function was used to generate public key.
Hashing is used for generation of a secret key with the help of public and private key.

As you can see from figure 2, we can see that we used random number generator to generate the key for 'P'. The keys generated by the following code are symmetric.

Sender computes: $B^a \cong (BASE^b)^a \cong BASE^{ba} \bmod P$
Receiver computes: $A^b \cong (BASE^a)^b \cong BASE^{ba} \bmod P$

```
def generate_private_key():
    random.randint(2, P-2)


def generate_public_key(Priv):
    return pow(BASE, Priv, P)


def generate_secret_key(Priv, Pub):
    s = pow(Pub, Priv, P)
    s = hashlib.sha1(str(s)).hexdigest()
    i = 0
    key = []
    while i < 32:
        key += [int(s[i:i+2], 16)]
        i += 2
return key
```

Figure 2: Diffie Hellman Key Exchange in Python.

2. AES & 3. CBC

Whenever Sender sends message following process takes place sequentially: Message encryption, block generation, and IV gnenration

```
def AESencrypt(message, key):
```

```
    blks = block(message)
    IV = generate_IV()
    E = CBC(blks, key, IV)
    return [E, IV]
```

```
def block(mess):
    i = 0
    xi = []
    while i < len(mess) - 16:
        xi.append(mess[i:i+16])
        i = i + 16
    xi.append(mess[i:] + "~~~~~~~~~~~~~~~~"[:16 - (len(mess) - i)])
    return xi
```

```
def CBC(xi, key, iv):
    i = 0
    IV = iv
    new = ''
    yi = []
    while i < len(xi):
        for x in range(0, 16):
            t = (ord(xi[i][x]) ^ IV[x // 4][x % 4])
            new += chr(t)
        yi += [encrypt(new, key)]
        new = ''
        IV = yi[i]
        i += 1
    return yi
```

```
def generate_IV():
    IV = [[random.randint(0, 255), random.randint(0, 255), random.randint(0, 255), random.randint(0,
255)],
          [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255), random.randint(0,
255)],
          [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255), random.randint(0,
255)],
          [random.randint(0, 255), random.randint(0, 255), random.randint(0, 255), random.randint(0,
255)]]

    return IV
```

When receiver receives the message following process takes place:
Decryption, CBC inverse and unblock.

```
def AESdecrypt(blks, key, IV):
    blked = CBC_Inv(blks, key, IV)
    return unblock(blked)
```

```
def CBC_Inv(yi, key, iv):
    new = ''
    i = 0
    IV = iv
    xi = []
    while i < len(yi):
        xi += [decrypt(yi[i], key)]
        for x in range(0, 16):
            new += chr(ord(xi[i][x]) ^ IV[x // 4][x % 4])
        xi[i] = new
        IV = yi[i]
        i += 1
        new = ''
    return xi
```

```
def unblock(blks):
    text =''
    x = 0
    while x < len(blks) - 1:
        text += blks[x]
        x += 1
    text += unpad(blks[x])
    return text
```

Working:

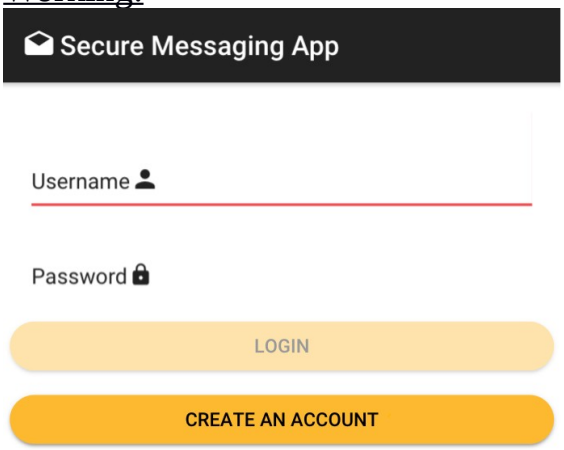

Figure 3: Homepage

The user logs into the system using his credentials as you can see from the figure above. After logging into the system he can perform following tasks.

a. Coversations:

User can view previous conversations with his friends and send new messages to them. All the messages are stored in an Encypted database.db file. Conversations can be viewed from figure 4 and figure 5.

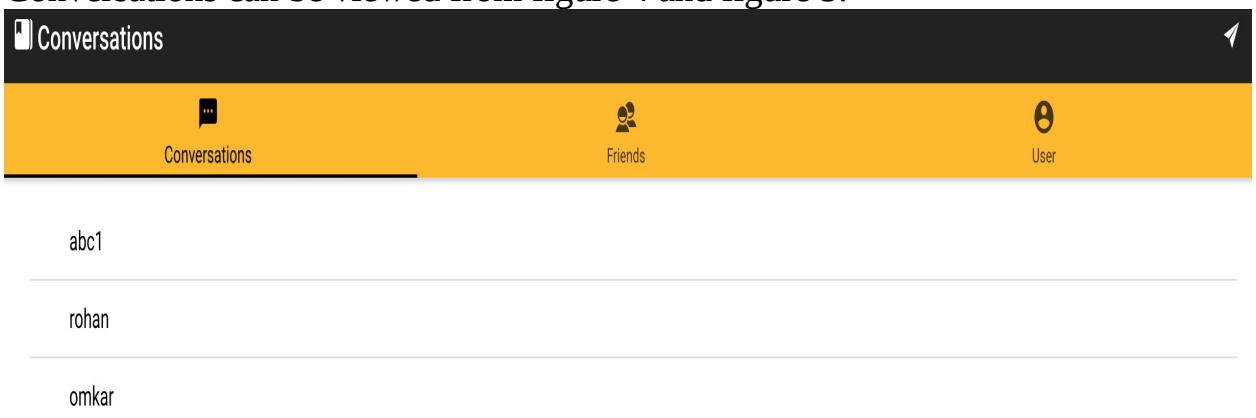

Figure 4: Conversations tab

b. Friends:

User can add new friends and can directly send messages to them after adding them. Every new friend added can later be seen in the conversation tab.

