Ex.No.1	IMPLEMENTATION AND CRYPTANALYSIS OF CAESAR CIPHER

AIM:

To implement

- Encryption
- Decryption
- Brute force Attack
- Frequency Analysis attack

in Caesar cipher using Java.

THEORY:

Encryption and Decryption in Caesar Cipher:

Caesar cipher is a basic letters substitution algorithm. It takes as input a message, and apply to every letter a particular shift. This shift used to be 3, according to history, when it was use by Caesar to encrypt war messages (so for example a would become d, b will be, and so on and so forth). Of course you can choose any shift you want. This is basically a modulo 26 addition; Caesar cipher, as Polybius Square cipher, is a monoalphabetical cipher. Like the others of this kind, the problem of this cipher is its really poor security. To break it, you can, like I do here, apply every shift to the ciphertext, and see if there's one that makes sense. The other way to break it is by looking to letters frequency

Example:

Encryption

Plain text: dhanuja

Key value: 4

Encrypted text: hylerne

Decryption

Encrypted text: hylerne

Key value: 4

Decrypted text: dhanuja

Brute Force Attack in Caesar Cipher:

A brute-force attack tries every possible decryption key for a cipher. Nothing stops a cryptanalyst from guessing one key, decrypting the ciphertext with that key, looking at the output, and then moving on to the next key if they didn't find the secret message. Because the brute-force technique is so effective against the Caesar cipher

Example:

Encrypted text: vhfxulwb

Key	Value	Meaningful word	status
1	ugewtkva	no	continue
2	tfdvsjuz	no	continue
3	security	yes	stop

Frequency Analysis Attack in Caesar Cipher :

It is based on the nature of the plain text language used for encryption. exploiting the nature of the plain text language.

Encryption in Caesar cipher is mere substitution

It is based on the study of the frequency of letters or groups of letters in a ciphertext. In all languages, different letters are used with different frequencies. It is possible to determine the correct order of letters from mixed words.

Example:

Encrypted text: rckncqr

Most frequently used letter in plain text: E

Most frequently used letter in encrypted text: C

K = C-P mod 26

K = 24

Decrypted text: tempest

ALGORITHM:

- A String of lower case letters, called Text.
- An Integer between 0-25 denoting the required shift.

ENCRYPTION :

Do until end of plaintext for each character :

Character at((value of character + key) % 26

DECRYPTION :

Do until end of plaintext for each character :

Character at((value of character - key) % 26

C = (P+K) % 26

P = (C-K) % 26

FREQUENCY ANALYSIS ATTACK:

Most frequently used letter in plain text: E

Most frequently used letter in encrypted text: C

K = C-P mod 26

Server.java :

1. Start
2. Import essential packages.
3. Create object for the ServerSocket class with the port number to provide service to client.
4. Based on the choice given by client invoke respective method and perform the action.
5. If the client needs to perform encryption use the formula (P + K) % 26 to find the cipher
text (where P is the numeric equivalent of the plain text sent by the client, K is the key
value).
6. If the client needs to perform decryption use the formula (C - K) % 26 to find the plain text (
where C is the numeric equivalent of the cipher text sent by the client, K is the key value).

7. If the client needs to perform the brute force approach try to convert the cipher text into
plain text with all possible key values from 1 to 26, until the client accepts the plain text
using the formula (C - K) % 26(where C is the numeric equivalent of the cipher text sent by
the client, K is the key value).
8. If the client needs to perform frequency analysis attack in the sense first find the key value
by subtracting the high frequency element in frequency analysis table from the frequent
element from the plain text. Using the key calculate the plain text.
9. End

Client.java :

1. Start
2. Import essential packages.
3. Create object for the Socket class with host name, required port number.
4. Get the service to needed from the client by displaying the menu like encryption, decryption,
brute force approach, frequency analysis attack.
5. Get the text to be converted, key value (if needed) and the choice.
6. Send the text, key, choice the client gave to the server using the DataOuputStream class and
WriteUTF() method available in Socket class.
7. Close the socket file, after getting the service.

8. End

CODING:

Server.java

```
import java.net.*;
import java.io.*;
import java.util.*;
public class Server
{
static String alpha = "abcdefghijklmnopqrstuvwxyz";
public static String encode (String P, int k)
{
P = P.toLowerCase();
String C="";
for(int i=0; i<P.length(); i++)
{
int pos = alpha.indexOf(P.charAt(i));
int key = (k+pos)%26;
char x = alpha.charAt(key);
C += x;
}
return(C);
}

public static String decode (String C, int k)
{
C = C.toLowerCase();
String P="";
for(int i=0; i<C.length(); i++)
{
int pos = alpha.indexOf(C.charAt(i));
int key = (pos-k)%26;
if(key<0)
```

```
{
key = alpha.length() + key;
}
char x = alpha.charAt(key);
P += x;
}
return(P);
}
public static void main(String[] args)
{
try
{
ServerSocket s = new ServerSocket(3000);
System.out.println("server started");
Socket socket = s.accept();
System.out.println("Client accepted");
DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
DataInputStream dis = new DataInputStream(socket.getInputStream());
int ch = dis.readInt();
if (ch==1)
{
String P = dis.readUTF();
int k = dis.readInt();
dos.writeUTF(encode(P, k));
}
if (ch==2)
{
String C = dis.readUTF();
int k = dis.readInt();
dos.writeUTF(decode (C, k));
}
if (ch==3)
{
```

```
String C = dis.readUTF();
for (int i=0;i<26; i++)
{
System.out.println("key: "+i+" Plain text: "+ decode(C, i));
}
}
if (ch==4)
{
String C = dis.readUTF();
int[] f = new int[C.length()];
char max = C.charAt(0);
char cipher[] = C.toCharArray();
int i, j, m, k, flag;
flag = dis.readInt();
for (i=0; i<cipher.length; i++)
{
f[i] = 1;
for (j = i+1; j<cipher.length; j++)
{
if(cipher[i]==cipher[j] && cipher[i]!='0')
{
f[i]++;
cipher[j]='0';

}
}
if (cipher[i]=='0')
{
f[i]=0;
}
}
int c=0;
while (flag==0)
{
```

```
m=f[0];
c++;
for (i=0; i<f.length; i++)
{
if (m<f[i])
{
m = f[i];
max = cipher[i];
}
}
System.out.println("The most frequently occuring element is: "+ max+" with frequency count: "+ m);
String P = dis.readUTF();
k = (alpha.indexOf(max)-alpha.indexOf(P))%26;
dos.writeUTF(decode(C, k));
flag = dis.readInt();
if (c!=0)
{
for(i=0; i<cipher.length; i++)
{
if (cipher[i]==max)
{
f[i]=0;
break;
}
}
}
}
}
socket.close();
}
catch(IOException e)
{
System.out.println(e);
```

```
}
}
}
```

