FACULTY OF INFORMATICS
COURSEWORK COVERSHEET

SUBJECT'S INFORMATION:			
Subject:	CSCI368 Network Security		
Session:	February 2020		
Programme / Section:	BCS		
Lecturer:	**Mohamad Faizal Alias**		
Coursework Type *(tick appropriate box)*	❑ Individual Assessment		
Coursework Title:	Assessment 2	Coursework Percentage:	14%
Hand-out Date:	Week 6	Received By : (signature)	
Due Date:	Week 12	Received Date :	
STUDENT'S INFORMATION:			
Student's Name & ID:	**TEH WIN SAM** **6306196**		
Contact Number / Email:	**0133696298** **me@tehwinsam.com**		
STUDENT'S DECLARATION			
By signing this, I / We declare that: 1. This assignment meets all the requirements for the subject as detailed in the relevant Subject Outline, which I/ we have read. 2. It is my / our own work and I / we did not collaborate with or copy from others. 3. I / we have read and understand my responsibilities under the University of Wollongong's policy on plagiarism. 4. I / we have not plagiarised from published work (including the internet). Where I have used the work from others, I / we have referenced it in the text and provided a reference list at the end of the assignment. I am / we are aware that late submission without an authorised extension from the subject co-ordinator may incur a penalty. *(See your subject outline for further information).*			
Name & Signature:	TEH WIN SAM		

----------✂ ----------✂ ----------✂ ----------✂ ----------✂ ----------✂ --

COURSEWORK SUBMISSION RECEIPT

Subject:	CSCI368 Network Security	Session:	February 2020
Programme / Section:	BCS	Lecturer:	**Mohamad Faizal Alias**
Coursework Type: (Tick appropriate box)	❑ Individual Assessment		
Coursework Title:	Assessment 2	Coursework Percentage:	14%
Hand-out Date:	Week 6	Received By: (Signature)	
Due date:	Week 12	Received Date:	
STUDENT'S INFORMATION:			
Student's Name & ID:	Tehwinsam 6306196		
Contact Number / Email:	0133696298 me@tehwinsam.com		

Assessment Criteria		**Total Marks**	**Given Marks**
1.	**Part 1:** Online Quiz 3 – PKI (Must be completed within **Week 6**)	2	
2.	**Part 2: Secret File Server System (Submission by Week 12)**	8	
	1. Authentication Server code 2. Client and creating account code and KeyGen 3. IDEA encryption and OFB Mode code 4. FTP file server, connection code, Upload and Download 5. Report **Note**: Presentation on Week 14 required	1. /2 2. /2 3. /2 4. /3 5. /1	
		10	
		Penalty	
Marked by: _____ Date: _____		**Final Mark (10 %)**	

Lecturer's Comments

Penalty for late submission:

1 day – minus 20% of total mark awarded
2 days – minus 50% of total mark awarded
3 days – 0 mark for this piece of coursework

University of Wollongong

CSCI368 NETWORK SECURITY
February 2020
Individual Assessment 2 (10 %)

Aims

This assignment consists of **TWO** parts. Part 1 consists of an Online Quiz 3 covering topic of Public Key Infrastructure.
Where by Part 2 is a program development for Secret File Server system

Objectives

On completion of this assignment you should be able to:

- Understand Cloud Computing and Cloud Security
- Understand the Public Key Infrastructure covering Key Management, Digital Certificates and Key Distribution
- Applying network programming.
- Applying Cryptographic techniques such as Key Generator, Hashing, block cipher and secured file transfer (Programming)

Part 1 – Online Quiz 3 – PKI (2%) – Week 6

- Quiz 3 is an online quiz on Moodle which consist of 10 MCQs. You are given only 20 minutes to complete the quiz.
- The coverage for this quiz is on Chp04 – Public Key Infrastructure.
- Make sure you have done necessary reading before attempting this quiz.
- Quiz will be open on Week 6 Monday and close by Week 5 Sunday.

Part 2 – Secret File Server System

Aims

This Part 2 assignment aims to establish a basic familiarity with an authentication system, encryption of block ciphers and file transfer with confidentiality.

Part 2 Objectives

The assignment involves the following tasks:

- Implementation of a working prototype of secret file server system.
- Socket programming.
- Security programming including block ciphers, hashed password file and authentication server

On completion of this assignment you should be able to:

- Understand advanced authentication systems.
- Understand block cipher and hashing during authentication and file transfer
- Write security code for communications in computer networks.
- Write socket code.

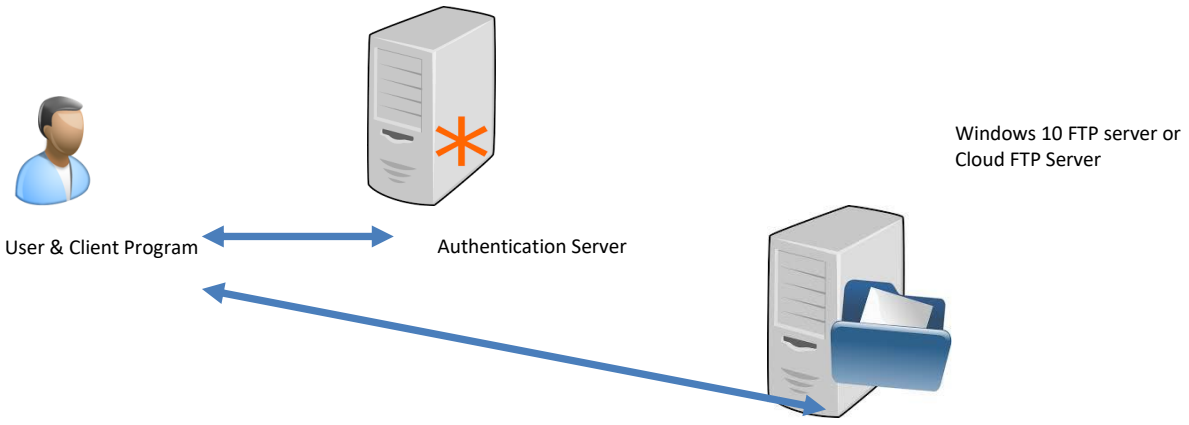
Specifications

Write a C++ socket program to implement a working prototype of secret file server system
Configuration of either internal or external (cloud service) FTP server
Use of a new library: ChilKat FTP Reference (library). Refer notes at the end of the specification.

SECRET FILE SERVER SYSTEM

Planning:

Client Program	**Authentication Server**	**FTP File Server**
Spec. Own program With IDEA Key 1 Crypto++ library C++ FTP library Reverse IDEA Key 2 to make IDEA Key 3 to be used during upload/download to FTP server.	**Spec.** Own program with IDEA Key 1 Store: User Account with IDEA Key 2 Crypto++ library C++ FTP library	**Spec.** Windows 10 FTP Server configuration (minimum) Alternatively, cloud FTP server



Requirements:

A. Execution and Network

1. Windows 10 FTP server is a configure server with IP and FTP service. It should be readily starts and with a fixed account and password (Single Administrator account). Alternatively, you may create account on free FTP cloud service, but read carefully the policy for free account. (Example Hostinger.my or any other FTP service)
2. Authentication server (AS) is your own program that handles user account management and authentication process during client program logon.
3. AS stores, all user accounts, Hash of the user's password in a secured file and randomly generated Key (in user account file).
4. The user account file is a standard text file but all user accounts, hash of password, key and date register are encrypted using **IDEA encryption with OFB Mode. The key (IDEA Key1) is hardcoded in the AS program itself.**
5. During startup of AS, it has its own IP address and opened port number (entered by the admin during AS execution) for client program to connect.
6. During this time, entry for IP address of FTP server, server name (if required), admin ID and password must be entered.
7. This information later will be used by AS to pass securely to the client program over the network.

B. Account Creation

1. Using the Client program, a user can register his/her account and decide a password.
2. The password should be between 8 to 12 characters long.
3. During account creation, AS also randomly generate 16 characters long key (**IDEA Key 2**). This Key is uniquely assigned to each user.
4. The user account, hash of password **(SHA-1 hashing)**, **IDEA Key 2**, registered date is securely stored by AS in a file mentioned above (in A.3).
5. With the client program, a user also can change his/her own password at later time. But the **IDEA Key 2** remain the same.
6. User account only **can be use after 24hours from the date created** for FTP operation. Use account created date to verify.
7. AS program will also update the random **IDEA Key 2** assigned to each user's account every 2 months, based on registered date stored.

C. Authentication

1. Using the client program, any user can request to connect to the FTP server via AS.
2. AS is required to authenticate the user login based on the user account file already exists.
3. The password entered by the user should be hash **(SHA-1)** and compared both user ID and hash of the password with the user account file already exists.
4. Upon authenticated by the AS, the **client program should receive the FTP server detail and IDEA Key 2 (uniquely assigned to the user) encrypted using OFB IDEA encryption with IDEA Key 1.**
5. The **IDEA key (IDEA Key 1)** used for both client program and AS can be hard-coded in both programs.

D. FTP file server setup (No programming requried)

1. FTP server is configured in a standard Windows 10 PC having one admin account and password. (Alternative, Cloud FTP server)
2. All FTP upload and download is managed by only this account. Possibly one folder.
3. The information regarding FTP server need to be entered during AS startup as mentioned in A.6 above.
4. FTP server handle upload and download of file between client program and itself.
5. **The file received by FTP server is already a secured file. Encrypted and decrypted by the client program.**
6. The client program is using **IDEA encryption with OFB** to encrypt and decrypt the file before it is uploaded to the FTP server.
7. The key used by the IDEA encryption (**IDEA Key 3**) is reverse character of **IDEA Key 2** received from AS during the login process and after authentication satisfied.
8. As mentioned, the encryption and decryption of file is handled by client program and using **IDEA Key 3** (the reverse of IDEA Key2 stored in AS and received during authentication between client and AS).
9. During file download, the client program should decrypt the file received from FTP server using similar process in D.7 and D.8 above.
10. File should use the same extension before and after encryption/decryption.

E. Secure client upload/download to/from FTP server

1. After authentication of user account via client program. Communication is established between client program and FTP server directly without intervention from AS.
2. Secured file upload/download process can be initiated by the user via client program. Refer D.6 to D.10 above.
3. Client program in charge of encryption and decryption of the file to create the original file with appropriate extension.
4. For our program simulation and testing, ensure that file of ***.docx, *.png, *.jpg, *.txt and *.pdf** are encrypted and decrypted correctly. These should be reported.

References:

1. C++ File Upload: https://www.example-code.com/vcpp/ftp_upload.asp
2. C++ File Download: https://www.example-code.com/cpp/ftp_download.asp
3. ChilKat FTP Reference (library): https://www.chilkatsoft.com/refdoc/vcCkFtp2Ref.html
   Warning: ChilKat does not give away the library for free anymore. You have limited time of using it. Please refer to his website for latest update.
4. How to setup FTP with Windows 10: https://www.windowscentral.com/how-set-and-manage-ftp-server-windows-10

Note:

- You may use any other C++ library deemed suitable for FTP apart from the above suggestion. But FTP server remains on Windows 10 PC or Cloud FTP service.
- If you would like to host the FTP server on other platform, please get permission from your lecturer.

Files to be submitted:

- All source code
- readme (text file)
- A report in pdf format containing some screen captures about the whole execution, and explanation of the working system.
- Complete test results of different file extension during uploading and downloading (encrypt/decrypt)

Generate a zip file named <yourname><assign2>.zip that includes all the above files to be submitted. Put your name and student number in all source codes.

Submission

This assignment should be submitted electronically via the assignment submission link on Moodle:
Comments in code files should be concise. A header should give your information (including name, student ID) and briefly summarize the contents of the file - identifying purpose of program, listing classes etc.
Classes may have brief header comments if these are considered necessary.
Individual functions should only require comment if they are complicated or result in non-obvious side effects etc.

The code that does not compile or/and failure of client/server connection will receive a zero.
Note: A presentation session will be prepared for you to show case your system in Week 14

Late Submission:

Penalty is 25% deduction per day.

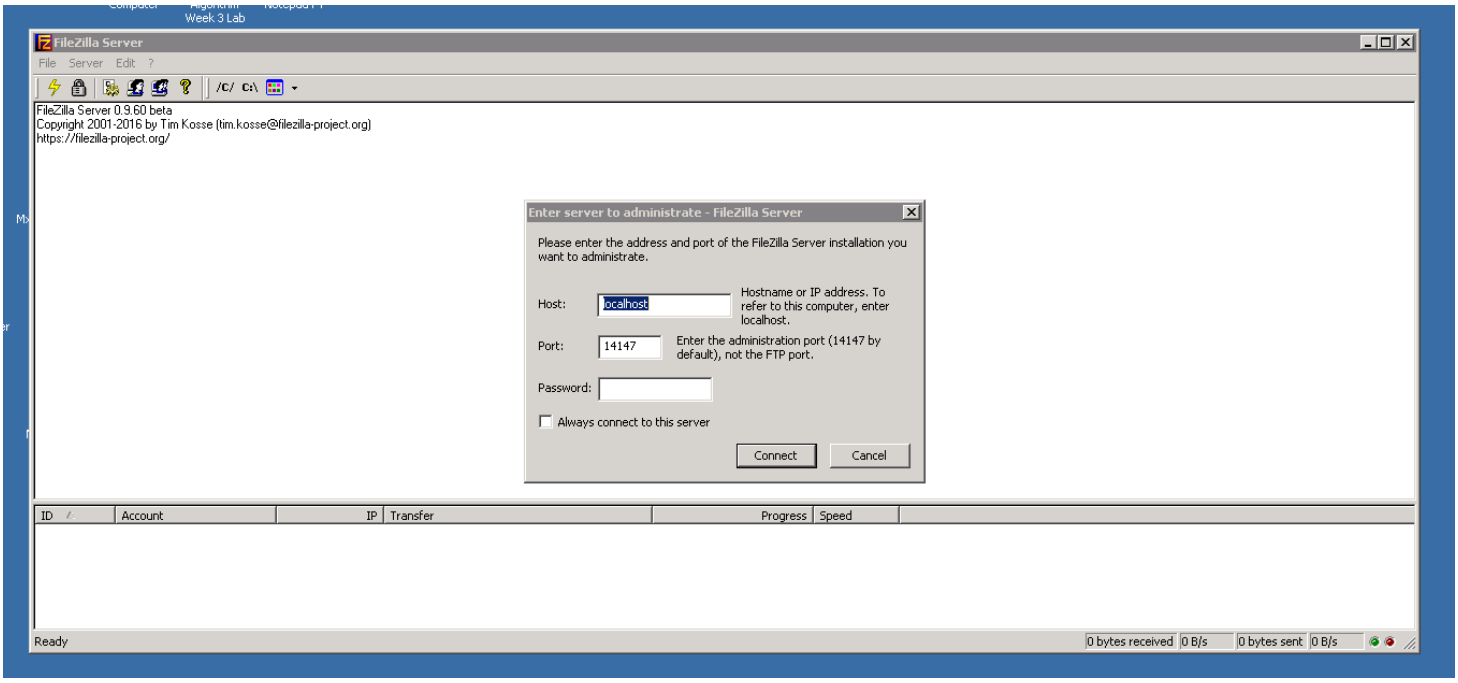
Plagiarism

A plagiarised assignment will receive a zero mark and be penalised according to the university rules. Plagiarism detection software will be used.

