Secure Notes Manager

GONE AKASH 21CSB0B19
T. JAYANTH VINNY 21CSB0B57

1. Introduction

The problem addressed in this project the goal is to create a system where clients can securely manage their notes, ensuring confidentiality , integrity and availability of their data. Implementing a client-server architecture allows clients to interact with a central server to perform operations like creating, updating, reading, and deleting notes.

Users expect their private notes to remain confidential. By employing symmetric cryptography and secure authentication, we must ensure that only authorized users can access and modify their notes, enhancing privacy protection. our project aims to provide a secure and efficient solution for note management while ensuring data privacy and integrity.

Many solutions use end-to-end encryption, where data is encrypted on the client side before being sent to the server. This ensures that even if the server is compromised, the data remains encrypted and unreadable without the decryption key. Secure solutions often employ strong authentication mechanisms. These approaches, along with advancements in encryption algorithms, secure communication protocols, and security best practices, contribute to the effectiveness of existing solutions in solving the problem of secure note management.

Key Contributions of our project are implementing robust authentication mechanisms, including salted hashed passwords and symmetric key encryption. Those who have username , password and the symmetric key only can access the data. So , we added an extra authenticated layer i.e symmetric key which decrypts the received data. The algorithm used is AES of CBC mode , which gives more security as encrypted and decrypted based on randomness of intialization vector. There is no key exchange as client only encrypting and decrypting , this solves us the major problem. We did encrypted and then encoding (in the form of ASCII characters) and also decoding and decryption , to minimize transmission errors while transferring encrypted data .

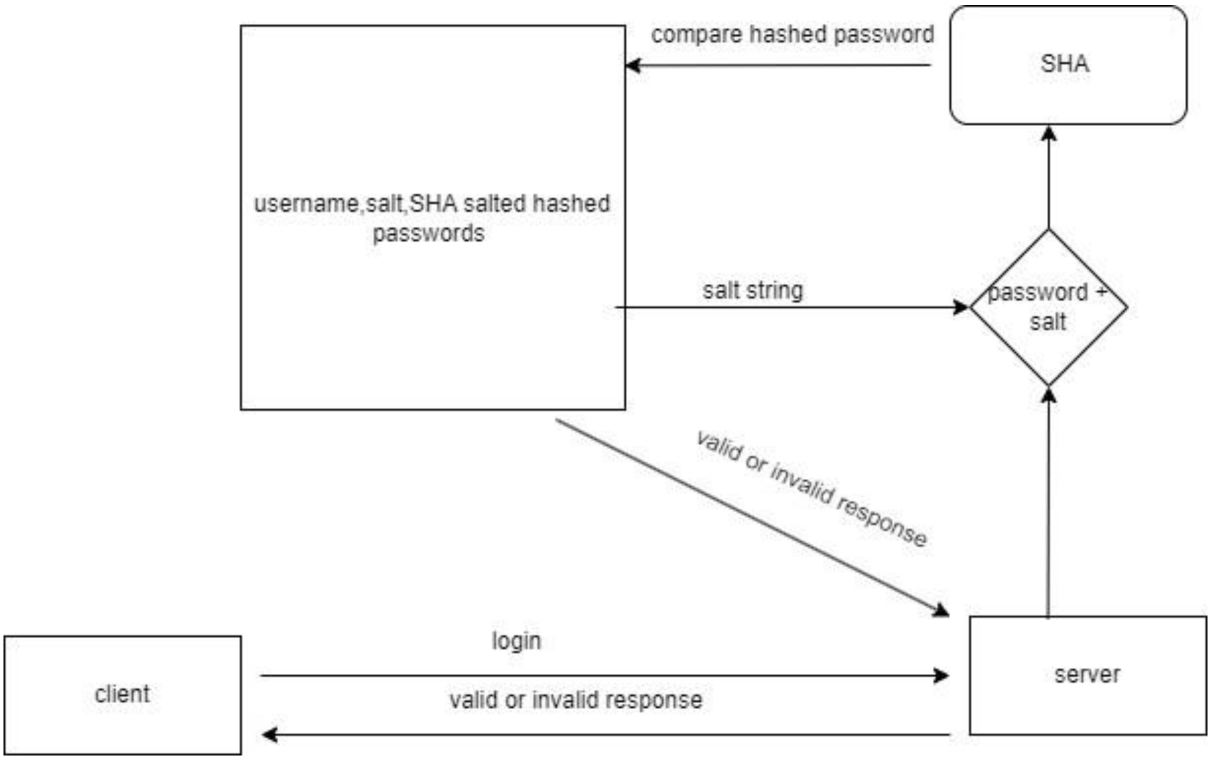
2. Objectives

- Must ensure that if server got attacked also , user privacy will be maintained . If user password leaked , then also wont effect confidentiality. Using symmetic key but no key exchange.
- Authenticating users for secure service. Avoid collisons in hashing by using salting.