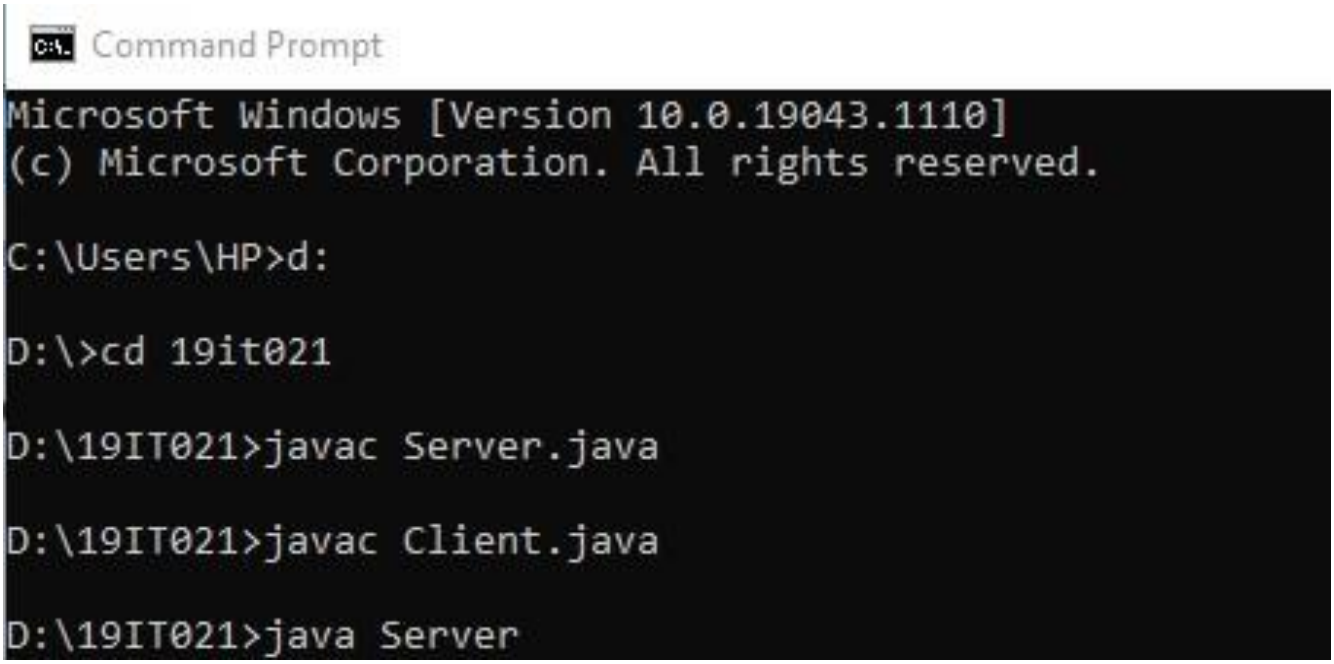
Client.java

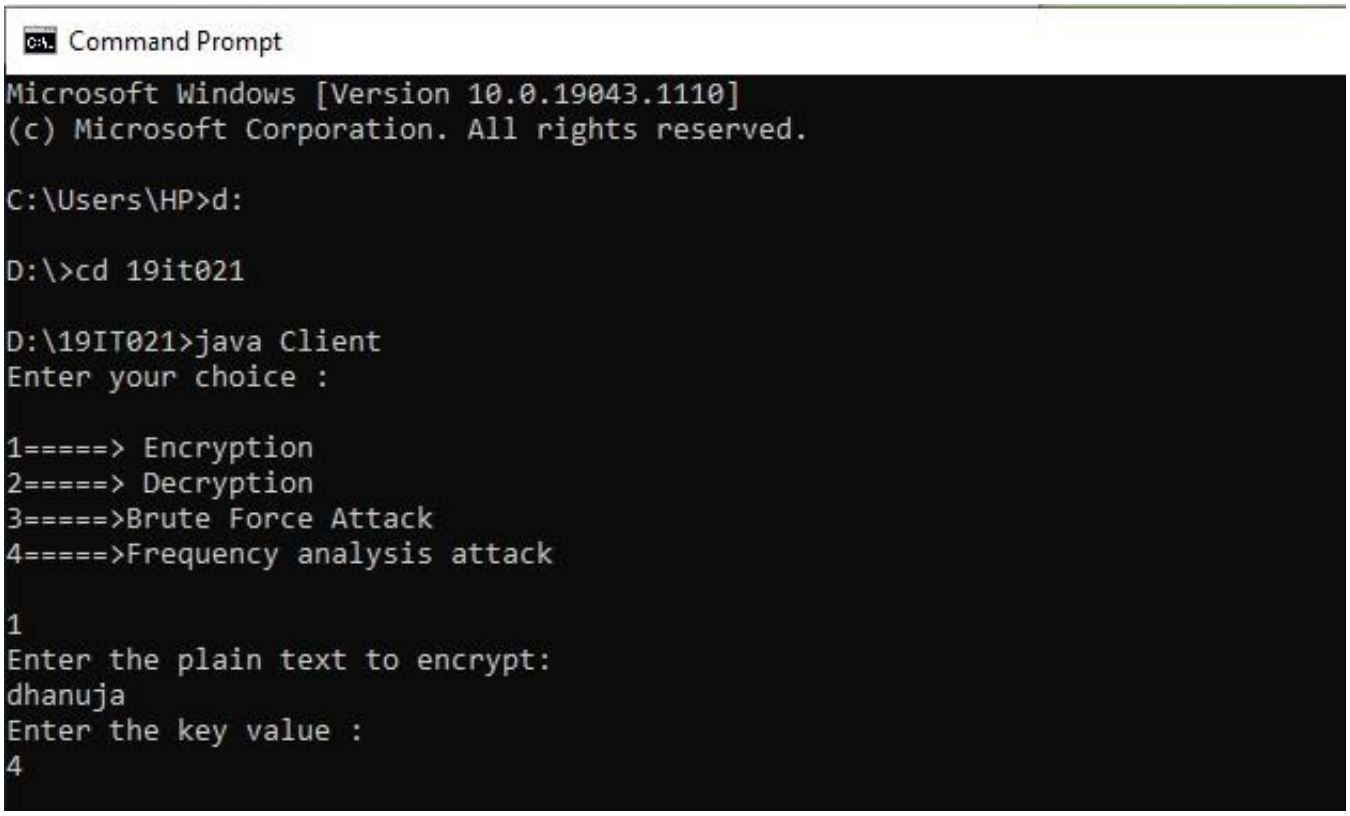
```
import java.io.*;
import java.net.*;
import java.util.*;
public class Client
{
public Client(String address,int port)
{
try
{
Socket socket = new Socket(address,port);
System.out.println("Server connected!");
Scanner ip = new Scanner(System.in);
DataInputStream dis = new DataInputStream(socket.getInputStream());
DataOutputStream dos = new DataOutputStream(socket.getOutputStream());
System.out.println("1===> Encryption\n2===> Decryption\n3===> Brute Force\n4===> Frequency analysis attack");
System.out.print("Enter your option: "); //getting input from user
int x = ip.nextInt();
dos.writeInt(x);
if(x==1)
{
System.out.println("ENCRYPTION!");
System.out.print("Enter plain text: ");
String P = ip.next();
System.out.print("Enter key: ");
int k = ip.nextInt();
dos.writeUTF(P);
dos.writeInt(k);
String res = dis.readUTF();
```

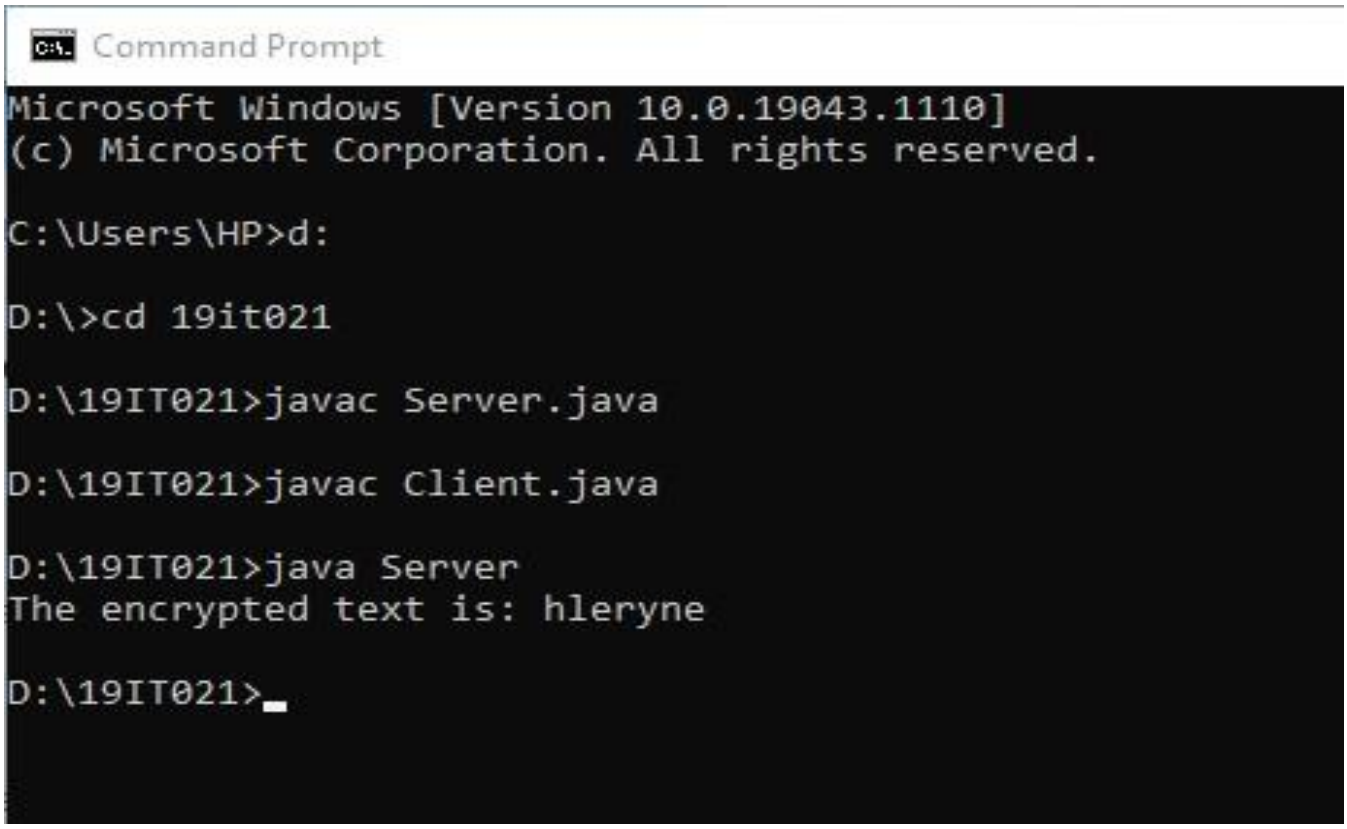
```
System.out.println("Cipher Text: "+res);
}
if(x==2)
{
System.out.println("DECRYPTION!");
System.out.print("Enter Cipher text: ");
String C = ip.next();
System.out.print("Enter key: ");
int k = ip.nextInt();
dos.writeUTF(C);
dos.writeInt(k);
String res = dis.readUTF();
System.out.println("Plain Text: "+res);
}
if (x==3)
{
System.out.println("BRUTE FORCE ATTACK!");
System.out.print("Enter Cipher text: ");
String C = ip.next();
dos.writeUTF(C);
}
if (x==4)
{
System.out.println("FREQUENCY ANALYSIS ATTACK!");
System.out.print("Enter Cipher text: ");
String C = ip.next();
dos.writeUTF(C);
String s, res;
int f=0;
dos.writeInt(f);
while(f==0)
{
System.out.print("Enter the (next) most frequently occuring in Plain Text:
");
```

