SIMPLIFIED SSL (SECURE SOCKET LAYER)

(CS 5780-01 (ADVANCED INFORMATION SECURITY): PROJECT 1))

Team Members:

Mitali Purohit

Bryan Nguyen

Ishita Kumbhani

Faculty Advisor:

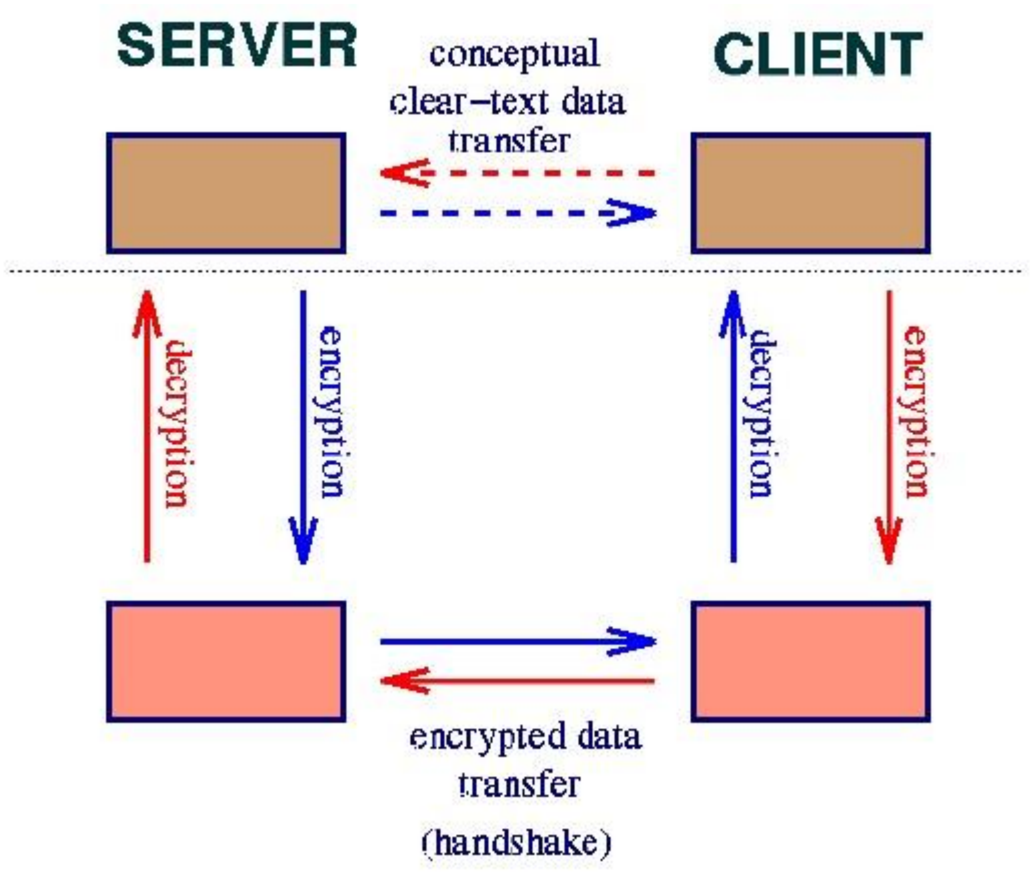
Dr. Huiping Guo

Abbreviations

SSL	Secure Socket Layer
K	One-time key proposed by the client
KRS	Server's Private Key
KRC	Client's Private Key
KUS	Server's Public Key
KRC	Client's Public Key
h(X)	Hash Function
ndatabytes	Number of data bytes in a packet
ncheckbytes	Number of checksum bytes in the packet

Project Description

The RSA ciphering in Java involves implementing the SSL (Secure Sockets Layer) protocol to establish a secure connection between a server and a client. This is achieved by performing a handshake process where cryptographic keys are exchanged to create a secure communication channel. Data transmission involves encrypting and hashing the data using a one-time key. To establish the secure network socket connection, the client authenticates itself to the server and exchanges a one-time key. After this exchange, data can be securely transmitted between the server and the client via the socket connection.


SERVER
conceptual clear−text data transfer
CLIENT
decryption
encryption
decryption
encryption
encrypted data transfer (handshake)

Server’s Perspective

➢ Authorized Client

To establish a secure connection between a user and a server, specific procedures are followed. The server stores the user's public key in the user's profile, while the user stores the server's public key in its own server profile.

When the user wants to connect to the server, it encrypts its username using the server's public key and encrypts its company name using its own private key. This ensures that the data remains confidential and can only be accessed by the intended recipients.