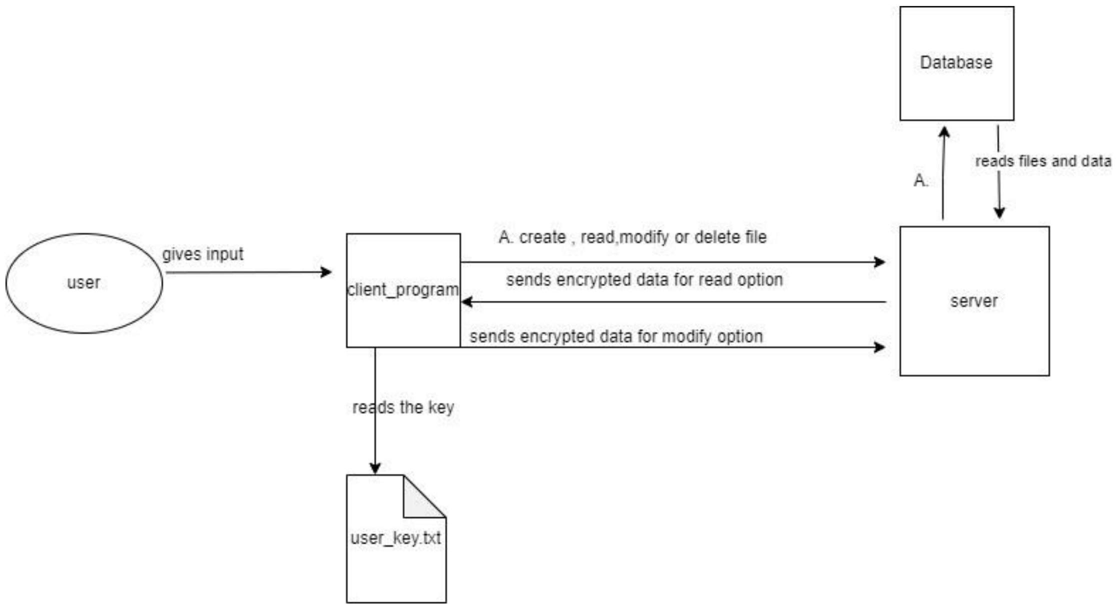
3. **Implementation and Results analysis**

We implement client-server architecture using bsd sockets and cryptopp library for encryption,decrytpion,key generation,salted hashing (SHA -512).

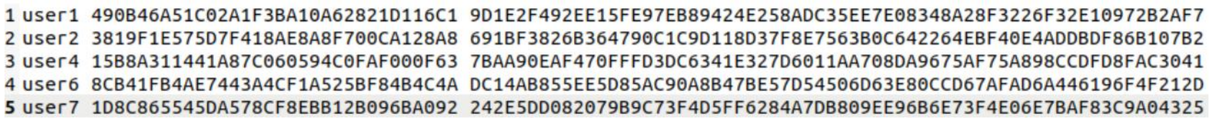
Authentication


compare hashed password
SHA
username,salt,SHA salted hashed passwords
salt string
password + salt
valid or invalid response
login
valid or invalid response
client
server

Functionality

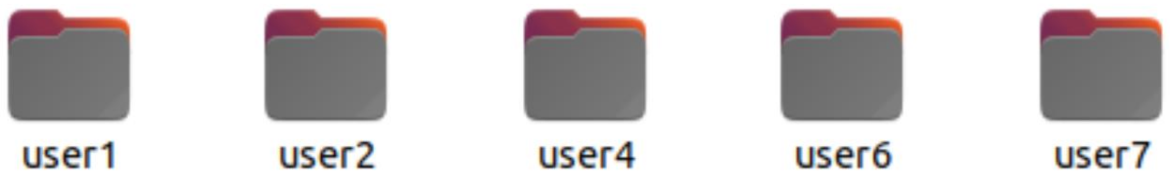

Database
reads files and data
A.
user
gives input
client_program
A. create , read,modify or delete file
sends encrypted data for read option
sends encrypted data for modify option
server
reads the key
user_key.txt