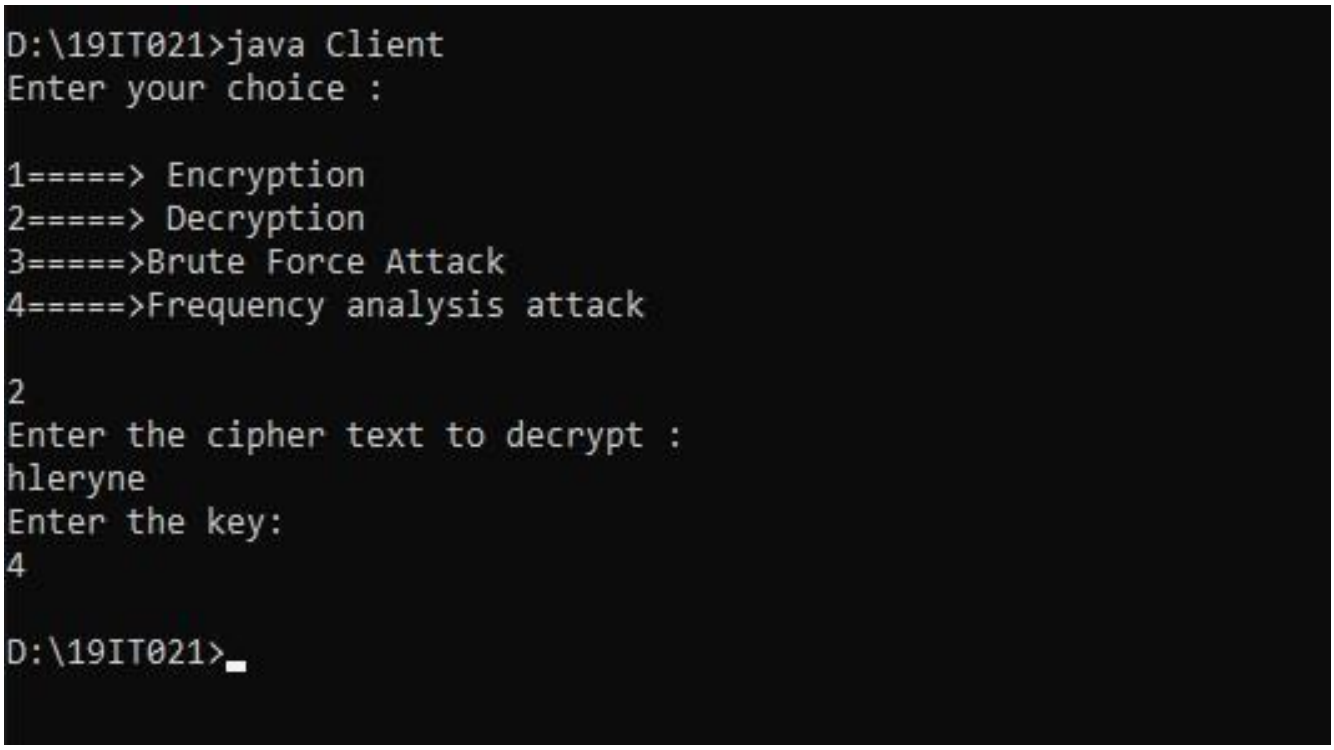
```
s = ip.next();
dos.writeUTF(s);
res = dis.readUTF();
System.out.println("Plain Text: "+res);
System.out.print("Is this the required plain text? (Enter '1' for yes and '0' for no): ");
f = ip.nextInt();
dos.writeInt(f);
}
}
}
catch(Exception e)
{
System.out.println(e);
}
}
public static void main(String[] args)
{
Client Client=new Client("localhost", 3000);
}
}
```

SCREEN SHOTS:





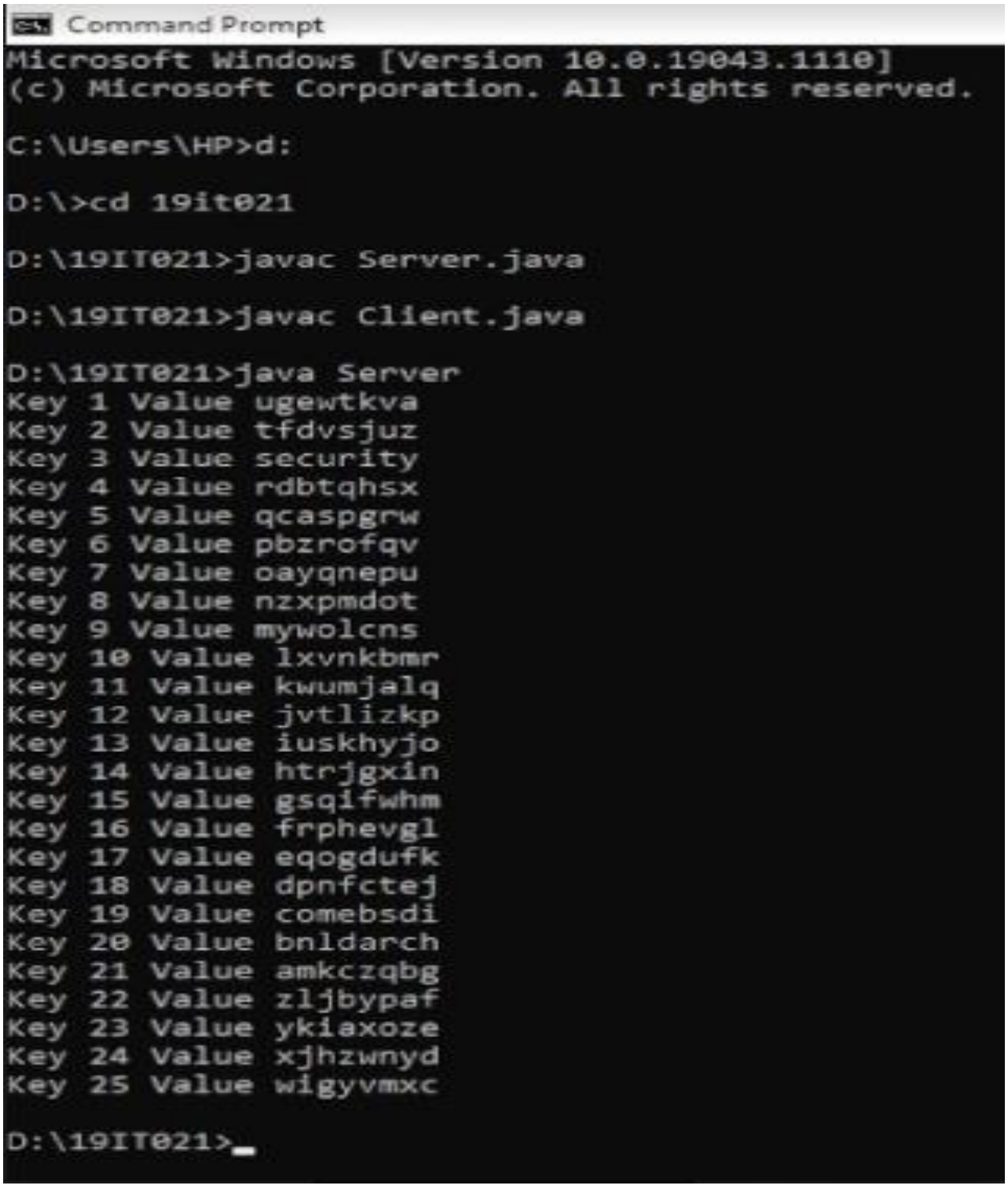

Command Prompt
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HP>d:
D:\>cd 19it021
D:\19IT021>javac Server.java
D:\19IT021>javac Client.java
D:\19IT021>java Server
The encrypted text is: hleryne
D:\19IT021>

