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 1	Coursework Percentage:	10%
Hand-out Date:	Week 3	Received By : (signature)	
Due Date:	Week 6	Received Date :	
STUDENT'S INFORMATION:			
Student's Name & ID:	**6306196**		
Contact Number / Email:	**TEH WIN SAM /** **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 1	Coursework Percentage:	10%
Hand-out Date:	Week 3	Received By: (Signature)	
Due date:	Week 6	Received Date:	
STUDENT'S INFORMATION:			
Student's Name & ID:	6306196		
Contact Number / Email:	TEHWINSAM 013-369-6298		

Assessment Criteria		**Total Marks**	**Given Marks**
1.	**Part 1:**		
	Quiz 1	2	
2.	**Part 2:**		
	Modified DH key exchange protocol and the shared key computations	3	
	AES Encryption and RSA PKC	3	
	CFB - Mode of Block Cipher	1	
	Presentation of work product	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 1 (10 %)

Aims

This assignment consists of two parts. Part 1 is a Quiz 1 that focuses on revision aspects of Computer Network concepts learnt in the earlier semester. This is to build-up a good foundation before further topics on Network Security are covered.

Part 2 of this assessment aims to establish a basic familiarity with the cryptographic methods and provides an exercise of key establishment and secure communication in a networked environment.

Objectives

On completion of this assignment you should be able to:

- Understand some basic concepts in cryptography and networking
- Understand key transport and secure communication.
- Understand network programming.
- Applying AES and Key Exchange concepts.

Part 1 – Quiz 1

Online Quiz 1 – Computer Networks Revisited

- The quiz consists of 10 MCQs each. You are given only 20 minutes to complete the quiz. Refer to Blended Learning Plan on Moodle.

Part 2 – Key Exchange Program Development:

Specifications

Write C++ UDP or TCP programs allowing two parties to establish a secure communication channel. For simplicity, let us call the programs "Host" and "Client"; each can be used by a user. Again, for simplicity, let us assume that Alice uses Host and Bob uses Client.

Alice and Bob want to establish a secure communication channel where messages are encrypted with **AES encryption** in which operates on **128-bit blocks** using a **192-bit key**.

The **key establishment** is done by using modified Diffie-Hellman Exchange scheme (modified for this assignment). Alice has a pair of private/public keys (x1, y1) and Bob has a pair of private/public keys (x2, y2), generated by your program via **KeyGen**. By key exchange, they obtain a share secret key, which is a **192 bit AES session key.** The PKC that handles both public key and private key should be **RSA**.

Place Host and Client in two separate directories: Alice and Bob. Alice's keys are stored in a file located at her directory (Alice) and Bob's keys are stored in a file located at his directory (Bob).

The protocol is described as follows:

1. Alice runs **KeyGen** to generate a pair of her private and public keys including all required parameters based on **RSA** requirements. These keys and parameters are stored in directory Alice.
2. Bob runs **KeyGen** to generate a pair of his private and public keys including all required parameters based on **RSA** requirements. These keys and parameters are stored in directory Bob.
3. Alice executes Host.
    a. Host is running and listening to the opened port (you need select a port for your code).

    b. Preferable that the moment the user execute Host program, the program will ask the user to setup the IP and a port number.
4. Bob executes Client.
    a. Preferable that the moment the user execute Client program, the program will ask the user identify the IP and a port number to connect to (which is the Host executed earlier)
    b. Client (Bob) sends his **public key y2** to Host (Alice) together with Hash of this key in SHA-1 format.
    c. Client is ready and listens to the port for next response.
5. Upon receiving the public key and SHA-1 hash from Bob, Alice first verify the integrity of the Public key. Alice then send a "Verified" message to Bob.
6. Alice then computes/generates the **192 bit AES session key and SHA-1 hash of this session key**. This session key is unique for each runs of the program.
7. Alice sends **her public key y1 + SHA-1 hash** to Bob (Client) – expose. Then she send **encrypted 192 bit AES session key + SHA-1 hash** with **Bob Public Key y2**.
8. Bob received the above components. He verify Alice public key y1 with the SHA-1 hash.
9. Then Bob decrypt the encrypted components of 192 bit AES session key + SHA-1 hash using **his Private Key x2**.
10. As for confirmation of receiving the session key, Bob encrypt back a message of "Acknowledge" with the **192 bit AES session key** and send back to Alice. (note: only when step 8 and 9 above are verified with their respective Hash).
11. Upon receiving the encrypted message, Alice Decrypt it (using **her local copy of 192 bit AES session key).** If the message represent **"Acknowledge"** she knows that the secure communication now can take place. If match, Alice send a signal message "Ready" encrypted using the AES session key to Bob.
12. Once the message "Ready" sent to Bob, **Alice reverse the 192 bit AES key** to be use for the next secured communication.
13. Bob, upon receiving this Encrypted message "Ready", he decrypts it and identifies this signal. With this, Bob now, **reverse the 192 bit AES key**.
14. Now, the secure channel is established.
    a. Either Alice or Bob can send a message encrypted with the **reverse 192 bit AES session key** (from the original key) now. They type the message on their own terminal. The message is encrypted by the program (Host or Client) and sent out.
    b. All message must use **Cipher Feed Back Mode (CFB) mode of block Cipher**.
    c. The received encrypted message and decrypted message are displayed on the screen.
    d. Either one Host or Client can quit the program by using "exit".

Questions:

Attempt these questions either by answering it in your report or directly implement in your code.

What happen if acknowledgment messages such as "verified", acknowledge" and "ready" never arrives or corrupted during transmission?

What happen if connection is terminated half-way?

How to ensure the program continue running and "catch" this error?

Suggest the best way to tackle the above problem. (**Note**: there are multiple strategies or combination of strategies for the above questions)

Note:

For presentation/demo purposes, each steps of generating Private, Public key by RSA, the 192 bit AES session key, The matching process for acknowledgement and the reverse of 192 bit AES session key should be displayed on your terminal. This is to verify that your client and server program conform with all the requirements stated above.

Coding requirement:

You need to write (or re-use from library) four functions:

1. KeyGen for RSA Public/Private keys
2. **192 bit AES session key** generation.
3. **Reversing process of 192 bit session key.**
4. **AES** encryption/Decryption using **reverse of 192 bit AES session key** for secure communication.
5. **CFB mode**
6. Host.
7. Client.

How to run during presentation?

Your programs should run according to the protocol mentioned above. Host and Client should be executed on different computers. You need a pair of computers. I suggested that you execute the Host program, and the Client program on your friend's computer (or computer available in the lab – if required). For simplicity, there is no GUI required in this assignment. That is, messages are simply typed on the terminal and printed on the receiver's terminal.

Remember, during secure communication between Alice and Bob, displayed text message should be as a pair, Encrypted message and the decrypted message. The looping should continue until the moment the user chooses to exit with some sort of termination command.

Presentation day and slot will be announced by your lecturer.

Submission:

- Part 1 – Both Quiz 1 is on-the-spot quizzes during Blended Hours.
- Part 2 – Submission is on or before the end of Week 7 via Submission link on Moodle

Part 2 Submission requirements:

You are required to prepare a report on your program development (part 2), the execution (with screen captures) and simple testing done. The softcopy of the report together with the program (source files) are required to be in a Zipped folder.

Plagiarism

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

Contents

Extra Miles

Attempt these questions either by answering it in your report or directly implement in your code.
What happen if acknowledgment messages such as “verified”, acknowledge” and “ready” never arrives or corrupted during transmission?
What happen if connection is terminated half-way?
How to ensure the program continue running and “catch” this error?
Suggest the best way to tackle the above problem. (**Note**: there are multiple strategies or combination of strategies for the above questions)

My recommendation is that if “Ready” or “Acknowledge” never arrives, in probably 30-64 seconds the program will quit or decide to restart the handshake process starting from KeyGen(). Definitely terminal the program will be the best option, we never knew how attacker can think out of box
But what if Acknowledge or Verified are ‘corrupted during transmission’, in my program after Server or Client have successfully decrypted the AES key and AES IV. Client will send ‘Acknowledge text’ in Encrypted format to recipient but I prefer to increase the complexity with certain algorithm probably we make use of *timetamp* + *username* + *destination port* + *some public key elements* + *Ready or Acknowledge text* concatenate and generate into a ‘SHA1 digest == 40 lengths long’ so that it make Attack difficult to ‘CRAFT’ the message, since hash is a one-way encryption algorithm. Attackers run at a blackbox meaning attackers have no prior knowledge about the source code. In other word, attackers doesn’t know how the ‘HASH’ are being made.

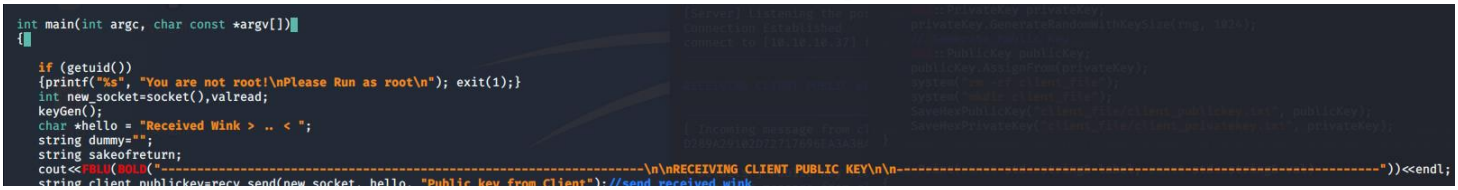
If any connection is terminated half-way, program will required to restart the process starting from ‘listening on a port’ and ‘connect to a port’. We never know what can be done by attackers, if you heard about ‘Socket Reuse’ that attacker can reuse back the socket to start his own ‘taking’ .

To catch the program, there definitely a ‘try and catch’ function. Mainly start from received buffer, let say we know that SHA1 length always 40, if we received 39 lengths but not 40 lengths. Program will be terminate, because the ‘content’ might have intercept by someone else.

Alice representing Server
Bob representing Client

Alice executes Host.

Alice runs **KeyGen** to generate a pair of her private and public keys including all required parameters based on **RSA** requirements. These keys and parameters are stored in directory Alice.