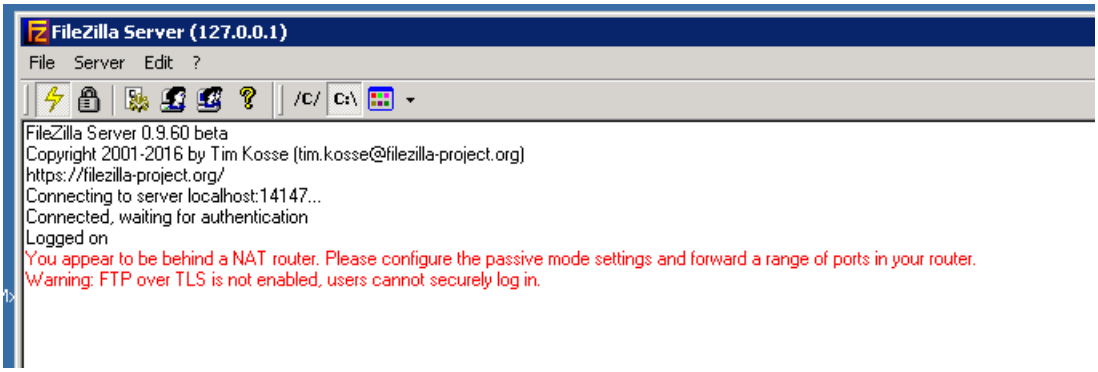
Table of Content

FileZilla Server Installation guide

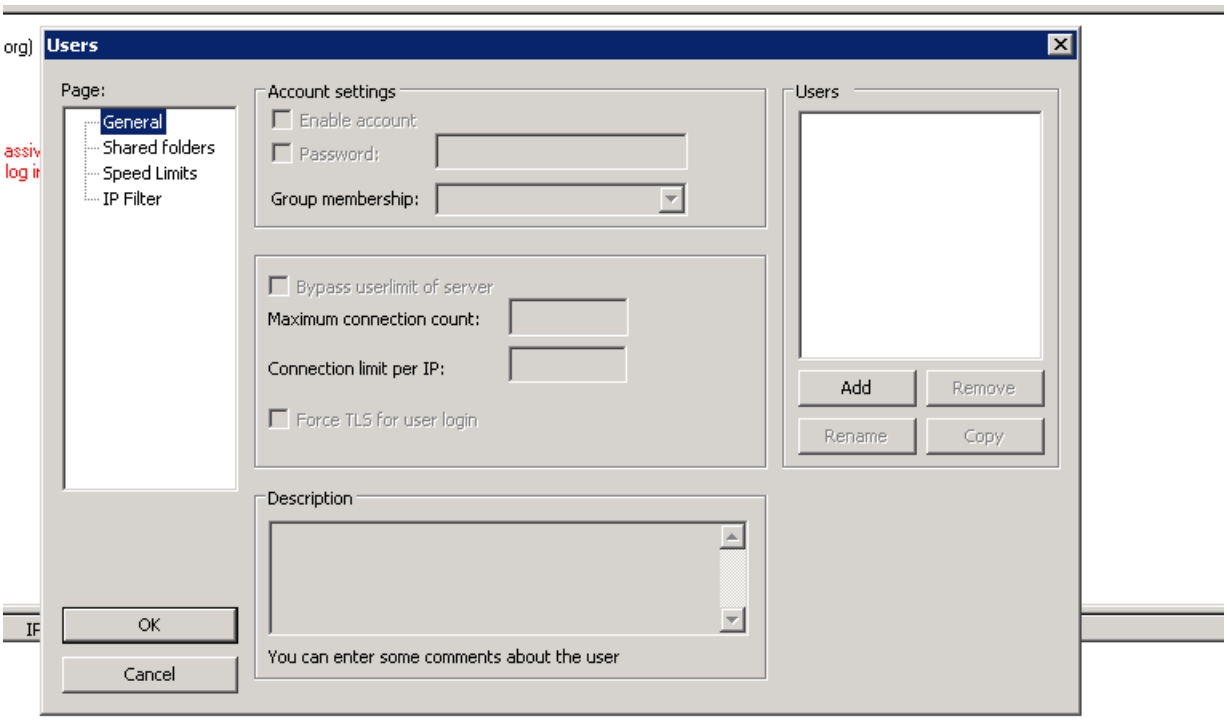
https://dl2.cdn.filezilla-project.org/server/FileZilla_Server-0_9_60_2.exe?h=8w_ImcR5NzTshjMgk8xong&x=1585833384

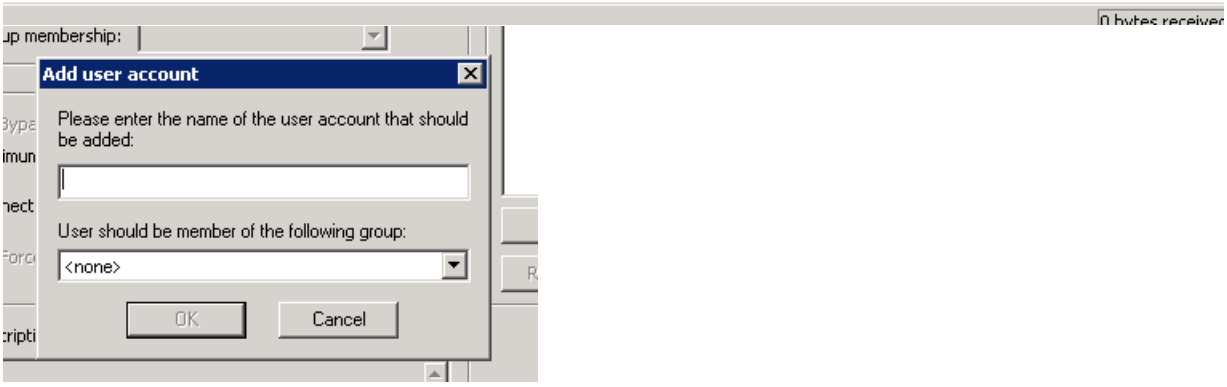


After we have completed the installation, the FileZilla Server interface should look like this.

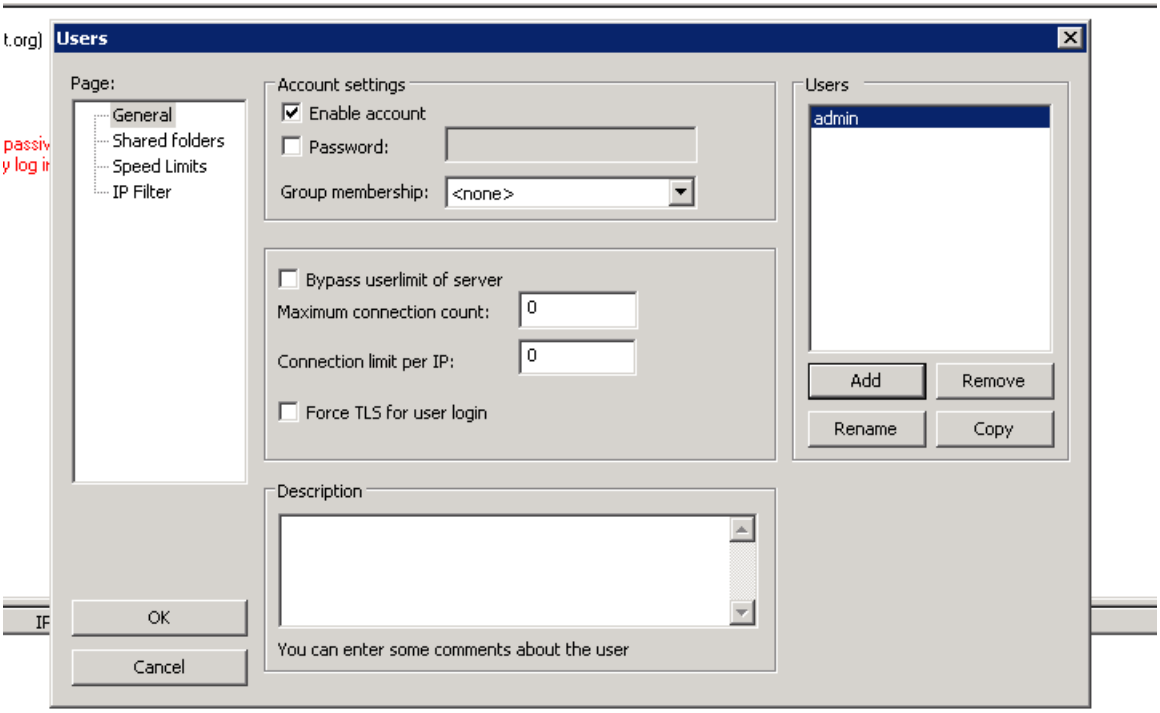


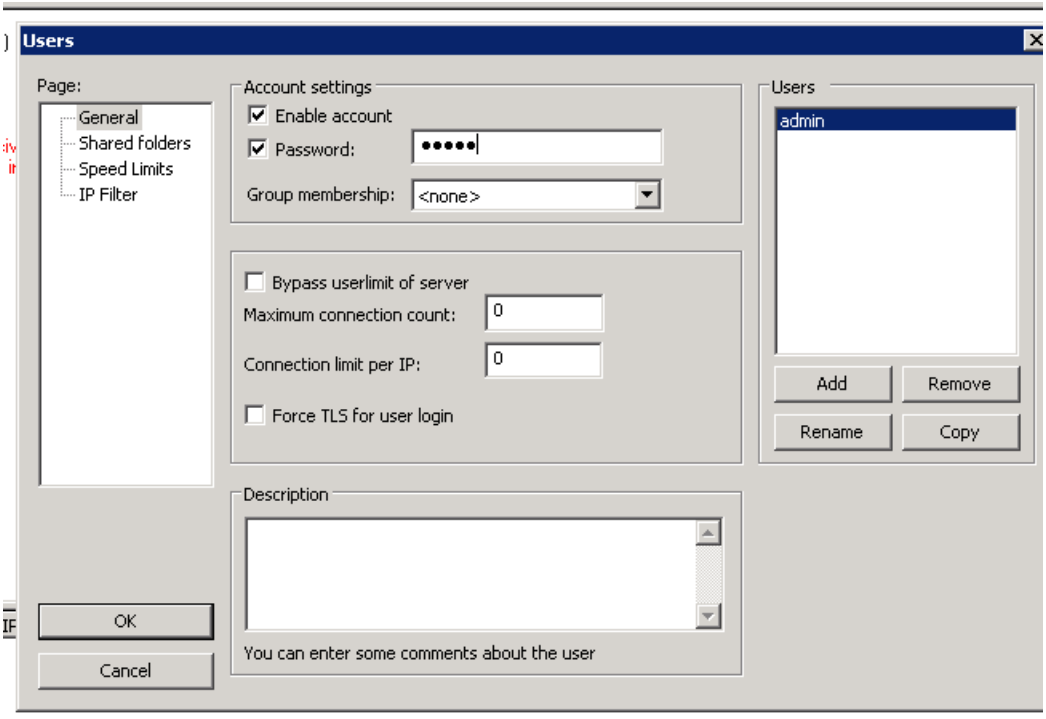
We navigate to *display the user account dialog*, which located at 4th icon starting from left handside.





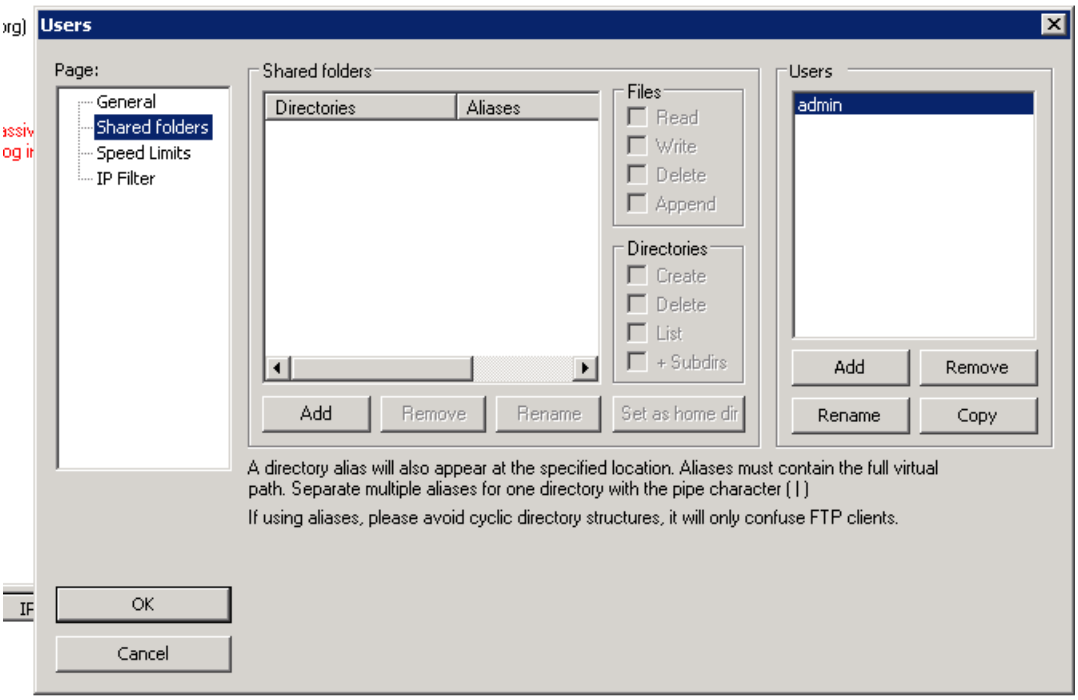
We will start create an FTP account, the ID for the FTP is *admin*



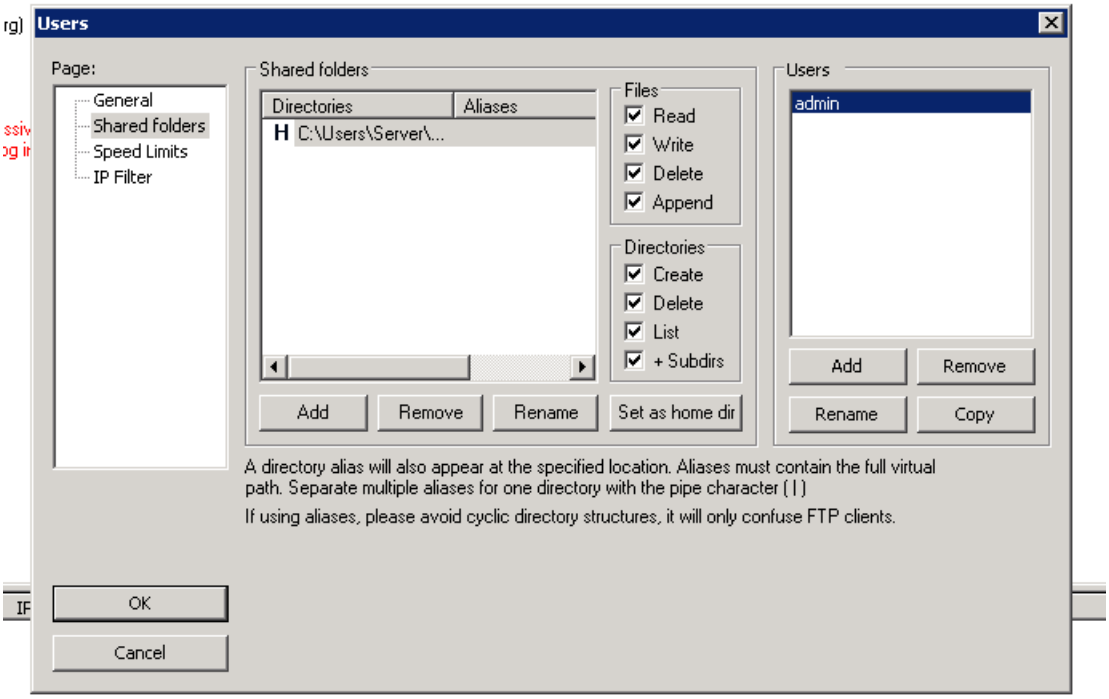


We will proceed to password setting for that account too, I set the password as *admin*

Now have successfully created an account with *admin:admin* credential

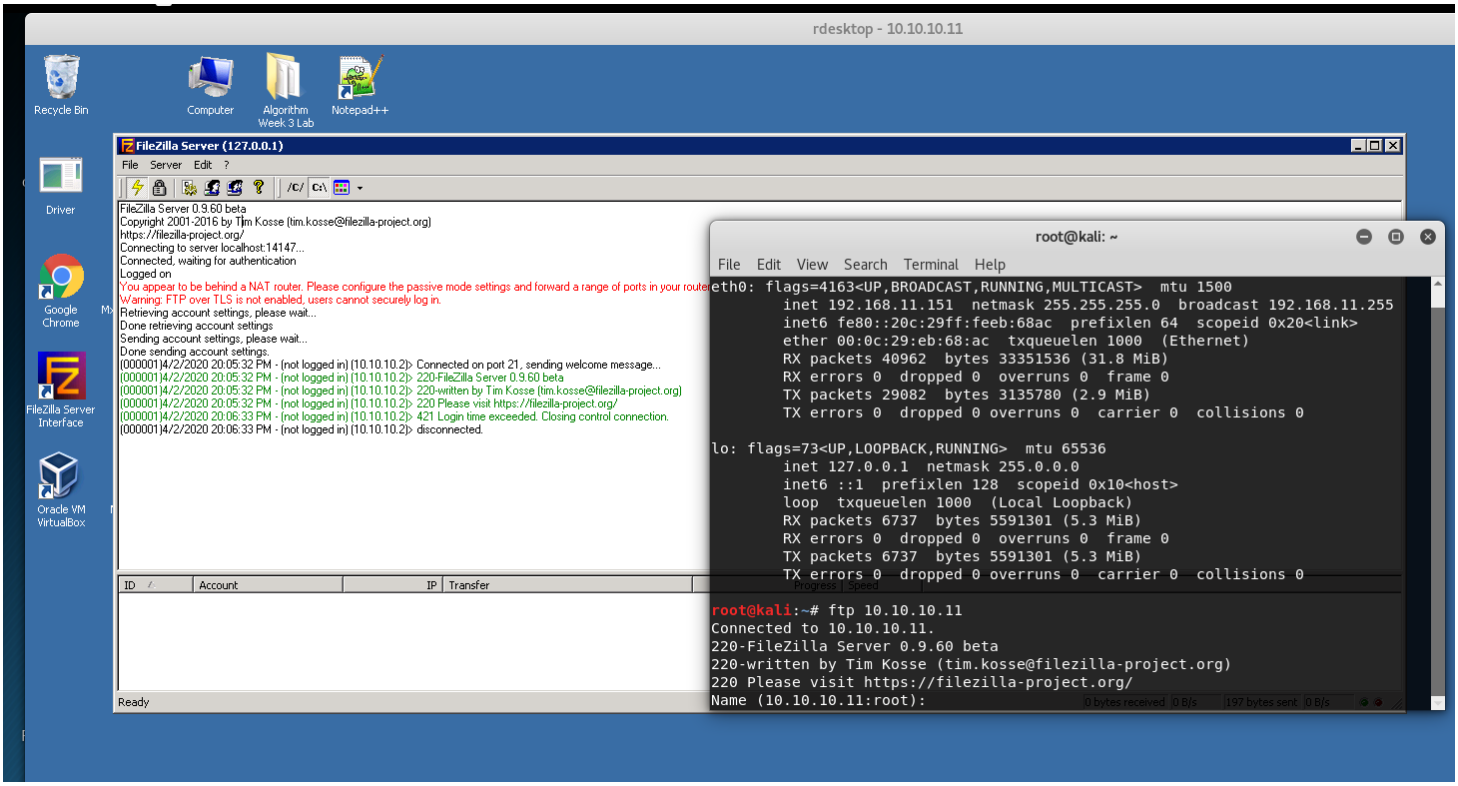


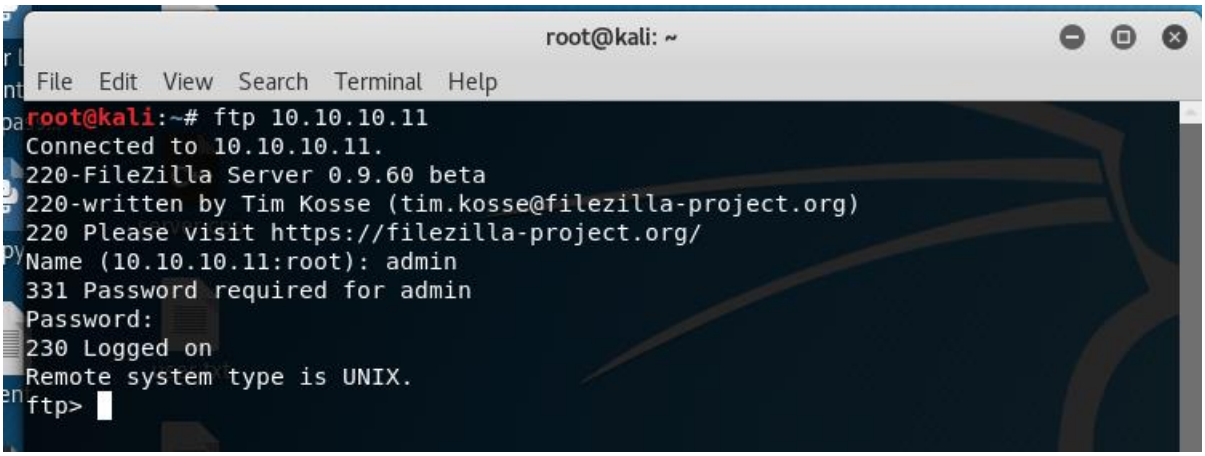
We have to specify the directory for ‘admin’ when any user are logged in as admin they will able to “read and write” on that directory



Great, we have successfully completed the FTP Server installation.