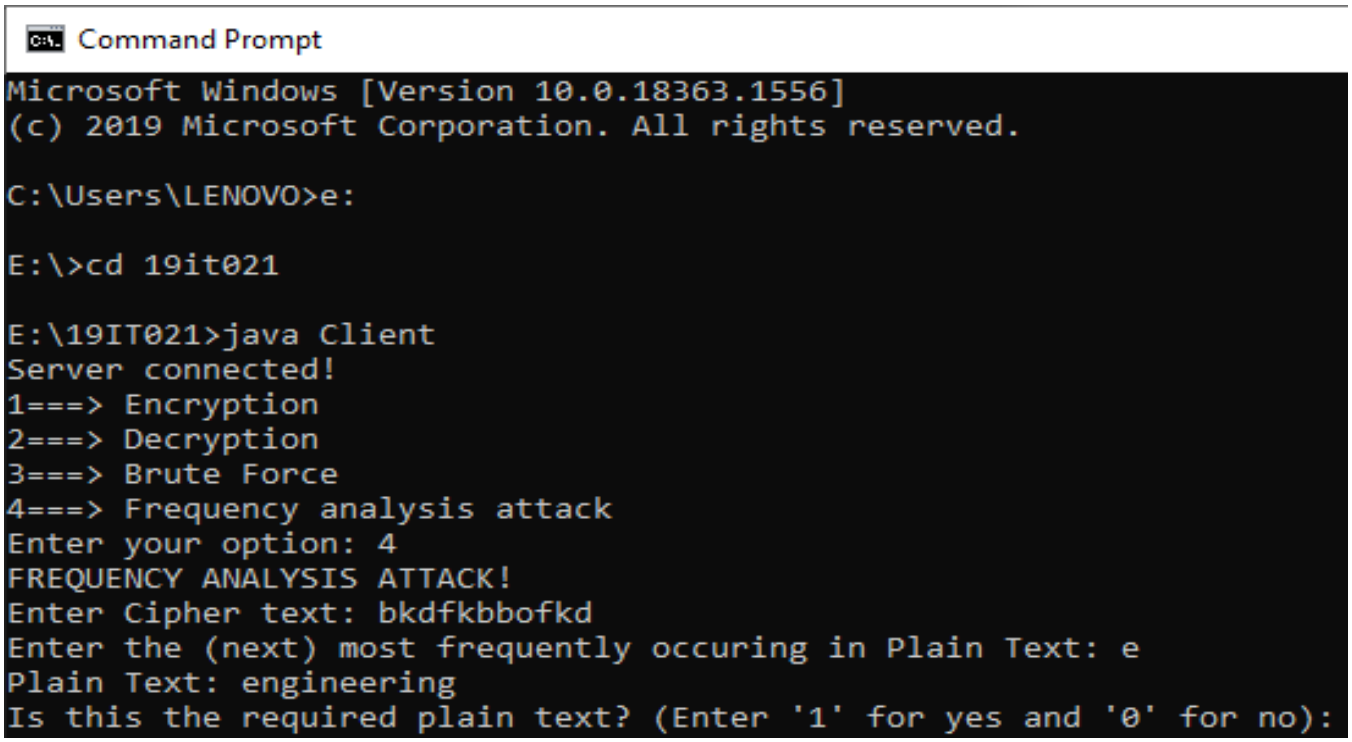

D:\19IT021>java Client
Enter your choice :
1=====> Encryption
2=====> Decryption
3=====>Brute Force Attack
4=====>Frequency analysis attack
2
Enter the cipher text to decrypt :
hleryne
Enter the key:
4
D:\19IT021>

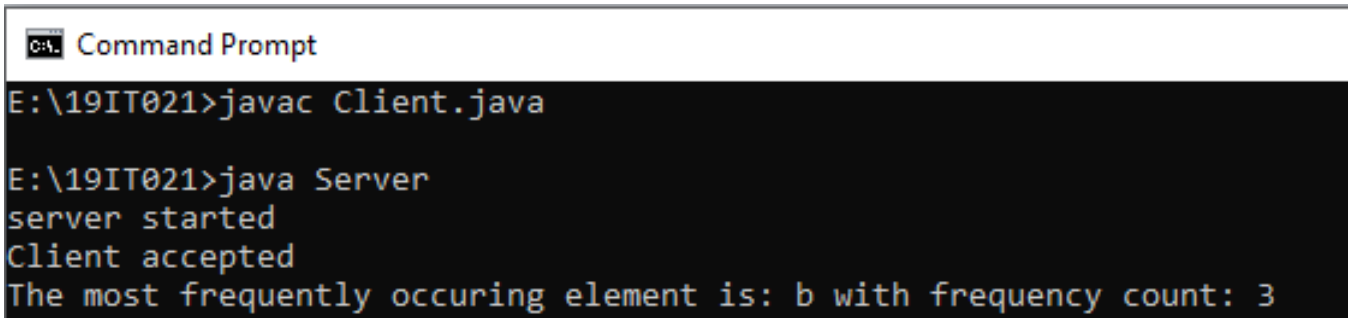
```
D:\19IT021>java Server
The decrypted plain text: dhanuja

D:\19IT021>
```

Command Prompt

```
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.

C:\Users\HP>d:

D:\>cd 19it021

D:\19IT021>java Client
Enter your choice :
1--->Encryption
2--->Decryption
3--->Brute Force Attack
4--->frequency analysis Attack
3
Enter the cipher text to decode for brute force :
vhfxulwb

D:\19IT021>
```


Command Prompt
Microsoft Windows [Version 10.0.19043.1110]
(c) Microsoft Corporation. All rights reserved.
C:\Users\HP>d:
D:\>cd 19it021
D:\19IT021>javac Server.java
D:\19IT021>javac Client.java
D:\19IT021>java Server
Key 1 Value ugewtkva
Key 2 Value tfdvsjuz
Key 3 Value security
Key 4 Value rdbtqhsx
Key 5 Value qcaspgrw
Key 6 Value pbzrofqv
Key 7 Value oayqnepu
Key 8 Value nzxpmdot
Key 9 Value mywolcns
Key 10 Value lxvnkbmr
Key 11 Value kwumjalq
Key 12 Value jvtlizkp
Key 13 Value iuskhyjo
Key 14 Value htrjgxin
Key 15 Value gsqifwhm
Key 16 Value frphevgl
Key 17 Value eqogdufk
Key 18 Value dpnfctej
Key 19 Value comebsdi
Key 20 Value bnldarch
Key 21 Value amkczqbg
Key 22 Value zljbypaf
Key 23 Value ykiaxoze
Key 24 Value xjhzwnyd
Key 25 Value wigyvmxc
D:\19IT021>





RESULT:

Thus the programs for

- Encryption
- Decryption
- Bruteforce Attack
- Frequency Analysis attack

in Caesar cipher are implemented in Java and the results are verified.

EVALUATION:

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	7	
Use of Comment lines and standard coding practices	3	
Viva	10	
Sub Total	20	
Completion of experiment on time	3	
Documentation	7	
Sub Total	10	
Signature of the faculty with Date		

Ex.No.2	**IMPLEMENTATION AND CRYPTANALYSIS OF HILL CIPHER**

AIM:

To implement

- Encryption
- Decryption
- Known Plain test – cipher text attack

in Hill cipher using Java.

THEORY:

Encryption

Caesar cipher is a basic letters substitution algorithm. It takes as input a message, and apply to every letter a particular shift. This shift used to be 3, according to history, when it was use by Caesar to encrypt war messages (so for example a would become d, b will be, and so on and so forth). Of course you can choose any shift you want. This is basically a modulo 26 addition; Caesar cipher, as Polybius Square cipher, is a mono alphabetical cipher. Like the others of this kind, the problem of this cipher is its really poor security. To break it, you can, like I do here, apply every shift to the ciphertext, and see if there's one that makes sense. The other way to break it is by looking to letters frequency

C = K * P mod 26

C – Cipher Matrix K – Key Matrix P – Plain Text Matrix

This cipher matrix is then converted back to letters.

Decryption

Decryption is the process of reversing the encryption. Since we are using matrix multiplication for the encryption process, the decryption requires finding the inverse of the key matrix. After finding the inverse of the key matrix used for encryption, the same process as encryption is carried out using the cipher text matrix. The formula used for this is:

P = K-1 * C mod 26

C – Cipher Matrix K – Key Matrix P – Plain Text Matrix

This plain text matrix is then converted back to letters.

Constraints for selecting Key Matrix in Hill Cipher

• The Key Matrix must be a square matrix

• The determinant of the Key Matrix must not be equal to 0

• The GCD of determinant of Key Matrix and 26 must be equal to 1, so that the inverse exists in the required Z26 modulo.

Steps for calculation of inverse of Key Matrix in Modulo arithmetic

- Calculate the determinant of the Key Matrix
- Find the multiplicative inverse of the determinant in the Z26
- Calculate the adjugate of the Key Matrix.
- Multiply the found multiplicative inverse with the adjugate to arrive at the inverse Key Matrix

Euclidean and Extended Euclidean algorithm

Euclidean algorithm provides an easy way to compute the Greatest Common Divisor of two

integers. Greatest Common Divisor(GCD) is the largest integer that divides both the given the integers without reminder. The Euclidean algorithm is based on the principle that the greatest common divisor of two numbers does not change if the larger number is replaced by its difference with the smaller number The function GCD(int a, int b) return integer a until b equals to 0 while recursively performing GCD(a % b, b).

The Extended Euclidean algorithm is used to find the inverse of the integers along with GCD of the two numbers. The extended Euclidean algorithm is particularly useful when a and b are co prime. With that provision, x is the modular multiplicative inverse of a modulo b, and y is the modular multiplicative inverse of b modulo a.

The Extended Euclidean algorithm performs the same operations as Euclidean algorithm and along with it parallelly calculate the value of s. This value of s is the required multiplicative inverse of the given two numbers.

Known Plaintext and Cipher text attack with an example

This means that if we have two variables in our hand, then we can easily identify the third variable. Taking advantage of this simple fact, if the cipher text and plain text are known then it is feasible to find the Key matrix. If the key matrix is found by this attack, then any further communication can be intercepted and decrypted with the found key matrix. Hill Cipher is vulnerable to the Known Plain Text Attack. From the existing formula for encryption, we can derive formula for the attack as below,

K = C * P-1 mod 26

ALGORITHM:

Encryption

- Convert the plain text and key into number matrix.