At the very first beginning of my server.cpp it will check is the program is running as root or not, if it's not running as root privilege the program will quit.
After finished verify the root access, we will start to run 'port binding' which will allow client to connect our server.
Code below is to "start a server listening on a 'specific' port". In our scenario we will be listening on port '4444'.
Lastly, keygen(), which will allow us to generate our 'unique KEY PAIR'. Then we will do some display and declare some variable for future.

```
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,peer_addr;
    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);
        }

        char *ip;

                    printf("Connection Established\n");

        char host[NI_MAXHOST];      // Client's remote name
        char service[NI_MAXSERV];   // Service (i.e. port) the client is connect on

        memset(host, 0, NI_MAXHOST); // same as memset(host, 0, NI_MAXHOST);
        memset(service, 0, NI_MAXSERV);

        if (getnameinfo((sockaddr*)&address, sizeof(address), host, NI_MAXHOST, service, NI_MAXSERV,
0) == 0)
        {
            string ls = GetStdoutFromCommand("ifconfig eth0 | grep -w inet | awk '{ print $2}'");
            int a =ls.length();
            std::string str2 = ls.substr (0,a-1);     // "think"
            cout <<"connect to ["<<str2 <<"] from (UNKNOWN) ["<<host<<"] "<< service << endl;
        }
            return new_socket ;
    }
```

```
File  Actions  Edit  View  Help
kali@kali:~/Desktop$ ./server
You are not root!
Please Run as root
kali@kali:~/Desktop$ sudo ./server
[sudo] password for kali:
Enter port number to start the listener
4444
[Server] Listening the port 4444 successfully.
```

Void keygen()

```
void keyGen()
{
    AutoSeededRandomPool rng;
    InvertibleRSAFunction privkey;
    privkey.Initialize(rng, 1024);
    // Generate Private Key
    RSA::PrivateKey privateKey;
    privateKey.GenerateRandomWithKeySize(rng, 1024);
    // Generate Public Key
    RSA::PublicKey publicKey;
    publicKey.AssignFrom(privateKey);
    system("sudo rm -rf server_file");
    system("mkdir server_file");
    SaveHexPublicKey("server_file/server_publickey.txt", publicKey);
    SaveHexPrivateKey("server_file/server_privatekey.txt", privateKey);
}
```

We create 1024 bits of public key, we ran *sudo rm-rf server_file* to clean the directory and sub-directory. Next will run *mkdir server_file* to create our "server_file" directory which will store our *publicKey* and *privateKey.*
Code below is code relating how we store our data from *raw* into *HexMessages* and store into *.txt* file.

```
void Save(const string& filename, const BufferedTransformation& bt)
{
    FileSink file(filename.c_str());
    bt.CopyTo(file);
    file.MessageEnd();
}

void SaveHex(const string& filename, const BufferedTransformation& bt)
{
    HexEncoder encoder;
    bt.CopyTo(encoder);
    encoder.MessageEnd();
    Save(filename, encoder);
}

void SaveHexPrivateKey(const string& filename, const PrivateKey& key)
{
    ByteQueue queue;
    key.Save(queue);
    SaveHex(filename, queue);
}


void SaveHexPublicKey(const string& filename, const PublicKey& key)
{
    ByteQueue queue;
    key.Save(queue);
    SaveHex(filename, queue);
}
```

Bob executes Client.

Bob runs **KeyGen** to generate a pair of his private and public keys including all required parameters based on **RSA** requirements. These keys and parameters are stored in directory Bob.

From client perspective, we declare some variables that related to 'listening buffer/messages from server'. Next after finished generating the key, the Client's *publickey* and *privatekey* are stored in *client_file.* Then it will encode the private key into '*Base64*' just for '***display purpose***'. Next it will go into "grabfilecontent" function to extract client's public from "*client_file/client_publickey.txt"*

```
int main()
{
    int sock=socket(),valread;
    char buffer[1024] = {0};
    string receive="Received Winked From Client";
    cout<<FRED(BOLD("[System] RSA Key is generating"))<<endl;
    cout<<FRED(BOLD("----------------------Sending Public-Key and Checksum----------------------"))<<endl;
    keyGen();
    string privatekey=grabprivatekey("client_file/client_privatekey.txt");

    string encoded;

    StringSource ss(privatekey,true,
        new Base64Encoder(
            new StringSink(encoded)
        ) // Base64Encoder
    ); // StringSource
    cout << "Private Key for Client in Base64"<<endl;
    cout<<"----BEGIN RSA PRIVATE KEY----"<<endl;
    cout << encoded;
    cout<<"----END RSA PRIVATE KEY----\n"<<endl;
    encoded= "";

    string publickey=grabfilecontent("client_file/client_publickey.txt");
    StringSource sss(publickey, true,
        new Base64Encoder(
            new StringSink(encoded)
        ) // Base64Encoder
    ); // StringSource

    cout<<"----BEGIN PUBLIC KEY----"<<endl;
    cout << encoded;
    cout<<"----END PUBLIC KEY----\n"<<endl;
    cout<<FRED(BOLD("Value of Public Key in Hex format"))<<endl;
```

String grabprivatekey(), which will return us the content of the file

```
string grabprivatekey(string filename)
{
 string inputdata,totaldata;
 ifstream file (filename);
  if (file.is_open())
  {
      getline (file,inputdata);
          file.close();
      return inputdata;
  }
}
```

string grabfilecontent

```
string grabfilecontent(string filename)
{
 string inputdata,totaldata;
 ifstream file (filename);
  if (file.is_open())
  {
int counter=0;
   while(getline (file,inputdata))
    {
                totaldata=totaldata+inputdata+"\n";
    }
   file.close();
      return totaldata;
  }
}
```

Difference between these two is ‘\n’.

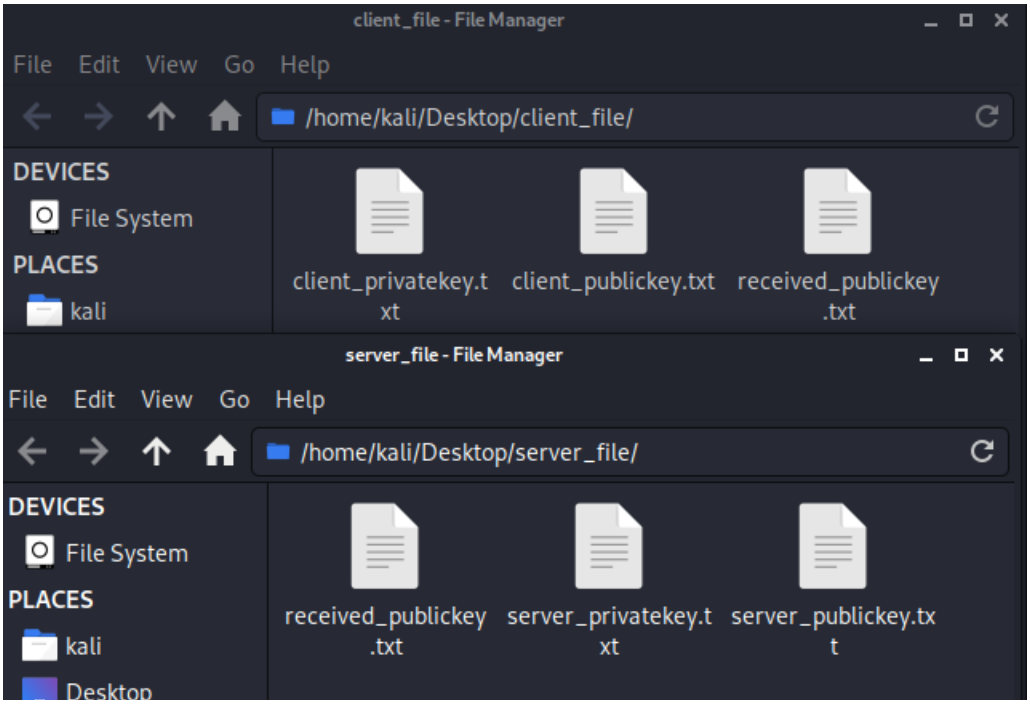
Take a look on client’s socket()

```
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;
}
```

After we start finished executing our client and *ENTER* we able to see that server and client has *successfully established the connection.* From the server perspective we can see that ***we got a connection from (UNKNOWN) localhost from source port 40842.***

```
kali@kali:~/Desktop$ ./server
You are not root!
Please Run as root
kali@kali:~/Desktop$ sudo ./server
[sudo] password for kali:
Enter port number to start the listener
4444
[Server] Listening the port 4444 successfully.
Connection Established
connect to [10.10.10.37] from (UNKNOWN) [localhost] 40842
------------------------------------------------------------------

RECEIVING CLIENT PUBLIC KEY

------------------------------------------------------------------
[ Incoming message from client ] : 30819D300D06092A864886F70D010101050003818B0030818702818100AA64057E4EBCDFF90FACCDB1F43FECE048
D289A29102D72717696EA3A38A4C41D88BCF160338DB3A955DC7A1C9175BDF75E9662F22610A5674850F133E54636E4F6E8F049418D991BB57020111

[ Details ] Public key from Client
[ Incoming message from client ] : 560E5682D27FD8603A3249FEF318F3B0F67A408A
[ Details ] Hash value from Client
--------------------[Received and Verifying the integrity]---------------------

Digest: 560E5682D27FD8603A3249FEF318F3B0F67A408A


kali@kali:~/Desktop$ ./client
Enter Server IP ADDRESS
127.0.0.1
Enter port number
4444
[System] RSA Key is generating
--------------------Sending Public-Key and Checksum--------------------
Private Key for Client in Base64
----BEGIN RSA PRIVATE KEY----
MzA4MjAyNzMwMjAxMDAzMDBEMDYwOTJBODY0ODg2RjcwRDAxMDEwMTA1MDAwNDgyMDI1RDMw
ODIwMjU5MDIwMTAwMDI4MTgxMDBBQTY0MDU3RTRFQkNERkY5MEZBQ0NEQjFGNDNGRUNFMDQ4
NjQ0RUFCRkExNjFFQUQzNkM4QjA5QUYxRjc0Q0FDMUYzNDk4OENDNzI4RTBGNTExQTdGREQz
MUMzOUU3QTVEOEMzOTY2QjVDMTRBNjY4MzFFNDlCNDY2MEU1NjRDMjI4QTY0OUI5MTZGRTE1
RDI4OUEyOTEwMkQ3MjcxNzY5NkVBM0EzOEE0QzQxRDg4QkNGMTYwMzM4REIzQTk1NURDN0Ex
QzkxNzVCREY3NUU5NjYyRjIyNjEwQTU2NzQ4NTBGMTMzRTU0NjM2RTRGNkU4RjA0OTQxOEQ5
```

After both of them has successfully establish the connection, their directory will store a few .txt which contain their own keys and sender's publickey in *HEX format.*



Client (Bob) sends his **public key y2** to Host (Alice) together with Hash of this key in SHA-1 format.

Client is ready and listens to the port for next response.