Additionally, the server's user profile includes information about the hash function used for data transfer, adding an extra layer of security and ensuring the integrity of the transmitted data.

The user's identity can only be decrypted by the server using public key cryptography. The server can also decrypt the company information using the client's public key to verify the user. However, this approach is susceptible to potential attacks where malicious entities intercept the encrypted data on the network to establish a connection with the server. To prevent such attacks, a hash function is employed during data exchange, and the parameters are agreed upon offline between the client and server. If the checksums of the transmitted data do not match the hashed values, the server terminates the connection immediately.

➢ Secure Communication

After the client has been authenticated, the communication takes place using a unique hash function specific to the client, and a one-time key suggested by the client.

Client's Perspective

➢ Contacting to the Real Server

The client's identity is encoded using the server's public key, and the corresponding private key is required to decrypt the client's identity and the proposed one-time key.

- Secure Communication

  In this approach, the client takes the initiative to initiate the connection and proposes the one-time key.

Handshake (Authorization & Key Exchange)

The client communicates the following three pieces of information to the server.

- username
- company
- proposed one-time key.

KUS (username) || KRC (company) || KUS(K)

The server decrypts the username and company information using the KRS and KUC obtained from the user's profile, respectively. If the received company information does not match the one stored in the user's profile, access is denied. If the decryption process is successful, the server also decrypts and stores K for future communication.

- Data Transfer

To exchange data, both the client and server utilize the client's hash function (h) and a one-time key (K). More specifically, message X is transmitted in the form of K(h(X)).

The initial step is to create packets that can be sent over the network. It is simpler to work with packets of a fixed size. The following scheme has been utilized to generate these packets:

- chopping the message into fixed sized (ndatabytes big) pieces.
- Calculating checksum and append it to the message.
- assembling the packet as

```
 +-----+--------------------------------+-------------------+
|  n   |           data bytes           |    checksum     |
+-----+--------------------------------+-------------------+


|<-1->|<------- ndatabytes -------- >|<- ncheckbytes->|
```

The data bytes field has a length of one byte and includes the value of n, representing the actual number of data bytes to be sent. If the number of bytes to be transmitted is less than ndatabytes, the receiving entity can determine that only the first n bytes of data bytes are significant. The checksum is computed using the user's specified hash function, which is described later. Once the packet is constructed, it is encrypted using the one-time key K.

➢ Hash Function (Calculating the checksum and Assembling packets)

$$h(X) := \left(\sum_{i=1}^{ndatabytes} (pattern \,\&\, data\_bytes[i]) * k\right) \bmod 2^{8*ncheckbytes}$$