- Perform matrix multiplication between key and plain text.
- Do modulo m with all element of matrix.
- Convert back the cipher text to letters and print the result

Decryption

- Convert the cipher text and key into number matrix.
- Find determinant of the key matrix and then do modulo.
- Calculate the inverse of the determinant.
- Calculate inverse of key matrix.
- Perform matrix multiplication between inverse of key (K) and C.
- Do modulo m with all element of matrix.
- Convert back the plain text to letters and print the result

Known Plaintext attack and Cipher text attack

- Get plain text and cipher text as inputs.
- Convert it to number matrix.
- Perform matrix multiplication between inverse of P and C.
- Do modulo m with all element of matrix.
- The resultant matrix is the key.

DESIGN

The functions that I have used in my code are,

- NumericEncoding
- Decode
- Matrix multiplication
- Multiple Inverse
- Determinant
- Cofactor
- Adjoint

- Encryption
- Decryption
- Known PT-CT

This is the design of my code that I have implemented.

MODULE WISE CODING AND UNIT TESTING

ENCODING

```
int[][] numencode(String ptext,int key)
{
int row=0;
int col=0;
int res=0;
int extracol=0;
int k=0;
int i=0;
int j=0;
int col1=0;
String alpha="abcdefghijklmnopqrstuvwxyz";
int len=ptext.length();
int[] enc=new int[100];
for(i=0;i<len;i++)
{
enc[i]=alpha.indexOf(ptext.toLowerCase().charAt(i));
}
if(len%key==0)
{
row=key;
col1=len/key;
}
else
```

```
{
row=key;
res=len%key;
res=key -res;
col=res+len;
col1=col/key;
for(i=len;i<col;i++) {
enc[i]=23; }}
int[][] ans=new int[key][col1];
k=0;
for(i=0;i<col1;i++) {
for(j=0;j<key;j++) {
ans[j][i]=enc[k];
k++; }}
for(i=0;i<key;i++) {
for(j=0;j<col1;j++) {
System.out.print(ans[i][j]);
System.out.print("
\t");
}
System.out.println(); }
return ans; }
```

DECODING

```
void decode(int c[][],int len) {
String p="";
//intlen=c.length;
int k=c.length;
int q=len/k;
for(int j=0;j<=q;j++)
```

```
{
for(int i=0;i<k;i++) {
//convert integer to character
p+=ALPHABET.charAt(c[i][j]); }
} System.out.print(p); }
```

DECODE

```
void decode1(int c[][],int len) {
String p="";
//intlen=c.length;
int k=c.length;
int q=len/k;
for(int j=0;j<q;j++) {
for(int i=0;i<k;i++) {
p+=ALPHABET.charAt(c[i][j]); }}
System.out.println(p.substring(0,len -1)); }
```

MATRIX MULTIPLICATION

```
int[][] matmul (int key[][],int a[][]) {
int row=key.length;
int col=a[0].length;
int col1=key[0].length;
int [][] c= new int[row][col];
for (int i = 0; i< row; i++) {
for (int j = 0; j < col; j++) {
for (int k = 0; k < row; k++) {
c[i][j] = c[i][j] + key[i][k] * a[k][j]; }}}
return c; }
```

DETERMINANT

```
static int determinant(int mat[][], int n)
{
```

```
int D = 0; // Initialize result
if (n == 1)
return mat[0][0];
// cofactors
int temp[][] = new int[n][n];
int sign = 1;
for (int f = 0; f < n; f++)
{
getCofactor(mat, temp, 0, f, n);
D += sign * mat[0][f]
* determinant(temp, n - 1);
sign = -sign; }
D=D%26;
if(D<0)
{
D=26+D;
return D;
}
return D;
}
```

MULTIPLE INVERSE

```
int mulinverse(int a,int m) {
a = a % m;
for (int x = 1; x < m; x++)
if ((a * x) % m == 1)
return x;
return 1; }
int[][] inverse(int adjoint1[][],int g,int N) {
int k[][]=new int[N][N];
```

```
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
k[i][j]=(g*adjoint1[i][j])%26;
}
}
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
if(k[i][j]<0)
{
k[i][j]=26+k[i][j];
}
}
}
return k; }
```

ADJOINT

```
int[][] adjoint(int mat[][],int N)
{
int si = 1;
int[][]temp=new int[N][N];
int[][]adjoint=new int[N][N];
for (int i=0; i<N; i++)
{
for (int j=0; j<N; j++)
{
//adjoint
```

```
getCofactor(mat, temp, i, j, N);
si = ((i+j)%2==0)? 1: -1;
adjoint[j][i] = (si)*(determinant(temp, N-1));
}
}
return adjoint;
}
```

COFACTOR

```
static void getCofactor(int mat[][],int temp[][], int p, int q, int n)
{
int i = 0, j = 0;
for (int row = 0; row < n; row++)
{
for (int col = 0; col < n; col++)
{
if (row != p && col != q)
{
temp[i][j++] = mat[row][col];
if (j == n
- 1)
{
j = 0;
i++;
}
}
}
} }
```

ENCRYPTION

```
int[][] encrypt(int key[][] ,int a[][] ,int N) {
```

```
int row=a.length;
int col=a[0].length;
int mat1[][]=new int[N][col];
char [][] replace=new char[N][col];
mat1 = matmul(key,a);
System.out.println("The encrypted matrix is: ");
for (int i = 0; i< N; i++) {
System.out.println();
for (int j = 0; j < col; j++) {
mat1[i][j]=(mat1[i][j])%26;
System.out.print("
\t"+mat1[i][j]);
}}
return mat1; }
```

DECRYPTION

```
int[][] decrypt(int key[][], int b[][],int N) {
int m=26;
int row=b.length;
int col=b[0].length;
int[][] adjoint1=new int [N][N];
int [][] keyinverse=new int[N][N];
int[][] mat2=new int[row][col];
int l=determinant(key, N);
System.out.println("Determinant " + "of the matrix is : "+ l );
int g=mulinverse(l,m);
System.out.println("Inverse of determinant: "+g);
adjoint1=adjoint(key,N);
keyinverse=inverse(adjoint1,g,N);
mat2=matmul(keyinverse,b);
```

```
System.out.println("Decrypted matrix: ");
for(int i=0;i<row;i++)
{
System.out.println();
for(int j=0;j<col;j++)
{
mat2[i][j]=(mat2[i][j])%26;
System.out.print("\t"+mat2[i][j]);
}
}
return mat2;
}
```

KNOWN PT-CT

```
int[][] knownptct(int a[][],int b[][],int N2)
{
int row=a.length;
int col=b[0].length;
int m=26;
int [][] adjoint1=new int[row][col];
int [][] keyinverse1=new int[row][col];
int [][] mat2=new int[row][col];
int l=determinant(b, row);
System.out.println("\n");
System.out.println("\n");
System.out.println("\n");
System.out.println("\nDeterminant " + "of the matrix is : "+ l );
int g=mulinverse(l,m);
System.out.println("Inverse of determinant: "+g);
adjoint1=adjoint(b,row);
```

```
keyinverse1=inverse(adjoint1,g,row);
mat2=matmul(a,keyinverse1);
System.out.println("Key matrix: ");
for(int i=0;i<N2;i++)
{
System.out.println();
for(int j=0;j<N2;j++)
{
mat2[i][j]=(mat2[i][j])%26;
System.out.print("\t"+mat2[i][j]);
}
}
return mat2;
}
```