We now will send the publickey and it digest to Server over the network.

```
cout<<"----END PUBLIC KEY----\n"<<endl;
cout<<FRED(BOLD("Value of Public Key in Hex format"))<<endl;
cout<<publickey<<endl;
send_recv(sock, publickey, "Public Key have sent");
string publicmd5=sha1string(grabfilecontent("client_file/client_publickey.txt"));
cout<<FRED(BOLD("\n[SHA -1 ]PUBLIC KEY : "));
cout<<publicmd5<<"\n"<<endl;
send_recv(sock, publicmd5, "Hash value \"SHA - 1\" of public key sent");
cout<<"Waiting Integerity Verified from Server"<<endl;
```

We will take grab the content of publickey from the '*client_file/client_publickey.txt*' using the grabfilecontent function and go into *sha1string* function

```
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;
}
```

The value will be return and store into a variable named "publicmd5", we take this both value 'publickey' and 'sha1 digest' go into send_recv function.

```
string send_recv(int socket, string message, string comments)
{
int valread;
char buffer[1024] = {0};
send(socket , message.c_str() , strlen(message.c_str())+1 , 0 );
cout<<"[ Successfully send to Server ] "<<comments<<endl;
valread = read(socket , buffer, 1024);
cout<<"[ Incoming message from server ] : ";
printf("%s\n",buffer );
return buffer;
}
```

This *send_recv* function will be sending message 1st and receive message after finish sending messages over the network.

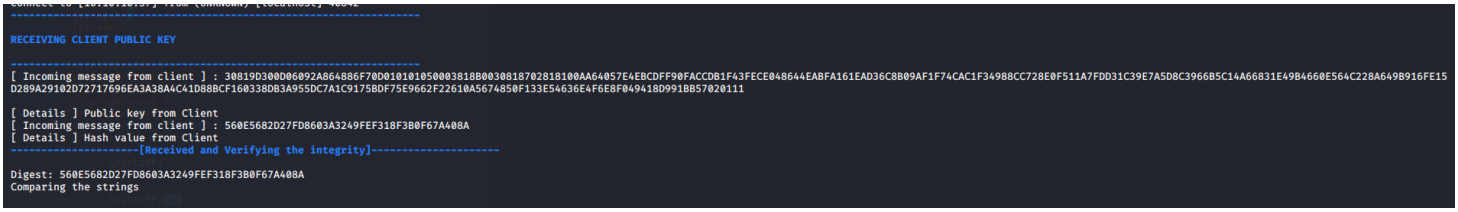
```
4444
[System] RSA Key is generating
--------------------Sending Public-Key and Checksum--------------------
Private Key for Client in Base64
----BEGIN RSA PRIVATE KEY----
MzA4MjAyNzMwMjAxMDAzMDBEMDYwOTJBODY0ODg2RjcwRDAxMDEwMTA1MDAwNDgyMDI1RDMw
ODIwMjU5MDIwMTAwMDI4MTgxMDBBQTY0MDU3RTRFQkVERkY5MEZBQ0NEQjFGNDNGRUNFMDQ4
NjQ0RUFCRkExNjFFQUQzNkM4QjA5QUYxRjc0QUFDMUYzNDk4OENDNzI4RTBGNTExQTdGREQz
MUM2OUU3QTVEOEM2OTY2QjVDMTRBNjY4M2FFNDlCNDY2MEU1NjRDMjI4QTY0OUI5MTZGRTE1
RDI4OUEyOTEwMkQ3MjcxNzY5NkVBM0EzOEE0QzQxRDg4QkNGMTYwMzM4REIzQTk1NURDN0Ex
QzkxNzVCREY3NUU5NjYyRjIyNjEwQTU2NzQ4NTBGMTMzRTU0NjM2RTRGNkU4RjA0OTQxOEQ5
OTFCQjU3MDIwMTExMDI4MTgwMTkwRUI1ODM4NDBDQjc4NjgyNEUxRTNGRDExODc1QTg4MzFE
Q0Y1NTg2QThFNjY0QzQ0QUIwOEY0MUIzNkQyODVFRjEyNTdFMUQ0OUM2QkFBMEI2RDI4MEVE
OTAwQjdBNDU4NjJDQjU2RjZDNzJEMzE2NjU2MjIwRjAyMUJCRkUzMUU1QjYyNEZBNjAzMkFF
MzU4ME03MDUwNzZDMkI1MEM0MDFCNTcxRkE5RTdCQTMzQjJENzYwMEM1Q0VGMDNFQUI4QTIx
MDE3NkNDRjNGOUMxMTU5QzBCNENCMzczOEFGQTQ5RkE0MzIxREM4M0FCNkQ2M0Y1QjhCQTY0
MEIyOTAyNDEwMEM3N0NGRjNERkY5QTVCNkE4MTRDNDgxMUEzMDE2RENFQjZCRTgyNDAyNjY1
MUMwNTZCMjQ1MzI1RTc1REUyRERCRTJGQkU2NDU3MTdFNzA0NjVDMEM0QTMxQ0J6QzdGNjI5
QUNBMkE5OUY0QzFCOTBCM0IyRUMzRTEyMDNBRkE5MDI0MTAwREFBOERBNEM3MEQ1OTRGQTlD
ODNDNjAyMzRERkI2NDI0RkQ5ODJBOTdEQTcwRDYzOTc4OTVBNzQxOTJBNTBENUUxNjA1RDc1
RENFRjdBNDJBQzQxRDIzREU0MDU5QURERDkwOTI5MTU3Nzc5RTdCNjhEMTA1NDI1RDQxOEYy
RkYwMjQwMTc3ODFFMDc0QjNGNTYwQzg3QUU5RjExMjIzQzY3NDU3RUU5M0M4MDA0ODQ1REE2
NDhENzE4RDc0ODY1NjVGQkY4NDFEQTI5RUMyMEVFMDA4NDcxMzMwOEYwODc3NTAzMjZE
N0Q1RTA4RjQyRjI0MzMyQURBMjAzQ0FCNDEwMjQwMjY5NjQ0QTQxM0U5NzRBNEIyMzU1RjJE
OTBEQzJGMzhFMEVBMjYxREU4RkY1Q0I3MzhEQzAwRTc0RkJDMkM2MUZBOTg4OEY2QUU4NDlE
MUFEMzFBQUNBMTgyOTc5M0NDQ0JGMjhFQzc4RDhERkJCNkNEOTk3ODQyRTkzMTk0NEIwMjQw
NTgyNkI4M0Q1QjI4MjJDMTU1REEyMzVBOTk1QTY5Q0Y2RjNFOEYzMjBEMTRDMkMzRDlEQjU3
QTRGMTdDNzI3MDVEOEQzMTQzQjEyM0JCMEM0NzA2NzYzMUEzMzUxRjYzOEU5QjlGMTYwNzA1
MUIwMUQ5MjlBQzhEODE4MzFGQzM=
----END RSA PRIVATE KEY----

----BEGIN PUBLIC KEY----
MzA4MTlEMzAwRDA2MDkyQTg2NDg4NkY3MEQwMTAxMDEwNTAwMDM4MThCMDAzMDgxODcwMjgx
ODEwMEFBNjQwNTdFNEVCQ0RGRjkwRkFDQ0RCMUY0M0ZFQ0UwNDg2NDRFQUJGQTE2MUVBRDM2
QzhCMDlBRjFGNzRDQUMxRjM0OTg4Q0M3MjhFMEY1MTFBN0ZERDMxQzM5RTdBNUQ4QzM5NjZC
NUMxNEE2NjgzMUU0OUI0NjYwRTU2NEMyMjhBNjQ5QjkxNkZFMTVEMjg5QTI5MTAyRDcyNzE3
Njk2RUEzQTM4QTRDNDFEODhCQ0YxNjAzMzhEQjNBOTU1REM3QTFDOTE3NUJERjc1RTk2NjJG
MjI2MTBBNTY3NDg1MEYxMzNFNTQ2MzZFNEY2RThGMDQ5NDE4RDk5MUJCNTcwMjAxMTEK
----END PUBLIC KEY----

Value of Public Key in Hex format
30819D300D06092A864886F70D010101050003818B0030818702818100AA64057E4EBCDFF90FACCDB1F43FECE048644EABFA161EAD36C8B09AF1F74CAC1F34988CC728E0F511A7FDD31C39E7A5D8C3966B5C14A66831E49B4660E564C228A649B916FE15D289A29102D72717696EA3A38A4C41D88BC
F160338DB3A955DC7A1C9175BDF75E9662F22610A5674850F133E54636E4F6E8F049418D991BB57020111

[ Successfully send to Server ] Public Key have sent
[ Incoming message from server ] : Received Wink > .. <

[SHA -1 ]PUBLIC KEY : 560E5682D27FD8603A3249FEF318F3B0F67A408A

[ Successfully send to Server ] Hash value "SHA - 1" of public key sent
[ Incoming message from server ] : Received Wink > .. <
Waiting Integerity Verified from Server
```

Upon receiving the public key and SHA-1 hash from Bob, Alice first verify the integrity of the Public key. Alice then send a “Verified” message to Bob.

Now ‘Alice == Server’ will be receiving ‘PublicKey’ and ‘it digest’ value from Client.

Next it will run ‘*recv_send*’ function

It mainly receive message from opponent/Client and response back to Server. The response message is “*Received Wink > .. <*” After received both messages (PublicKey and Hash) from Client, we will run a *comparison* to making sure the ‘*value of digest that we received from client*’ is matched with (“*the publickey received and hash it*”)



After finished the verify function,

```
void verify(string a, string b)
{

int result = strcmp(a.c_str(), b.c_str());
cout<<"Comparing the strings"<<endl;
if(result==0)
{
        cout<<FRED(BOLD("Verified"));
}
else
{
        cout<<FRED(BOLD("File is tampered"))<<endl;
        exit(0);
}
}
```

If it's matched, it will go into receiveddummy(new_socket) function which is just to received message or dummy message from Client because in socket programming it have to be "**Send and Received**" and cannot be "**Send Send and Recieved**". **Else program quit.**

```
string receivedummy(int new_socket)
{
        char buffer[1024] = {0};
        string compare;
        int valread = read( new_socket , buffer, 1024);
        int b_size = sizeof(buffer) / sizeof(char);
        string s_b = convertToString(buffer, b_size);
        received_dummy=s_b;
        return s_b;
}
```

After,we have received *dummy_messages* from Client, we will send Verified message over network to Client.

```
RECEIVING CLIENT PUBLIC KEY

[ Incoming message from client ] : 30819D300D06092A864886F70D010101050003818B0030818702818100AA64057E4EBCDFF90FACCDB1F43FECE048644EABFA161EAD36C8B09AF1F74CAC1F34988CC728E0F511A7FDD31C39E7A5D8C3966B5C14A66831E49B4660E564C228A649B916FE15
D289A29102D72717696EA3A38A4C41D88BCF160338DB3A955DC7A1C9175BDF75E9662F22610A5674850F133E54636E4F6E8F049418D991BB57020111

[ Details ] Public key from Client
[ Incoming message from client ] : 560E5682D27FD8603A3249FEF318F3B0F67A408A
[ Details ] Hash value from Client
----------[Received and Verifying the integrity]----------

Digest: 560E5682D27FD8603A3249FEF318F3B0F67A408A
Comparing the strings
Verified
Hold on, while we're sending our result to Client
Sent
```

Client perspective, it state that they received a response from Server and if it's matched then Continue to compare the strings else the program will quit.

```
[SHA -1 ]PUBLIC KEY : 560E5682D27FD8603A3249FEF318F3B0F67A408A

[ Successfully send to Server ] Hash value "SHA - 1" of public key sent
[ Incoming message from server ] : Received Wink > .. <
Waiting Integerity Verified from Server
We recevied a response from Server
Comparing the strings
Verified
```