USERNAME,SALT STRING, HASHED PASSWORD FILE IN SERVER DATABASE

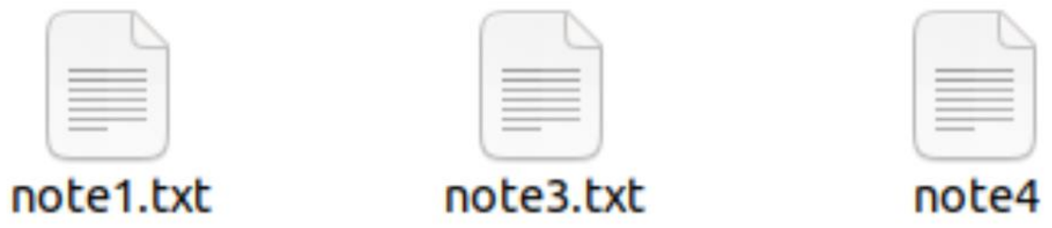


```
user1 490B46A51C02A1F3BA10A62821D116C1 9D1E2F492EE15FE97EB89424E258ADC35EE7E08348A28F3226F32E10972B2AF7
user2 3819F1E575D7F418AE8A8F700CA128A8 691BF3826B364790C1C9D118D37F8E7563B0C642264EBF40E4ADDBDF86B107B2
user4 15B8A311441A87C060594C0FAF000F63 7BAA90EAF470FFFD3DC6341E327D6011AA708DA9675AF75A898CCDFD8FAC3041
user6 8CB41FB4AE7443A4CF1A525BF84B4C4A DC14AB855EE5D85AC90A8B47BE57D54506D63E80CCD67AFAD6A446196F4F212D
user7 1D8C865545DA578CF8EBB12B096BA092 242E5DD082079B9C73F4D5FF6284A7DB809EE96B6E73F4E06E7BAF83C9A04325
```