INTEGRATED CODE AND TESTING

```
import java.util.*;
class Hillcipher
{
static final String ALPHABET="abcdefghijklmnopqrstuvwxyz";
int[][] numencode(String ptext,int key)
{
int row=0;
int col=0;
int res=0;
int extracol=0;
int k=0;
int i=0;
int j=0;
int col1=0;
```

```
String alpha="abcdefghijklmnopqrstuvwxyz";
int len=ptext.length();
int[] enc=new int[100];
for(i=0;i<len;i++)
{
enc[i]=alpha.indexOf(ptext.toLowerCase().charAt(i));
}
if(len%key==0) {
row=key;
col1=len/key; }
else {
row=key;
res=len%key;
res=key -res;
col=res+len;
col1=col/key;
for(i=len;i<col;i++) {
enc[i]=23; }}
int[][] ans=new int[key][col1];
k=0;
for(i=0;i<col1;i++) {
for(j=0;j<key;j++) {
ans[j][i]=enc[k];
k++; }}
for(i=0;i<key;i++) {
for(j=0;j<col1;j++) {
System.out.print(ans[i][j]);
System.out.print("\t");
}
```

```
System.out.println(); }
return ans; }
void decode(int c[][],int len) {
String p="";
//intlen=c.length;
int k=c.length;
int q=len/k;
for(int j=0;j<=q;j++) {
for(int i=0;i<k;i++) {
//convert integer to character
p+=ALPHABET.charAt(c[i][j]); }}
System.out.print(p); }
void decode1(int c[][],int len) {
String p="";
//intlen=c.length;
int k=c.length;
int q=len/k;
for(int j=0;j<q;j++) {
for(int i=0;i<k;i++) {
p+=ALPHABET.charAt(c[i][j]); }}
System.out.println(p.substring(0,len -1)); }
int[][] matmul (int key[][],int a[][]) {
int row=key.length;
int col=a[0].length;
int col1=key[0].length;
int [][] c= new int[row][col];
for (int i = 0; i< row; i++) {
for (int j = 0; j < col; j++) {
for (int k = 0; k < row; k++) {
```

```
c[i][j] = c[i][j] + key[i][k] * a[k][j]; }}}
return c; }
static int determinant(int mat[][], int n)
{
int D = 0; // Initialize result
if (n == 1)
return mat[0][0];
// cofactors
int temp[][] = new int[n][n];
int sign = 1;
for (int f = 0; f < n; f++)
{
getCofactor(mat, temp, 0, f, n);
D += sign * mat[0][f]* determinant(temp, n- 1);
sign =-sign; }
D=D%26;
if(D<0)
{
D=26+D;
return D;
}
return D;
}
int mulinverse(int a,int m) {
a = a % m;
for (int x = 1; x < m; x++)
if ((a * x) % m == 1)
return x;
return 1; }
```

```
int[][] inverse(int adjoint1[][],int g,int N) {
int k[][]=new int[N][N];
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
k[i][j]=(g*adjoint1[i][j])%26;
}
}
for(int i=0;i<N;i++)
{
for(int j=0;j<N;j++)
{
if(k[i][j]<0)
{
k[i][j]=26+k[i][j];
}
}
}
return k; }
int[][] adjoint(int mat[][],int N)
{
int si = 1;
int[][]temp=new int[N][N];
int[][]adjoint=new int[N][N];
for (int i=0; i<N; i++)
{
for (int j=0; j<N; j++)
{
```

```
//adjoint
getCofactor(mat, temp, i, j, N);
si = ((i+j)%2==0)? 1: -1;
adjoint[j][i] = (si)*(determinant(temp, N-1));
}
}
return adjoint;
}
static void getCofactor(int mat[][],int temp[][], int p, int q, int n)
{
int i = 0, j = 0;
for (int row = 0; row < n; row++)
{
for (int col = 0; col < n; col++)
{
if (row != p && col != q)
{
temp[i][j++] = mat[row][col];
if (j == n- 1)
{
j = 0;
i++;
}
}
}
}
}
//encryption
int[][] encrypt(int key[][] ,int a[][] ,int N) {
```

```
int row=a.length;
int col=a[0].length;
int mat1[][]=new int[N][col];
char [][] replace=new char[N][col];
mat1 = matmul(key,a);
System.out.println("The encrypted matrix is: ");
for (int i = 0; i< N; i++) {
System.out.println();
for (int j = 0; j < col; j++) {
mat1[i][j]=(mat1[i][j])%26;
System.out.print("\t"+mat1[i][j]);
}}
return mat1; }
// decryption
int[][] decrypt(int key[][], int b[][],int N) {
int m=26;
int row=b.length;
int col=b[0].length;
int[][] adjoint1=new int [N][N];
int [][] keyinverse=new int[N][N];
int[][] mat2=new int[row][col];
int l=determinant(key, N);
System.out.println("Determinant " + "of the matrix is : "+ l );
int g=mulinverse(l,m);
System.out.println("Inverse of determinant: "+g);
adjoint1=adjoint(key,N);
keyinverse=inverse(adjoint1,g,N);
mat2=matmul(keyinverse,b);
System.out.println("Decrypted matrix: ");
```

```
for(int i=0;i<row;i++)
{
System.out.println();
for(int j=0;j<col;j++)
{
mat2[i][j]=(mat2[i][j])%26;
System.out.print("\t"+mat2[i][j]);
}
}
return mat2;
}
// Known plain text – cipher text attack
int[][] knownptct(int a[][],int b[][],int N2)
{
int row=a.length;
int col=b[0].length;
int m=26;
int [][] adjoint1=new int[row][col];
int [][] keyinverse1=new int[row][col];
int [][] mat2=new int[row][col];
int l=determinant(b, row);
System.out.println("\n");
System.out.println("\n");
System.out.println("\n");
System.out.println("\nDeterminant " + "of the matrix is : "+ l );
int g=mulinverse(l,m);
System.out.println("Inverse of determinant: "+g);
adjoint1=adjoint(b,row);
keyinverse1=inverse(adjoint1,g,row);
```

```
mat2=matmul(a,keyinverse1);
System.out.println("Key matrix: ");
for(int i=0;i<N2;i++)
{
System.out.println();
for(int j=0;j<N2;j++)
{
mat2[i][j]=(mat2[i][j])%26;
System.out.print("\t"+mat2[i][j]);
}
}
return mat2;
}
}
// main method to perform main operations
public class Hills{
public static void main (String[] args)
{
int m=26;
Hillcipher h=new Hillcipher();
Scanner input = new Scanner(System.in);
Scanner sc= new Scanner(System.in);
System.out.println("*******ENCRYPTION*******");
System.out.println("Enter the order of key: ");
int o=input.nextInt();
int[][] key=new int[o][o];
System.out.println("Enter the elements of key matrix "+"("+o*o+")"+":");
for(int i=0;i<o;i++)
{
```

```
for(int j=0;j<o;j++)
{
key[i][j]=input.nextInt();
}
}
System.out.println("Enter the plain text to encode :");
String pt=sc.nextLine();
int len=pt.length();
int [][]a=h.numencode(pt,o);
int[][]mat1=h.encrypt(key,a,o);
System.out.println("\nThe encrypted cipher text is: ");
h.decode(mat1,len);
System.out.println("\n*******DECRYPTION*******");
System.out.println("Enter the order of key matrix: ");
int o1=input.nextInt();
int[][] key1=new int[o1][o1];
System.out.println("\nEnter elements of key matrix:"+"("+o1*o1+")"+":");
for(int i=0;i<o1;i++)
{
for(int j=0;j<o1;j++)
{
key1[i][j]=input.nextInt();
}
}
System.out.println("Enter the cipher text to decode: ");
String pt1=sc.nextLine();
int len1=pt1.length();
int[][]a1=h.numencode(pt1,o1);
int mat2[][]=h.decrypt(key1,a1,o1);
```

```
System.out.println("\nThe decrypted plain text is: ");
h.decode1(mat2,len1);
System.out.println("*******KNOWN PLAIN TEXT - CIPHER TEXT*******");
System.out.println("Enter the Plain text ");
String pt3 = sc.nextLine();
System.out.println("Enter the cipher text ");
String ct = sc.nextLine();
System.out.println("Enter order of key matrix: ");
int o2=input.nextInt();
int [][] a2=new int[o2][o2];
int[][] a3=new int[o2][o2];
int [][]mat3=new int[o2][o2];
a2=h.numencode(pt3,o2);
a3=h.numencode(ct,o2);
mat3= h.knownptct(a2,a3,o2);
}
}
```

DIFFICULTIES FACED DURING INTEGRATION

- Construction of functions in same program since it required objects and interconnecting all.
- In this program, have used numerous variable names for denotation

SCREEN SHOTS:

```
E:\19IT021>javac Hills.java

E:\19IT021>java Hills
*******ENCRYPTION*******
Enter the order of key:
2
Enter the elements of key matrix (4):
1
2
3
4
Enter the plain text to encode :
tce
19      4
2       23
The encrypted matrix is:

        23      24
        13      0
The encrypted cipher text is:
xnya
*******DECRYPTION*******
Enter the order of key matrix:
2

Enter elements of key matrix:(4):
1
2
3
8
Enter the cipher text to decode:
aaws
0       22
0       18
Determinant of the matrix is : 2
Inverse of determinant: 1
Decrypted matrix:

        0       10
        0       4
The decrypted plain text is:
aak
```

```
*******KNOWN PLAIN TEXT - CIPHER TEXT*******
Enter the Plain text
friday
Enter the cipher text
pqcfku
Enter order of key matrix:
2
5       8       0
17      3       24
15      2       10
16      5       20




Determinant of the matrix is : 17
Inverse of determinant: 23
Key matrix:

        23      8
        19      19
E:\19IT021>
```

RESULT:

Thus the programs for

- Encryption
- Decryption
- Known Plain text – Cipher text attack

in Hill Cipher are implemented in Java and the results are verified.

Evaluation

Criteria	Ratings				Pts
Correctness	5.0 pts Excellent • Program runs and completes all required tasks • Handles special cases • Executes without errors	3.0 pts Good • Program is complete in all aspects and competes most tasks appropriately • Program fails to work for special cases	2.0 pts Satisfactory • Individual modules produce expected output • Program fails to handle errors due to integration	0.0 pts Unsatisfactory • Individual modules do not execute due to errors. • No integration of modules has been performed • Incorrect results for most or all independent modules	
Coding Standards	5.0 pts Excellent • Includes name, date, and assignment title. • Excellent use of white space. • Creatively organized work. • Excellent use of variables (no global variables,	3.0 pts Good • Includes name, date, and assignment title. • Good use of white space. • Organized work. • Good use of variables (no global variables,	2.0 pts Satisfactory • Completed between 70-80% of the requirements. • Delivered on time, and in correct format (disk, email, etc.) global variables, unambiguous	0.0 pts Unsatisfactory • Completed less than 70% of the requirements. • Not delivered on time or not in correct format (disk, email, etc.)	