data_bytes[i] - ith data byte in the packet

pattern - one byte bit pattern

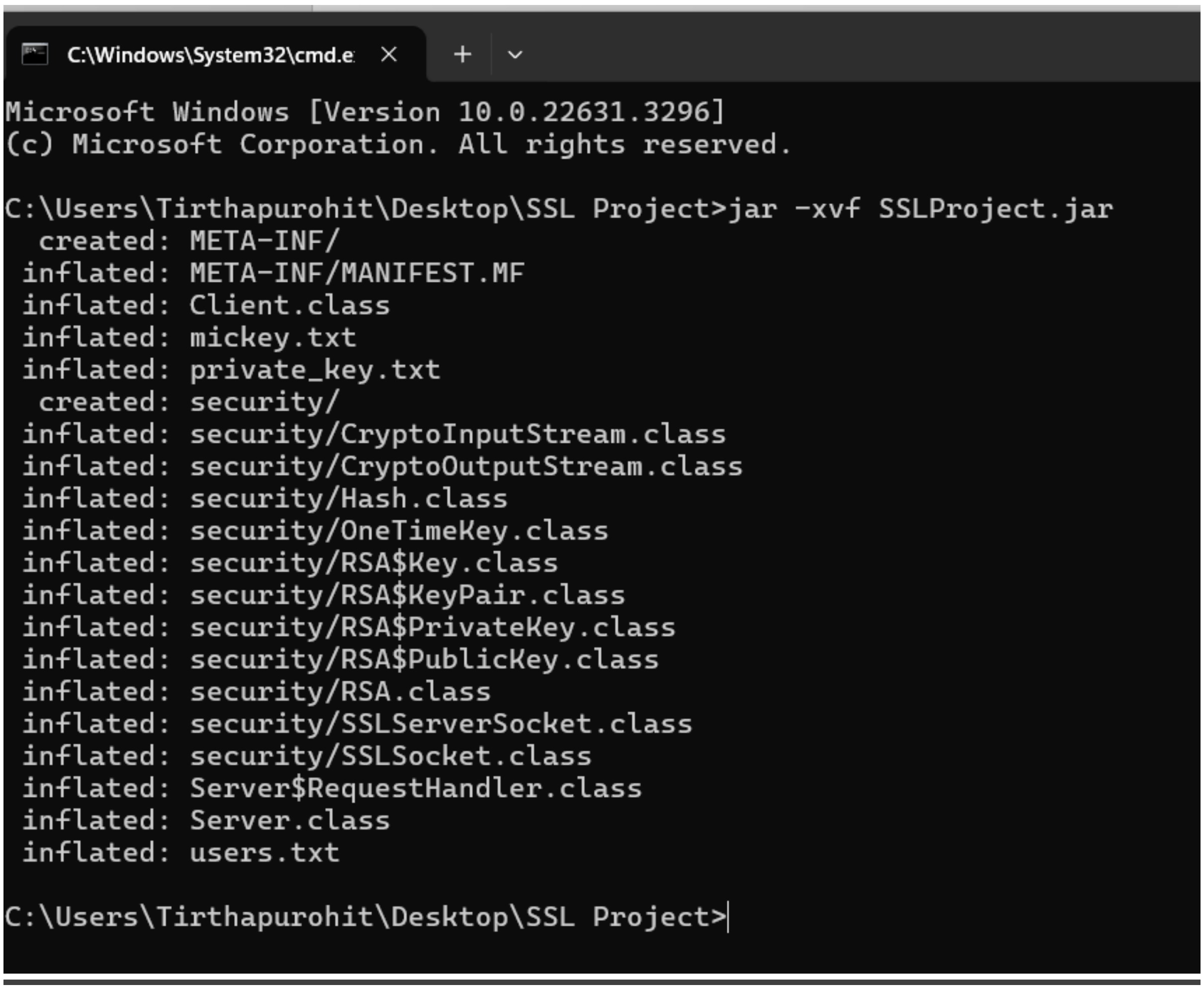
k - odd integer

& - bitwise AND operator

"ndatabytes" indicates the count of bytes contained within the packet, and "data_bytes[i]" represents the i-th byte of data in the packet. Meanwhile, "ncheckbytes" denotes the number of check bytes, "pattern" is a one-byte bit pattern, "k" is an odd integer, and the "&" operator signifies the bitwise AND operation. Both the user and the server possess knowledge of the pattern and k, which were exchanged offline.

How to Run Program

C:\Users\Tirthapurohit\Desktop\SSL Project>jar -xvf SSLProject.jar

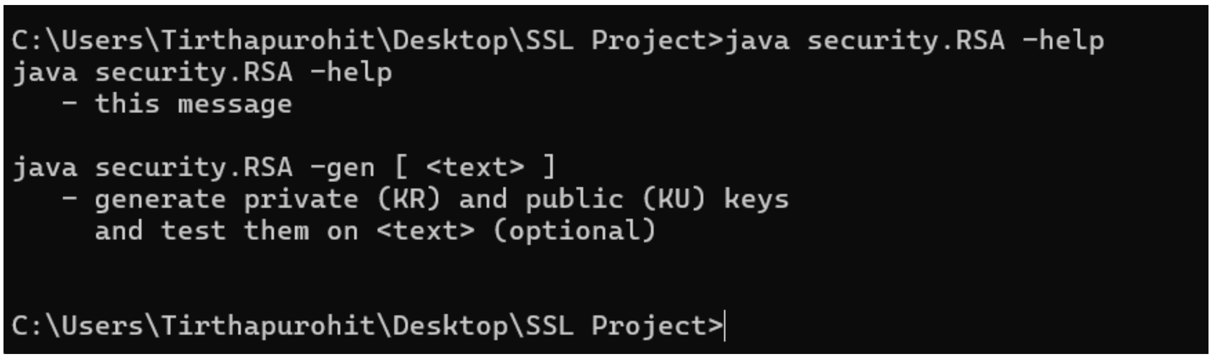


```
Microsoft Windows [Version 10.0.22631.3296]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Tirthapurohit\Desktop\SSL Project>jar -xvf SSLProject.jar
  created: META-INF/
 inflated: META-INF/MANIFEST.MF
 inflated: Client.class
 inflated: mickey.txt
 inflated: private_key.txt
  created: security/
 inflated: security/CryptoInputStream.class
 inflated: security/CryptoOutputStream.class
 inflated: security/Hash.class
 inflated: security/OneTimeKey.class
 inflated: security/RSA$Key.class
 inflated: security/RSA$KeyPair.class
 inflated: security/RSA$PrivateKey.class
 inflated: security/RSA$PublicKey.class
 inflated: security/RSA.class
 inflated: security/SSLServerSocket.class
 inflated: security/SSLSocket.class
 inflated: Server$RequestHandler.class
 inflated: Server.class
 inflated: users.txt

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

Part A:

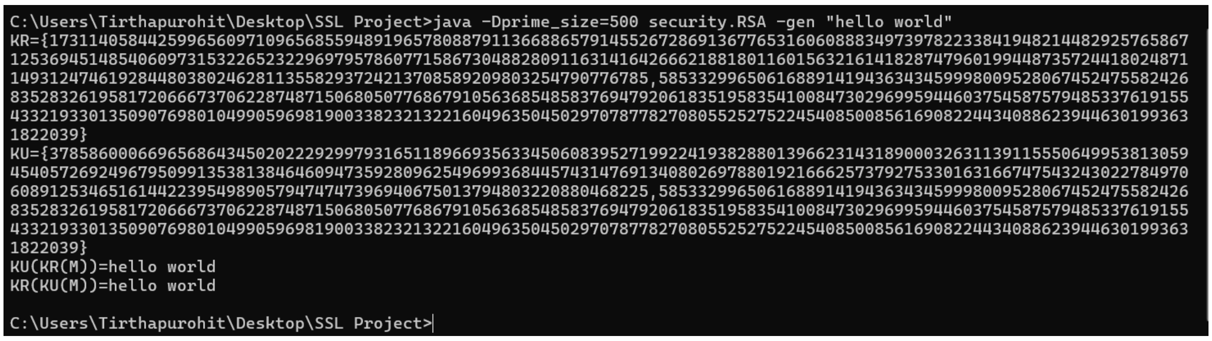
Implementing the RSA key generation and ciphering algorithm.

1) **C:\Users\Tirthapurohit\Desktop\SSL Project>java security.RSA -help**



```
C:\Users\Tirthapurohit\Desktop\SSL Project>java security.RSA -help
java security.RSA -help
   - this message

java security.RSA -gen [ <text> ]
   - generate private (KR) and public (KU) keys
     and test them on <text> (optional)


C:\Users\Tirthapurohit\Desktop\SSL Project>
```

2) **C:\Users\Tirthapurohit\Desktop\SSL Project>java -Dprime_size=500 security.RSA -gen "hello world"**

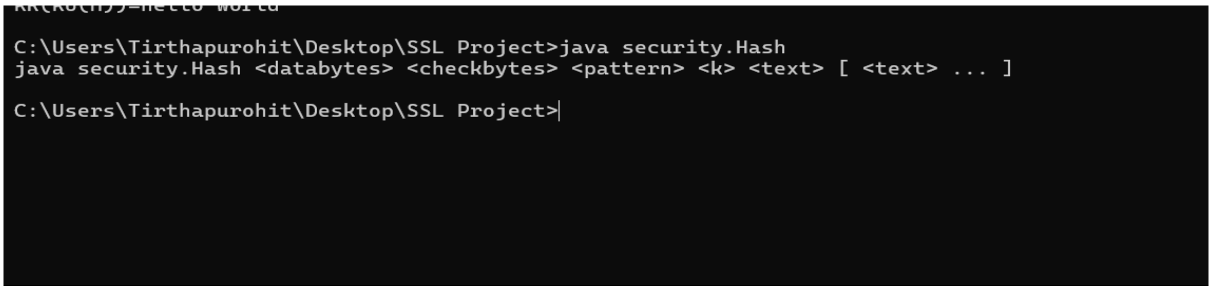


```
C:\Users\Tirthapurohit\Desktop\SSL Project>java -Dprime_size=500 security.RSA -gen "hello world"
KR={17311405844259965609710965685594891965780887911366886579145526728691367765316060888349739782233841948214482925765867
1253694514854060973153226523229697957860771586730488280911631416426662188180116015632161418287479601994487357244180248 71
14931247461928448038024628113558293724213708589209803254790776785,585332996506168891419436343459998009528067452475582426
835283261958172066673706228748715068050776867910563685485837694792061835195835410084730296995944603754587579485337619155
43321933013509076980104990596981900338232132216049635045029707877827080555252752245408500856169082244340886239444630199363
1822039}
KU={37858600066965686434502022292997931651189669356334506083952719922419382880139662314318900032631139115550649953813059
45405726924967950991353813846460947359280962549699368445743147691340802697880192166625737927533016316674754324302278 4970
60891253465161442239549890579474747396940675013794803220880468225,585332996506168891419436343459998009528067452475582426
835283261958172066673706228748715068050776867910563685485837694792061835195835410084730296995944603754587579485337619155
43321933013509076980104990596981900338232132216049635045029707877827080555252752245408500856169082244340886239444630199363
1822039}
KU(KR(M))=hello world
KR(KU(M))=hello world

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

Part B:

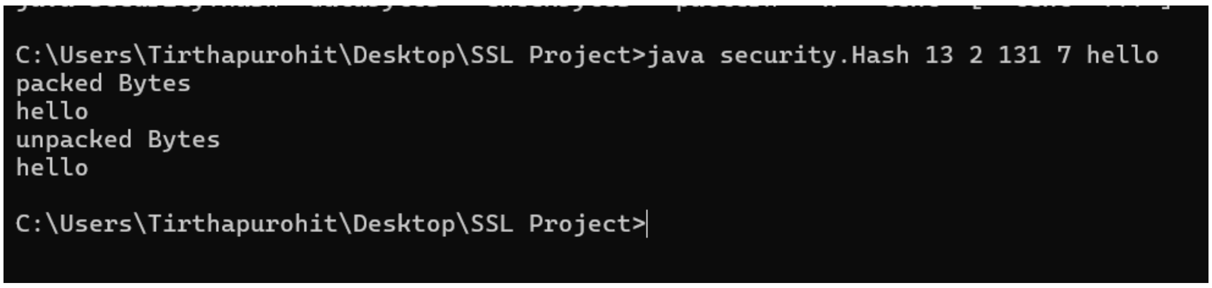
Implementing the hash function and one-time key encryption.

1) **C:\Users\Tirthapurohit\Desktop\SSL Project>java security.Hash**



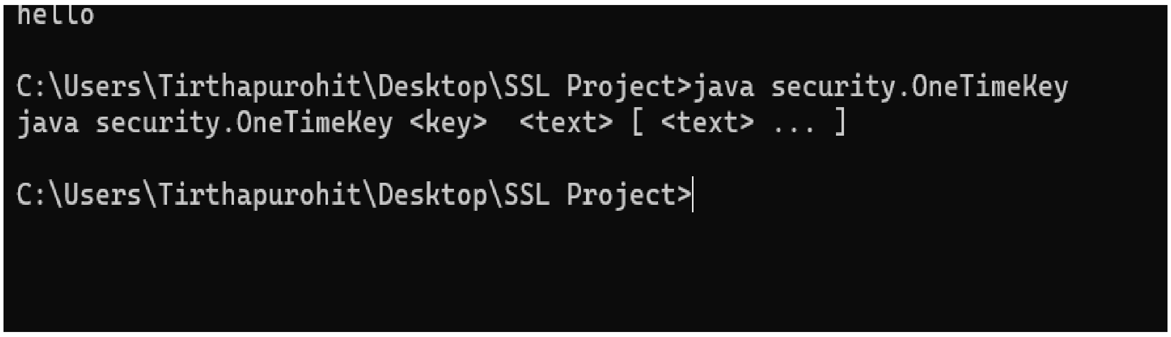
```
C:\Users\Tirthapurohit\Desktop\SSL Project>java security.Hash
java security.Hash <databytes> <checkbytes> <pattern> <k> <text> [ <text> ... ]

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

2) **C:\Users\Tirthapurohit\Desktop\SSL Project>java security.Hash 13 2 131 7 hello**



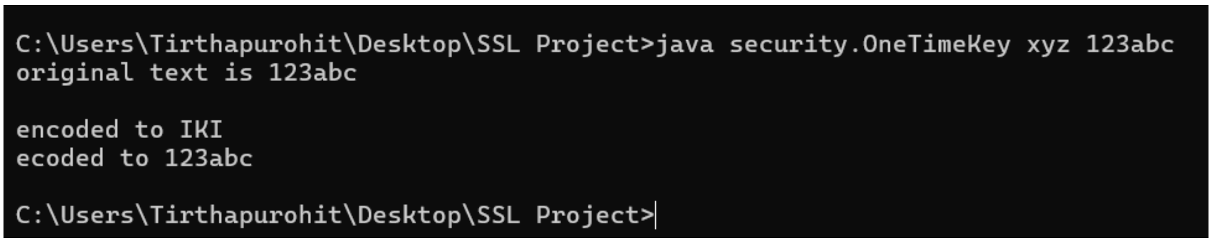
```
C:\Users\Tirthapurohit\Desktop\SSL Project>java security.Hash 13 2 131 7 hello
packed Bytes
hello
unpacked Bytes
hello

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

3) **C:\Users\Tirthapurohit\Desktop\SSL Project>java security.OneTimeKey**



```
hello

C:\Users\Tirthapurohit\Desktop\SSL Project>java security.OneTimeKey
java security.OneTimeKey <key>  <text> [ <text> ... ]

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

4) **C:\Users\Tirthapurohit\Desktop\SSL Project>java security.OneTimeKey xyz 123abc**



```
C:\Users\Tirthapurohit\Desktop\SSL Project>java security.OneTimeKey xyz 123abc
original text is 123abc

encoded to IKI
ecoded to 123abc

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

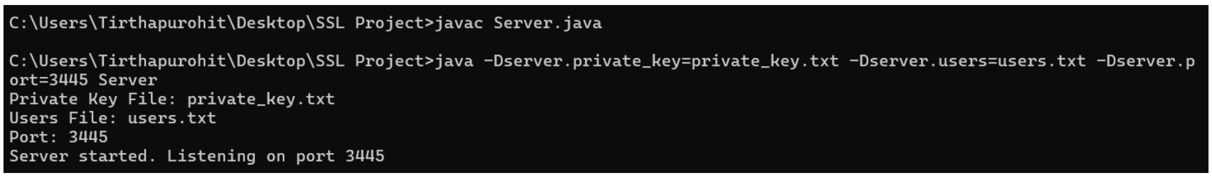
Part C:

<u>Open Two Terminals</u>

In Terminal 1

javac Server.java

java -Dserver.private_key=private_key.txt -D server.users=users.txt -Dserver.port=3445 Server

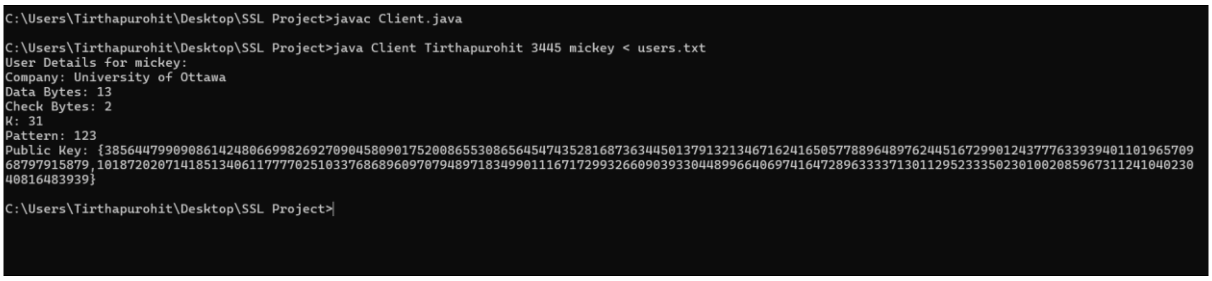


```
C:\Users\Tirthapurohit\Desktop\SSL Project>javac Server.java

C:\Users\Tirthapurohit\Desktop\SSL Project>java -Dserver.private_key=private_key.txt -Dserver.users=users.txt -Dserver.p
ort=3445 Server
Private Key File: private_key.txt
Users File: users.txt
Port: 3445
Server started. Listening on port 3445
```

In Terminal 2

javac Client.java

java Client hostname_of_system 3445 mickey < users.txt



```
C:\Users\Tirthapurohit\Desktop\SSL Project>javac Client.java

C:\Users\Tirthapurohit\Desktop\SSL Project>java Client Tirthapurohit 3445 mickey < users.txt
User Details for mickey:
Company: University of Ottawa
Data Bytes: 13
Check Bytes: 2
K: 31
Pattern: 123
Public Key: {38564479909086142480669982692709045809017520086553086564547435281687363445013791321346716241650577889648976244516729901243777633939401101965709
68797915879,101872020714185134061177770251033768689609707948971834990111671729932660903933044899664069741647289633337130112952333502301002085967311241040230
40816483939}

C:\Users\Tirthapurohit\Desktop\SSL Project>
```

Member’s Contribution

Mitali Purohit	33.33%
Bryan Nguyen	33.33%
Ishita Kumbhani	33.33%