Alice then computes/generates the **192 bit AES session key and SHA-1 hash of this session key**. This session key is unique for each runs of the program.

```
string privatekey=grabprivatekey("server_file/server_privatekey.txt");
string publickey=grabfilecontent("server_file/server_publickey.txt");

string encoded;

 StringSource ss(privatekey,true,
     new Base64Encoder(
         new StringSink(encoded)
     ) // Base64Encoder
 ); // StringSource
 cout << "Private Key for Server in Base64"<<endl;
 cout<<"----BEGIN RSA PRIVATE KEY----"<<endl;
 cout << encoded;
 cout<<"----END RSA PRIVATE KEY----\n"<<endl;
 encoded= "";

 StringSource sss(publickey, true,
     new Base64Encoder(
         new StringSink(encoded)
     ) // Base64Encoder
 ); // StringSource

 cout << "Public Key for Server in Base64"<<endl;
 cout<<"----BEGIN PUBLIC KEY----"<<endl;
 cout << encoded;
 cout<<"----END PUBLIC KEY----\n"<<endl;


 cout<<FRED(BOLD("Value of Public Key in Hex format"))<<endl;
 receivedummy(new_socket);

 cout<<publickey<<endl;
 send_recv(new_socket, publickey, "Public Key have sent");
 string publicsha1=sha1string(grabfilecontent("server_file/server_publickey.txt"));
 cout<<FRED(BOLD("\n[SHA -1 ]PUBLIC KEY : "));
 cout<<publicsha1<<"\n"<<endl;
 send_recv(new_socket, publicsha1, "Hash value \"SHA - 1\" of public key sent");
 cout<<"Waiting Integerity Verified from Client"<<endl;

 sendpacket(new_socket,"dummy");
 receivedummy(new_socket);
 cout<<"We recevied a response from Server"<<endl;
 verify("Verified",received_dummy);

 cout<<FRED(BOLD("\n------------------Complete sending publickey--------------------"))<<endl;
 cout<<FBLU(BOLD("------------------Generating AES Session Key--------------------"))<<endl;
 generateAES_session();
 cout<<FBLU(BOLD("----------------------------------------------------------------"))<<endl;
```

After finished executing server's previous code, we reach to extract our publickey and privatekey from "*server_file/server_publickey.txt*" & "*server_file/server_privatekey.txt*" into 2 different variables, same as Client, we encode it into Base64 just for "**Display purpose**" and we will received dummy packet from receivedummy() function, as I mentioned previously "*the structure of socket programming, send and received*".
Then we will run "send recv()" function to send Server's publickey and hashes over the network to Client.

```
string send_recv(int socket, string message, string comments)
{
int valread;
char buffer[1024] = {0};
send(socket , message.c_str() , strlen(message.c_str())+1 , 0 );
cout<<"[ Successfully send to Client ] "<<comments<<endl;
valread = read(socket , buffer, 1024);
cout<<"[ Incoming message from client ] : ";
printf("%s\n",buffer );
return buffer;
}
```

Which is send message over network and received response from Client.

After finished the sending process
We will jump to generate “AES Key” in *generateAES_session()*

```
void generateAES_session()
{
        AutoSeededRandomPool prng;
        string encoded;
        //24 byte, 24*8 = 192bits
        byte key[24];
        prng.GenerateBlock(key, sizeof(key));

        // byte iv[AES::BLOCKSIZE];
        byte iv[24];
        prng.GenerateBlock(iv, sizeof(iv));

        // Pretty print key
        encoded.clear();
        StringSource(key, sizeof(key), true,
                                new HexEncoder(
                                        new StringSink(encoded)) // HexEncoder
        );                                                                  // StringSource
        cout << "Key: " << encoded << endl;
        cout <<"Key length in Hex Format "<<encoded.length()<<endl;
        IV_session_digest=sha1string(encoded);
        AES_session=encoded;

        // Pretty print iv
        encoded.clear();
        StringSource(iv, sizeof(iv), true,
                                new HexEncoder(
                                        new StringSink(encoded)) // HexEncoder
        );                                                                  // StringSource
        cout << "\n\nIV: " << encoded << endl;
        cout <<"IV length in Hex Format "<<encoded.length()<<endl;
        IV_session_digest=sha1string(encoded);
        cout<<endl;
        IV_session=encoded;
}
```

Take note , IV_session and AES_session are **global variable, I will attach all the relevant library and global variable at the end of section.** After finish executed the generateAES_session(), the AES KEY and IV are store in IV_session and AES_session variables.

Alice sends her public key y1 + SHA-1 hash to Bob (Client) – expose. Then she send encrypted 192 bit AES session key + SHA-1 hash with Bob Public Key y2.

```
    cout<<FBLU(BOLD("------------------Generating AES Session Key--------------------"))<<endl;
    generateAES_session();
    cout<<FBLU(BOLD("----------------------------------------------------------------"))<<endl;
//////////////////ENCRYPT AND SEND////////////////////////////
    string AES_Sess_Key=Encrypt_AES(contentofpublickeytoverify,AES_session,"SESSION KEY");
    cout<<BOLD(FYEL("\n\nSending Encrypted Session Key to Client"))<<endl;
    send_recv(new_socket,AES_Sess_Key,"Encrypted AES Session Key");
////////////////////////////////////////////////////////////
////////SEND ENCRYPTED AES SESSION'S CHECKSUM SESSION//////
    cout<<BOLD(FYEL("\n\nSending SHA - 1 FROM ENCRYPTED Session Key to Client"))<<endl;
    send_recv(new_socket,sha1string(AES_Sess_Key),"SHA-1 digest from Encrypted Session Key");

////////////////////////////////////////////////////////////
////////SEND PLAIN AES SESSION'S CHECKSUM SESSION///////////
    cout<<BOLD(FYEL("\n\nSending SHA - 1 FROM PLAIN Session Key to Client"))<<endl;
    send_recv(new_socket,sha1string(AES_session),"SHA-1 digest from Plain Session Key");

    cout<<BOLD(FYEL("----------------------------------------------------------------"))<<endl<<endl;
```

After we get the value of “KEY” and “IV” We will encrypting our KEY using Encrypt_AES(), which using Client’s public key to run the encryption process on the “AES KEY” which we get a ‘cipher text’ which is encryptedkey. We will be sending 3 type of content over network

1.Encrypted AES Key X + K = Y

2.Hash of Encrypted AES Key H(Y)

3.Hash of AES Key H(P)

```
Value of Public Key in Hex format
30819D300D06092A864886F70D010101050003818B0030818702818100CBED2D03C7BE11696ED16D29920C44C67045B3BE12D5B7C7A8FC78406F9679EFF1D9AB22A086A109A29A97CAAED32B360993C1A2C65AC618C01A85234A468681E44CECF18C3309F368A59300FF401EC25B0E472666D96285A
A4EAB68E9278D614A8A14D1DC473DAD0C7C7B9118E9967616D74CE13074E1B91A3D9AE3657BE169020111

[ Successfully send to Client ] Public Key have sent
[ Incoming message from client ] : Received Winked From Client
Digest: 957250BEF0B19EAC96A276F38B0000CF30A416C7

[SHA -1 ]PUBLIC KEY : 957250BEF0B19EAC96A276F38B0000CF30A416C7

[ Successfully send to Client ] Hash value "SHA - 1" of public key sent
[ Incoming message from client ] : Received Winked From Client
Waiting Integerity Verified from Client
We recevied a response from Server
Comparing the strings
Verified
--------------------Complete sending publickey--------------------
--------------------Generating AES Session Key--------------------
Key: 3AB3A45B7B09AF857F9679E9BA29D658150CFFE0D3A6C611
Key length in Hex Format 48
Digest: AD82AB5E622DAF025FF5D28D5F7F7C4E51285551


IV: ABF67878A050D20420C17E326FAB7F414DB645DAED3031F3
IV length in Hex Format 48
Digest: 7B504F6AC02984CD5FA862F5C1ACB2528C3FCAE4

----------------------------------------------------------------------
```

```
----------------------------------------------------------------------
[-------ENCRYPTING SESSION KEY USING CLIENT PUBLIC KEY----------]
[Client-PublicKey]30819D300D06092A864886F70D010101050003818B0030818702818100AA64057E4EBCDFF90FACCDB1F43FECE048644EABFA161EAD36C8B09AF1F74CAC1F34988CC728E0F511A7FDD31C39E7A5D8C3966B5C14A66831E49B4660E564C228A649B916FE15D289A29102D727176
96EA3A38A4C41D88BCF160338DB3A955DC7A1C9175BDF75E9662F22610A5674850F133E54636E4F6E8F049418D991BB57020111
+
[PlainText AES_SESSION KEY]3AB3A45B7B09AF857F9679E9BA29D658150CFFE0D3A6C611
=
[ENCRYPTED]A38C6E5506297128C87D967CA4589D36C9CC08A8D09A6E2F99F92A1FFAB42E0AB982CB92C04429A19DE8FFC94E4C81AFE47B4FFC22C5ABF8CC8A70A50713732C12E67CB3960B2C20950B8D413F8BC1D01F4E1ECDDB4D09660AF569C90B9608D06FFB3F2C4E2005994D99B1CC5FB273F8
F83B8CC859301267CE9AACE5B600BA1B


Sending Encrypted Session Key to Client
[ Successfully send to Client ] Encrypted AES Session Key
[ Incoming message from client ] : Received Winked From Client


Sending SHA - 1 FROM ENCRYPTED Session Key to Client
Digest: 5BA6D1676434E4BFCF6D45738F028949877A2081
[ Successfully send to Client ] SHA-1 digest from Encrypted Session Key
[ Incoming message from client ] : Received Winked From Client


Sending SHA - 1 FROM PLAIN Session Key to Client
Digest: AD82AB5E622DAF025FF5D28D5F7F7C4E51285551
[ Successfully send to Client ] SHA-1 digest from Plain Session Key
[ Incoming message from client ] : Received Winked From Client
----------------------------------------------------------------------

[-------ENCRYPTING IV USING CLIENT PUBLIC KEY----------]
[Client-PublicKey]30819D300D06092A864886F70D010101050003818B0030818702818100AA64057E4EBCDFF90FACCDB1F43FECE048644EABFA161EAD36C8B09AF1F74CAC1F34988CC728E0F511A7FDD31C39E7A5D8C3966B5C14A66831E49B4660E564C228A649B916FE15D289A29102D727176
96EA3A38A4C41D88BCF160338DB3A955DC7A1C9175BDF75E9662F22610A5674850F133E54636E4F6E8F049418D991BB57020111
+
[PlainText AES_IV]ABF67878A050D20420C17E326FAB7F414DB645DAED3031F3
=
[ENCRYPTED]56F6A2D46350415D0D6B410601D064368EC99E7EA0B18952BFFC77836EF71B3E69E77EDFB4D699D8108C6A9A2FA30626C812E845BC9095FC341D7246A5163940470973 45BAE57FAC883CAE83B261BA54B585D85F155A0A2297A893E1AC5512CECC282D33C84944D2B11B910D030559E0
347E25AA6FAAB8972A57F326ACBC98A0


Sending Encrypted AES IV to client
[ Successfully send to Client ] Encrypted AES IV
[ Incoming message from client ] : Received Winked From Client


Sending SHA - 1 VALUE FROM ENCRYPTED AES IV to client
Digest: 842983E293461EE4F2BB24AC3A454C56748C7C44
[ Successfully send to Client ] SHA-1 digest from Encrypted IV Key
[ Incoming message from client ] : Received Winked From Client


Sending SHA - 1 VALUE FROM PLAIN AES IV to client
Digest: 7B504F6AC02984CD5FA862F5C1ACB2528C3FCAE4
[ Successfully send to Client ] SHA-1 digest from Plain IV Key
[ Incoming message from client ] : Received Winked From Client
----------------------------------------------------------------------
```

Bob received the above components. He verify Alice public key y1 with the SHA-1 hash.

```
    cout<<FRED(BOLD("\n--------------------Complete sending publickey--------------------"))<<endl;
    cout<<FBLU(BOLD("--------------------RECEIVING SERVER PUBLIC KEY--------------------\n\n"));
    string message="Send me your public key";
    sendpacket(sock,message);
    string serverpublickey=recv_send(sock, receive, "Public key from Server");//send received wink
    cout<<""<<endl;
    SaveContent(serverpublickey,"client_file/received_publickey.txt");                //store server public key
    string hashvalue=recv_send(sock, receive, "Hash value from Client"); // send received wink
    string contentofpublickeytoverify=grabfilecontent("client_file/received_publickey.txt");
    cout<<FBLU(BOLD("[--------------------Received and Verifying the integrity--------------------]"))<<endl;
    verify(hashvalue,sha1string(contentofpublickeytoverify));
    cout<<"\nHold on, while we're sending our result to Server"<<endl;
    receivedummy(sock);
    sendpacket(sock,"Verified");
    cout<<"Sent"<<endl;
    cout<<FRED(BOLD("\n--------------------------------------------------------------------"))<<endl;
    cout<<FBLU(BOLD("Receiving Encrypted AES Session Key from Server"))<<endl;
////////////////////////RECEIVE KEY//////////////////////////
    string Encrypted_AES_SESS_KEY=recv_send(sock,receive,"Encrypted AES Session Key");
    cout<<endl;
    string sha1_AES_SESS_KEY=recv_send(sock,receive,"VALUE OF SHA - 1 FROM \"ENCRYPTED\" AES SESSION KEY");
    cout<<endl;
    string sha1_AES_PLAIN_SESS_KEY=recv_send(sock,receive,"VALUE OF SHA - 1 FROM \"PLAIN\" AES SESSION KEY");
    cout<<endl;
////////////////////////////////////////////////////////////

    cout<<FBLU(BOLD("Receiving Encrypted AES IV from Server"))<<endl;

////////////////////////RECEIVE IV//////////////////////////
    string Encrypted_AES_IV=recv_send(sock,receive,"Encrypted AES IV");
    cout<<endl;
    string sha1_AES_IV=recv_send(sock,receive,"SHA - 1 FROM \"ENCRYPTED\"AES IV");
    cout<<endl;
    string sha1_PLAIN_AES_IV=recv_send(sock,receive,"SHA - 1 FROM \"PLAIN\"AES IV");
    cout<<endl;
////////////////////////////////////////////////////////////
    cout<<"Verifying the Intergrity of recieved content"<<endl;
    cout<<"Verifiying integrity of Encrypted AES Session"<<endl;
    verify(sha1_AES_SESS_KEY,sha1string(Encrypted_AES_SESS_KEY));
         ///VERFIYING ENCRYPTED AES IV
    cout<<"\nVerifiying integrity of Encrypted AES IV"<<endl;
    verify(sha1_AES_IV,sha1string(Encrypted_AES_IV));
    cout<<endl;

        cout<<FBLU(BOLD("Process of verifying is almost complete, 20% remaining left"))<<endl;

        ///START DECRYPT VERIFYING THE DECRYPTED HASH
    cout<<"\nProcess of decryption is running and \"Verifying PLAIN's AES SESSION hash\""<<endl<<endl;
    string HexSESSION=Decryption_PKI(Encrypted_AES_SESS_KEY,privatekey,"Session Key ⟶");
    verify(sha1string(HexSESSION),sha1_AES_PLAIN_SESS_KEY);

        ///START DECRYPT IV TO VERIFYING THE DECRYPTED HASH
    cout<<"\nProcess of decryption is running to \"Verifying PLAIN's AES IV\""<<endl<<endl;
    string HexIV=Decryption_PKI(Encrypted_AES_IV,privatekey,"    AES IV  ⟶");
    verify(sha1string(HexIV),sha1_PLAIN_AES_IV);

    cout<<"\nProcess of integrity checking is completed "<<endl<<endl;
```

After finished the process of sending Client's Public_Key to Server, run a function called "SaveContent()" to store the "Server's Public key" into a text file at "*client_file/received_publickey.txt*".

```
void SaveContent(string content, string filename)
{
        ofstream file;
        file.open (filename);
        file << content;
        file.close();
}
```

Then we will received the publickey's hashes, next will grabthecontent of the publickey from the textfile.
Run a *verify()* function

```
void verify(string a, string b)
{

int result = strcmp(a.c_str(), b.c_str());
cout<<"Comparing the strings"<<endl;
if(result==0)
{

        cout<<BOLD(FRED("Verified"));
}
else
{
        cout<<a<<endl;
        cout<<b<<endl;
        cout<<BOLD(FRED("NOT MATCH!"))<<endl;
        exit(0);
}
}
```

After finished the verifying process, we will be receivedummy() as I mentioned previously "*structure of socket programming*".
Then we will send a response to Server tell to Server that Client has successfully received and verified successfully the hashes with it encrypted key and it hashes.

Then Bob decrypt the encrypted components of 192 bit AES session key + SHA-1 hash using **his Private Key x2**.

Assume that Client has successfully verify the *Encrypted Key* with *Hash from Encrypted Key* && the *Encrypted IV* with *Hash from Encrypted IV*

Keys		HASH	
Y	✓	Y	✓
H(Y)	✓	H(Y)	✓
H(X)	?	H(X)	?

```
------------------------------------------------------------------
Receiving Encrypted AES Session Key from Server
[ Incoming message from server ] : A38C6E5506297128C87D967CA4589D36C9CC08A8D09A6E2F99F92A1FFAB42E0AB982CB92C04429A19DE8FFC94E4C81AFE47B4FFC22C5ABF8CC8A70A50713732C12E67CB3960B2C20950B8D413F8BC1D01F4E1ECDDB4D09660AF569C90B9608D06FFB3F2C
4E2005994D99B1CC5FB273F8F83B8CC859301267CE9AACE5B600BA1B
[ Details ] Encrypted AES Session Key

[ Incoming message from server ] : 5BA6D1676434E4BFCF6D45738F028949877A2081
[ Details ] VALUE OF SHA - 1 FROM "ENCRYPTED" AES SESSION KEY

[ Incoming message from server ] : AD82AB5E622DAF025FF5D28D5F7F7C4E51285551
[ Details ] VALUE OF SHA - 1 FROM "PLAIN" AES SESSION KEY

Receiving Encrypted AES IV from Server
[ Incoming message from server ] : 56F6A2D46350415D0D6B410601D064368EC99E7EA0B18952BFFC77836EF71B3E69E77EDFB4D699D8108C6A9A2FA30626C812E845BC9095FC341D7246A5163940470973345BAE57FAC883CAE83B261BA54B585D85F155A0A2297A893E1AC5512CECC282D33
C84944D2B11B910D030559E0347E25AA6FAAB8972A57F326ACBC98A0
[ Details ] Encrypted AES IV

[ Incoming message from server ] : 842983E293461EE4F2BB24AC3A454C56748C7C44
[ Details ] SHA - 1 FROM "ENCRYPTED"AES IV

[ Incoming message from server ] : 7B504F6AC02984CD5FA862F5C1ACB2528C3FCAE4
[ Details ] SHA - 1 FROM "PLAIN"AES IV

Verifying the Intergrity of recieved content
Verifiying integrity of Encrypted AES Session
Comparing the strings
Verified
Verifiying integrity of Encrypted AES IV
Comparing the strings
Verified
Process of verifying is almost complete, 20% remaining left
```

Hold on, we have not verify the HASH that came from plaintext. But we only have encrypted content so we need to run decryption on the encrypted content and take the decrypted content to hash it and compare with H(X).

```
    verify(sha1_AES_IV,sha1string(Encrypted_AES_IV));
    cout<<endl;

        cout<<FBLU(BOLD("Process of verifying is almost complete, 20% remaining left"))<<endl;

        ///START DECRYPT VERIFYING THE DECRYPTED HASH
    cout<<"\nProcess of decryption is running and \"Verifying PLAIN's AES SESSION hash\""<<endl<<endl;
    string HexSESSION=Decryption_PKI(Encrypted_AES_SESS_KEY,privatekey,"Session Key ⟶");
    verify(sha1string(HexSESSION),sha1_AES_PLAIN_SESS_KEY);

        ///START DECRYPT IV TO VERIFYING THE DECRYPTED HASH
    cout<<"\nProcess of decryption is running to \"Verifying PLAIN's AES IV\""<<endl<<endl;
    string HexIV=Decryption_PKI(Encrypted_AES_IV,privatekey,"    AES IV  ⟶");
    verify(sha1string(HexIV),sha1_PLAIN_AES_IV);

    cout<<"\nProcess of integrity checking is completed "<<endl<<endl;
```

We have a look at *Decryption_PKI()*, that will be decrypt our encrypted content and bring the decrypted content into hash and verify.

```
string Decryption_PKI(string encryptedcontent,string privKey,string title)
{
AutoSeededRandomPool rng;
InvertibleRSAFunction parameters;
parameters.GenerateRandomWithKeySize(rng,1024);
RSA::PrivateKey privateKey(parameters);
string decodedPrivKey;


////Load Private Key
StringSource ss2(privKey,true,(new HexDecoder( new StringSink(decodedPrivKey))));      //decode the privkey from hex to symbol stuff
StringSource PrivKeySS(decodedPrivKey,true);           //load it into bytes
privateKey.Load(PrivKeySS);             //load the private key

RSAES_OAEP_SHA_Decryptor d(privateKey);
string plaintext;
StringSource ss3(encryptedcontent ,true,(new HexDecoder (new PK_DecryptorFilter(rng, d, (new StringSink(plaintext))))));
cout<<"--------------------------------Decryption is in progress-------------------"<<endl;
cout<<BOLD(FRED("[ **"<< title <<" found* ]"));
cout<<plaintext<<"         |"<<" SHA-1 :";
cout<<sha1string(plaintext)<<endl;
cout<<"--------------------------------Process of decryption is completed-----------"<<endl<<endl;
return plaintext;
}

string send_recv(int socket, string message, string comments)
{
int valread;
char buffer[1024] = {0};
send(socket , message.c_str() , strlen(message.c_str())+1 , 0 );
cout<<"[ Successfully send to Server ] "<<comments<<endl;
valread = read(socket , buffer, 1024);
cout<<"[ Incoming message from server ] : ";
printf("%s\n",buffer );
return buffer;
}
```

It will take the “HexEncoded PrivateKey” run HexDecoder into raw and load into privatekey and decrypt on the encrypted content (“it can be KEY or IV”), It will go into HexDecoder first and only start the PK_DecryptorFiltering decryption and DONE, we have our plaintext.

Assume, everything went GOOD.

```
Process of verifying is almost complete, 20% remaining left

Process of decryption is running and "Verifying PLAIN's AES SESSION hash"

----------------------------------Decryption is in progress------------------
[ **Session Key ⟶ found* ]3AB3A45B7B09AF857F9679E9BA29D658150CFFE0D3A6C611        | SHA-1 :AD82AB5E622DAF025FF5D28D5F7F7C4E51285551
----------------------------------Process of decryption is completed----------

Comparing the strings
Verified
Process of decryption is running to "Verifying PLAIN's AES IV"

----------------------------------Decryption is in progress------------------
[ **    AES IV  ⟶ found* ]ABF67878A050D20420C17E326FAB7F414DB645DAED3031F3        | SHA-1 :7B504F6AC02984CD5FA862F5C1ACB2528C3FCAE4
----------------------------------Process of decryption is completed----------

Comparing the strings
Verified
Process of integrity checking is completed

-----------------------------------------------------------------
```

As for confirmation of receiving the session key, Bob encrypt back a message of "Acknowledge" with the **192 bit AES session key** and send back to Alice. (note: only when step 8 and 9 above are verified with their respective Hash).

```
cout<<FRED(BOLD("----------\n\nTRYING TO SEND ACKNOWLEDGE FLAG\n\n----------"))<<endl;
string nonevalue="";
AES_Encryption("Acknowledge",HexIV,HexSESSION,sock,false);
sendpacket(sock,dummy);
cout<<endl<<endl<<endl;
```

We declare a variable with "NULL" value inside.
Client will run an AES_Encryption using the "AES_IV" and "AES_KEY" that decrypt from the Encrypted IV and Encrypted Key that send from "Server".

Due to code too long, unable to screenshot. I have pasted it here as text format

```
void AES_Encryption(string temp, string AESiv, string AESkey,int socket,bool haha)
{
string plain;
if(haha==false)
{
        plain = temp;
}
else
{
do
{
cout<<"Enter message send to server"<<endl;
std::getline(std::cin, plain);
if(plain.size()>1024)
{
cout<<BOLD(FRED("Message is exceed the length"))<<endl;
}

}while(plain.size()>1024);
}


    string  recovered;
    AutoSeededRandomPool prng;
    string decodedkey,decodediv;
    StringSource s(AESkey, true,(new HexDecoder(
                new StringSink(decodedkey))
       ) // StreamTransformationFilter
      ); // StringSource
```

```
     StringSource ss(AESiv, true,(new HexDecoder(
                new StringSink(decodediv))
       ) // StreamTransformationFilter
      ); // StringSource

SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
SecByteBlock iv((const byte*)decodediv.data(), decodediv.size());


string cipher,encoded;

/*********************************\
\*********************************/
try
{
        cout << "plain text: " << plain << endl;
        CFB_Mode<AES>::Encryption e;
        e.SetKeyWithIV(key, sizeof(key), iv);

        // CFB mode must not use padding. Specifying
        //  a scheme will result in an exception
        StringSource(plain, true,
         new StreamTransformationFilter(e,
        new StringSink(cipher)) // StreamTransformationFilter
        );              // StringSource
        }
        catch (const CryptoPP::Exception &e)
        {
                cerr << e.what() << endl;
                exit(1);
        }

        /*********************************\
        \*********************************/
        // Pretty print
        encoded.clear();
        StringSource(cipher, true,
        new HexEncoder( new StringSink(encoded)
                ) // HexEncoder
        );  // StringSource
        cout << "cipher text: " << encoded << endl;
        cout<<"encoded length:" << encoded.length()<<endl;
       if(haha==false)
        {
        send_recv(socket,encoded , "Client send \"Received\" to Server");
        }
        else
        {
        sendpacket(socket,encoded);
        }
        if(plain=="quit")
        {
              cout<<FRED(BOLD("PROGRAM TERMINAL GRACEFULLY"))<<endl;
              exit(1);
        }
}
```

Under the *AES_Encryption()*

```
cout << "cipher text: " << encoded << endl;
cout<<"encoded length:" << encoded.length()<<endl;
if(haha==false)
{
send_recv(socket,encoded , "Client send \"Received\" to Server");
}
else
{
sendpacket(socket,encoded);
}
if(plain=="quit")
{
      cout<<FRED(BOLD("PROGRAM TERMINAL GRACEFULLY"))<<endl;
      exit(1);
}
```

```
AES_Encryption("Acknowledge",HexIV,HexSESSION,sock,false);
```

If haha == false, we will run “send_recv” function which we will send “Encrypted Acknowledge text ” over network,
at here, the variable haha is Boolean type, if false meaning this is before 3 way handshake , if it’s true meaning it’s after establish 3way handshake. We just want to reuse back the function.

```
------------------------------------------------------------------
TRYING TO SEND ACKNOWLEDGE FLAG
------------------------------------------------------------------
plain text: Acknowledge
cipher text: 3A759922AC05FAF20A8091
encoded length:22
[ Successfully send to Server ] Client send "Received" to Server
[ Incoming message from server ] : Server responed "Received Acknowledge", sending "Ready"
```

Upon receiving the encrypted message, Alice Decrypt it (using **her local copy of 192 bit AES session key).** If the message represent **“Acknowledge”** she knows that the secure communication now can take place. If match, Alice send a signal message “Ready” encrypted using the AES session key to Bob.

```
cout<<BOLD(FRED("------------------------------------------------------------------"))<<endl<<endl;
cout<<FRED(BOLD("TRYING TO RECEIVE ACKNOWLEDGE FLAG FROM CLIENT\n\n----------"))<<endl;
string nonevalue="";
AES_Decryption(nonevalue,IV_session, AES_session, new_socket,false);
receivedummy(new_socket);
cout<<FRED(BOLD("------------------------------------------------------------------\n\nTRYING TO SEND READY FLAG\n\n----------"))<<endl;
AES_Encryption("Ready",IV_session, AES_session,new_socket,false);
```

After received “Encrypted Acknowledge” from Client, Server will take the encrypted go into process of *AES_Decryption()* to find out what is the decrypted text.

Due to code too long, unable to screenshot. I have pasted it here as text format

AES_Decryption()

```
string AES_Decryption(string cipher,string AESiv,string AESkey, int socket,bool haha)
{
    AutoSeededRandomPool prng;
    string decodedkey,decodediv;
    StringSource s(AESkey, true,(new HexDecoder(
                new StringSink(decodedkey))
       ) // StreamTransformationFilter
      ); // StringSource
```

```
         StringSource ss(AESiv, true,(new HexDecoder(
                    new StringSink(decodediv))
           ) // StreamTransformationFilter
          ); // StringSource

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
    SecByteBlock iv((const byte*)decodediv.data(), decodediv.size());

        if(haha==false)
        {
        cipher= recv_send(socket, "Server responed \"Received Acknowledge\", sending \"Ready\"",
"Acknowledge send from Client");
        }
        else
        {
        int valread;
        char buffer[1024] = {0};
        string compare;
        valread = read(socket , buffer, 1024);
        cipher=buffer;
        }
        cout<<"Cipher Text : "<<cipher<<endl;
        string rawcipher;
            StringSource ss2(cipher, true,
            new HexDecoder(
                    new StringSink(rawcipher)
                ) // HexEncoder
            ); // StringSource



        string recovered;
       try
            {
                    CFB_Mode<AES>::Decryption d;
                    d.SetKeyWithIV(key, sizeof(key), iv);

                    // The StreamTransformationFilter removes
                    //  padding as required.
                    StringSource s(rawcipher, true,
                       new StreamTransformationFilter(d,
                  d.SetKeyWithIV(key, sizeof(key), iv);

                    // The StreamTransformationFilter removes
                    //  padding as required.
                    StringSource s(rawcipher, true,
                       new StreamTransformationFilter(d,
                       new StringSink(recovered)) // StreamTransformationFilter
                    );

                    cout <<FCYN(BOLD("Message Received: " << recovered <<""<<))<<endl;
                    if(recovered=="quit")
                    {
                            cout<<FRED(BOLD("PROGRAM TERMINAL GRACEFULLY"))<<endl;
                            exit(1);
                    }
            }
            catch (const CryptoPP::Exception &e)
            {
                    cerr << e.what() << endl;
                    exit(1);
            }

            /*********************************\
            \*********************************/
            if(haha==false)
```

```
        {
                verify("Acknowledge",recovered);
                cout<<FGRN(BOLD("\nAcknowledge Recevied\n\n"))<<endl;
        }
return recovered;
}
```

```
TRYING TO RECEIVE ACKNOWLEDGE FLAG FROM CLIENT

[ Incoming message from client ] : 3A759922AC05FAF20A8091
[ Details ] Acknowledge send from Client
Cipher Text : 3A759922AC05FAF20A8091
Message Received: Acknowledge
Comparing the strings
Verified
Acknowledge Recevied
```

After, we received the decrypted content, we will receive dummy packet again, so we can start the send function.
We now can start to encrypt 'Ready' text and send over network.

```
void AES_Encryption(string temp, string AESiv, string AESkey,int socket,bool haha)
{
string plain;
if(haha==false)
{
        plain = temp;
}
else
{
do
{
cout<<"Enter message send to server"<<endl;
std::getline(std::cin, plain);
if(plain.size()>1024)
{
cout<<BOLD(FRED("Message is exceed the length"))<<endl;
}

}while(plain.size()>1024);
}


    string  recovered;
    AutoSeededRandomPool prng;
    string decodedkey,decodediv;
    StringSource s(AESkey, true,(new HexDecoder(
                new StringSink(decodedkey))
       ) // StreamTransformationFilter
      ); // StringSource

     StringSource ss(AESiv, true,(new HexDecoder(
                new StringSink(decodediv))
       ) // StreamTransformationFilter
      ); // StringSource

SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
SecByteBlock iv((const byte*)decodediv.data(), decodediv.size());
string cipher,encoded;
/*********************************\
\*********************************/
try
```

```
{
        cout << "plain text: " << plain << endl;
        CFB_Mode<AES>::Encryption e;
        e.SetKeyWithIV(key, sizeof(key), iv);

        // CFB mode must not use padding. Specifying
        //  a scheme will result in an exception
        StringSource(plain, true,
         new StreamTransformationFilter(e,
        new StringSink(cipher)) // StreamTransformationFilter
        );              // StringSource
        }
        catch (const CryptoPP::Exception &e)
        {
                cerr << e.what() << endl;
                exit(1);
        }
        /*********************************\
        \*********************************/
        // Pretty print
        encoded.clear();
        StringSource(cipher, true,
        new HexEncoder( new StringSink(encoded)
                ) // HexEncoder
        );  // StringSource
        cout << "cipher text: " << encoded << endl;
        cout<<"encoded length:" << encoded.length()<<endl;
        if(haha==false)
        {
        send_recv(socket,encoded , "Server send \"Ready\" message to Client");
        }
       else
        {
        sendpacket(socket,encoded);
        }
        if(plain=="quit")
        {
              cout<<FRED(BOLD("PROGRAM TERMINAL GRACEFULLY"))<<endl;
              exit(1);
        }
}
```

Code above is segment of code in *AES_Encryption()* ,
if we found out the variable *haha* is "*false*", we will send_recv packet the encoded content over
We go back to our *int main()* we can clearly see that the 5th parameter is "False"

```
------------------------------------------------------------------

TRYING TO SEND READY FLAG

------------------------------------------------------------------
plain text: Ready
cipher text: 29739328BA
encoded length:10
[ Successfully send to Client ] Server send "Ready" message to Client
[ Incoming message from client ] : Client responed "Received Ready"
```

Once the message “Ready” sent to Bob, Alice reverse the 192 bit AES key to be use for the next secured communication.

We now take a look on *AES_Decryption()*

Due to code too long, unable to screenshot. I have pasted it here as text format

```
    string AES_Decryption(string cipher,string AESiv,string AESkey, int socket,bool haha)
    {
        AutoSeededRandomPool prng;
        string decodedkey,decodediv;
        StringSource s(AESkey, true,(new HexDecoder(
                    new StringSink(decodedkey))
           ) // StreamTransformationFilter
          ); // StringSource       if(haha==false)

         StringSource ss(AESiv, true,(new HexDecoder(
                    new StringSink(decodediv))
           ) // StreamTransformationFilter
          ); // StringSource

    SecByteBlock key((const byte*)decodedkey.data(), decodedkey.size());
    SecByteBlock iv((const byte*)decodediv.data(), decodediv.size());

       if(haha==false)
        {
        cipher=recv_send(socket,"Client responed \"Received Ready\"", "Acknowledge send from
Client");
    //    sendpacket(socket,"Client responed \"Received Ready\"");
        }
        else
        {
        int valread;
        char buffer[1024] = {0};
        string compare;
        valread = read(socket , buffer, 1024);
        cipher=buffer;
        }
        cout<<"Cipher Text : "<<cipher<<endl;
        string rawcipher;
            StringSource ss2(cipher, true,
            new HexDecoder(
                    new StringSink(rawcipher)
                ) // HexEncoder
            ); // StringSource

        string recovered;
       try
            {
                    CFB_Mode<AES>::Decryption d;
                    d.SetKeyWithIV(key, sizeof(key), iv);

                    // The StreamTransformationFilter removes
                    //  padding as required.
                    StringSource s(rawcipher, true,
                       new StreamTransformationFilter(d,
                      new StringSink(recovered)) // StreamTransformationFilter
                    );

                    cout << FCYN(BOLD("Message Received: " << recovered<<""))<< endl;
                    if(recovered=="quit")
                    {
                            cout<<FRED(BOLD("PROGRAM TERMINAL GRACEFULLY"))<<endl;
                            exit(1);
                    }
```

```
            }

            catch (const CryptoPP::Exception &e)

        {
                cerr << e.what() << endl;
                exit(1);
            }

            /*********************************\
            \*********************************/
            if(haha==false)
            {
                verify("Ready",recovered);
                cout<<FGRN(BOLD("\nReady Recevied"))<<endl;
            }
    return recovered;
    }
```

Inside the *AES_Decryption()*, it have a function what will '*received encrypted message that sent from Server*'.

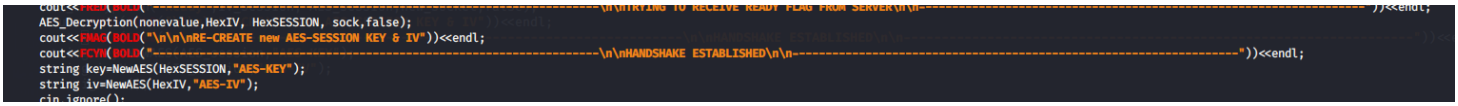
```
    if(haha==false)
    {
    cipher=recv_send(socket,"Client responed \"Received Ready\"", "Acknowledge send from Client");
//    sendpacket(socket,"Client responed \"Received Ready\"");
    }
    else
    {
    int valread;
    char buffer[1024] = {0};
    string compare;
    valread = read(socket , buffer, 1024);
    cipher=buffer;
    }
    cout<<"Cipher Text : "<<cipher<<endl;
    string rawcipher;
        StringSource ss2(cipher, true,
        new HexDecoder(
            new StringSink(rawcipher)
        ) // HexEncoder
        ); // StringSource
```

Same, if (haha == true) will run send_recv function. Else will only run "Recv function"

```
string nonevalue="";
AES_Encryption("Acknowledge",HexIV,HexSESSION,sock,false);
sendpacket(sock,dummy);
cout<<endl<<endl<<endl;
cout<<FRED(BOLD("----------\n\nTRYING TO RECEIVE READY FLAG FROM SERVER\n\n----------"))<<endl;
AES_Decryption(nonevalue,HexIV, HexSESSION, sock,false);
```

```
------------------------------------------------------------------

TRYING TO RECEIVE READY FLAG FROM SERVER

------------------------------------------------------------------
[ Incoming message from server ] : 29739328BA
[ Details ] Acknowledge send from Client
Cipher Text : 29739328BA
Message Received: Ready
Comparing the strings
Verified
Ready Recevied
```

Bob, upon receiving this Encrypted message “Ready”, he decrypts it and identifies this signal. With this, Bob now, reverse the 192 bit AES key.



After finish the Decryption, the HANDSHAKE is *establish*, we will run another function which will reverse the key.

```
string NewAES(string value,string title){
string tmp,reverse;
cout<<"[OLD] ["<<title<<"] :"<<value<<endl;
for(int i=0; i <= value.length();i++)
 {
                tmp[i]=value[value.length()-i];
                reverse= reverse+tmp[i];
 }
                cout<<"[NEW] ["<<title<<"] :"<<reverse<<endl<<endl;
                return reverse;
}
```

```
------------------------------------------------------------------

TRYING TO RECEIVE READY FLAG FROM SERVER

------------------------------------------------------------------
[ Incoming message from server ] : 29739328BA
[ Details ] Acknowledge send from Client
Cipher Text : 29739328BA
Message Received: Ready
Comparing the strings
Verified
Ready Recevied

RE-CREATE new AES-SESSION KEY & IV
------------------------------------------------------------------
HANDSHAKE ESTABLISHED
------------------------------------------------------------------
[OLD] [AES-KEY] :3AB3A45B7B09AF857F9679E9BA29D658150CFFE0D3A6C611
[NEW] [AES-KEY] :116C6A3D0EFFC051856D92AB9E9769F758FA90B7B54A3BA3

[OLD] [AES-IV] :ABF67878A050D20420C17E326FAB7F414DB645DAED3031F3
[NEW] [AES-IV] :3F1303DEAD546BD414F7BAF623E71C02402D050A87876FBA

Enter message send to server
```

Now, the secure channel is established.

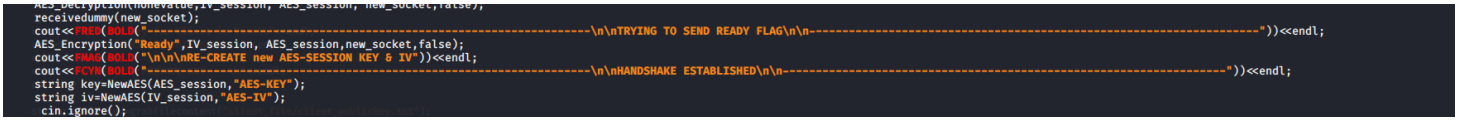
Either Alice or Bob can send a message encrypted with the **reverse 192 bit AES session key** (from the original key) now. They type the message on their own terminal. The message is encrypted by the program (Host or Client) and sent out.

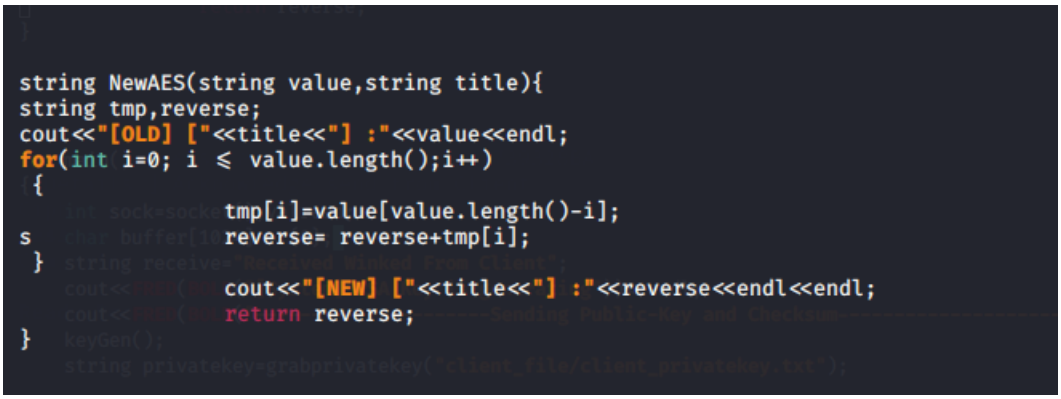
All message must use Cipher Feed Back Mode (CFB) mode of block Cipher.

The received encrypted message and decrypted message are displayed on the screen.

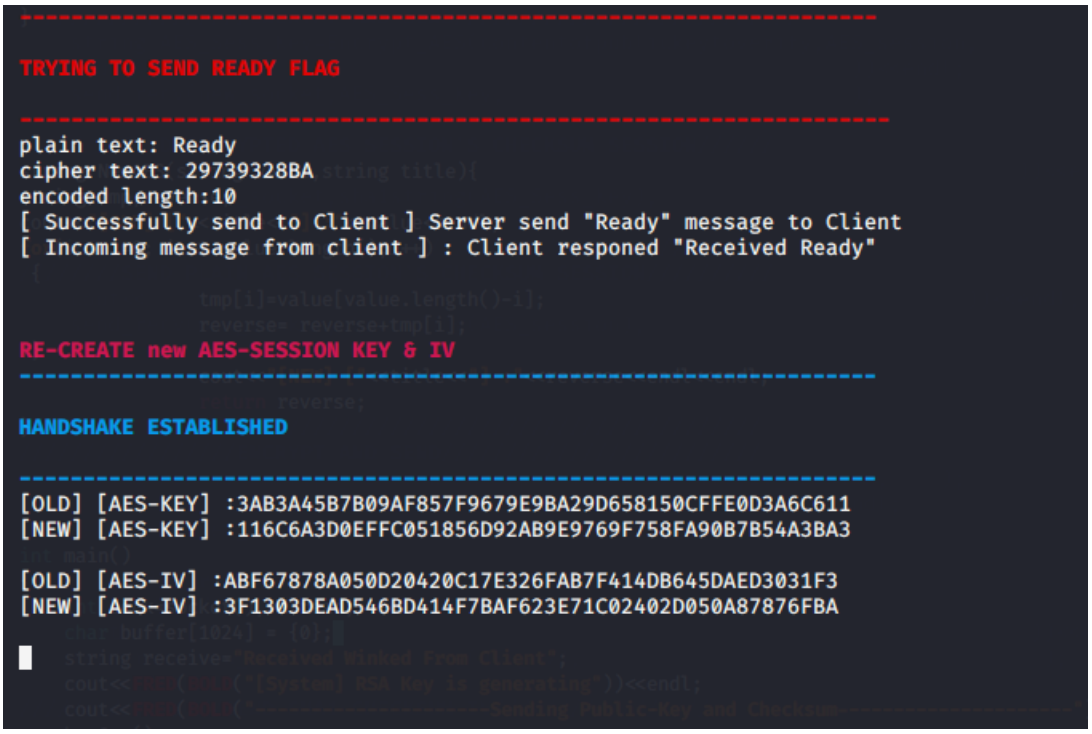
Either one Host or Client can quit the program by using “exit”.

On Server side, Alice will have to reverse the IV and the KEY too.



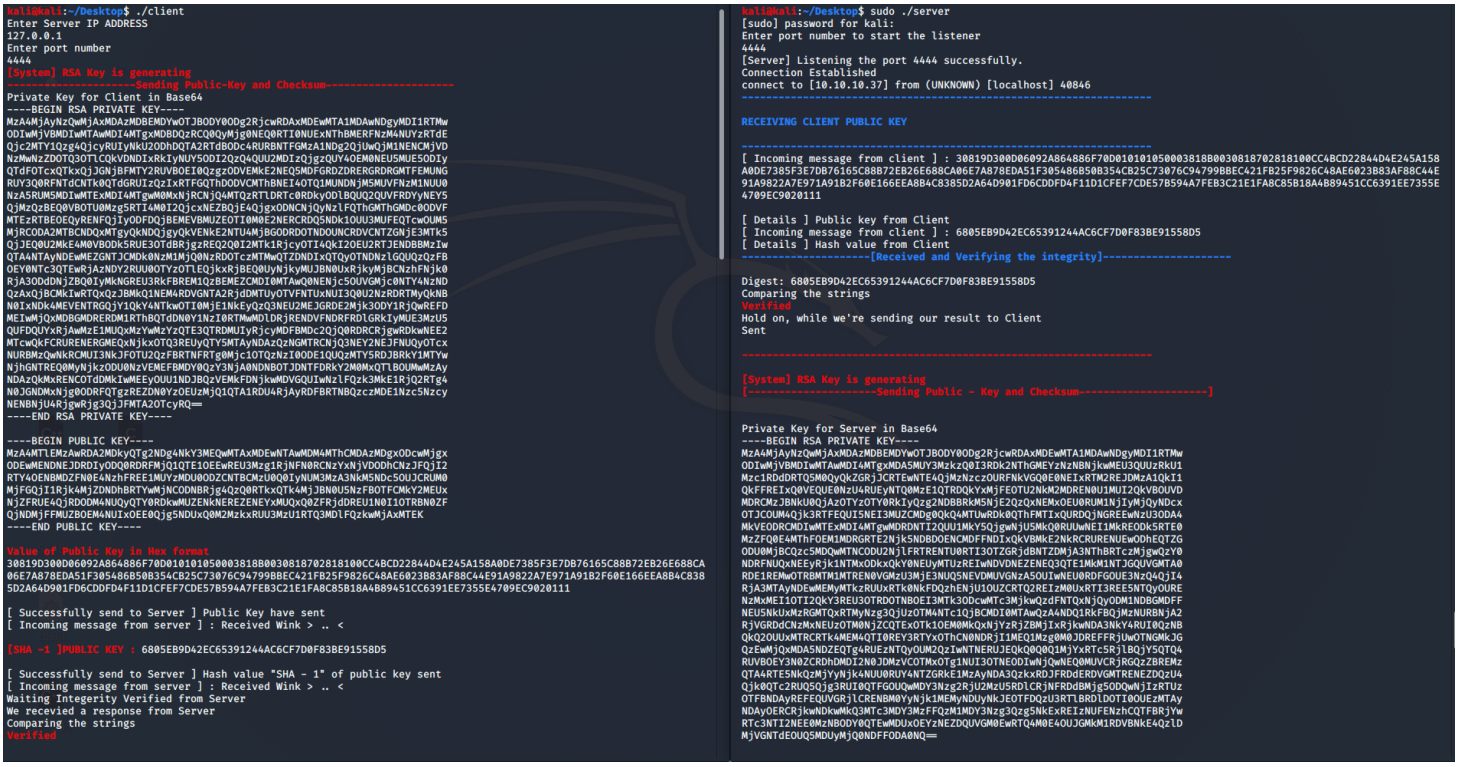


```
string NewAES(string value,string title){
string tmp,reverse;
cout<<"[OLD] ["<<title<<"] :"<<value<<endl;
for(int i=0; i ≤ value.length();i++)
 {
                tmp[i]=value[value.length()-i];
s               reverse= reverse+tmp[i];
 }
                cout<<"[NEW] ["<<title<<"] :"<<reverse<<endl<<endl;
                return reverse;
}
```

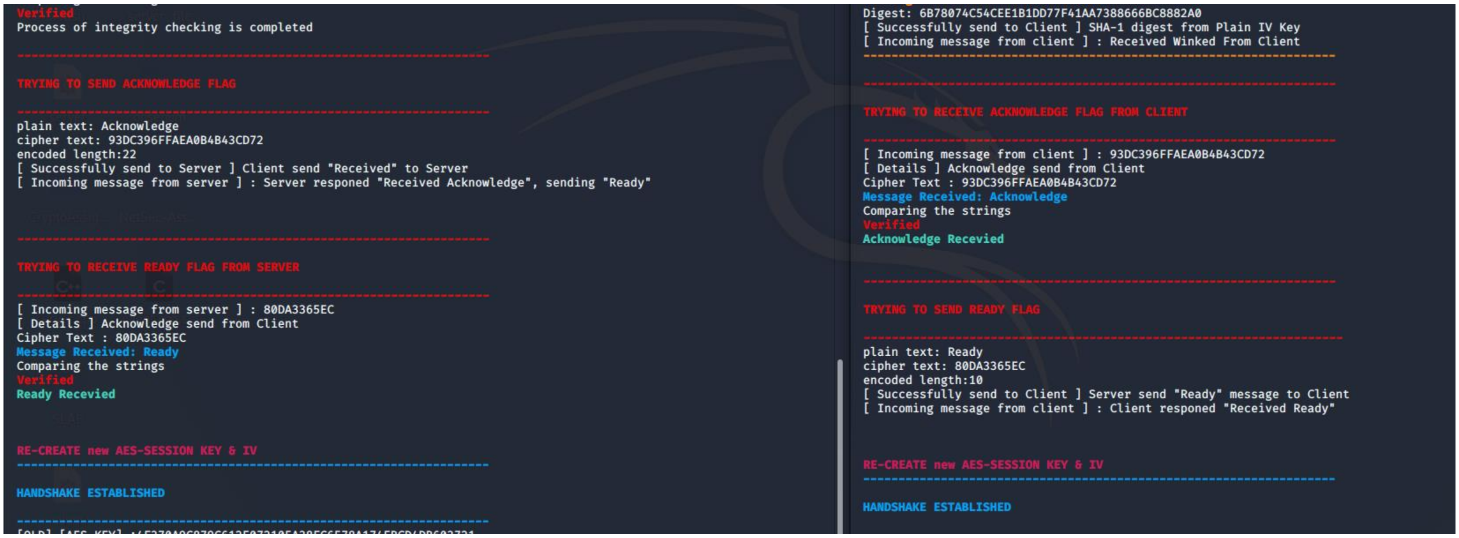


```
----------------------------------------------------------------

TRYING TO SEND READY FLAG

----------------------------------------------------------------
plain text: Ready
cipher text: 29739328BA
encoded length:10
[ Successfully send to Client ] Server send "Ready" message to Client
[ Incoming message from client ] : Client responed "Received Ready"


RE-CREATE new AES-SESSION KEY & IV
----------------------------------------------------------------

HANDSHAKE ESTABLISHED

----------------------------------------------------------------
[OLD] [AES-KEY] :3AB3A45B7B09AF857F9679E9BA29D658150CFFE0D3A6C611
[NEW] [AES-KEY] :116C6A3D0EFFC051856D92AB9E9769F758FA90B7B54A3BA3

[OLD] [AES-IV] :ABF67878A050D20420C17E326FAB7F414DB645DAED3031F3
[NEW] [AES-IV] :3F1303DEAD546BD414F7BAF623E71C02402D050A87876FBA
```

Before Communication Established



Key Reversing + sending text ready to pair.



After Communication Established

USER MANUAL (GITHUB)

README.MD

```
README.MD
```

GIT CLONE FROM GITHUB

To download the program in Ubuntu or Linux environment

```
kali@kali:~/Desktop$ git clone https://github.com/Applebois/NetSec-Assignment1
Cloning into 'NetSec-Assignment1'...
remote: Enumerating objects: 19, done.
remote: Counting objects: 100% (19/19), done.
remote: Compressing objects: 100% (12/12), done.
remote: Total 19 (delta 5), reused 19 (delta 5), pack-reused 0
Unpacking objects: 100% (19/19), 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
```

COMMAND TO COMPILE/BUILD

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

```
# cd NetSec-Assignment1
# cd Source
# g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o server server.cpp -lcryptopp
# g++ -g3 -ggdb -O0 -Wall -Wextra -Wno-unused -o client client.cpp -lcryptopp
```

ADD PRIVILEGE TO THE BINARY FILE

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

```
#chmod u+x server
#chmod u+x client
#./server
You are not root!
Please Run as root
#sudo ./server
[sudo] password for kali:
Enter port number to start the listener
^C
```

```
#./client
Enter Server IP ADDRESS
^C
```