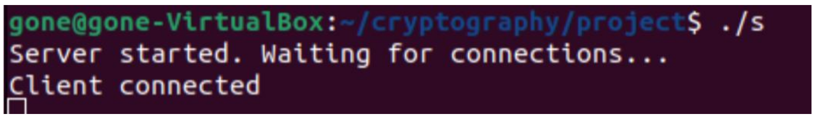
SERVER STORE USERS FILES IN SEPARATE FOLDER WITH USERNAME AS FOLDER NAME



USER NOTES FILES IN USER FOLDER IN SERVER DATABASE

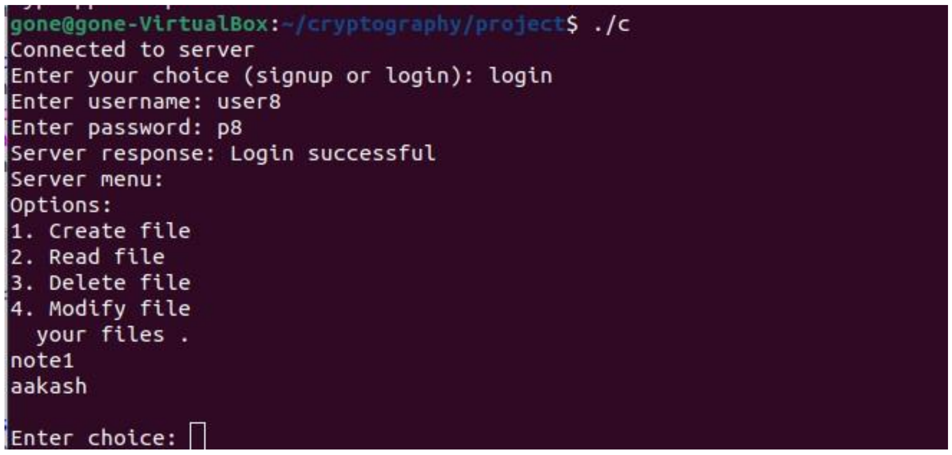


Server



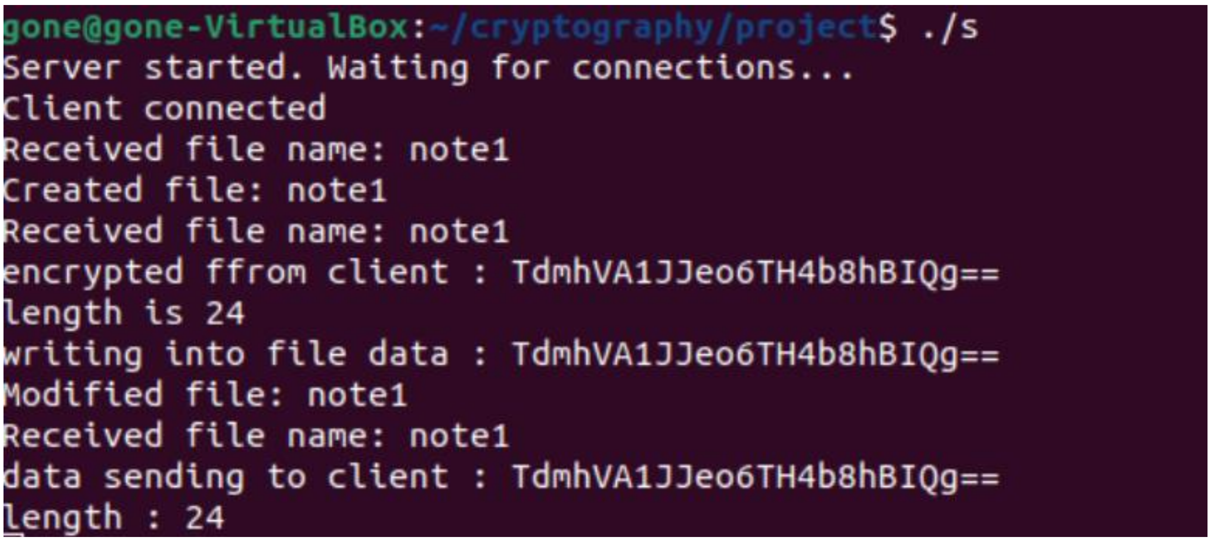
```
gone@gone-VirtualBox:~/cryptography/project$ ./s
Server started. Waiting for connections...
Client connected
```

Client



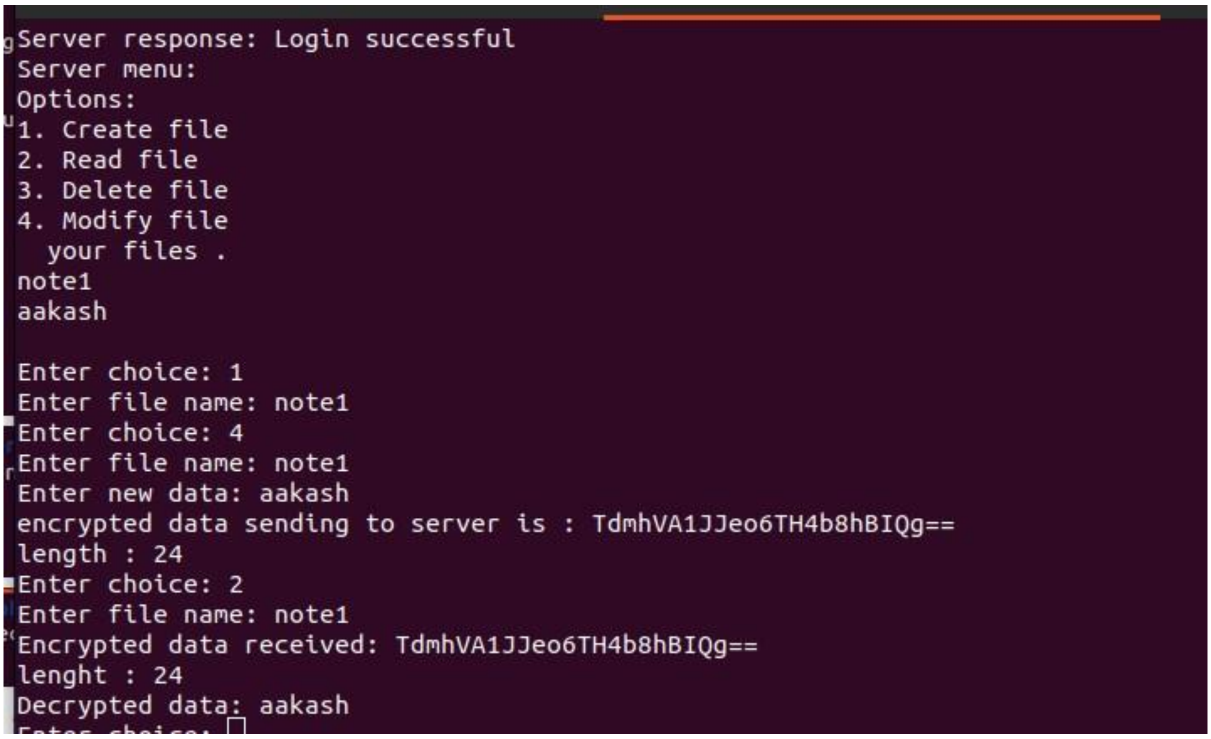
```
gone@gone-VirtualBox:~/cryptography/project$ ./c
Connected to server
Enter your choice (signup or login): login
Enter username: user8
Enter password: p8
Server response: Login successful
Server menu:
Options:
1. Create file
2. Read file
3. Delete file
4. Modify file
  your files .
note1
aakash

Enter choice: 
```