Great, let test it out.





Client

The concept for this assignment2 is that from the Client Side, all process have to verified by Authentication Server before proceed to upload and download file from the FTP Server.
Client will required to logged into Client Program, and the credential will be send to 'Server' and Server will do the authentication process from the user.txt file.
Firstly, purely look at Client source first

Client's *libraries*

```
//////////////INCLUDE COLOUR CODE//////////////////
#ifndef _COLORS_
#define _COLORS_
#define RST  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define FRED(x) KRED x RST
#define FGRN(x) KGRN x RST
#define FYEL(x) KYEL x RST
#define FBLU(x) KBLU x RST
#define FMAG(x) KMAG x RST
#define FCYN(x) KCYN x RST
#define FWHT(x) KWHT x RST
#define BOLD(x) "\x1B[1m" x RST
#define UNDL(x) "\x1B[4m" x RST
#endif  /* _COLORS_ */
//////////////END OF INCLUDE COLOR CODE////////////////////
#include "chilkat/include/CkFtp2.h"



#include <sys/socket.h>
#include <arpa/inet.h>
#include <iostream>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <fstream>

#include "cryptopp/idea.h"
using CryptoPP::IDEA;

#include "cryptopp/modes.h"
using CryptoPP::CBC_Mode;

#include "cryptopp/secblock.h"
using CryptoPP::SecByteBlock;

#include "cryptopp/hex.h"
#include "cryptopp/sha.h"
#include "cryptopp/osrng.h"

using namespace std;
using namespace CryptoPP;
string encrypt_conversation(string plain,string hard_keys,string hard_IVs);
string decrypt_conversation(string cipher,string hard_keys, string hard_IVs);
CkFtp2 ftp;
////////////////////////////////////////////
```

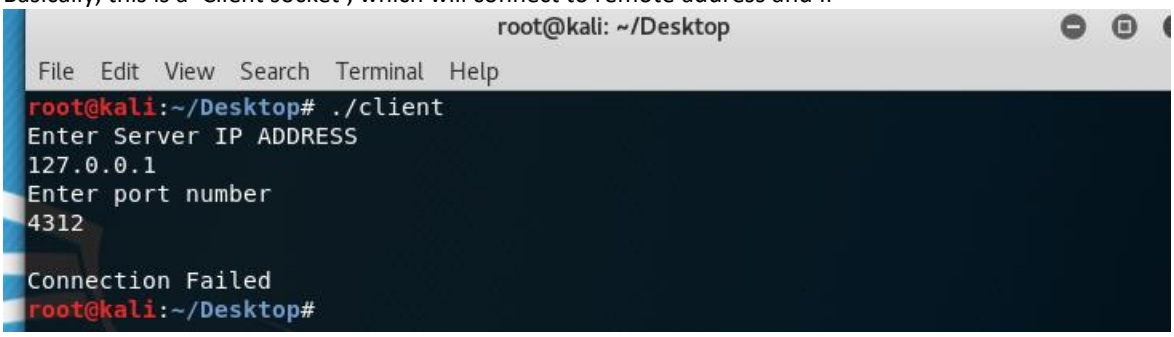
Client's *global variables with hardcoded values, function declarations and namespaces*

```
//////////////////////////////////////////
string IV_from_file,Key_from_file,ftp_host,ftp_name,ftp_password;
string hard_keys="65FD379515B4410BAA2EACAA1C0E865D";
string hard_IVs="30D37831B2FF9063";

string encrypt_conversation(string plain,string hard_keys,string hard_IVs);
string decrypt_conversation(string cipher,string hard_keys, string hard_IVs);
void menu(int sock);
```

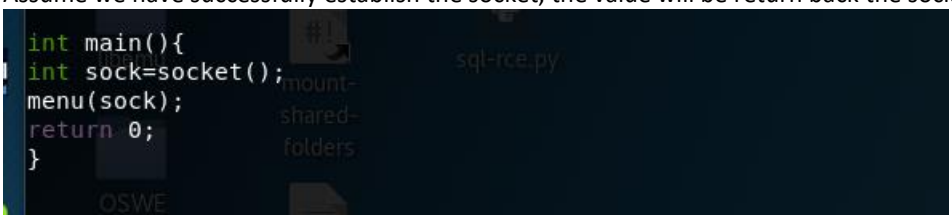
Client's *Socket()*

```
int socket()
{
    cout<<"Enter Server IP ADDRESS"<<endl;
    string ip;
    cin>>ip;
    int PORT;
    do{
    cout<<"Enter port number"<<endl;
    cin>>PORT;
    if(PORT > 65535 || PORT <1)
    {
    cout<<"are you dumb ? the port range is \"0 - 65535\" "<<endl;
     }
    }while(PORT > 65535 || PORT < 1);
    int sock = 0, valread;
    struct sockaddr_in serv_addr;
    if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
    {
        printf("\n Socket creation error \n");
    exit(0);
        return -1;
    }
    serv_addr.sin_family = AF_INET;
    serv_addr.sin_port = htons(PORT);

    // Convert IPv4 and IPv6 addresses from text to binary form
    if(inet_pton(AF_INET, ip.c_str(), &serv_addr.sin_addr)<=0)
    {
        printf("\nInvalid address/ Address not supported \n");
    exit(0);
        return -1;
    }

    if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
    {
        printf("\nConnection Failed \n");
    exit(0);
        return -1;
    }
    return sock;
}
```

Basically, this is a ‘Client socket’, which will connect to remote address and IP



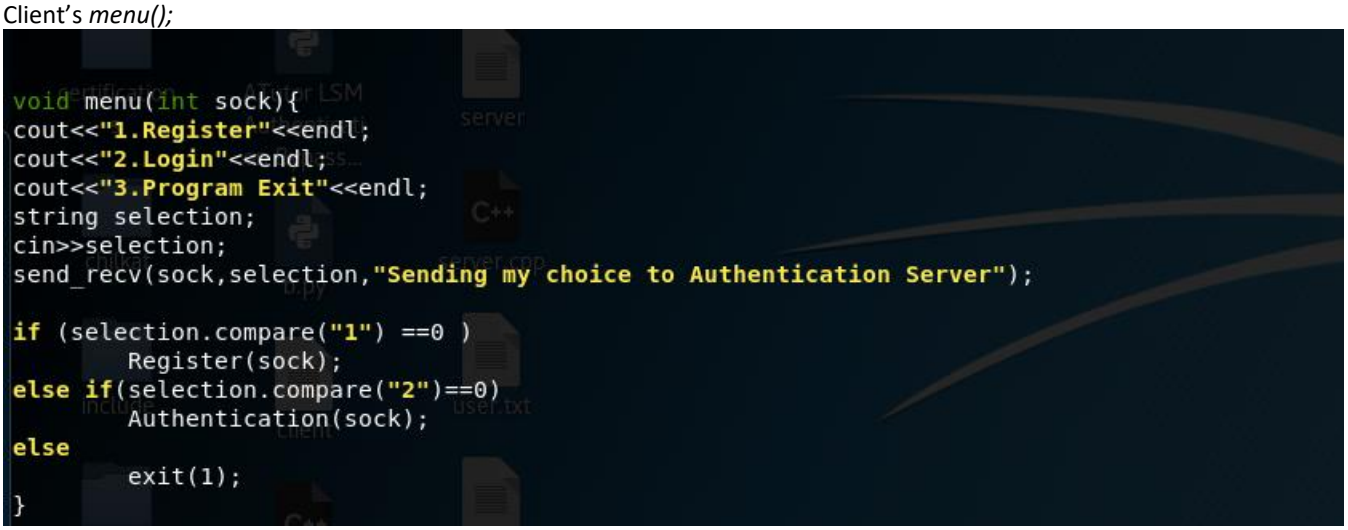
Assume we have successfully establish the socket, the value will be return back the socket value and socket descriptor.



```
int main(){
int sock=socket();
menu(sock);
return 0;
}
```

Which will proceed to *menu();*

Client’s *menu();*



```
void menu(int sock){
cout<<"1.Register"<<endl;
cout<<"2.Login"<<endl;
cout<<"3.Program Exit"<<endl;
string selection;
cin>>selection;
send_recv(sock,selection,"Sending my choice to Authentication Server");

if (selection.compare("1") ==0 )
        Register(sock);
else if(selection.compare("2")==0)
        Authentication(sock);
else
        exit(1);
}
```

The menu have only 3 choices, the user can decide to *Register* themselves a new account , *Login* into the Client program and *Quit.*

We look at registration process.

Client's *register();*

```
////////////////////////////////REGISTER//////////////////
void Register(int socket){
cout<<"Register"<<endl;
string user,password;
cout<<"Enter user name"<<endl;
cin>>user;
cout<<"Enter Password"<<endl;
cin>>password;
while(password.length() < 8  || password.length() > 12 )
{
cout<<"Password requirement length 8 - 12"<<endl;
cin>>password;
}
password=sha1string(password);
cout<<password<<endl;
send_recv(socket,user,"Sending UserID to Authentication Server");
send_recv(socket,password,"Sending Hash to Authentication Server");
//////////time stamp////////////
time_t current_time;
current_time = time(NULL);
int timestamp=current_time/60/60/24;
////////////////////////////

stringstream ss;
ss<<timestamp;
string s;
ss>>s;
//////////////////////////

string response=send_recv(socket,s,"Sending TimeStamp as representative to ask Server, everything received and waiting for authentication result?");

if (response.compare("Failed")==0)
{
cout<<"USED ID is USED by other user"<<endl;
}
menu(socket);
}
```

We will first prompt the user to enter their desire USER ID and PASSWORD, with password policy.
Next we take the password run into some algorithm into a unique 40lengths long digest, the algorithm called SHA-1.
We send the user and SHA-1 password to Server. Later we will look at *send_recv()* that does the sending and receiving message.

```
time_t current_time;
current_time = time(NULL);
int timestamp=current_time/60/60/24;
////////////////////////////
stringstream ss;
ss<<timestamp;
string s;
ss>>s;
```

Take note on this *time_t current_time* until *ss>s;*
This code is to generate a number that represent day, meaning `int timestamp=current_time/60/60/24;` the value for timestamp should be around 10485 for today, so the next day the same algorithm `int timestamp=current_time/60/60/24;` we will get the value of 10486, the stringstream is just a converter from *INT* to *STRING.*
After we successfully send the USER ID and PASSWORD to Server, client will send a timestamp to Server. This value have no meaning just *sending a dummy message* because we want to received the '*Response from Server, either the registration is failed or success'.*

```
string response=send_recv(socket,s,"Sending TimeStamp as representative to ask Server, everything received and waiting for authentication result?");

if (response.compare("Failed")==0)
{
cout<<"USED ID is USED by other user"<<endl;
}
menu(socket);
}
```

*If the response we received is **failed**, it will display "USED ID is USED by other user". If we received "Other type of messages" it will prompt nothing.* No matter what result, it will always go back the menu page.
We now look at send and recv function

Client's *send_recv()*

```
//////////////////////SEND_RECV_/////////////////////

string send_recv(int socket, string message, string comments)
{
int valread;
char buffer[1024] = {0};
string encrypted=encrypt_conversation(message.c_str(),hard_keys,hard_IVs);
send(socket , encrypted.c_str() , strlen(encrypted.c_str())+1 , 0 );
cout<<"[ Successfully send to Server ] "<<comments<<endl;
valread = read(socket , buffer, 1024);
cout<<"[ Encrypted message from Authentication Server ] : ";
printf("%s\n",buffer );
encrypted=buffer;
string recovered=decrypt_conversation(encrypted,hard_keys,hard_IVs);
cout<<"[ Recovered message from Authentication Server ] : "<<recovered<<endl;
return recovered;
}
```

Mainly this function received 3 parameter
1st parameter = Socket value/Socket descriptor
2nd parameter = Plaintext Messages
3rd parameter = Comments || Just inform Client/User what is this about, what content are we sending
It first go into *encrypt_conversation(),* We will come back to this encrypt_conversation in later while.

We will be using *send()* which is a *pre-loaded function*, which will send the encryptedmessage to Server with the socket value that we have. Next, the *read()* is a pre-loaded function which will receive any 'incoming message that sent from Server'. We will received 'Encrypted Message' as our incoming message which we will pass the Encrypted Content into *decrypt_conversation().* We will dive this function in a minute.
Lastly, we will display the "Decrypted Content" and we will *'return the decrypted text'* back to *menu()*

Client's *encrypt_conversation()*

```
//////////////////////ENCRYPT_ IDEA_KEY1///////////////

string encrypt_conversation(string plain,string hard_keys,string hard_IVs){

string keys,IVs;
StringSource ss(hard_keys,true,new HexDecoder(new StringSink(keys)));
StringSource sss(hard_IVs,true,new HexDecoder(new StringSink(IVs)));

AutoSeededRandomPool prng;

SecByteBlock key((const byte*)keys.data(), keys.size());
SecByteBlock iv((const byte*)IVs.data(), IVs.size());

string encoded,cipher;
try
{
//    cout << "Plain text: " << plain << endl;

    OFB_Mode< IDEA >::Encryption e;
    e.SetKeyWithIV(key, key.size(), iv);

    // The StreamTransformationFilter adds padding
    //  as required. ECB and CBC Mode must be padded
    //  to the block size of the cipher.
    StringSource(plain, true,
        new StreamTransformationFilter(e,
            new StringSink(cipher)
        ) // StreamTransformationFilter
```

```
        ); // StringSource
    }
    catch(const CryptoPP::Exception& e)
    {
        cerr << e.what() << endl;
        exit(1);
    }

    /*********************************\
    \*********************************/

    // Pretty print
    StringSource(cipher, true,
        new HexEncoder(
            new StringSink(encoded)
        ) // HexEncoder
    ); // StringSource

    //cout << "Cipher text: " << encoded << endl;
    return encoded;
    }
```

Code above is the Client's *encrypt_conversation();*

The 2nd and 3rd parameters is our 'HARDCODED VALUE', which is KEY and IV.

We will take these keys and IV run a HexDecoder into a 'raw', we now will load the Key and IV. Then will do the encryption process and return us the 'Encrypted Content'.

Client's *Decryption function()*

```
//////////////////////DECRYPT CONVERSATION////////////////////////////

string decrypt_conversation(string cipher,string hard_keys,string hard_IVs){
string recovered;
string keys,IVs;

StringSource ss(hard_keys,true,new HexDecoder(new StringSink(keys)));
StringSource sss(hard_IVs,true,new HexDecoder(new StringSink(IVs)));

AutoSeededRandomPool prng;

SecByteBlock key((const byte*)keys.data(), keys.size());
SecByteBlock iv((const byte*)IVs.data(), IVs.size());
string rawcipher;
try
{
        StringSource ss2(cipher, true,
        new HexDecoder(
                new StringSink(rawcipher)
            ) // HexEncoder
        ); // StringSource


    OFB_Mode< IDEA >::Decryption d;
    d.SetKeyWithIV(key, key.size(), iv);

    StringSource s(rawcipher, true,
        new StreamTransformationFilter(d,
            new StringSink(recovered)
        ) // StreamTransformationFilter
    ); // StringSource
}

catch(const CryptoPP::Exception& e)
{
    cerr << e.what() << endl;
    exit(1);
}
return recovered;
}
```

Code above is the Client's *decrypt_conversation();*

The 2nd and 3rd parameters is our 'HARDCODED VALUE', which is KEY and IV.

We will take these keys and IV run a HexDecoder into a 'raw', we now will load the Key and IV. Then will do the decryption process and return us the 'Decrypted Content'.