Criteria	Ratings				Pts
	unambiguous naming).	unambiguous naming)	naming).		
Documentation	5.0 pts Excellent • Clearly and effectively documented including descriptions of all variables. • Specific purpose is noted for each function, control structure, input requirements, and output results.	3.0 pts Good • Clearly documented including descriptions of all variables. • Specific purpose is noted for each function and control structure.	2.0 pts Satisfactory • Basic documentation has been completed including descriptions of all variables. • Purpose is noted for each function.	0.0 pts Unsatisfactory • No documentation included.	
Runtime	5.0 pts Excellent • Executes without errors excellent user prompts, good use of symbols, spacing in output. • Thorough and organized testing has been completed and output from test cases is included.	3.0 pts Good • Executes without errors. • User prompts are understandable, minimum use of symbols or spacing in output. • Thorough testing has been completed	2.0 pts Satisfactory • Executes without errors. • User prompts contain little information, poor design . • Some testing has been completed.	0.0 pts Unsatisfactory • Does not execute due to errors. • User prompts are misleading or non-existent. • No testing has been completed.	

Criteria	Ratings				Pts
Team Work	5.0 pts Excellent • Equal Participation and contribution • Excellent support for each other • Able to explain the logic of other modules	3.0 pts Good • Contribution from few members • Provide moderate explanation of other Modules • Moderate Support for Team members	2.0 pts Satisfactory • Contribution from one or two members in a group • No clear idea on the work of others	0.0 pts Unsatisfactory • No cooperation among team members • No support for each other • No idea on the work of other team members	
Completed on time	5.0 pts Excellent Program is completed on time	3.0 pts Good Program is one day late	2.0 pts Satisfactory Program is three day late	0.0 pts Unsatisfactory Program is late by more than three days	

Ex.No.3	**IMPLEMENTATION OF RSA**

AIM:

To simulate the working of RSA in Virtual lab environment and to implement the same in Java/Python

THEORY:

One way function

A one-way function is a function that is easy to compute on every input, but hard to invert given the image of a random input. The existence of such one-way functions is still an open conjecture. Key generation is the production of two large prime numbers in RSA algorithm.

Modular exponentiation:

When 2 largely prime numbers p and q are multiplied (n = p*q), we get a very large prime number which cannot be decomposed easily. Even when the values of p and n or q and n are known it is difficult to calculate q and p respectively. P and q are used for computing private keys. Decomposition may lead to loss of authenticity. This (one-way function) makes the decryption tough.

Key generation:

1. Choose two different large random prime numbers say “p” and “q”.

2. Calculate n = p * q. Since “n” is the modulus for the public key and the private keys

3. Calculate the totient: Ø (n) = (p - 1) (q - 1)

4. Choose an integer “e” such that 1 < e < Ø (n) and “e” is co-prime to Ø (n) i.e., “e” and Ø

(n) share no factors other than 1.

5. Find out decryption key “d” such that e * d = 1 mod (p - 1) (q - 1).

RSA Encryption:

RSA encryption is often used in combination with other encryption schemes, or for digital signatures which can prove the authenticity and integrity of a message. A file will generally be encrypted with a symmetric-key algorithm, and then the symmetric key will be encrypted with RSA encryption. Under this process, only an entity that has access to the RSA private key will be able to decrypt the symmetric key. Encrypt the message “m” using encryption key e, c = m^e mod n.

RSA Decryption

The decryption process is very straightforward and includes analytics for calculation in a systematic approach. Decrypt the message “m” using decryption key d, m = c^d mod n.

ALGORITHM:

Key generation:

1. Select two large primes, p and q.
2. Calculate n=p*q
3. Number e must be greater than 1 and less than (p − 1) (q − 1).
4. There must be no common factor for e and (p − 1) (q − 1) except for 1. In other words, two numbers e and (p – 1) (q – 1) are co-prime.
5. The pair of numbers (e,n) form the RSA public key and is made public.

6. Though n is part of the public key, difficulty in factorizing a large prime number ensures that attacker cannot find in finite time the two primes (p & q) used to obtain n. This is strength of RSA.

7. Private Key d is calculated from p, q, and e. For given n and e, there is unique number d.

8. Number d is the inverse of e mod (p - 1) (q – 1). This means that d is the number less than (p - 1) (q - 1) such that when multiplied by e, it is equal to 1 modulo (p -1) (q - 1).

RSA Encryption

1. The sender represents the plaintext as a series of numbers less than n.

2. To encrypt the first plaintext P, which is a number modulo n. (C = Pe mod n)

3. In other words, the ciphertext C is equal to the plaintext P multiplied by itself e times and then reduced modulo n. This means that C is also a number less than n.

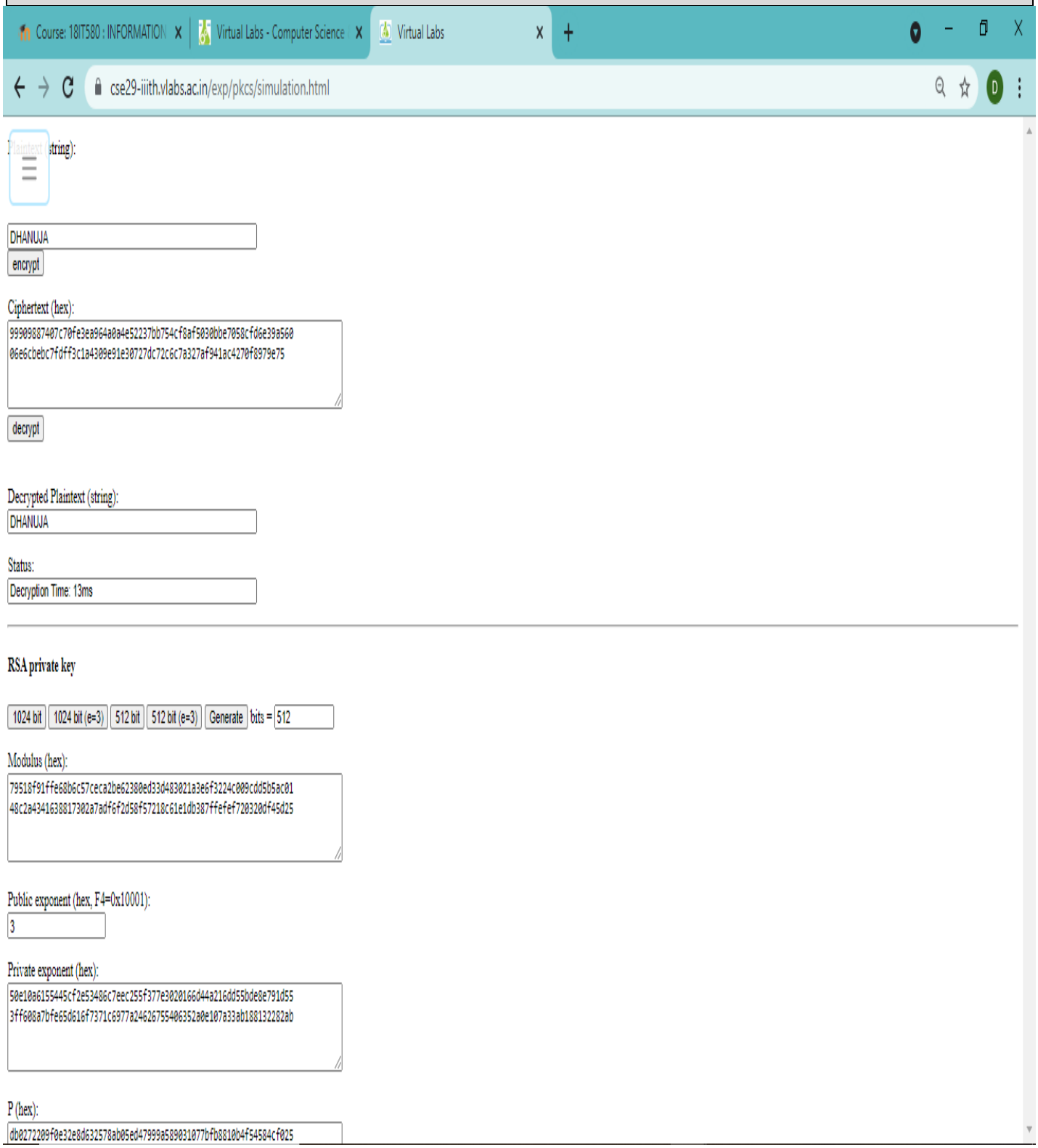
RSA Decryption

• Suppose that the receiver of public-key pair (e,n) has received a ciphertext C.

• Receiver raises C to the power of his private key d. The result modulo n will be the plaintext P.