Server



```
gone@gone-VirtualBox:~/cryptography/project$ ./s
Server started. Waiting for connections...
Client connected
Received file name: note1
Created file: note1
Received file name: note1
encrypted ffrom client : TdmhVA1JJeo6TH4b8hBIQg==
length is 24
writing into file data : TdmhVA1JJeo6TH4b8hBIQg==
Modified file: note1
Received file name: note1
data sending to client : TdmhVA1JJeo6TH4b8hBIQg==
length : 24
```

Client



```
Server response: Login successful
Server menu:
Options:
1. Create file
2. Read file
3. Delete file
4. Modify file
  your files .
note1
aakash

Enter choice: 1
Enter file name: note1
Enter choice: 4
Enter file name: note1
Enter new data: aakash
encrypted data sending to server is : TdmhVA1JJeo6TH4b8hBIQg==
length : 24
Enter choice: 2
Enter file name: note1
Encrypted data received: TdmhVA1JJeo6TH4b8hBIQg==
lenght : 24
Decrypted data: aakash
```

For encrypting and decrypting data , we use AES algorithm CBC mode , so introduce randomness as intilization vector in CBC mode. We do encoding and decoding for encrypted and decrypted ,(to convert to ASCII characters) by using Base64 encoding as encrypted generate binary symbols and some non-ASCII characters . so it leads to transmission errors as we use string from one socket to another socket .

Store the files of user in server database in folders according to their username.for authentication , server assigns unique salt string to the each user , it concatenates with password and computes hash using SHA 512 and stores it in text file.for each login , server adds salt string to given password according to their username and calculate hashed password and compare it with existing hashed password in file , based on this it decides valid or invalid response .

Client program generates symmetric key during signup and stores it in user’s storage and reads every time when it logins. No need of key exchange , as client only encrypting and decrypting data.

4. Conclusion

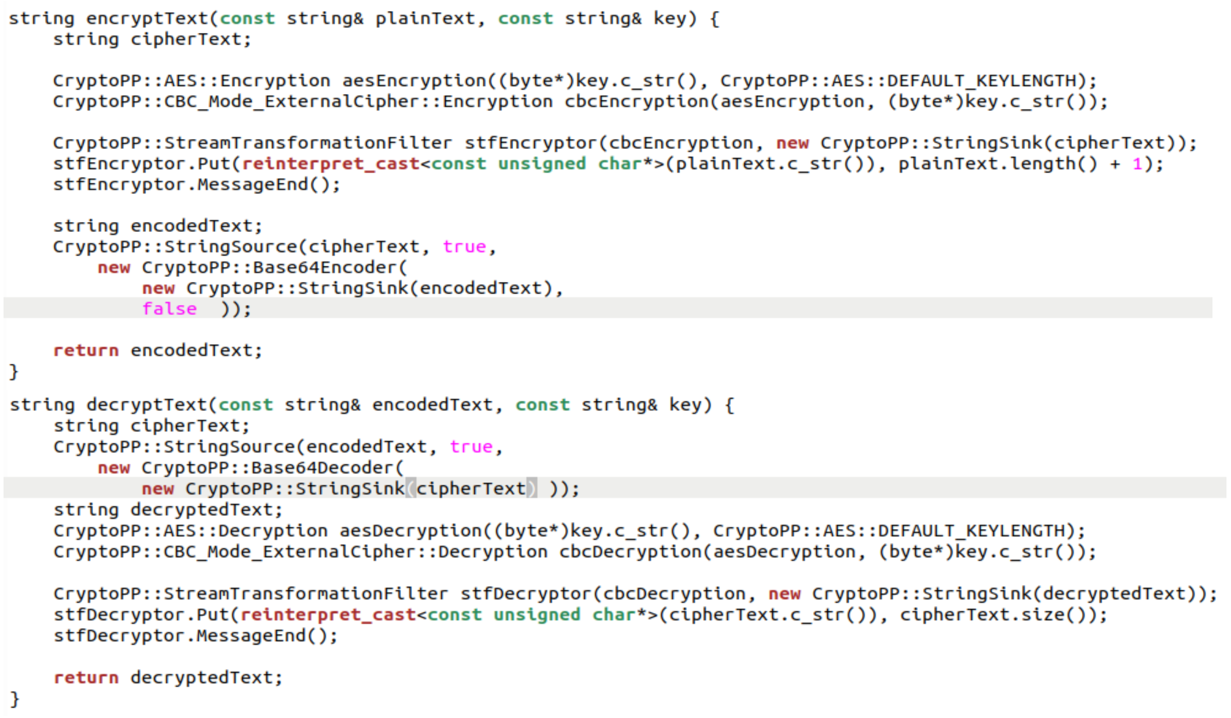
Our project is to implement secure notes manager by client server architecture , where clients can securely manage their notes, ensuring confidentiality , integrity and availability of their data . We have implemented the client such that it has both encrypting and decrypting capabilities and no key exchange , this also ensures efficieny of system. User can share password and the key to acess files in the same account. If other person knows password , they get only encrypted data as they don’t have the key. Salting for hashed passwords is beneficial as occurrence of collisons is less.Also used encoding and decoding to minimize transmission errors.

5. Learning outcomes

- Generating salt strings for each user and calculating hash using SHA-512 using cryptopp library,useful for authentication as compares original hashed password each time when user logged in.
- Generation of symmetric key , encrypting and decrypting data using AES algorithm , CBC mode and encoding and decoding to convert binary symbols to ASCII characters to reduce transmission errors.
- No key exchange , as client only encrypting and decrypted data.

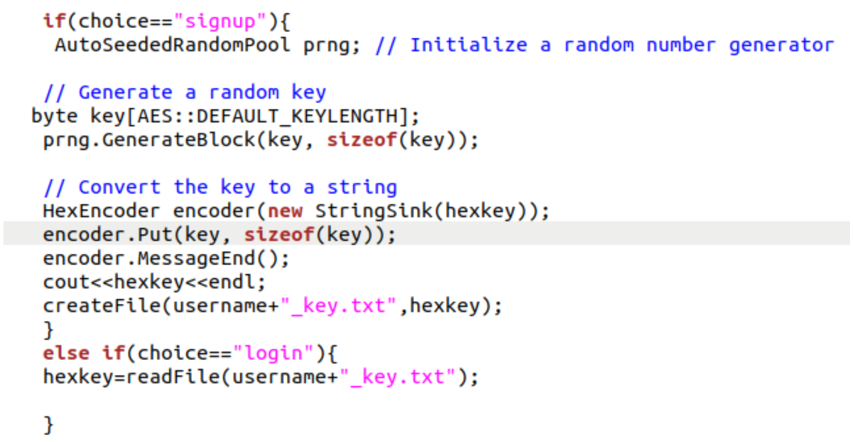
6. Source code:

Client.cpp



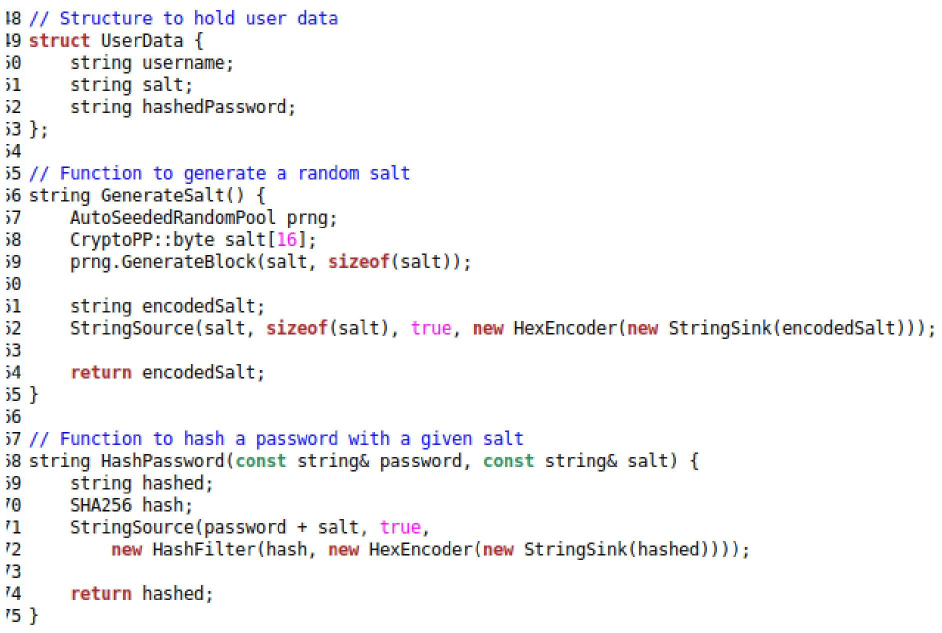
```
string encryptText(const string& plainText, const string& key) {
    string cipherText;

    CryptoPP::AES::Encryption aesEncryption((byte*)key.c_str(), CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Encryption cbcEncryption(aesEncryption, (byte*)key.c_str());

    CryptoPP::StreamTransformationFilter stfEncryptor(cbcEncryption, new CryptoPP::StringSink(cipherText));
    stfEncryptor.Put(reinterpret_cast<const unsigned char*>(plainText.c_str()), plainText.length() + 1);
    stfEncryptor.MessageEnd();

    string encodedText;
    CryptoPP::StringSource(cipherText, true,
        new CryptoPP::Base64Encoder(
            new CryptoPP::StringSink(encodedText),
            false  ));

    return encodedText;
}
string decryptText(const string& encodedText, const string& key) {
    string cipherText;
    CryptoPP::StringSource(encodedText, true,
        new CryptoPP::Base64Decoder(
            new CryptoPP::StringSink(cipherText) ));
    string decryptedText;
    CryptoPP::AES::Decryption aesDecryption((byte*)key.c_str(), CryptoPP::AES::DEFAULT_KEYLENGTH);
    CryptoPP::CBC_Mode_ExternalCipher::Decryption cbcDecryption(aesDecryption, (byte*)key.c_str());

    CryptoPP::StreamTransformationFilter stfDecryptor(cbcDecryption, new CryptoPP::StringSink(decryptedText));
    stfDecryptor.Put(reinterpret_cast<const unsigned char*>(cipherText.c_str()), cipherText.size());
    stfDecryptor.MessageEnd();

    return decryptedText;
}
```

Generating key…..



```
 if(choice=="signup"){
  AutoSeededRandomPool prng; // Initialize a random number generator

 // Generate a random key
byte key[AES::DEFAULT_KEYLENGTH];
 prng.GenerateBlock(key, sizeof(key));

 // Convert the key to a string
 HexEncoder encoder(new StringSink(hexkey));
 encoder.Put(key, sizeof(key));
 encoder.MessageEnd();
 cout<<hexkey<<endl;
 createFile(username+"_key.txt",hexkey);
 }
 else if(choice=="login"){
 hexkey=readFile(username+"_key.txt");

 }
```

Server.cpp



```
// Structure to hold user data
struct UserData {
    string username;
    string salt;
    string hashedPassword;
};

// Function to generate a random salt
string GenerateSalt() {
    AutoSeededRandomPool prng;
    CryptoPP::byte salt[16];
    prng.GenerateBlock(salt, sizeof(salt));

    string encodedSalt;
    StringSource(salt, sizeof(salt), true, new HexEncoder(new StringSink(encodedSalt)));

    return encodedSalt;
}

// Function to hash a password with a given salt
string HashPassword(const string& password, const string& salt) {
    string hashed;
    SHA256 hash;
    StringSource(password + salt, true,
        new HashFilter(hash, new HexEncoder(new StringSink(hashed))));

    return hashed;
}
```

References

1. Cryptopp library
2. Crptography and network security principles by William stallings .