After Registration process

```
string response=send_recv(socket,s,"Sending TimeStamp as representative to ask Server, everything received and waiting for authentication result?");

if (response.compare("Failed")==0)
{
cout<<"USED ID is USED by other user"<<endl;
}
menu(socket);
}
```

We now back to *menu()*

```
//////////////////MENU/////////////////


void menu(int sock){
cout<<"1.Register"<<endl;
cout<<"2.Login"<<endl;
cout<<"3.Program Exit"<<endl;
string selection;
cin>>selection;
send_recv(sock,selection,"Sending my choice to Authentication Server");

if (selection.compare("1") ==0 )
        Register(sock);
else if(selection.compare("2")==0)
        Authentication(sock);
else
        exit(1);
}
```

We now will be explain “Login process”, *Authentication()*

```
    //////////////////////AUTHENTICATION//////////////////

    void Authentication(int socket){
    cout<<"Login"<<endl;
    string user,password;
    cout<<"Enter your ID"<<endl;
    cin>>user;
    cout<<"Enter your password"<<endl;
    cin>>password;
    password=sha1string(password);
    cout<<password<<endl;
    send_recv(socket,user,"Sending UserID to Authentication Server");
    send_recv(socket,password,"Sending Hash to Authentication Server");
    ///////////time stamp//////////
    time_t current_time;
    current_time = time(NULL);
    int timestamp=current_time/60/60/24;
    ////////////////////////////////

    stringstream ss;
    ss<<timestamp;
    string s;
    ss>>s;
    ////////////////////////////
    string response=send_recv(socket,s,"Sending TimeStamp as representative to ask Server,
everything received and waiting for authentication result?");
    if (verify(response,"Verified") == true)
    {
```

```
    char *hello = "Received Wink > .. < ";
    ftp_host=send_recv(socket,hello,"Asking for FTP HOSTNAME");
    ftp_name=send_recv(socket,hello,"Asking for FTP ID");
    ftp_password=send_recv(socket,hello,"Asking for FTP PASSWORD");
    Key_from_file=send_recv(socket,hello,"Asking for IDEA KEY");
    IV_from_file=send_recv(socket,hello,"Asking for IDEA IV");

    cout<<"FTP CREDENTIAL"<<endl;
    cout<<"FTP HOSTNAME: "<<ftp_host<<endl;
    cout<<"FTP ID: "<<ftp_name<<endl;
    cout<<"FTP PASSWORD: "<<ftp_password<<endl;
    cout<<"IDEA KEY: "<<Key_from_file<<endl;
    cout<<"IDEA IV: "<<IV_from_file<<endl;
            logged_menu(socket);
    }
    else
    {
            menu(socket);
    }}
```

System will prompt us to enter ID and PASSWORD, instead of sending the password we will be send the 40 lengths of digest take generate from SHA1 function.

Client's *sha1string()*

```
//////////////////////SHA1////////////////////////

string sha1string(string haha)
{
string digest="";
CryptoPP::SHA1 sha1;
CryptoPP::StringSource(haha, true, new CryptoPP::HashFilter(sha1, new CryptoPP::HexEncoder(new CryptoPP::StringSink(digest))));
return digest;
}
```

Meaning it will send the USER ID and HASH over network to Server side. Same we will be sending timestamp to server as a dummy message just to ask Server to send 'Verified' or 'Failed'. If it's verified, meaning we're authenticate else failed to log in.

Client's *verify()*

```
////////////////////////////VERIFY//////////////////

bool verify(string a, string b)
{

int result = strcmp(a.c_str(), b.c_str());
cout<<"Hold on, while we're verifying"<<endl;
if(result==0)
{
        cout<<FRED(BOLD("Matched"))<<endl;
        return true;
}
else
{
        cout<<FRED(BOLD("Failed"))<<endl;
        return false;
}
}
```

```
if (verify(response,"Verified") == true)
{
char *hello = "Received Wink > .. < ";
ftp_host=send_recv(socket,hello,"Asking for FTP HOSTNAME");
<snip>
…
…
</snip>
        logged_menu(socket);
}
else
{
        menu(socket);
}}
```

If **Failed to login**, system will navigate them to *menu().*

If **Log in successfully**, Client will ask same data from Server, data which include FTP IP, ID ,PASSOWRD and more.

```
ftp_host=send_recv(socket,hello,"Asking for FTP HOSTNAME");
ftp_name=send_recv(socket,hello,"Asking for FTP ID");
ftp_password=send_recv(socket,hello,"Asking for FTP PASSWORD");
Key_from_file=send_recv(socket,hello,"Asking for IDEA KEY");
IV_from_file=send_recv(socket,hello,"Asking for IDEA IV");
```

After we received all the data that we require, Client will display all contents that Client received and navigate to *logged_menu()*

Client's *logged_menu()*

```
void logged_menu(int sock)
{
string selection;
cout<<"1.Change Password"<<endl;
cout<<"2.Upload File"<<endl;
cout<<"3.Download File"<<endl;
cout<<"4.Log out"<<endl;
cin >> selection;
if(selection.compare("1")==0)
{
send_recv(sock,selection,"Sending Choice 1");
        changepassword(sock);
        logged_menu(sock);

}
else if(selection.compare("2")==0)
{
char* ask="Ask";
send_recv(sock,selection,"Sending Choice 2");
string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
if(status.compare("Okay")!=0)
logged_menu(sock);


file_upload();
logged_menu(sock);
}
else if(selection.compare("3")==0)
{
char* ask="Ask";
send_recv(sock,selection,"Sending Choice 3 ");
string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
if(status.compare("Okay")!=0)
logged_menu(sock);


file_download();
logged_menu(sock);
}
else
{send_recv(sock,selection,"Sending Choice 4 ");
menu(sock);
}
}
```

Now, User have few options,

1. Change Password
2. Upload File
3. Download File
4. Log out

Client's *logout function*

We start from Log out() since it's easy to explain

```
void logged_menu(int sock)
{
string selection;
cout<<"1.Change Password"<<endl;
cout<<"2.Upload File"<<endl;
cout<<"3.Download File"<<endl;
cout<<"4.Log out"<<endl;
cin >> selection;
if(selection.compare("1")==0)
{
}
else if(selection.compare("2")==0)
{
}
else if(selection.compare("3")==0)
{
}
else
{
send_recv(sock,selection,"Sending Choice 4 ");
menu(sock);
}
}
```

We will be telling "Server" that we have selected selection '4' in *send_recv()* and we will back to *menu()*

Now we talk about *download file* from FTP Server.

```
send_recv(sock,selection,"Sending Choice 3 ");
string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
if(status.compare("Okay")!=0)
logged_menu(sock);
file_download();
logged_menu(sock);
```

After we selected choice '3', we will be sending our decision to Server. Therefore, server know that we're on choice 3, which server will also know that it's a *Download File function*

But before proceeding to the FTP operation, according to the specification the account have to be registered more than 24 hours only allowed to use the FTP operation.

Therefore, we will send a dummy message to seek approval from Server, if the response from Server is "Okay". Then we will proceed to *file_download()* and after completing the *file_download()*, User will be navigate back the *logged_menu()*. Also, if the response from Server is **NOT "Okay"**, it will also navigate to *logged_menu()*.

Let view File Download

Client's *file_download()*

```
void file_download(){
// Connect and login to the FTP server.
    bool success = ftp.Connect();
    success = ftp.UnlockComponent("Anything for 30-day trial");
 if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
        ftp.put_Hostname(ftp_host.c_str());
        ftp.put_Username(ftp_name.c_str());
        ftp.put_Password(ftp_password.c_str());

    success = ftp.Connect();
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }

 //  Change to the remote directory where the file will be downloaded
    success = ftp.ChangeRemoteDir("AS");
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
    cout<<"\n\n----------------------------\n\n\n"<<endl;

    // The ListPattern property is our directory listing filter.
    // The default value is "*", which includes everything.
    std::cout << ftp.listPattern() << "\r\n";

    // To get file and sub-directory information, simply
    // loop from 0 to ftp.GetDirCount() - 1
    int i;
    int n;
    n = ftp.GetDirCount();
    if (n < 0) {
        std::cout << ftp.lastErrorText() << "\r\n";
 //      return;
    }

    if (n > 0) {
        for (i = 0; i <= n - 1; i++) {
            // Display the filename
            std::cout << ftp.getFilename(i) << "\r\n";
        }

    }
    string filename;
    cout<<"\n\n\n\n\n----------------------------"<<endl;
    cout<<"Enter the file you want to Download"<<endl;
    cin>>filename;

    const char *remoteFilename = filename.c_str();

    // Download a file.
    success = ftp.GetFile(remoteFilename,remoteFilename);
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        return;
    }

    success = ftp.Disconnect();
```

```
        std::cout << "File Downloaded!" << "\r\n";

            cout<<"Decrypting File"<<endl;

        string keys=keyreverse(Key_from_file);
        string ivs=keyreverse(IV_from_file);
        string data=grabdata(filename);
        string plain=decrypt_conversation(data,keys,ivs);
        cout<<"------------------------------------------------"<<endl;
        cout<<"---------------Decrypted Content----------------"<<endl;
        cout<<"------------------------------------------------"<<endl;
        cout<<"\n\n\n"<<endl;
        cout<<plain<<endl;
        cout<<"\n\n\n"<<endl;
        cout<<"------------------------------------------------"<<endl;
        cout<<"------------------------------------------------"<<endl;
        std::ofstream ofs(filename.c_str(), std::ofstream::trunc);
            ofs << plain;
            ofs.close();
    }
```

Code Explanation for CHIKAT FTP

This portion of code, is something like “Try 30 days free trail”

```
      bool success = ftp.Connect();
      success = ftp.UnlockComponent("Anything for 30-day trial");
    if (success != true) {
            std::cout << ftp.lastErrorText() << "\r\n";
            exit(1);
      }
```

This portion of code, is to assign the FTP IP, ID and PASSWORD

```
            ftp.put_Hostname(ftp_host.c_str());
            ftp.put_Username(ftp_name.c_str());
            ftp.put_Password(ftp_password.c_str());
```

This portion of code, is to changeDirectory from “ROOT Directory” to “AS Directory which located at ROOT directory”
Assume “C:\Users\User\Desktop” is our ROOT directory after running code below the FTP connection is referring to “C:\Users\User\Desktop\AS” directory

```
    //  Change to the remote directory where the file will be downloaded
      success = ftp.ChangeRemoteDir("AS");
      if (success != true) {
            std::cout << ftp.lastErrorText() << "\r\n";
            exit(1);
      }
```

Assume “C:\Users\User\Desktop\AS” is our current FTP directory it will list down all the files and subdirectory in that directory.

```
  cout<<"\n\n----------------------------\n\n\n"<<endl;
  // The ListPattern property is our directory listing filter.
  // The default value is "*", which includes everything.
  std::cout << ftp.listPattern() << "\r\n";

  // To get file and sub-directory information, simply
  // loop from 0 to ftp.GetDirCount() - 1
  int i;
  int n;
  n = ftp.GetDirCount();
  if (n < 0) {
      std::cout << ftp.lastErrorText() << "\r\n";
//      return;
  }

  if (n > 0) {
      for (i = 0; i <= n - 1; i++) {
          // Display the filename
          std::cout << ftp.getFilename(i) << "\r\n";
      }
  }
```

Code below is about downloading the file from FTP Remote Server

```
  string filename;
  cout<<"\n\n\n\n\n----------------------------"<<endl;
  cout<<"Enter the file you want to Download"<<endl;
  cin>>filename;

  const char *remoteFilename = filename.c_str();

  // Download a file.
  success = ftp.GetFile(remoteFilename,remoteFilename);
  if (success != true) {
      std::cout << ftp.lastErrorText() << "\r\n";
      return;
  }
```

FTP disconnect from the FTP Server

```
  success = ftp.Disconnect();
```

After everything went well, we will start to decrypt the downloaded content.

Starting with reversing the IDEA-KEY2 into IDEA-KEY3

```
  string keys=keyreverse(Key_from_file);
  string ivs=keyreverse(IV_from_file);
  string data=grabdata(filename);
  string plain=decrypt_conversation(data,keys,ivs);
```

Client's *keyreverse();*

```
//////////////////////KEY///////////REVERSE////////////

string keyreverse(string plain){
string tmp,reverse;
cout<<"Before Reverse : "<<plain<<endl;
for(int i=0; i <= plain.length();i++)
{
    tmp[i]=plain[plain.length()-i];
     reverse= reverse+tmp[i];
}
cout<<"After Reverse : "<< reverse<<endl;
return reverse;
}
```

Mainly, just 'opposite the string'. Eg "ABCDEFG" into "GFEDCBA"

After we reversed all KEYS and IVS we start to grab the encrypted content from the 'files' that we downloaded a minute ago.

Client's *grabdata()*

```
///////////////GRAB DATA/////////////////


string grabdata(string plain_file){
ifstream file (plain_file);
string totaldata,inputdata;
if (file.is_open())
{
int counter=0;
    while(getline (file,inputdata))
     {
             totaldata=totaldata+inputdata+"\n";
     }
    file.close();
}
cout<<totaldata<<endl;
return totaldata;
}
```

Decrypt_conversation()

I have explained the code earlier, but I will roughly re-summary this function, this function will take "Encrypted Message ,IV and Key" into this function and will run the decryption content and it will *"Display" the decrypted content*. Not only that it will also return the *decrypted content* from the function.

string plain=decrypt_conversation(data,keys,ivs); ***meaning variable plain now hold the decrypted message***

After the decryption was completed, Client will *overwrite the "downloaded file" with the decrypted content*

```
    std::ofstream ofs(filename.c_str(), std::ofstream::trunc);
        ofs << plain;
        ofs.close();
```

Then everything went well, we will back to *menu();*

```
void logged_menu(int sock)
{
string selection;
cout<<"1.Change Password"<<endl;
cout<<"2.Upload File"<<endl;
cout<<"3.Download File"<<endl;
cout<<"4.Log out"<<endl;
cin >> selection;
if(selection.compare("1")==0)
{
send_recv(sock,selection,"Sending Choice 1");
        changepassword(sock);
        logged_menu(sock);

}
else if(selection.compare("2")==0)
{
char* ask="Ask";
send_recv(sock,selection,"Sending Choice 2");
string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
if(status.compare("Okay")!=0)
logged_menu(sock);

file_upload();
logged_menu(sock);
}
else if(selection.compare("3")==0)
{
char* ask="Ask";
send_recv(sock,selection,"Sending Choice 3 ");
string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
if(status.compare("Okay")!=0)
logged_menu(sock);

file_download();
logged_menu(sock);
}
else
{
send_recv(sock,selection,"Sending Choice 4 ");
menu(sock);
}

}
```

We now will discuss uploading a files into a FTP Server

```
else if(selection.compare("2")==0)
     {
     char* ask="Ask";
     send_recv(sock,selection,"Sending Choice 2");
     string status=send_recv(sock,ask,"Asking approval to run FTP Operation");
     if(status.compare("Okay")!=0)
     logged_menu(sock);

     file_upload();
     logged_menu(sock);
     }
```

As I mentioned earlier, specification clearly state those accounts that registered more than 24 hours are permitted enjoy FTP operation.
Therefore, it will send User's selection '2' to server, to tell server that Client are selection choice 2, Next Client will send dummy message to server to ask permission for FTP operation, if the response given by Server is "Okay" we're allowed to run *file_upload()*.

Client's *file_*upload*()*

```
void file_upload(){
    cout<<"\n\n----------------------------\n\n\n"<<endl;
    string filename;
    system("ls | xargs -n 1 basename");
    cout<<"\n\n\n\n\n----------------------------"<<endl;
    cout<<"Enter the file you want to upload"<<endl;
    cin>>filename;
        cout<<"Encrypting File"<<endl;

    string keys=keyreverse(Key_from_file);
    string ivs=keyreverse(IV_from_file);
    string data=grabdata(filename);
        string encrypted_data=encrypt_conversation(data,keys,ivs);
    cout<<"Encrypted Data:"<<encrypted_data<<endl;
    string random_file=savefile(encrypted_data);
// Connect and login to the FTP server.
    bool success = ftp.Connect();
    success = ftp.UnlockComponent("Anything for 30-day trial");

        ftp.put_Hostname(ftp_host.c_str());
        ftp.put_Username(ftp_name.c_str());
        ftp.put_Password(ftp_password.c_str());

    //  Connect and login to the FTP server.
    success = ftp.Connect();
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }

   //  Change to the remote directory where the file will be uploaded.
    success = ftp.ChangeRemoteDir("AS");
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }

   //  Upload a file.
    const char *localFilename = random_file.c_str();
//    const char *remoteFilename = remotefile.c_str();

    success = ftp.PutFile(localFilename,filename.c_str());
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
    success = ftp.Disconnect();
    std::cout << "File Uploaded!" << "\r\n";
    string rmcommand="rm " + random_file;
    system(rmcommand.c_str());

}
```

Code Explanation for CHIKAT FTP

This portion of code is to prompt user to input filename and system will list the subdirectory and files

```
void file_upload(){
    cout<<"\n\n----------------------------\n\n\n"<<endl;
    string filename;
    system("ls | xargs -n 1 basename");
    cout<<"\n\n\n\n\n----------------------------"<<endl;

cout<<"Enter the file you want to upload"<<endl;
    cin>>filename;
        cout<<"Encrypting File"<<endl;
```

This portion of code is to reverse the keys. Eg “ABCDEF” to “FEDCBA” and grab the content of file that user wish to upload.
Next will take the PlainText go into *encrypt_conversation();* which generate our ciphertext.

We mentioned before for *encrypt_conversation*, well this basically accept 3 parameter which are ‘MESSAGES’, ‘KEY’ & ‘IV’ and generate some cipher text and return it back.