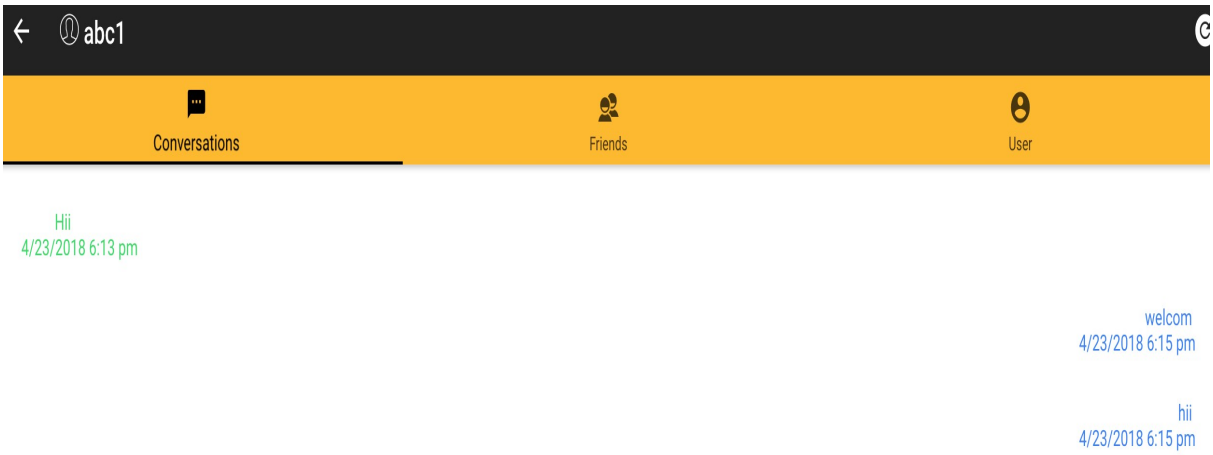

Figure 5: Messaging Interface

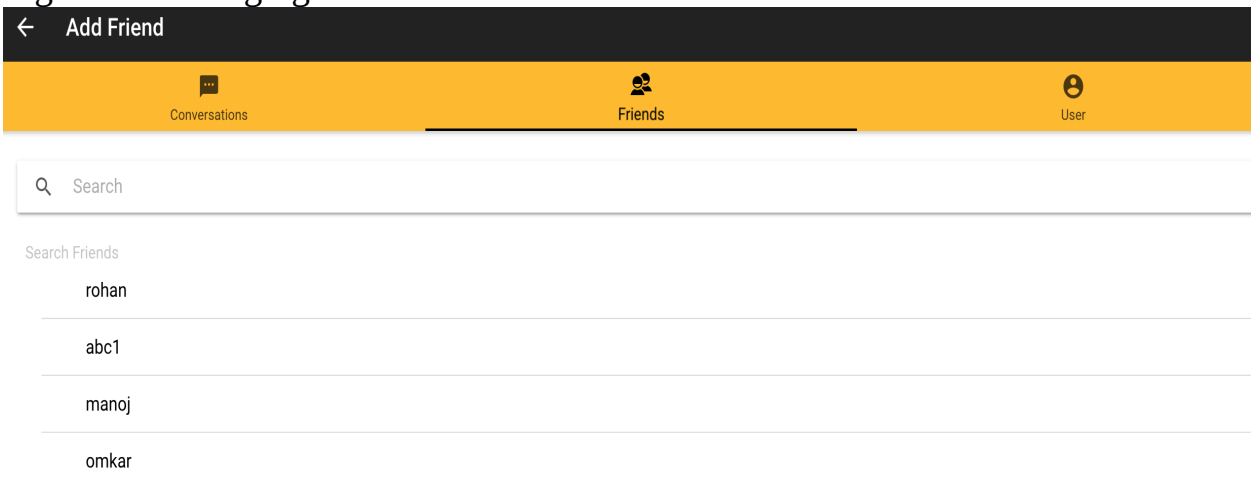

Figure 6: Friends tab.

c. Users Tab:

Users can view their profile and logout from the system in this tab. Once logged out all the messages are encrypted and stored in the Database.db file and cannot be accessed by any third party.

What we achieved: We tried to implement all the concepts explained in the lectures in form of an encrypted messaging application using the materials available to us from various technologies such as Angular framework and Python.

Future Work:

- Refine the UI
- Creating Profile page and statistics of messages sent
- Implement file exchange along with messages.

References:

AES: https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
https://en.wikipedia.org/wiki/Rijndael_mix_columns
https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development

Ionic Framework: https://ionicframework.com/docs/api/
Diffie-hellmman: https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange