Chat Client Using RSA and AES with Key Based Dynamic S- Box

PROJECT REPORT

Submitted by

B. Adarsh Reddy, 18BCI0196

Sumanth Kumar, 18BCE0413

P. Rohith Reddy, 18BCI0186

Course Code: CSE3501

Course Title: Information Security Analysis and Audit

Under the guidance of

DR. Arun Kumar K

VIT University, Vellore.

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April 2020

INDEX

1.	Introduction	Page no.
	1.1. Hybrid Encryption	3
	1.2. Dynamic S-Box	4
2.	Background of the Work	4-5
3.	Overview of the Work	
	3.1. Problem description	5
	3.2. Working model	5
	3.3. Design description	6
4.	Implementation	
	4.1. Description of Modules/Programs	7-8
	4.2. Source Code	8-18
	4.3. Test cases	18
	4.4. Execution snapshots	18-19
5.	Conclusion	19
6.	References	19-20

ABSTRACT

Security is nothing but protecting data and other information from unauthorised persons to access, destruction or change our data.In this world, cybersecurity is incredibly important because we've got our privacy at stake.So to counter this we'll design an encrypted chat employing a hybrid cryptographic system in java using AES(advanced encryption standard) and RSA(Rivest–Shamir–Adleman) for encryption.

RSA is employed to verify the intended recipient and to transfer the AES key. AES is employed then to encrypt the messages passed between the 2 users. First the Client transfers the AES key (encrypted using public key of recipient with RSA) to the recipient(Server).Server then decrypts the message using its private key to urge AES key which is employed to encrypt and decrypt further messages.RSA utilized in EBC(electronic code book) mode with 1024 bit key and AES with 128 bit key. AES is implemented with dynamic Key based SBox. Using this method messages will be securely passed between the 2 users.

1. INTRODUCTION

1.1

Encryption has a vital role in data protection. The importance of encryption realised with increasing communication. Encryption is smart when data packets using open channels, which they'll be reached by other devices or people, to transfer their contents. Encryption is knowledge of adjusting data with cipher key by using cipher algorithms, so someone who knows the cipher key and cipher algorithm can export the plain text from cipher text. The meaning of Encryption isn't only hiding information, but also it means sending information with another form, in order that ensure security of information.

An Encryption system contains set of transformations that convert plain text into cipher text. within the block cipher system, plain text converts into blocks that cipher algorithm applies on them to make cipher text. The block cipher systems divided into two general principles: Diffusion and Confusion. In Diffusion principle, each little bit of plain text converts into many bits. However, in Confusion principle, number of bits doesn't change and only transformations apply to plain text, hence in Confusion principle, size of plain text and cipher text

is equal. Usually in both principles, using round repetition to make cipher text. Repeating a single round contributes to cipher's simplicity.

1.2

Cipher algorithms have the 2 general categories: Private Key algorithms and public key algorithms. Private Key algorithms using single key to encrypt plain text and decrypt cipher text in sender and receiver side. Private Key algorithm samples are: DES, 3DES and Advanced Encryption Standard (AES). Public Key algorithms, like the Rivest-Shamir-Adleman (RSA), using two different key for encrypt plain text and decrypt cipher text in sender and receiver sides. Block cipher systems rely upon the S- Boxes, which are fixed and no relation with a cipher key. So only changeable parameter is cipher key. Since the sole nonlinear component of AES is S- Boxes, they're a vital source of cryptographic strength. So we intend use cipher key to come up with dynamic S-Box that's changed with every changing of cipher key. That cause increasing the cryptographic strength of AES algorithm. Other systems using key- dependent S- Boxes are proposed within the past, the foremost well-known is Blowfish and Khufu. Each of those two systems uses the crypto system itself to come up with the S- Boxes.

2. BACKGROUND OF THE WORK

Encryption plays a very important role in data protection. The importance of encryption is realized by increasing communications. Encryption is the knowledge of changing data with cipher key by using cipher algorithms. after encryption someone who knows the cipher key and cipher algorithm can extract the plain text from the cipher text. Encryption is a very effective way to protect the security data. The representative encryption algorithms like RSA and AES were not able to meet the requirements of efficiency and security when used separately. A hybrid encryption algorithm combining AES and RSA algorithm is proposed to will solve the above problems and increase efficiency and decreases the security problems in file encryption. The experimental results have backed that the hybrid encryption using RSA and AES was not only able to encrypt files but also have the advantage of algorithm performance and security.

Cipher algorithms have two general categories: Private Key and public key algorithms. Private Key algorithms using a single key to encrypt plain text and decrypt ciphertext on the sender and receiver side. some of the Private Key algorithm samples are DES, 3DES, and AES. Public Key algorithms, such as the RSA, using two different keys for encrypt plain text and decrypt ciphertext in the sender and receiver sides. Block cipher systems are generally dependent on the S-Boxes, which are fixed and has no relation with a cipher key. So the only changeable parameter is the cipher key. Since the only nonlinear component of AES is Substitution boxes, they are an important source to increase cryptographic strength. So the use the cipher key to generate a dynamic S-Box that is changed with every changing of the cipher key. That causes increasing the cryptographic strength of the AES algorithm.

3. OVERVIEW OF THE WORK

3.1 Problem description

To design a secure server-client chat system using various cryptography techniques and methods so that the client would never bother about his privacy and to create a system where hacking is literally not possible as the dynamic s-box comes into picture.

3.2. Working model

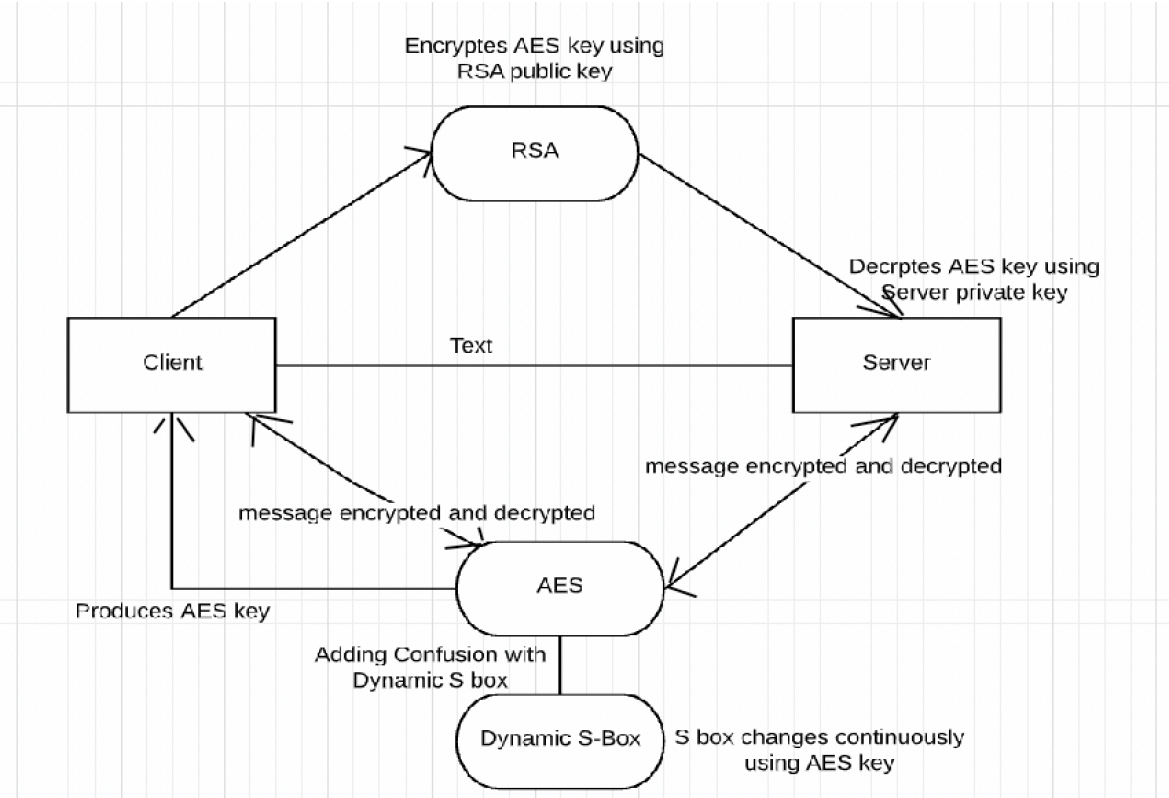


Fig 1

3.3. Design description

First the and the server generates public and private key using RSA. Then Client transfers the AES key (encrypted using public key of recipient with RSA) to the Server. Then Server decrypts the message using its private key to use the AES key which is employed to encrypt and decrypt further messages. The RSA is utilized in EBC(electronic code book) mode with 1024 bit key and AES with 128 bit key. AES is implemented with dynamic Key based SBox. The and the Dynamic S box will keeps on changing in the process of encryption with the help of the cipher key we use. Giving possibility for S-Box to be able to generate $(10^{38})! = \infty$ possible S-boxes when n = 128 is great. Below example shows that by just changing 2 bits in the key, 87% of the S box has been changed .

For key_hex1={5e,d3,f1,b4,7c,18,51,9a,ae,81,42,57,42,78,dc,8f}

67	1E	77	7B	4B	4F	6F	46	1F	F8	6A	87	B9	71	60	CD
AD	99	85	AF	48	D5	81	0E	17	82	9C	C9	35	13	47	0C
E5	7C	B8	A4	E1	55	E3	DF	02	01	31	70	B0	B7	BD	C1
80	D4	8F	E2	BF	54	B2	68	3B	A1	18	0F	9A	03	DC	EC
D6	E6	AC	B3	3F	7A	22	C5	5E	1D	EB	CA	D2	A6	5A	A0
D0	EF	AA	FB	43	16	AE	07	7D	F9	0A	33	93	26	E9	44
9B	4D	53	D1	00	04	20	5B	5D	51	76	CB	DB	6E	4A	6B
25	AB	D9	E7	73	BA	9D	DA	59	12	74	C0	C4	F7	11	63
94	C7	ED	6C	27	61	CE	39	30	FA	08	C3	36	EA	7F	E8
19	A2	A8	05	52	D7	9F	8C	4C	BE	9E	96	3E	FE	8E	A7
49	D3	8B	38	F1	1C	B6	41	1A	E4	FD	F5	89	D8	24	CF
5F	72	37	6D	92	95	4E	FC	83	0B	65	45	0D	A5	14	3D
E0	2D	C2	3A	B1	06	B4	C6	2E	32	10	62	8D	34	DE	BC
CC	C8	B5	66	2F	58	F6	98	79	56	86	F0	2A	2B	57	3C
88	78	84	90	91	97	EE	5C	BB	DD	69	15	42	40	28	64
75	A3	F2	23	2C	50	7E	1B	29	F4	8A	21	A9	FF	F3	09

S-Box-1 (Fig 2)

key_hex2={5d,d3,f1,b4,7c,18,51,9a,ae,81,42,57,42,78,dc,8f}

4D	CA	D7	59	3F	50	DE	5B	BC	21	16	2D	3A	55	28	A9
82	7D	EB	FA	9E	FE	F0	51	E2	0C	72	68	A2	B9	99	73
E3	C5	B6	01	46	AA	5D	DA	42	F2	BE	B5	77	B7	52	F9
0D	9C	27	B2	B3	86	C7	FD	29	C0	6D	9A	7B	81	DC	EC
BF	AD	17	22	45	89	83	31	5E	35	2F	E6	64	A4	3B	69
D0	A1	7C	FB	43	65	AE	38	47	D9	24	87	67	26	02	E7
80	E1	BD	D4	00	7F	20	8F	61	23	11	05	6F	6E	4A	6B
2C	94	40	88	75	2E	D1	10	07	A0	F3	C9	E5	F7	19	CF
6C	04	66	25	62	30	54	39	C3	5F	09	ED	1F	EA	18	CB
48	A3	7A	12	92	FC	9F	F5	4C	53	76	8B	B8	06	AC	A7
A8	91	0B	1E	F1	C6	79	08	5A	E4	15	95	44	F8	D3	96
BA	70	37	0F	D8	8A	57	CC	1B	98	33	AF	D5	A5	85	8E
C2	B1	E8	DD	CE	41	32	71	4E	1A	E0	78	8C	A6	49	E9
13	C8	C1	1D	34	58	3E	6A	B4	56	2B	0E	9D	84	FF	3C
DB	7E	4F	EE	9B	97	4B	5C	BB	D6	14	74	60	0A	1C	D2
B0	DF	93	90	63	C4	EF	03	36	8D	2A	F4	CD	F6	AB	3D

S-Box-2 (Fig 3)

4. IMPLEMENTATION

4.1 Description of Modules/Programs

Client module

This module consists of all the parts required for us to make a client work successfully and to connect it with the server

- The client Connects to server port 8002
- Send encrypted AES key using the servers public key.
- Sends and receive encrypted message using AES key.

Following files can be considered to be a part of Client module:-

1.Client.java

2.Client1.java

Client.java has all the function descriptions and we are using Client1.java to express the functionality by just using function calls to make the code look simple

Server module

This module consists of all the parts required for us to successfully connect it with the Client.

- Generate public and private keys.
- Start listener on port 8002.
- Send public key to connected client.
- Receive encrypted AES key and decrypt using private key.
- Sends and receive encrypted message using AES key.

Following files can be considered to be a part of Server module:-

Server.java

Server1.java

Additional Files

- AES, RSA are put in separate files so that both client and server can use whenever they need it.
- Sbox1.java file contains the dynamic substitution box which will be used to create a different and more random key every time the conversation happens.
- Polynomial.java:-This contains the functions for RSA and AES to work like left-shift, right-shift, circular-shift and etc.

4.2 Source code

Server1:

```
import java.security.*;
 import java.security.spec.RSAPrivateKeySpec; import java.security.spec.RSAPublicKey-
Spec; import java.util.Scanner;
 import java.io.*;
 import java.math.BigInteger;
 import java.net.ServerSocket;
 import java.net.Socket;
 import java.util.Base64;
 import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec; import javax.crypto.spec.SecretKeySpec;
public class Server1 {
 private ObjectOutputStream sOutput;
 private ObjectInputStream sInput;
 private Cipher keyDecipher;
 private Cipher ServerDecryptCipher;
 private Cipher ServerEncryptCipher;
 SecretKey AESkey;
 int i;
 byte[] input;
 private message m;
```

```
 int port;
 static String IV = "AAAAAAAAAAAAAAAA"; message toSend;
 public Server1(int port){
 this.port = port;
 }
public static String toHexString(byte[] ba) { StringBuilder str = new StringBuilder();
for(int i = 0; i < ba.length; i++) str.append(String.format("%x", ba[i])); return str.toString();
 }
public static String fromHexString(String hex) { StringBuilder str = new StringBuilder();
 for (int i = 0; i < hex.length(); i+=2) {
 str.append((char) Integer.parseInt(hex.substring(i, i + 2), 16)); }
return str.toString(); }
public static void main(String[] args) throws IOException, GeneralSecurityException{ RSA
rsa = new RSA();
 rsa.createRSA();
 int port = 8002;
Server server = new Server(port); server.start();
 }
void start() throws IOException{
 ServerSocket serverSocket = new ServerSocket(port); System.out.print("Receiver listening
on the port " + port + "."); Socket socket = serverSocket.accept();
clientThread t = new clientThread(socket); t.run();
 serverSocket.close();
 }
class clientThread extends Thread{
 Socket socket;
 clientThread( Socket socket) throws IOException{
this.socket = socket;
 sOutput = new ObjectOutputStream(socket.getOutputStream()); sInput = new ObjectInput-
Stream(socket.getInputStream());
 new listenFromClient().start();
```

```
 new sendToClient().start();
 }
}
class listenFromClient extends Thread{ public void run(){
while(true){ try {
m = (message) sInput.readObject();
 } catch (ClassNotFoundException e) { e.printStackTrace();
 } catch (IOException e) {e.printStackTrace();
}
if (i == 0) {
 if(m.getData() != null){
decryptAESKey(m.getData()); System.out.println();
 i++;}
 else{
System.exit(1);}}
else
 {
 if(m.getData() != null){
decryptMessage(m.getData()); }
} }
} }
class sendToClient extends Thread {
public void run(){ try{
PublicKey pK = readPublicKeyFromFile("public.key"); sOutput.writeObject(pK);
}catch (Exception e){ e.printStackTrace();
System.out.println("No message sent to server");
}
 while(true){
 try{
 System.out.println("Server: Enter message : > ");
Scanner sc = new Scanner(System.in); String s = sc.nextLine();
```

```
 toSend = null;
 toSend = new message(encryptMessage(s)); write();
}
 catch (Exception e){
e.printStackTrace();
 System.out.println("No message sent to server"); break;
} }
}
public synchronized void write() throws IOException{ sOutput.writeObject(toSend);
 sOutput.reset();
 }
}
private void decryptAESKey(byte[] encryptedKey) {
 SecretKey key = null; PrivateKey privKey = null; keyDecipher = null;
 try
 {
 privKey = readPrivateKeyFromFile("private.key");
   keyDecipher = Cipher.getInstance("RSA/ECB/PKCS1Padding"); keyDecipher.init(Ci-
pher.DECRYPT_MODE, privKey );
 key = new SecretKeySpec (keyDecipher.doFinal(encryptedKey), "AES"); System.out.print-
ln();
 System.out.println(" AES key after decryption: " +key.getEncoded());
 i = 1;
 AESkey = key;
 }
 catch(Exception e)
 { e.printStackTrace();
}
}
 private void decryptMessage(byte[] encryptedMessage) {
ServerDecryptCipher = null;
```

```
try {
String encodedKey,keyy;
 encodedKey = Base64.getEncoder().encodeToString(AESkey.getEncoded()); keyy=toHex-
String(encodedKey.getBytes());
 keyy=keyy.substring(0,32);
 String ret=new String(encryptedMessage);
 AES aes1= new AES(ret,keyy);
 String pt=aes1.decrypt();
 pt=fromHexString(pt);
 pt=pt.replaceAll("-*$","");
System.out.println("Message From Client >> " + pt);
System.out.println("Server: Enter message : > "); }
catch(Exception e) {
 e.getCause(); e.printStackTrace();
} }
private byte[] encryptMessage(String s) throws NoSuchAlgorithmException, NoSuch-
PaddingException,
  InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException,
BadPaddingException{
ServerEncryptCipher = null; byte[] cipherText = null;
 int k=s.length();
 if(k<16){
for(int i=0;i<(16-k);i++){ s=s.concat("-");
} }
String encodedKey,key;
 try{
 encodedKey = Base64.getEncoder().encodeToString(AESkey.getEncoded()); key=toHex-
String(encodedKey.getBytes());
 key=key.substring(0,32);
 String st=toHexString(s.getBytes());
 AES aes= new AES(st,key);
```

```
 String et=aes.encrypt();
 cipherText=et.getBytes();
 }catch(Exception e){}
 return cipherText;
 }
PrivateKey readPrivateKeyFromFile(String fileName) throws IOException { FileInputStream
in = new FileInputStream(fileName);
ObjectInputStream readObj = new ObjectInputStream(new BufferedInputStream(in)); try {
BigInteger m = (BigInteger) readObj.readObject();
  BigInteger d = (BigInteger) readObj.readObject(); RSAPrivateKeySpec keySpec = new
RSAPrivateKeySpec(m, d); KeyFactory fact = KeyFactory.getInstance("RSA");
 PrivateKey priKey = fact.generatePrivate(keySpec);
 return priKey;
} catch (Exception e) {
 throw new RuntimeException("private key err", e);
} finally { readObj.close();
} }
PublicKey readPublicKeyFromFile(String fileName) throws IOException {
FileInputStream in = new FileInputStream(fileName);
 ObjectInputStream oin = new ObjectInputStream(new BufferedInputStream(in));
try {
BigInteger m = (BigInteger) oin.readObject();
 BigInteger e = (BigInteger) oin.readObject();
 RSAPublicKeySpec keySpecifications = new RSAPublicKeySpec(m, e);
KeyFactory kF = KeyFactory.getInstance("RSA"); PublicKey pubK =
kF.generatePublic(keySpecifications); return pubK;
} catch (Exception e) {
 throw new RuntimeException("public key err", e);
} finally { oin.close();
} }
}
```

Client1:

```
import java.io.*;
 import java.math.BigInteger;
 import java.net.Socket;
 import java.security.*;
 import java.security.spec.RSAPublicKeySpec; import java.util.Scanner;
 import javax.crypto.*;
import javax.crypto.spec.IvParameterSpec; import java.util.Base64;
 public class Client1 {
private ObjectOutputStream sOutput; private ObjectInputStream sInput; private Socket sock-
et;
 private String server;
private int port;
 private Cipher cipher1;
 private Cipher cipher2;
 int i = 0,j=0;
 message m;
 SecretKey AESkey;
 PublicKey pK ;
 message toSend;
 static String IV = "AAAAAAAAAAAAAAAA";
Client1 (String server, int port){ this.server = server;
 this.port = port;
 }
public static String toHexString(byte[] ba) {
StringBuilder str = new StringBuilder(); for(int i = 0; i < ba.length; i++) str.append(String.-
format("%x", ba[i])); return str.toString();
}
public static String fromHexString(String hex) { StringBuilder str = new StringBuilder();
 for (int i = 0; i < hex.length(); i+=2) {
 str.append((char) Integer.parseInt(hex.substring(i, i + 2), 16));
```

```
}
return str.toString(); }
public static void main(String[] args) throws IOException, NoSuchAlgorithmException
{ String serverAddress;
int portNumber = 8002;
 serverAddress = "localhost";
 Client client = new Client(serverAddress, portNumber); client.generateAESkey();
 client.start();
 }
void start() throws IOException{
 socket = new Socket(server, port);
   System.out.println("connection accepted " + socket.getInetAddress() + " :" +
socket.getPort());
sInput = new ObjectInputStream(socket.getInputStream()); sOutput = new ObjectOutput-
Stream(socket.getOutputStream());
new sendToServer().start(); new listenFromServer().start(); }
class listenFromServer extends Thread { public void run(){
while(true){ try{
if(j == 0){
 pK = (PublicKey) sInput.readObject(); j=1;
 }else{
 m = (message) sInput.readObject(); decryptMessage(m.getData());}
 } catch (Exception e){ e.printStackTrace();
System.out.println("connection closed"); }
} }
}
class sendToServer extends Thread { public void run(){
 while(true){
 try{
if (i == 0){ toSend = null;
toSend = new message(encryptAESKey()); sOutput.writeObject(toSend);
```

```
 i =1;
 }
else{
System.out.println("Client: Enter message > "); Scanner sc = new Scanner(System.in);
String s = sc.nextLine();
 toSend = new message(encryptMessage(s)); sOutput.writeObject(toSend);
 }
} catch (Exception e){ e.printStackTrace(); break;
} }
void generateAESkey() throws NoSuchAlgorithmException{ AESkey = null;
 KeyGenerator Gen = KeyGenerator.getInstance("AES"); Gen.init(128);
AESkey = Gen.generateKey();
 System.out.println("Genereated the AES key : " + AESkey.getEncoded()); }
private byte[] encryptAESKey (){ cipher1 = null;
 byte[] key = null;
 try
{
System.out.println("Encrypting the AES key using RSA Public Key:\n" + pK); cipher1 = Ci-
pher.getInstance("RSA/ECB/PKCS1Padding");   cipher1.init(Cipher.ENCRYPT_MODE,
pK );
 key = cipher1.doFinal(AESkey.getEncoded());
i = 1; }
catch(Exception e ) {
 System.out.println ( "" + e.getMessage() ); e.printStackTrace();
}
return key; }
private byte[] encryptMessage(String s) throws NoSuchAlgorithmException, NoSuch-
PaddingException,
InvalidKeyException, InvalidAlgorithmParameterException, IllegalBlockSizeException,
BadPaddingException{ cipher2 = null;
} }
```

```
byte[] cipherText = null; int k=s.length(); if(k<16){
 for(int i=0;i<(16-k);i++){ s=s.concat("-");
} }
String encodedKey,key;
 try{
  encodedKey = Base64.getEncoder().encodeToString(AESkey.getEncoded()); key=toHex-
String(encodedKey.getBytes());
 key=key.substring(0,32);
 String st=toHexString(s.getBytes());
 AES aes= new AES(st,key);
 String et=aes.encrypt();
 cipherText=et.getBytes();
 }catch(Exception e){}
 return cipherText;
 }
 private void decryptMessage(byte[] encryptedMessage) {
 cipher2 = null;
 try
{
 String encodedKey,key;
 encodedKey = Base64.getEncoder().encodeToString(AESkey.getEncoded());
key=toHexString(encodedKey.getBytes()); key=key.substring(0,32);
 String ret=new String(encryptedMessage); AES aes1= new AES(ret,key);
String pt=aes1.decrypt(); pt=fromHexString(pt); pt=pt.replaceAll("-*$","");
System.out.println("Message From Server
>> " + pt); System.out.println("Client: Enter message > ");
}
catch(Exception e)
 {
 e.getCause();
 e.printStackTrace();
```

```
 System.out.println ( "" + e.getMessage() );
} }
public void closeSocket() { try{
if(sInput !=null) sInput.close(); if(sOutput !=null) sOutput.close(); if(socket !=null) socket.-
close(); }catch (IOException ioe){
} }
}
```

4.3 Test case

S-Box is substitution table that get number and return another number. This action is nonlinear. In S-Box, n input bits are represented as one of 2n different characters. The set of 2n characters are then transposed to one of the others in the set. The character is then converted back to an n-bit output. It can be easily shown that there are (2n)! different substitution or connection patterns possible. Thus if n is large then the possible S-Boxes that can be generate is large. If the cryptanalyst want to decode AES algorithm he should try to generate possible S-Box and use them in SubBytes function in AES cipher system. The cryptanalyst's task becomes n 38 38 computionally unfeasible as n gets large, say n = 128; then 2 = 10 , and (10)! possible S-Box can be generate which is an astronomical number (1038)! = ∞ Possible S-Box can be generate when n = 128

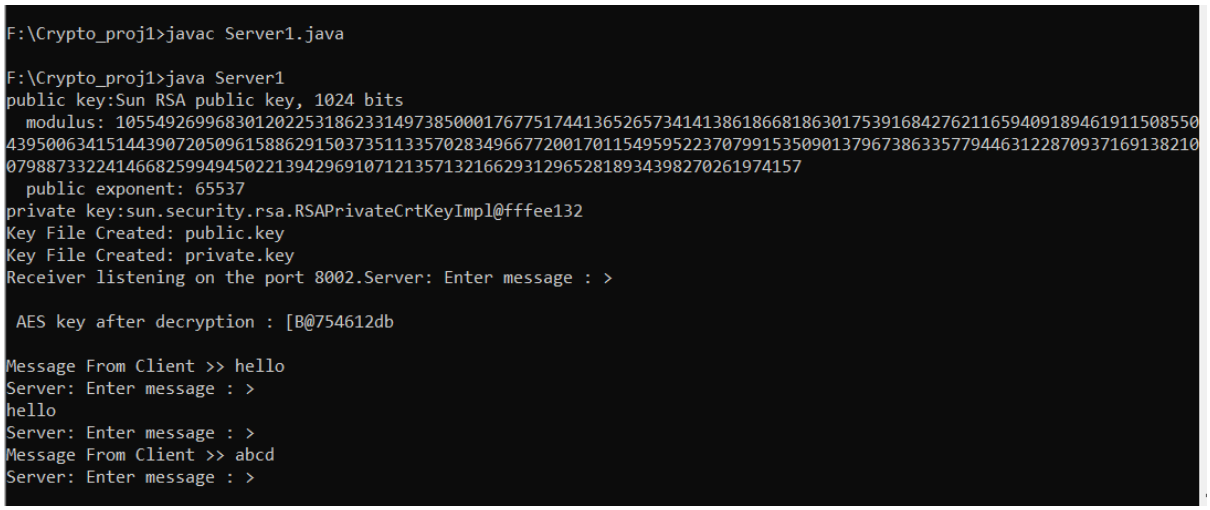
4.4 Outputs

Client:

```
F:\Crypto_proj1>java Client1
Genereated the AES key : javax.crypto.spec.SecretKeySpec@17bf9
connection accepted localhost/127.0.0.1 :8002
Encrypting the AES key using RSA Public KeySun RSA public key, 1024 bits
  modulus: 105549269968301202253186233149738500017677517441365265734141386186681863017539168427621165940918946191150855043950063415144390720509615886291503735113357028349667720017011549595223707991535090137967386335779446312287093716913821007988733224146682599494502213942969107121357132166293129652818934398270261974157
  public exponent: 65537
Time taken by RSA Encryption(Nano Seconds): 258200
CLIENT: Enter message >
hello
Time taken by AES Encryption (Nano Seconds) 54900
CLIENT: Enter message >
Message From Server   >> hello
CLIENT: Enter message >
abcd
Time taken by AES Encryption (Nano Seconds) 15600
CLIENT: Enter message >
```

Client side Output(Fig 4)

Server:



Server side output(Fig 5)

5. CONCLUSION

Designed chat client which uses rsa and modified AES to provide secure communication.We introduced a new algorithm to generate dynamic S-Box from cipher key for AES. The quality of this algorithm tested by changing only two bits of cipher key to generate new S-Boxes. This algorithm will lead to generate more secure block ciphers, solve the problem of the fixed structure S-Boxes and will increase the security level of the AES block cipher system. The main advantage of this algorithm is that many S-Boxes can be generated by changing Cipher key.

6. REFERENCES

[1] Razi Hosseinkhani & H. Haj Seyyed Javadi.” Using Cipher Key to Generate Dynamic S-Box in AES Cipher System” IEEE Trans. on Circuits and Systems – I: Volume: 53 Issue: 6 – 2006

[2] B. Schneier. Applied Cryptography: Protocols, Algorithms, and Source Code in C, New York: Wiley. 1996

[3] Merkle, R. Fast software encryption functions. In Advances in Cryptology: Proceedings of CRYPTO’90, Berlin: Springer-Verlag, 1991

[4] Federal Information Processing Standards, “Advanced Encryption Standard (AES)” Publication 197, November 26 - 2001

[5] Kazys KAZLAUSKAS, Janunius KAZLAUSKAS,” Key-Dependent S-Box Generation in AES Block Cipher System” , Inoformatica Volume: 20 - 2009

[6] Michael J.Quinn, Designing efficient algorithms for parallel computers, University of New Hamoshire, 1987

[7] Cui, J, Huang, L, Zhong, H. An improved AES S-Box and its performance analysis. Int J Innov Comput I 2011; 7: 2291–2302.

[8] Hosseinkhani, R, Haj Seyyed Javadi, H. Using cipher key to generate dynamic S-Box in AES cipher system. Int J Comput Sci Secur 2012; 6: 19–28.

[9] Kazys, K, Jaunius, K. Key-dependent S-Box generation in AES block cipher system. Informatica 2009; 20: 23–34.

[10] Arrag, S, Hamdoun, A, Tragha, A. Implementation of stronger AES by using dynamic S-Box dependent of master key. J Theor Appl Inf Technol 2013; 53: 196–204.