Then it will save the output into a ‘temporary random file name.

```
    string keys=keyreverse(Key_from_file);
    string ivs=keyreverse(IV_from_file);
    string data=grabdata(filename);
        string encrypted_data=encrypt_conversation(data,keys,ivs);
    cout<<"Encrypted Data:"<<encrypted_data<<endl;
    string random_file=savefile(encrypted_data);
```

Something unlock 30days free trail and setting for FTP IP,ID & PASSWORD

```
// Connect and login to the FTP server.
    bool success = ftp.Connect();
    success = ftp.UnlockComponent("Anything for 30-day trial");

        ftp.put_Hostname(ftp_host.c_str());
        ftp.put_Username(ftp_name.c_str());
        ftp.put_Password(ftp_password.c_str());

    //  Connect and login to the FTP server.
    success = ftp.Connect();
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
```

We change the FTP directory, we mentioned this earlier in the *FILE DOWNLOAD PORTION*

```
   //  Change to the remote directory where the file will be uploaded.
    success = ftp.ChangeRemoteDir("Server");
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
```

This portion of code , is to specific which file to upload and after successfully uploaded what ‘name’ will be named.

```
   //  Upload a file.
    const char *localFilename = random_file.c_str();
//    const char *remoteFilename = remotefile.c_str();

    success = ftp.PutFile(localFilename,filename.c_str());
    if (success != true) {
        std::cout << ftp.lastErrorText() << "\r\n";
        exit(1);
    }
    success = ftp.Disconnect();
```

If everything went smooth, which mean file upload successfully.

```
    std::cout << "File Uploaded!" << "\r\n";
    string rmcommand="rm " + random_file;
    system(rmcommand.c_str());

}
```

We now go to the last section,

Client's *Change Password()*

```
void changepassword(int sock){
string pass,pass2,pass3;
cout<<"Enter your existing password"<<endl;
cout<<"We will not verify your existing password until we send it over network"<<endl;
cin>>pass;
while(pass.length() < 8  || pass.length() > 12 )
{
cout<<"Password requirement length 8 - 12"<<endl;
cin>>pass;
}

do
{
cout<<"Enter your new password"<<endl;
cin>>pass2;

while(pass2.length() < 8  || pass2.length() > 12 )
{
cout<<"Password requirement length 8 - 12"<<endl;
cin>>pass2;
}f

cout<<"Reconfirm your password"<<endl;
cin>>pass3;

}while(pass2.compare(pass3) != 0);

char* how="???";
send_recv(sock,sha1string(pass),"Sending OLD Password over network");
send_recv(sock,sha1string(pass2),"Sending NEW Password over network");
string status=send_recv(sock,how,"Is the password have been changed ?");
if(status.compare("Changed")==0)
cout<<"Your password have been changed"<<endl;
else
cout<<"You have entered wrong current password"<<endl;
}
```

Basically, first segment ask user to input the current password

```
void changepassword(int sock){
string pass,pass2,pass3;
cout<<"Enter your existing password"<<endl;
cout<<"We will not verify your existing password until we send it over network"<<endl;
cin>>pass;
while(pass.length() < 8  || pass.length() > 12 )
{
cout<<"Password requirement length 8 - 12"<<endl;
cin>>pass;
}
```

Prompt user to input new password and retype the same password as confirmation.

```
do
{
cout<<"Enter your new password"<<endl;
cin>>pass2;
```

```
while(pass2.length() < 8  || pass2.length() > 12 )
{
cout<<"Password requirement length 8 - 12"<<endl;
cin>>pass2;
}

cout<<"Reconfirm your password"<<endl;
cin>>pass3;

}while(pass2.compare(pass3) != 0);
```

After everything went smooth, system will send 2 digests generated from ‘old password’ and ‘new password’ send over Server. Server will do the verification process and reply back ‘Changed’ is it’s matched .

```
char* how="???";
send_recv(sock,sha1string(pass),"Sending OLD Password over network");
send_recv(sock,sha1string(pass2),"Sending NEW Password over network");
string status=send_recv(sock,how,"Is the password have been changed ?");
if(status.compare("Changed")==0)
cout<<"Your password have been changed"<<endl;
else
cout<<"You have entered wrong current password"<<endl;
}
```

Server
Libraries

```
//////////////INCLUDE COLOUR CODE//////////////////
#ifndef _COLORS_
#define _COLORS_
#define RST  "\x1B[0m"
#define KRED  "\x1B[31m"
#define KGRN  "\x1B[32m"
#define KYEL  "\x1B[33m"
#define KBLU  "\x1B[34m"
#define KMAG  "\x1B[35m"
#define KCYN  "\x1B[36m"
#define KWHT  "\x1B[37m"
#define FRED(x) KRED x RST
#define FGRN(x) KGRN x RST
#define FYEL(x) KYEL x RST
#define FBLU(x) KBLU x RST
#define FMAG(x) KMAG x RST
#define FCYN(x) KCYN x RST
#define FWHT(x) KWHT x RST
#define BOLD(x) "\x1B[1m" x RST
#define UNDL(x) "\x1B[4m" x RST
#endif  /* _COLORS_ */
//////////////END OF INCLUDE COLOR CODE////////////////////
#include <unistd.h>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <sys/socket.h>
#include <netinet/in.h>
#include <fstream>
#include <ctime>
#include "chilkat/include/CkFtp2.h"
#include "cryptopp/idea.h"
using CryptoPP::IDEA;
#include "cryptopp/modes.h"
using CryptoPP::CBC_Mode;
#include "cryptopp/secblock.h"
using CryptoPP::SecByteBlock;
#include "cryptopp/osrng.h"
#include "cryptopp/hex.h"
#include "cryptopp/idea.h"
using CryptoPP::IDEA;
#include "cryptopp/sha.h"
using namespace std;
using namespace CryptoPP;
CkFtp2 ftp;
////////////////////////////
```

Hardcoded Values in Server

```
//////////////////////////////////////////
string IV_from_file,Key_from_file,ftp_host,ftp_name,ftp_password;
string hard_keys="65FD379515B4410BAA2EACAA1C0E865D";
string hard_IVs="30D37831B2FF9063";

////////////////////PRINT//////////////////
```

Server's Function Declaration

```
int main();
void listen_menu(int sock);
bool check_credential(string id,string password,int type);
string decrypt_conversation(string cipher,string hard_key,string hard_IVs);
string encrypt_conversation(string plain,string hard_key,string hard_IVs);
```

Start from

Server's int *main*()

```
int main()
{
int sock=socket();
bool connection=connect_ftp();
listen_menu(sock);
return 0;
}
```

First it will establish socket from socket() and return back the socket descriptor into sock variable as int data type. Next will run connect_ftp(), this function is to prompt server's user input FTP credential which include 'IP,USER & PASS'.

Server's int Socket()

```
int PORT;
int socket()
{
    do{
    cout<<"Enter port number to start the listener"<<endl;
    cin >> PORT;
     if(PORT > 65535 || PORT <1)
    {
        cout<<"are you dumb ? the port range is \"1 - 65535\" "<<endl;
     }
    }while(PORT > 65535 || PORT < 1);

    printf ("[Server] Listening the port %d successfully.\n", PORT);
    int server_fd, new_socket, valread;
    struct sockaddr_in address;
    int opt = 1;
    int addrlen = sizeof(address);

    // Creating socket file descriptor
    if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
    {
        perror("socket failed");
        exit(EXIT_FAILURE);
    }

    // Forcefully attaching socket to the port 8080
```

```
        if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                                    &opt, sizeof(opt)))
        {
           perror("setsockopt");
           exit(EXIT_FAILURE);
        }
        address.sin_family = AF_INET;
        address.sin_addr.s_addr = INADDR_ANY;
        address.sin_port = htons( PORT );

        // Forcefully attaching socket to the port 8080
        if (bind(server_fd, (struct sockaddr *)&address,
                                 sizeof(address))<0)
        {
            perror("bind failed");
            exit(EXIT_FAILURE);
        }
        if (listen(server_fd, 3) < 0)
        {
            perror("listen");
            exit(EXIT_FAILURE);
        }
        if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                         (socklen_t*)&addrlen))<0)
        {
            perror("accept");
            exit(EXIT_FAILURE);
        }
        return new_socket ;
    }
```

Server's Connect_FTP()

```
bool connect_ftp(){
        cout<<"Enter IP address of FTP Server"<<endl;
        cin>>ftp_host;
        cout<<"Enter FTP ID"<<endl;
        cin>>ftp_name;
        cout<<"Enter FTP Password"<<endl;
        cin>>ftp_password;
}
```

After everything have done, we will be landing on menu_page(), listening on ‘what client input when Client have reach their MENU ’

```
void listen_menu(int sock){
char *hello = "Received Wink > .. < ";
string menu_selection=recv_send(sock,hello,"Client menu selection");
if(menu_selection.compare("1") == 0 )
        register_account(sock);
else if(menu_selection.compare("2") == 0)
        login_account(sock);
else if(menu_selection.compare("3")==0)
{
    cout<<"{Program Terminate}"<<endl;
        exit(1);
}
}
```

Client perspective

We going to SKIP the ‘selection 3’ since it’s just a quit function.

Let focus on Selection ‘1’ Register.

We will received all input from client, client will input the ID and HASH’s PASSWORD, which will received at this function. TAKE NOTE, all the sending and received message are encrypted using ***OFB MODE IDEA ENCRYPTION***, *USING* ***IDEA_KEY 1***

```
void register_account(int sock){
char *hello = "Received Wink > .. < ";
string id=recv_send(sock,hello,"Client's ID");
string password=recv_send(sock,hello,"Client's Hash Password");
```

We assign the SecByteBlock *key and iv* which our desired length, and we generate ‘a’ *random key and iv* which is IDEA2 Key, and will store into variable EncodedKey and EncodedIV.

```
AutoSeededRandomPool prng;
SecByteBlock key(IDEA::DEFAULT_KEYLENGTH);
SecByteBlock iv(IDEA::BLOCKSIZE);

prng.GenerateBlock(key, key.size());
prng.GenerateBlock(iv, sizeof(iv));
string EncodedKey=Print("key", std::string((const char*)key.begin(), key.size()));
string EncodedIV=Print("iv", std::string((const char*)iv.begin(), iv.size()));
```

Generate TimeStamp

```
  char date[9];
  time_t t = time(0);
  struct tm *tm;
  tm = gmtime(&t);
  strftime(date, sizeof(date), "%Y%m%d", tm);

string s=date;
cout<<s<<endl;
```

Encrypt all the hash's password,KEY,IV, TIMESTAMP using our *IDEA KEY 1*

```
password = encrypt_conversation(password,hard_keys,hard_IVs);
EncodedKey = encrypt_conversation(EncodedKey,hard_keys,hard_IVs);
EncodedIV = encrypt_conversation(EncodedIV,hard_keys,hard_IVs);
s = encrypt_conversation(s,hard_keys,hard_IVs);
```

After encrypted everything we will concatenate together into a string variable 'data'

```
string data=id+":"+password+":"+EncodedKey+":"+EncodedIV+":"+s+":\n";
```

We will check the USERID, ***making sure no one is using the USERID*** before write into *user.txt* , if it found duplicate USERID, it will send Failed to Client and Server perspective will back to *listen_menu()*

```
if (check_credential(id,password,1) == true)
{
char *fail ="Failed";
string password=recv_send(sock,fail,"Failed");
listen_menu(sock);
}
else if(check_credential(id,password,1)==false)
{
char *ver ="Verified";
string password=recv_send(sock,ver,"Verified");
}
cout<<"User INPUT "<<data<<endl;
```

Assume we search nothing in user.txt meaning no duplicate userid, we will 'append' the user.txt and write the concatenate data into the user.txt from variable 'data', and we will back to *listen_menu()*.

```
ofstream outfile;
outfile.open("user.txt", std::ios_base::app); // append instead of overwrite
outfile << data;
outfile.close();
cout<<"Waiting for user's selection"<<endl;
listen_menu(sock);           // listen function;
}
```

We look at some function's that called within this '*register_account()*'. ' check_credential(),Print(), encrypt_conversation, '

Server's Check_Credential()

```
    bool check_credential(string id,string password,int type){
    std::ifstream file("user.txt");
    int counter=0;
    if(file.good())
    {
    cout<<"Reading user.txt file"<<endl;
    }
    else
    {
    counter=1;
    file.open ("user.txt", fstream::app);
    }


    std::string str;
    if (type==2)
    {
    while(!file.eof())
        {
           string file_id="",file_pass="",dump;
           Key_from_file,IV_from_file,registered_time="";
           getline(file,file_id,':');                                    ///Read words before :
           getline(file,file_pass,':');                                    ///Read words before :
           getline(file,Key_from_file,':');                                    ///Read words before :
           getline(file,IV_from_file,':');                                    ///Read words before :
           getline(file,registered_time,':');                                    ///Read after the
first word dumps all trash value just to go to nextline
        getline(file,dump);

         file_pass=decrypt_conversation(file_pass,hard_keys,hard_IVs);

        if(file_id.compare(id)==0 && password.compare(file_pass)==0){
         Key_from_file=decrypt_conversation(Key_from_file,hard_keys,hard_IVs);
         IV_from_file=decrypt_conversation(IV_from_file,hard_keys,hard_IVs);
         registered_time=decrypt_conversation(registered_time,hard_keys,hard_IVs);

        cout<<"Matched"<<endl;
        file.close();
        return true;
        }

        }

    file.close();
    return false;

    }

    else if(type==1)
    {

    while(!file.eof())
    {
        string dump;
        string file_a;
           getline(file,file_a,':');                                    ///Read words before :
           getline(file,dump);                                    ///Read after the first word dumps
all trash valu$
           if(counter==1)
```

```
        {
                return false;
        }
           if(id.compare(file_a) == 0 )
           {
            cout<<"USER ID is used, Change another USER ID"<<endl;
        file.close();
        return true;
           }
    }
    file.close();
    return false;
    }
    }
```

This function have separate into 2 type, 1 and 2.

If it's type '1', it will do ID and PASSWORD comparison, we will not jump into details first.
If it's **type '2',** it will do ID comparison, making sure no duplicate ID is found.

It will start to grab lines by lines in user.txt and getline with delimiter to check 'user input id' with 'stored id in user.txt'.

If no duplicate, it will return false. But if found same ID exist in user.txt it will return true to terminate the registration process and back to *listen_menu()*.

Server's Print()

```
string Print(const std::string& label, const std::string& val)
{
   std::string encoded;
   StringSource(val, true,
      new HexEncoder(
         new StringSink(encoded)
      ) // HexEncoder
   ); // StringSource
   std::cout << label << ": " << encoded << std::endl;
    return encoded;
}
```

This function is to print 'bytes value' in Hex Format.

Server's Encrypt Function

```
//////////////////////ENCRYPT CONVERSATION//////////////////////////////
string encrypt_conversation(string plain,string hard_keys,string hard_IVs){
string keys,IVs;
StringSource ss(hard_keys,true,new HexDecoder(new StringSink(keys)));
StringSource sss(hard_IVs,true,new HexDecoder(new StringSink(IVs)));

AutoSeededRandomPool prng;

SecByteBlock key((const byte*)keys.data(), keys.size());
SecByteBlock iv((const byte*)IVs.data(), IVs.size());

string encoded,cipher;
try
{
    cout << "Plain text: " << plain << endl;
```

```
    OFB_Mode< IDEA >::Encryption e;
    e.SetKeyWithIV(key, key.size(), iv);

    // The StreamTransformationFilter adds padding
    //  as required. ECB and CBC Mode must be padded
    //  to the block size of the cipher.
    StringSource(plain, true,
        new StreamTransformationFilter(e,
            new StringSink(cipher)
        ) // StreamTransformationFilter
    ); // StringSource
}
catch(const CryptoPP::Exception& e)
{
    cerr << e.what() << endl;
    exit(1);
}

/*********************************\
\*********************************/

// Pretty print
StringSource(cipher, true,
    new HexEncoder(
        new StringSink(encoded)
    ) // HexEncoder
); // StringSource

cout << "[ Sending Encrypted Content over network ]" << encoded << endl;
return encoded;
}
```

Declare variable

```
//////////////////////ENCRYPT CONVERSATION//////////////////////////////
string encrypt_conversation(string plain,string hard_keys,string hard_IVs){
string keys,IVs;
StringSource ss(hard_keys,true,new HexDecoder(new StringSink(keys)));
StringSource sss(hard_IVs,true,new HexDecoder(new StringSink(IVs)));
```

Load keys and IV from global variables

```
AutoSeededRandomPool prng;

SecByteBlock key((const byte*)keys.data(), keys.size());
SecByteBlock iv((const byte*)IVs.data(), IVs.size());
```

Start the encryption process using OFB_Mode

```
string encoded,cipher;
try
{
    cout << "Plain text: " << plain << endl;

    OFB_Mode< IDEA >::Encryption e;
    e.SetKeyWithIV(key, key.size(), iv);

    // The StreamTransformationFilter adds padding
    //  as required. ECB and CBC Mode must be padded
```

```
    //  to the block size of the cipher.
    StringSource(plain, true,
        new StreamTransformationFilter(e,
            new StringSink(cipher)
        ) // StreamTransformationFilter
    ); // StringSource
}
catch(const CryptoPP::Exception& e)
{
    cerr << e.what() << endl;
    exit(1);
}

/*********************************\
\*********************************/

// Pretty print
StringSource(cipher, true,
    new HexEncoder(
        new StringSink(encoded)
    ) // HexEncoder
); // StringSource
```

Display the encrypted content

```
cout << "[ Sending Encrypted Content over network ]" << encoded << endl;
return encoded;
}
```

Server's Login_account()

```
bool login_account(int sock){
char *hello = "Received Wink > .. < ";
char *ver = "Verified";
char *fail = "Failed";
string id=recv_send(sock,hello,"Client's ID");
string password=recv_send(sock,hello,"Client's Hash Password");
bool check=check_credential(id,password,2);
if (check==true)
{
int i = std::stoi(registered_time);

  char date[9];
  time_t t = time(0);
  struct tm *tm;

  tm = gmtime(&t);
  strftime(date, sizeof(date), "%Y%m%d", tm);

   string s=date;
   cout<<s<<endl;
int b =std::stoi(s);

if( b - i >=200)     // if register date + 60 days equal or more than then renew KEY
{
cout<<"your IDEA KEY2 and IV is renew"<<endl;
renewKEY_IV(id);
}
string timestamp=recv_send(sock,ver,"Verified");
cout<<ftp_host<<endl;
recv_send(sock,ftp_host,"FTP HOSTNAME");
recv_send(sock,ftp_name,"FTP ID");
recv_send(sock,ftp_password,"FTP PASSWORD");
recv_send(sock,Key_from_file,"IDEA KEY");
recv_send(sock,IV_from_file,"IDEA IV");

string login_selection;
do{
login_selection=recv_send(sock,hello,"Received Login Menu Choice");

if(login_selection.compare("1")==0)
    {
            char *hello = "Received Wink > .. < ";
            char * ok="Changed";
            char * fail="Failed";
            string existpw=recv_send(sock,hello,"OLD PASSWORD");
            bool stat=check_credential(id,existpw,2);
            string newpw=recv_send(sock,hello,"New PASSWORD");
            if (stat==true)
            {
                    changedpassword(id,newpw);
                    recv_send(sock,ok,"Password changed");
            }
            else
            recv_send(sock,fail,"Password failed to change");


    }
    else if(login_selection.compare("2")==0)
    {
            if( b - i >= 1 )         //difference of 24hous
            {
            char* ok="Okay";
            recv_send(sock,ok,"Operation allowed");
            cout<<"File Upload Selected"<<endl;
            }else
```

```
                {
                char* fail="failed";
                recv_send(sock,fail,"Operation failed");
                cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
                }
        }
        else if(login_selection.compare("3")==0)
        {
                if( b - i >=1 )          //difference of 24 hours
                {
                char* ok="Okay";
                recv_send(sock,ok,"Received Login Menu Choice");
                cout<<"File Download Selected"<<endl;
                }else
                {
                char* fail="failed";
                recv_send(sock,fail,"Received Login Menu Choice");
                cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
                }
        }
        else if (login_selection.compare("4")==0)
        {
                listen_menu(sock);
        }
    }while(login_selection.compare("4")!=0);

    }
    else
    {
    string timestamp=recv_send(sock,fail,"Failed");
    listen_menu(sock);
    }

    }
```

Explanation:

Received encrypted credentials from Client, it will run “Check_Credential() as type 2, as I mentioned type2 is for login purpose credential validation on ID and PASSWORD ”

```
bool login_account(int sock){
char *hello = "Received Wink > .. < ";
char *ver = "Verified";
char *fail = "Failed";
string id=recv_send(sock,hello,"Client's ID");
string password=recv_send(sock,hello,"Client's Hash Password");
bool check=check_credential(id,password,2);
```

If the result *true*, server will run a timestamp on current time in INT datatype. Format Eg “20200315” == Year 2020, March, 15

```
if (check==true)
{
int i = std::stoi(registered_time);

  char date[9];
  time_t t = time(0);
  struct tm *tm;
  tm = gmtime(&t);
  strftime(date, sizeof(date), "%Y%m%d", tm);
   string s=date;
   cout<<s<<endl;
int b =std::stoi(s);
```

If current time are 2 month ahead the 'user's registration timestamp', *USER'S UNIQUE KEY AND IV* will be assigned to a *NEW KEY*

```
if( b - i >=200)     // if register date + 60 days equal or more than then renew KEY
{
cout<<"your IDEA KEY2 and IV is renew"<<endl;
renewKEY_IV(id);
}
```

Server will send a '**Verified'**, for fun 'as validation', then ***Authentcation Server will send all FTP credentials and USER's UNIQUE key and iv to client.***

```
string timestamp=recv_send(sock,ver,"Verified");
cout<<ftp_host<<endl;
recv_send(sock,ftp_host,"FTP HOSTNAME");
recv_send(sock,ftp_name,"FTP ID");
recv_send(sock,ftp_password,"FTP PASSWORD");
recv_send(sock,Key_from_file,"IDEA KEY");
recv_send(sock,IV_from_file,"IDEA IV");
```

After everything done, Server will fall into a do while loop, it will keep login in this menu. To quit this menu Server have to received message with content '4' which represent 'Log Out' to quit this menu and back to *listen_menu()*

Client's Perspective

```
[ Recovered message from Authentication Server ] : E3CD821F6293F46E
FTP CREDENTIAL
FTP HOSTNAME: 123
FTP ID: 123
FTP PASSWORD: 123
IDEA KEY: 03B4EEC0A119ED38025CB818BF2463AD
IDEA IV: E3CD821F6293F46E
1.Change Password
2.Upload File
3.Download File
4.Log out
```

```
**       NOTE: FTP IP , ID && PASSWORD ARE JUST FOR DEMONSTRATION PURPOSES   **
```

```
string login_selection;
do{
login_selection=recv_send(sock,hello,"Received Login Menu Choice");

if(login_selection.compare("1")==0)
    {
            char *hello = "Received Wink > .. < ";
            char * ok="Changed";
            char * fail="Failed";
            string existpw=recv_send(sock,hello,"OLD PASSWORD");
            bool stat=check_credential(id,existpw,2);
            string newpw=recv_send(sock,hello,"New PASSWORD");
            if (stat==true)
            {
                    changedpassword(id,newpw);
                    recv_send(sock,ok,"Password changed");
            }
            else
            recv_send(sock,fail,"Password failed to change");
```

```
        }
        else if(login_selection.compare("2")==0)
        {
                if( b - i >= 1 )          //difference of 24hous
                {
                char* ok="Okay";
                recv_send(sock,ok,"Operation allowed");
                cout<<"File Upload Selected"<<endl;
                }else
                {
                char* fail="failed";
                recv_send(sock,fail,"Operation failed");
                cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
                }
        }
        else if(login_selection.compare("3")==0)
        {
                if( b - i >=1 )           //difference of 24 hours
                {
                char* ok="Okay";
                recv_send(sock,ok,"Received Login Menu Choice");
                cout<<"File Download Selected"<<endl;
                }else
                {
                char* fail="failed";
                recv_send(sock,fail,"Received Login Menu Choice");
                cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
                }
        }
        else if (login_selection.compare("4")==0)
        {
                listen_menu(sock);
        }
    }while(login_selection.compare("4")!=0);

    }
    else
    {
    string timestamp=recv_send(sock,fail,"Failed");
    listen_menu(sock);
    }
    }
```

Before proceed to selection ‘1,2,3 or 4 . Let discuss functions’ that execute within this function();

As we mentioned previously, we will discuss more detail on *check_credential()*

Server’s check_credential() type 2

```
bool check_credential(string id,string password,int type){
std::ifstream file("user.txt");
int counter=0;
if(file.good())
{
cout<<"Reading user.txt file"<<endl;
}
else
{
counter=1;
file.open ("user.txt", fstream::app);
}
```

```
    std::string str;
    if (type==2)
    {
    while(!file.eof())
        {
           string file_id="",file_pass="",dump;
           Key_from_file,IV_from_file,registered_time="";
           getline(file,file_id,':');                                    ///Read words before :
           getline(file,file_pass,':');                                    ///Read words before :
           getline(file,Key_from_file,':');                                    ///Read words before :
           getline(file,IV_from_file,':');                                    ///Read words before :
           getline(file,registered_time,':');                                    ///Read after the
first word dumps all trash value just to go to nextline
        getline(file,dump);

         file_pass=decrypt_conversation(file_pass,hard_keys,hard_IVs);

        if(file_id.compare(id)==0 && password.compare(file_pass)==0){
         Key_from_file=decrypt_conversation(Key_from_file,hard_keys,hard_IVs);
         IV_from_file=decrypt_conversation(IV_from_file,hard_keys,hard_IVs);
         registered_time=decrypt_conversation(registered_time,hard_keys,hard_IVs);

        cout<<"Matched"<<endl;
        file.close();
        return true;
        }

        }

    file.close();
    return false;

    }

    else if(type==1)
    {
    …<snip>
    }
```

Server will read “user.txt” ,and grab the lines by lines from user.txt and get ID and encrypted password using delimiter and run decryption on the password and start the comparison to determine they login successfully or not.

If login success it will return ***true***.

Server’s renewKEY_IV()

```
    void renewKEY_IV(string id){
    AutoSeededRandomPool prng;
    SecByteBlock key(IDEA::DEFAULT_KEYLENGTH);
    SecByteBlock iv(IDEA::BLOCKSIZE);
    prng.GenerateBlock(key, key.size());
    prng.GenerateBlock(iv, sizeof(iv));
    Key_from_file=Print("key", std::string((const char*)key.begin(), key.size()));
    IV_from_file=Print("iv", std::string((const char*)iv.begin(), iv.size()));
    std::ifstream file("user.txt");
    if(file.good())
    {
    cout<<"File found,verifying"<<endl;

    }else
    {
    file.open ("user.txt", fstream::app);
    }
    std::string str,total_data;
    while(!file.eof())
    {
```

```
            string file_id,dump,pass,IVs,KEYs,stamp;
            getline(file,file_id,':');                                    ///Read words before :
            getline(file,pass,':');                                     ///Read words before :
            getline(file,KEYs,':');                                     ///Read words before :
            getline(file,IVs,':');                                    ///Read words before :
            getline(file,stamp,':');                                    ///Read words before :
            getline(file,dump);                                   ///Read words before :
            if(id.compare(file_id) == 0 )
            {
         total_data=total_data+file_id+":"+pass+":"+Key_from_file+":"+IV_from_file+":"+stamp+":"+"\n"
;
     //
total_data=total_data+file_id+":"+pass+":"+Key_from_file+":"+IV_from_file+":"+stamp+":";
            }
             else if(file_id.compare("")!=0)
             {
             total_data=total_data+file_id+":"+pass+":"+KEYs+":"+IVs+":"+stamp+":"+"\n";
             }
     }
        file.close();
        std::ofstream ofs("user.txt", std::ofstream::trunc);
        ofs << total_data;
        ofs.close();

     }
```

Basically, same methodology read line by line and concatenate all string into a variable called *'total_data'*. If that particular line is not relate to 'current user', it will leave it remain everything. If, it current line is related to 'current user', it will grab *id, password and timestamp* as remain but the KEYS and IV will be NEW VALUE and concatenate back into the variable *'total_data'*.

After finish reading the file until END OF FILE (EOF), it overwrite existing USER.TXT data with our 'CONCATENATE DATA'. Meaning our USER.TXT now with *UPDATE KEY and IV value on current user.*

*Let look at what happened if "Client have selected option 1 , 2 or 3 ", let look at option '**1**' Change Password*

Server's Change Passowrd

```
     Segmented code in login_account()

                 char *hello = "Received Wink > .. < ";
                 char * ok="Changed";
                 char * fail="Failed";
                 string existpw=recv_send(sock,hello,"OLD PASSWORD");
                 bool stat=check_credential(id,existpw,2);
                 string newpw=recv_send(sock,hello,"New PASSWORD");
                 if (stat==true)
                 {
                         changedpassword(id,newpw);
                         recv_send(sock,ok,"Password changed");
                 }
                 else
                 recv_send(sock,fail,"Password failed to change");
```

It will received old password, and run *check_credential* on type 2 check whether it matches the ID and EXISTING PASSWORD.

If it matches, it will **return true else it will return false**.

If true , it will run *changedpassword()* and send an encrypted message with content 'PasswordChanged' to Client.

Let look at the function *changedpassword()*

```
void changedpassword(string id,string password){
std::ifstream file("user.txt");
if(file.good())
{
cout<<"File found,verifying"<<endl;
}else
{
file.open ("user.txt", fstream::app);
}
std::string str;
string file_id,dump,total_data,pass;
while(!file.eof())
{
       getline(file,file_id,':');                                    ///Read words before :
       getline(file,pass,':');                                   ///Read words before :
       getline(file,dump);                                  ///Read words before :
       if(id.compare(file_id) == 0 )
       {
    password=encrypt_conversation(password,hard_keys,hard_IVs);
    total_data=total_data+file_id+":"+password+":"+dump+"\n";
       }
    else if(file_id.compare("")!=0)
    {
    total_data=total_data+file_id+":"+pass+":"+dump+"\n";
    }
}
   system("rm user.txt");
   std::ofstream ofs("user.txt", std::ofstream::trunc);
   ofs << total_data;
   ofs.close();
}
```

It will take 2 parameters , ID and ‘NEW ’ PASSWORD , look for user.txt, grab all files. If related to current user, it will concatenate ID + NEW PASSWORD+ KEY,IV & TIMESTAMP and merge into a variable ‘total_data’, if the lines is not related to the current user it will leave the data as remain but will merge/concatenate into ‘total_data’ variable.

After finished Reading Files (EOF), it will remove existing ‘user.txt’ and write new ‘user.txt’ into disk

*After finished the process, now it will back to *do while loop**

We now look at **File Upload**

Server's response to File Upload

```
*Segment of code in login_account*
else if(login_selection.compare("2")==0)
    {
            if( b - i >= 1 )         //difference of 24hous
            {
            char* ok="Okay";
            recv_send(sock,ok,"Operation allowed");
            cout<<"File Upload Selected"<<endl;
            }else
            {
            char* fail="failed";
            recv_send(sock,fail,"Operation failed");
            cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
            }
    }
```

Now, if (B – I) larger than 1 or equal 1 meaning it's 24 hours or more than 24 hours. It's in TIMEFORMAT just that it's STRING DATA TYPE
Why so ? 20200320 – 20200319 = 1 meaning it's a new day, which can mean 24 hours have passed.

It will an encrypted "Okay" to Client as confirmation to use FTP operation **if** it's larger or equal than 1.
Now Client will do the *File Upload Function*

If's not more than 24hours, Server will replied back an encrypted message with 'Failed'

We now look at **File Download**
Server's response to FileDownload

```
else if(login_selection.compare("3")==0)
    {
            if( b - i >=1 )          //difference of 24 hours
            {
            char* ok="Okay";
            recv_send(sock,ok,"Received Login Menu Choice");
            cout<<"File Download Selected"<<endl;
            }else
            {
            char* fail="failed";
            recv_send(sock,fail,"Received Login Menu Choice");
            cout<<"You're within 24 hours activiation,FTP operation is disabled"<<endl;
            }
    }
```

Now, if (B – I) larger than 1 or equal 1 meaning it's 24 hours or more than 24 hours. It's in TIMEFORMAT just that it's STRING DATA TYPE
Why so ? 20200320 – 20200319 = 1 meaning it's a new day, which can mean 24 hours have passed.

It will an encrypted "Okay" to Client as confirmation to use FTP operation **if** it's larger or equal than 1.
Now Client will do the *File Download Function*

If's not more than 24hours, Server will replied back an encrypted message with 'Failed'

USER MANUAL (GITHUB)

README.MD

```
README.MD
```

GIT CLONE FROM GITHUB

To download the program in Ubuntu or Linux environment

```
root@kali:~# git clone https://github.com/Applebois/NetSec-Assignment2
Cloning into 'NetSec-Assignment2'...
remote: Enumerating objects: 14, done.
remote: Counting objects: 100% (14/14), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 14 (delta 2), reused 14 (delta 2), pack-reused 0
Unpacking objects: 100% (14/14), done.
```

INSTALLATION

Install the crypto++ library or else might not able to build/compile in later.

```
#sudo apt-get update
```

Then, issue next command to install crypto++

```
#sudo apt-get install libcrypto++-dev libcrypto++-doc libcrypto++utils
```

Download chilkat library and extract it on our desktop

```
tar -xf chilkat-9.5.0-x86_64-linux-gcc.tar
```

and rename `chilkat-9.5.0-x86_64-linux-gcc to chilkat`

```
export LD_LIBRARY_PATH=chilkat/lib:$LD_LIBRARY_PATH
```

COMMAND TO COMPILE/BUILD

How to compile the source code after issued “git clone” command

```
# cd NetSec-Assignment2
# cd Source
# cp client.cpp /root/Desktop/
# cp server.cpp /root/Desktop/
# cd /root/Desktop
# g++ -o server server.cpp -Lchilkat/lib -lchilkat-9.5.0 -lresolv -lpthread -
lcryptopp
# g++ -o client client.cpp -Lchilkat/lib -lchilkat-9.5.0 -lresolv -lpthread -
lcryptopp
```

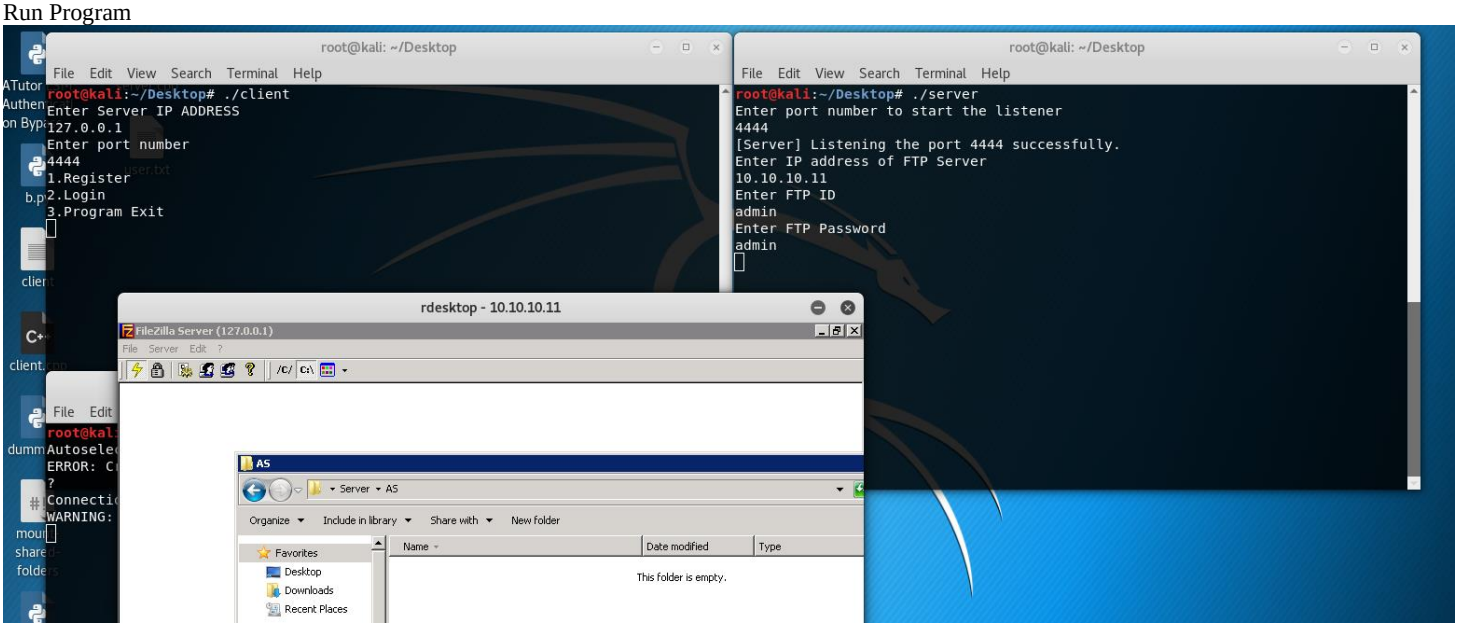
ADD PRIVILEGE TO THE BINARY FILE

Change program privilege to allow execution after issued “git clone” command

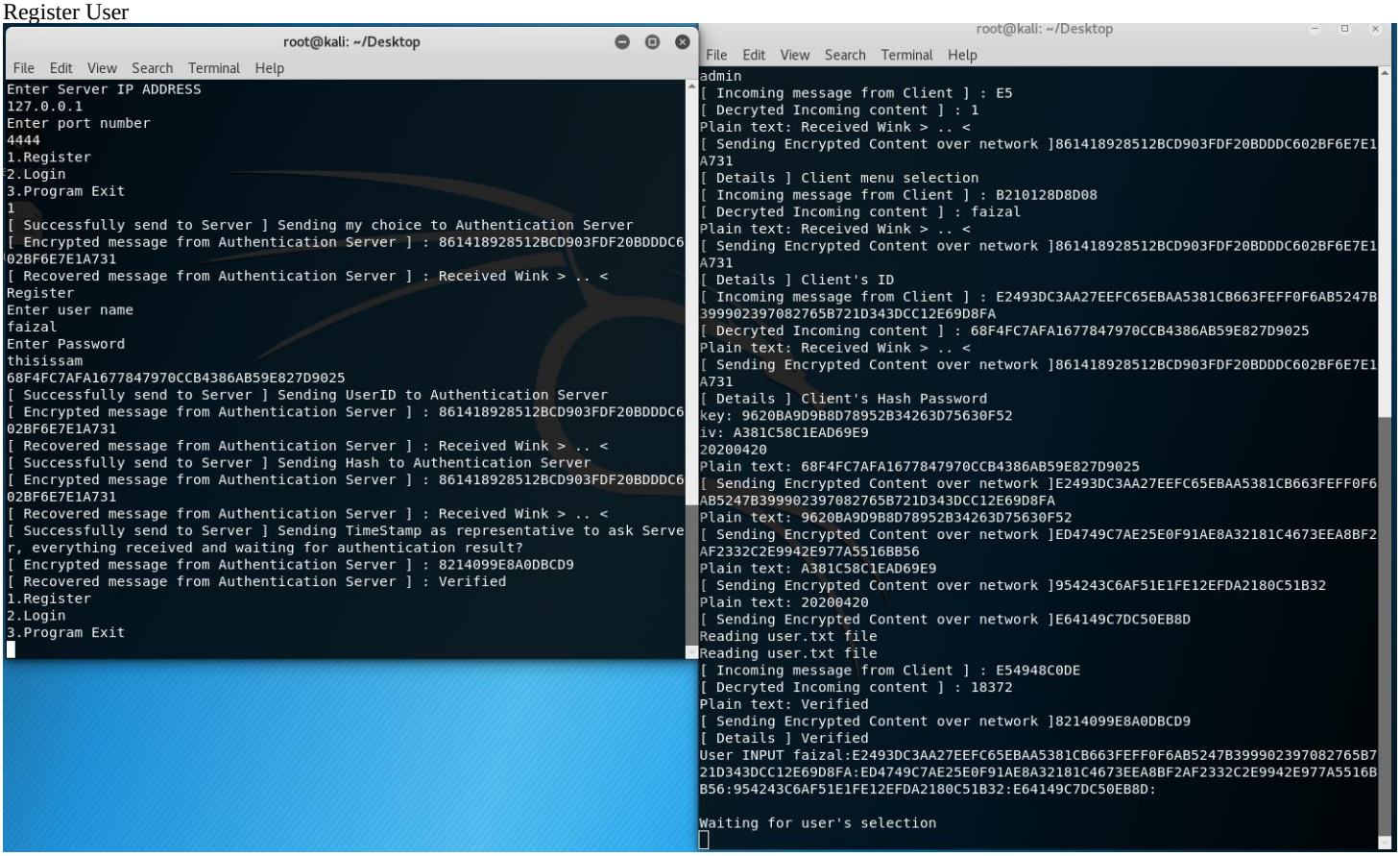
```
# ./server
Enter port number to start the listener
4444
[Server] Listening the port 4444 successfully.
# chmod u+x client
# ./client
Enter Server IP ADDRESS
127.0.0.1
Enter port number
4444
1.Register
2.Login
3.Program Exit
```

Program's Flow

Run Program



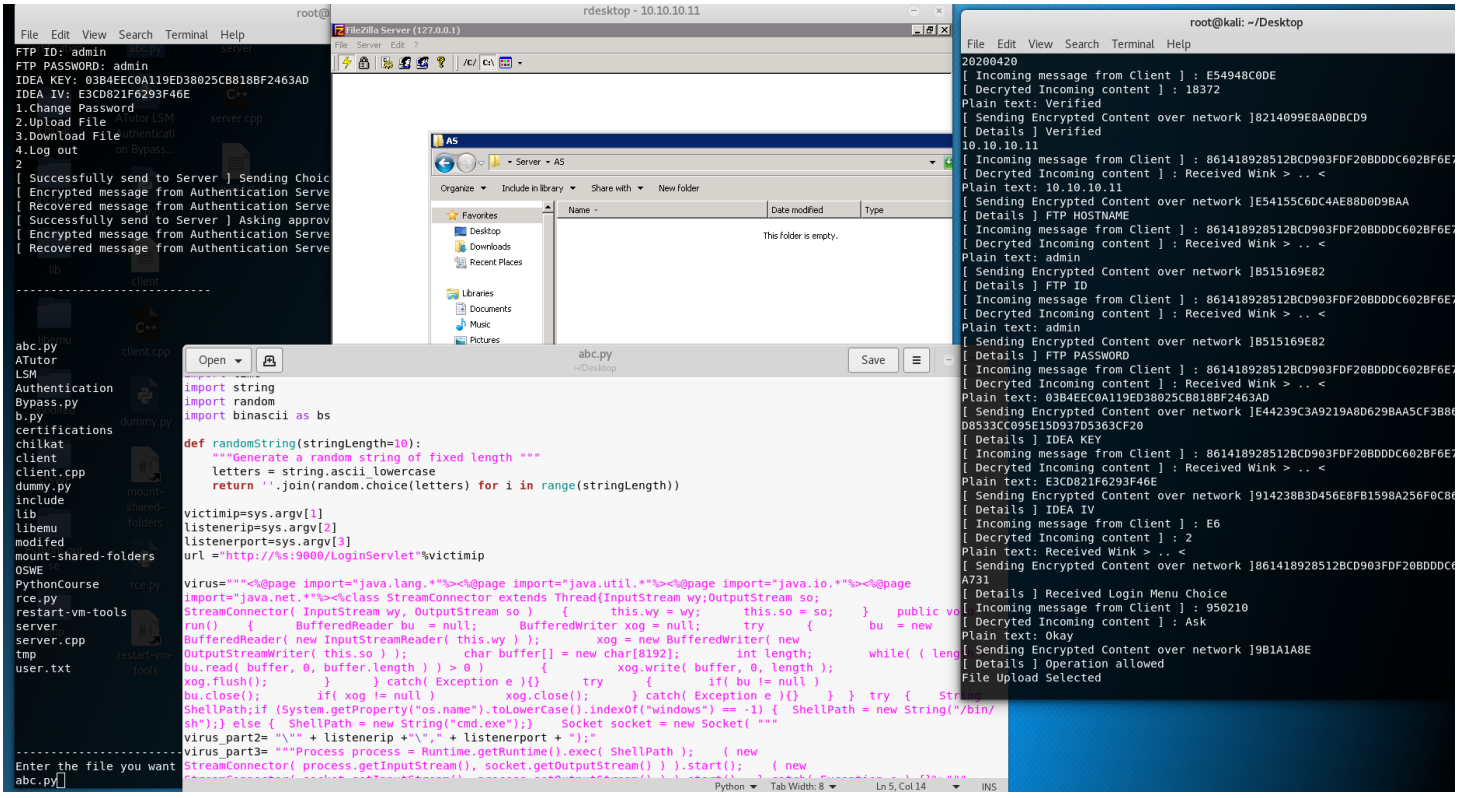
Register User



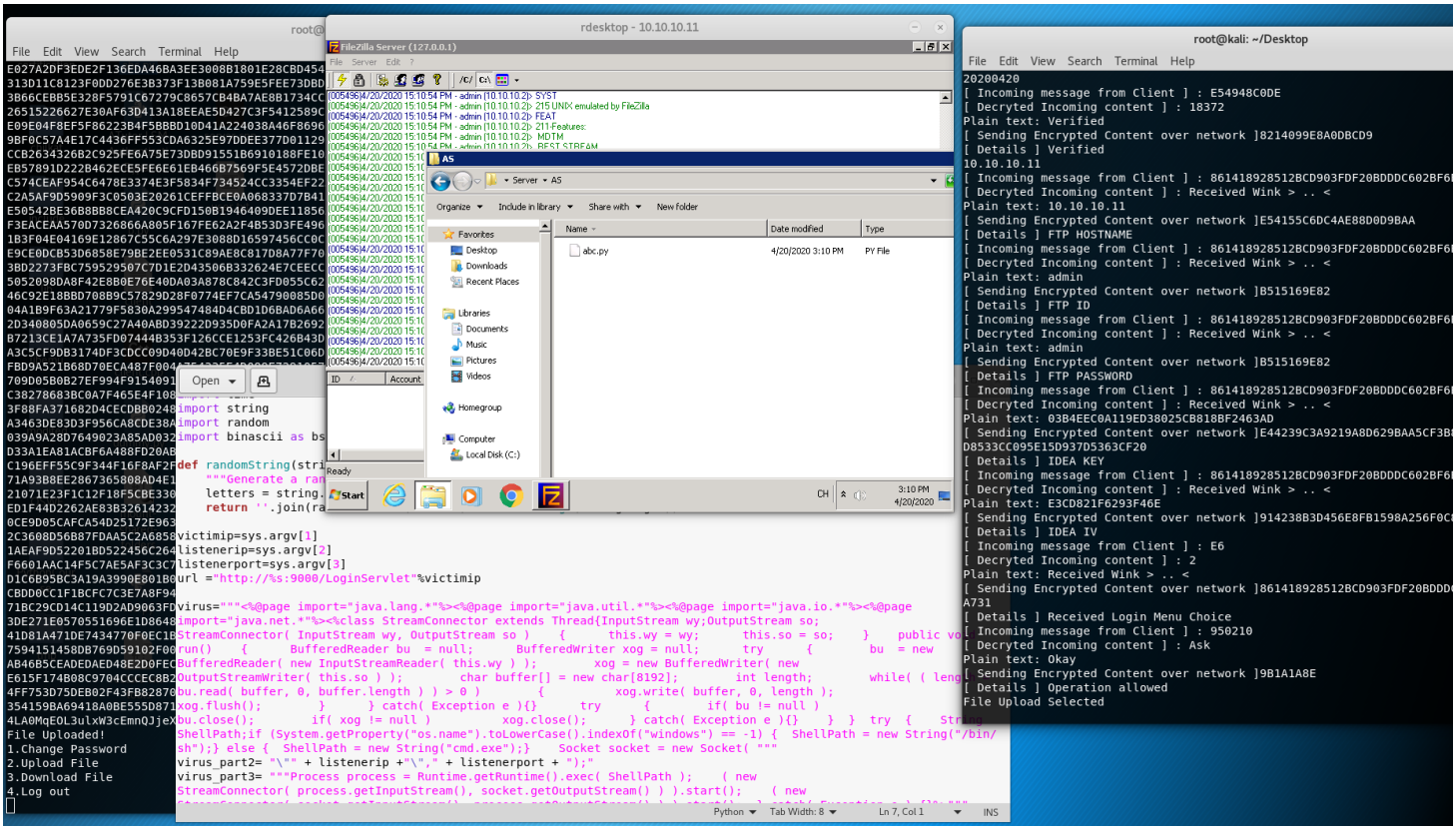
Content of “user.txt”

```
    winsamwinsam:963343B4DC26E08A66EBAA26F3B96E3D9AF0F3A82746B594E622E4702464B757D246A
8BC511DD88D:E23739C3AA279CF96592AF218FCC68329EFF80AC2735C298E028E2705715C825:ED484AC2D
9529A8E1A9EA22685B96B39:E64149C7DC50E88B:
    winsam:E5474BB5D55CE18962EEDF5581C81D339CF9F0AD2946C1E8E65D917A2367CF53DA42DBB0556
1AE89:E44239C3A9219A8D629BAA5CF3B86D33E8FBF4D8533CC095E15D937D5363CF20:914238B3D456E8F
B1598A256F0C8684E:E64149C7DC50E88B:
    abc:96453ACFDF5DEF8A6293A95C85B96D3D9CFEF0AB5437C698945AE2715C12B65DDE44D3C32F60AC
F7:92453FCEDD529884179CD824F5C96F33EEF180DE2930C69EE522E47C2065B653:EC474BC3DD219F8C17
9CAD51F0CA1F4E:E64149C7DC50EB8D:
    faizal:E2493DC3AA27EEFC65EBAA5381CB663FEFF0F6AB5247B399902397082765B721D343DCC12E6
9D8FA:ED4749C7AE25E0F91AE8A32181C4673EEA8BF2AF2332C2E9942E977A5516BB56:954243C6AF51E1F
E12EFDA2180C51B32:E64149C7DC50EB8D:
```

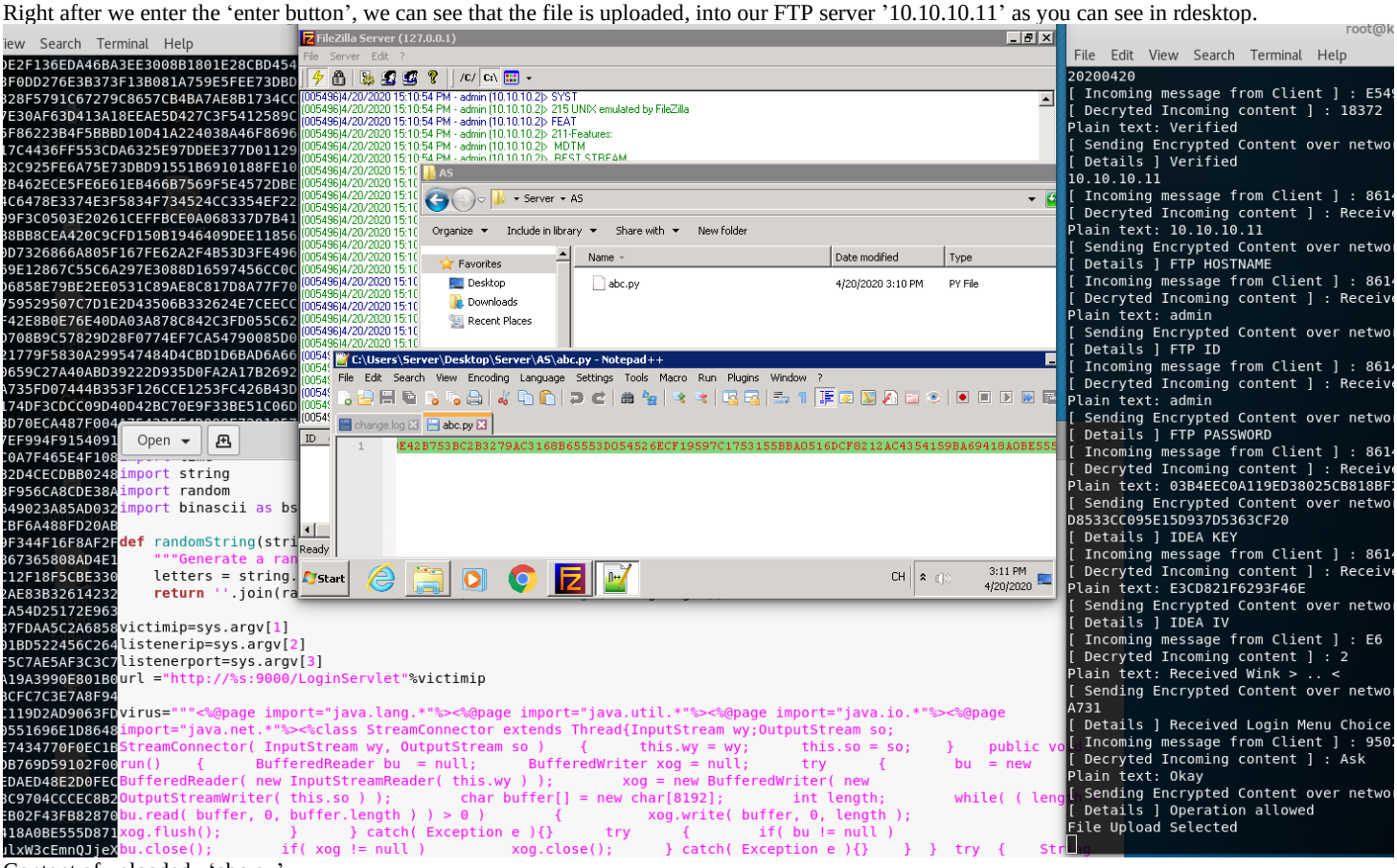
Format
ID : PASSWORD : IV : KEY : TIMESTAMP
ID : HASH PASSWORD : IV : KEY : TIMESTAMP
ID : ENCRYPTED HASH PASSWORD : ENCRYPTED IV : ENCRYPTED KEY : ENCRYPTED TIMESTAMP



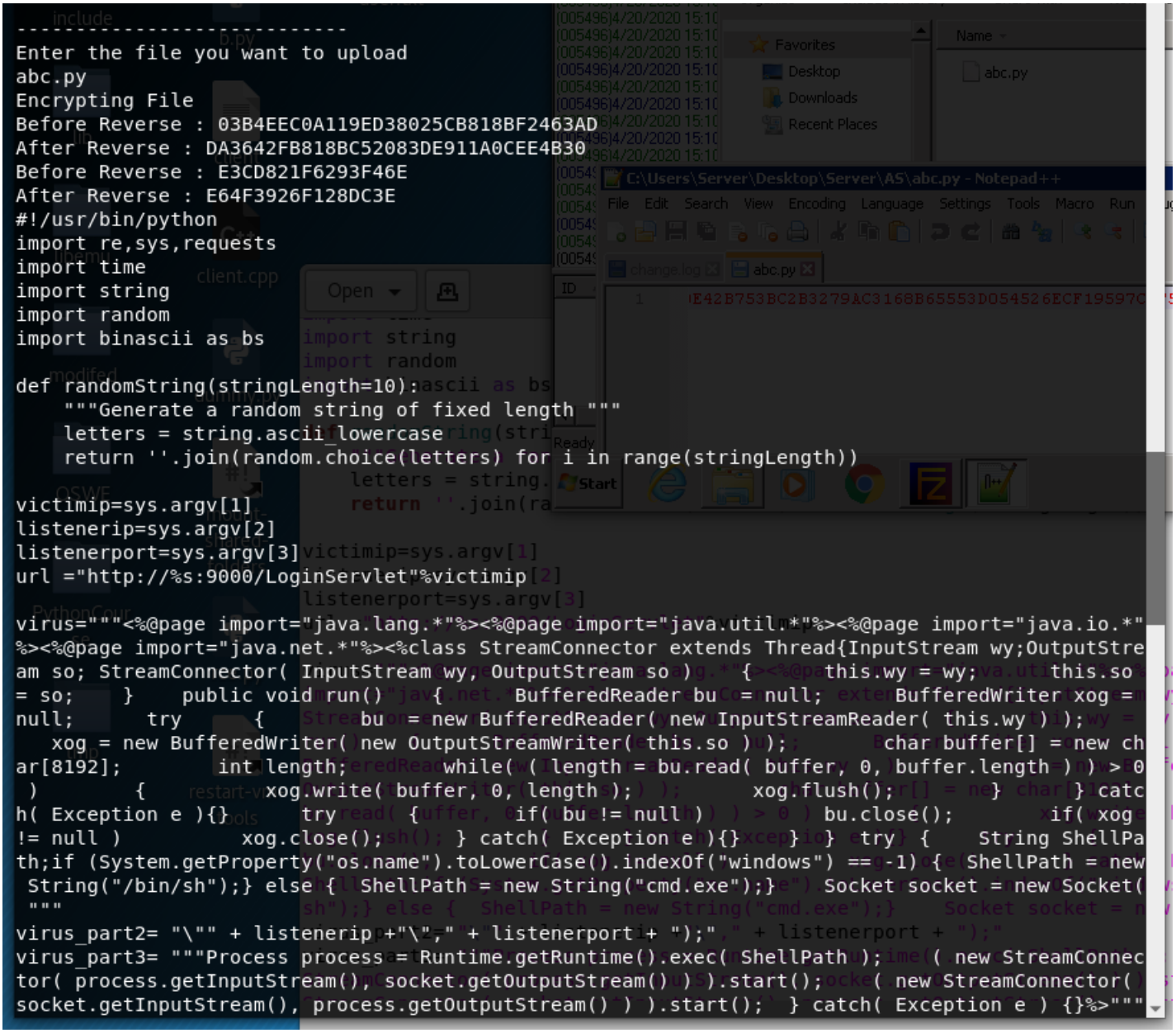
As you can see , client allow us to see what files to upload, we select “ABC.PY”, and we will encrypt this file first before proceed to the process of upload.



Right after we enter the ‘enter button’, we can see that the file is uploaded, into our FTP server ’10.10.10.11’ as you can see in rdesktop.

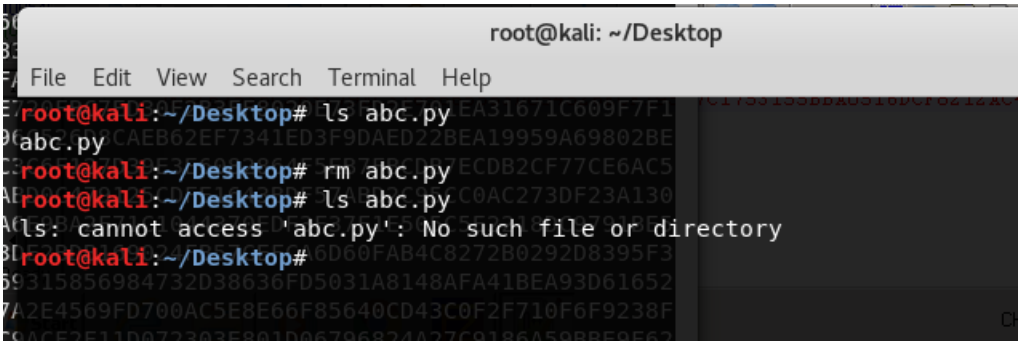


Content of uploaded , ‘abc.py’

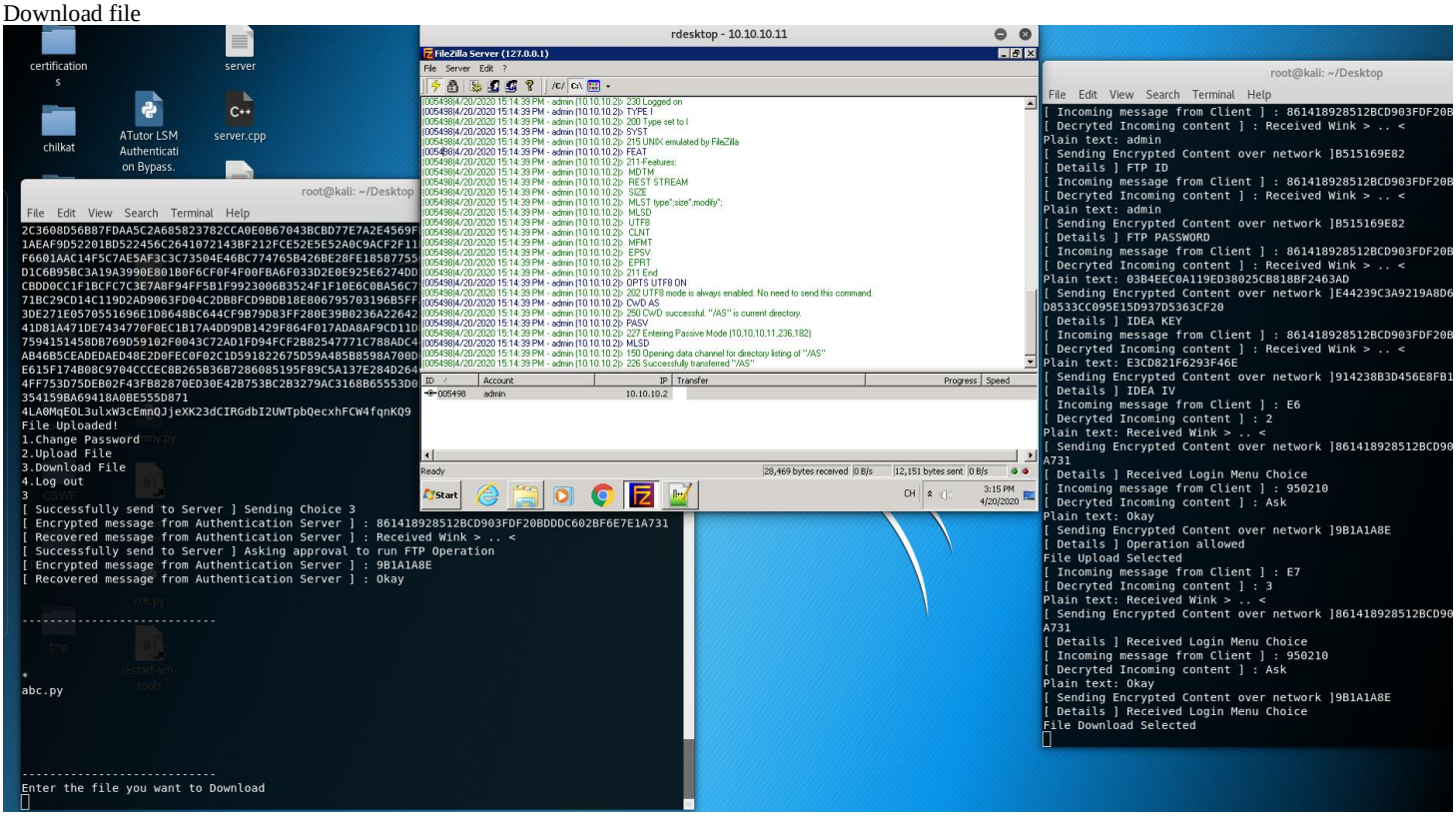


The client will show the ‘KEY’ && ‘IV’. Before and After.

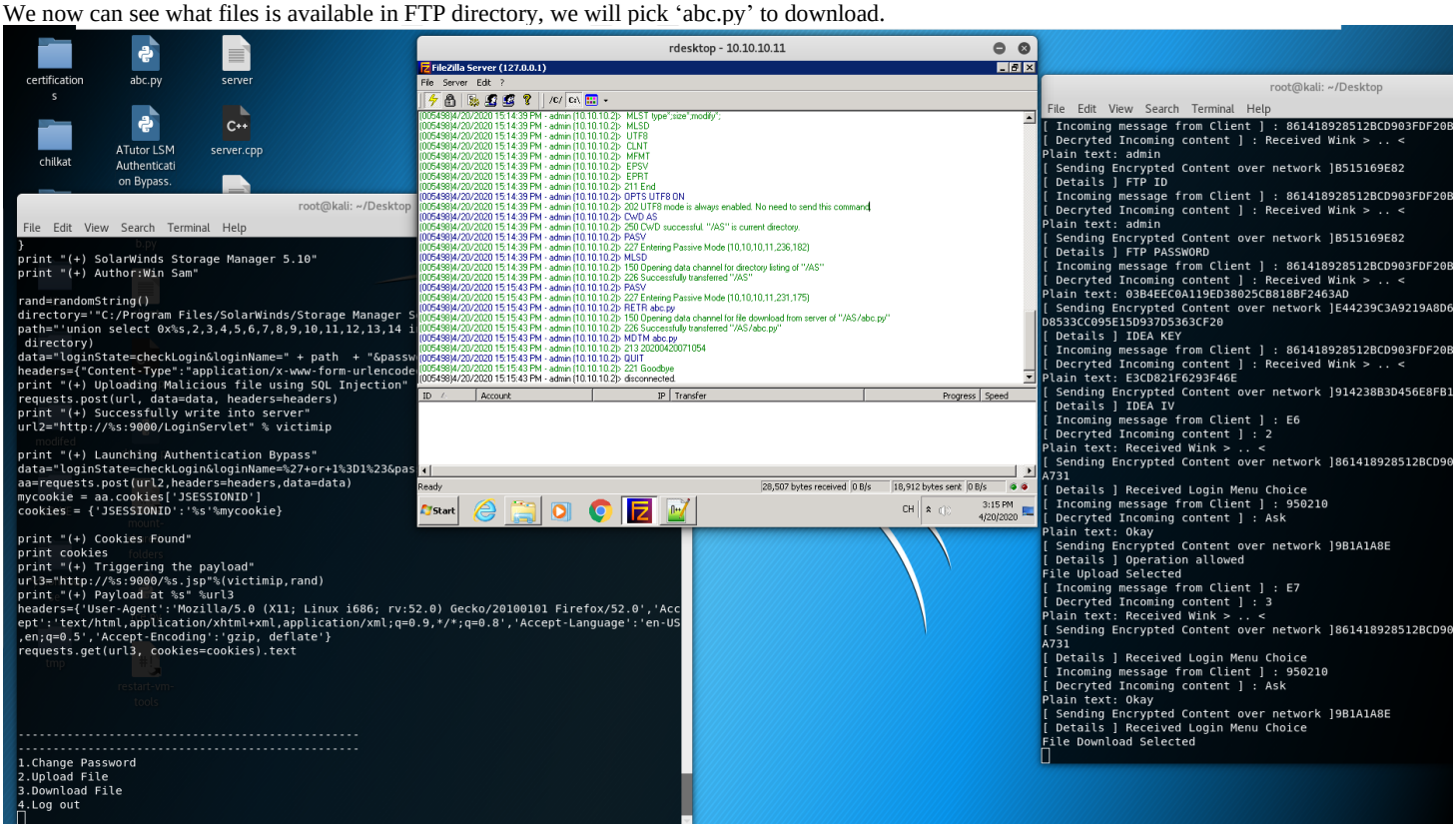
We now remove our local file, so we can download it from the FTP Server.



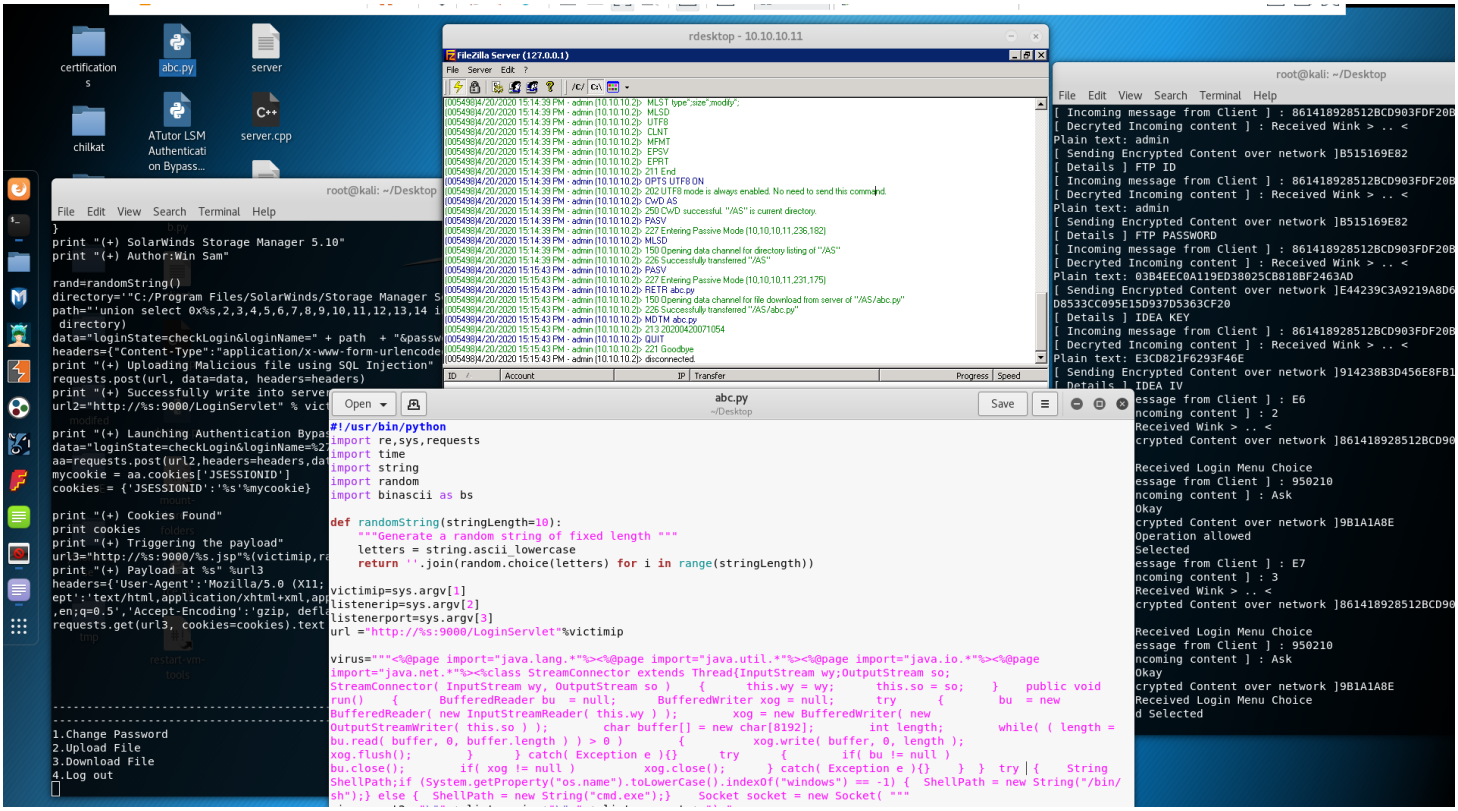
Download file



We now can see what files is available in FTP directory, we will pick ‘abc.py’ to download.



As you can see, on the right hand side 2nd icon it’s abc.py get downloaded from FTP Server, and from the logs in FTP FileZilla we can see transmission logs too.



Our file get downloaded with ‘decrypted’ form

I have submitted a youtube URL, to view the entire login flow and stuff.

https://youtu.be/axRW6uOHpTk