Plaintext = Cd mod n

Screen Shots of simulation in Virtual labs



Coding

```
import java.io.*;
import java.util.*;
import java.math.*;
class rsaAlgo
{
Scanner in = new Scanner (System.in);
 int gcd, d;
 void key (int p, int q)
 {
 int n = p*q;
 int fi = (p-1)*(q-1);
 System.out.println("Enter e (co-prime): ");
 int e = in.nextInt();
 gcd_cal(e,fi);
 if(gcd==1)
 {
 System.out.println("Modular multiplicative inverse is: "+key1(e,fi));
 }
 else
 {
 System.out.println("E is not co prime, provide another value");
 key(p,q);
 }
 System.out.println("Public key (e,n) is: "+e+","+n);
 System.out.println("Public key (d,n) is: "+key1(e,fi)+","+n);
 }
 void gcd_cal(int e, int fi)
 {
```

```
for (int i = 1; i <= e && i <= fi; ++i)
{
if (e % i==0 && fi % i==0)
gcd = i;
}
}
static int key1(int a, int b)
{
a = a % b;
for (int x = 1; x < b; x++)
if ((a * x) % b == 1)
return x;
return 1;
}
public long exponentiation (long base, long exp, long N)
{
if (exp == 0)
return 1;
if (exp == 1)
return base % N;
long t = exponentiation (base, exp / 2, N);
t = (t * t) % N;
// if exponent is even value
if (exp % 2 == 0)
return t;
// if exponent is odd value
else
return ((base % N) * t) % N;
}
```

```
}
class rsa
{
 public static void main (String args[])
 {
 rsaAlgo r1 = new rsaAlgo();
 BigInteger p1, q1;
 int p,q, num,n1, e1, charPosition,i,value,count;
 long e,n,d;
 String m, ALPHABET = "abcdefghijklmnopqrstuvwxyz";
 Scanner in = new Scanner (System.in);
 do
 {
 System.out.println("\n1. Key Generation");
 System.out.println("2. Encryption");
System.out.println("3. Decryption");
 System.out.println("\n Enter your choice: ");
 int choice = in.nextInt();
 switch(choice)
 {
 case 1:
 System.out.println("Enter prime number p: ");
 p1 = in.nextBigInteger();
 System.out.println("Enter prime number q: ");
 q1 = in.nextBigInteger();
 p = p1.intValue();
 q = q1.intValue();
 r1.key(p,q);
 break;
```

```
case 2:
System.out.println("Enter public key e: ");
e = in.nextLong();
System.out.println("Enter public key n: ");
n = in.nextLong();
System.out.println("Enter plain text: ");
m = in.next();
m = m.toLowerCase();
System.out.println("\nAfter encryption: ");
for(i=0; i<m.length(); i++)
{
charPosition = ALPHABET.indexOf(m.charAt(i));
System.out.println(""+r1.exponentiation(charPosition,e,n));
}
break;
case 3:
System.out.println("Enter private key d: ");
d = in.nextLong();
System.out.println("Enter private key n: ");
n = in.nextLong();
System.out.println("How many words you want to decrypt? ");
count = in.nextInt();
System.out.println("Enter cipher text:");
for (i=0; i<count; i++)
{
value = in.nextInt();
System.out.println("value of letter "+i+" is: "+r1.exponentiation(value,d,n));
}
break;
```

```
default:
System.out.println("Enter a valid choice");
}
System.out.println("\nEnter 1 to continue.....");
num = in.nextInt();
}
while(num==1);
}
}
```

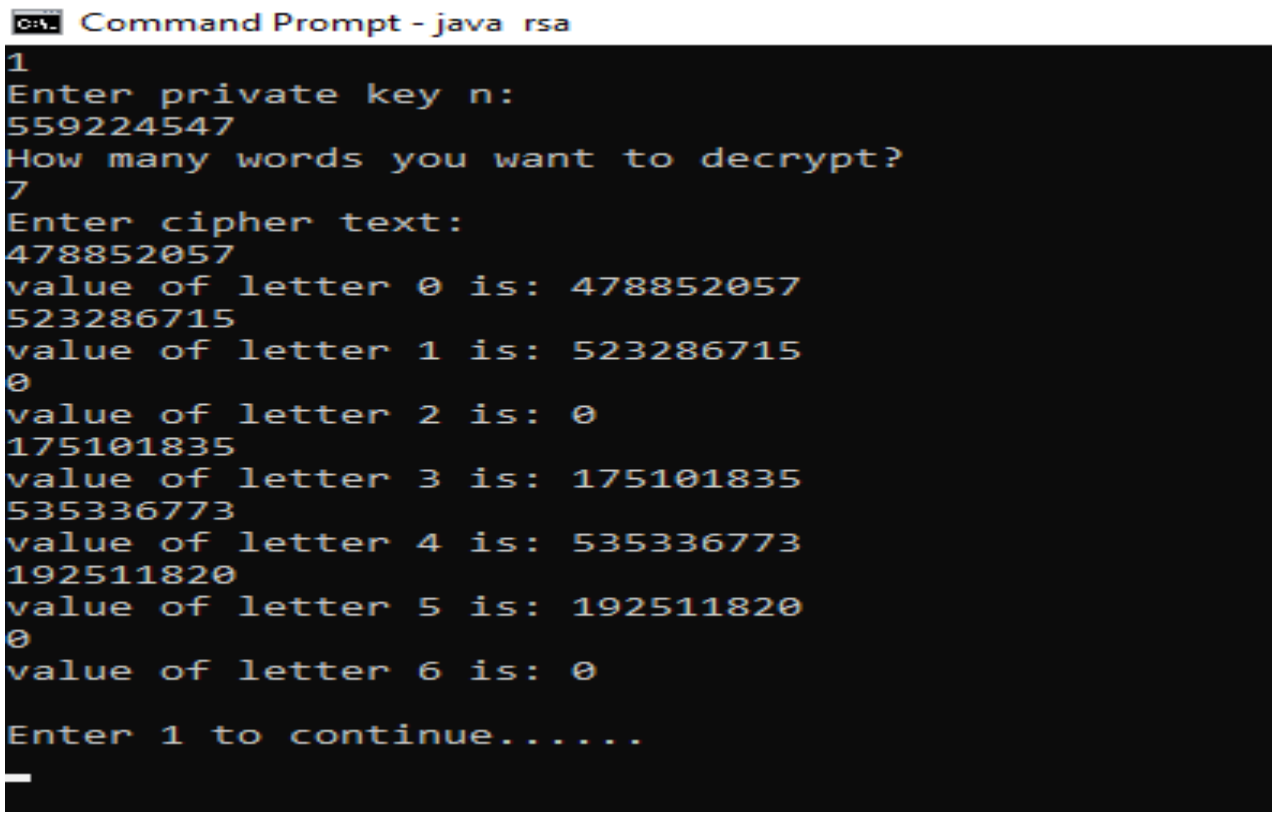
SCREEN SHOTS:

Command Prompt - java rsa

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\LENOVO>e:

E:\>cd 19it021

E:\19IT021>java rsa

1. Key Generation
2. Encryption
3. Decryption

 Enter your choice:
1
Enter prime number p:
3167572029241906662499918444262149202401482915547265781928016033797990360793371581435324201897676033
Enter prime number q:
252605411662216004885545986179504618452159789654050588376741883552141444084170818665539794498127869 1
Enter e (co-prime):
59
Modular multiplicative inverse is: 1
Public key (e,n) is: 59,559224547
Public key (d,n) is: 1,559224547

Enter 1 to continue.....
1
```

Command Prompt - java rsa

```
Enter 1 to continue......
1

1. Key Generation
2. Encryption
3. Decryption

 Enter your choice:
2
Enter public key e:
59
Enter public key n:
559224547
Enter plain text:
dhanuja

After encryption:
478852057
523286715
0
175101835
535336773
192511820
0

Enter 1 to continue......
1

1. Key Generation
2. Encryption
3. Decryption

 Enter your choice:
3
Enter private key d:
1
Enter private key n:
559224547
How many words you want to decrypt?
7
```



RESULT:

Hence, the experiment has been executed and the result is verified.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	15	
Completion of experiment on time	5	
Documentation	5	
Simulation in Vlabs	5	
Total	30	
Signature of the faculty with Date		

Ex.No.4	**IMPLEMENTATION OF DIFFIE HELLMAN KEY EXCHANGE**

AIM:

To simulate the working of Diffie Hellman in Virtual lab environment and to implement the same in Client/ Server model using Java

THEORY:

Steps Involved:

- The first trick here is that given x (with g and p known), it's very easy to find r. But given r (with g and p known) it's difficult to deduce x. This is also called the discrete logarithmic problem (one-way function for diffie hellman).
- All the three Bob, Alice and eve now know g and p. Now, Alice selects a random private number xa and calculates (g to the power xa) mod p = ra. This resultant ra is sent on the communication channel to Bob. Intercepting in between, eve also comes to know ra.
- Similarly, Bob selects his own random private number xb, calculates (g to the power xb) mod p = rb and sends this rb to Alice through the same communication channel. Obviously, eve also comes to know about rb.
- So, eve now has information about g, p, ra and rb. ▪ Now comes the heart of this algorithm. Alice calculates (rb to the power xa) mod p = Final key which is equivalent to (g to the power (xa * xb)) mod p .
- Similarly, Bob calculates (ra to the power xb) mod p = Final key which is again equivalent to (g to the power (xb * xa)) mod p.
- So, both Alice and Bob were able to calculate a common Final key without sharing each other's private random number and eve sitting in between will not be able to determine the Final key as the private numbers were never transferred.

ALGORITHM:

Process 1(Server)

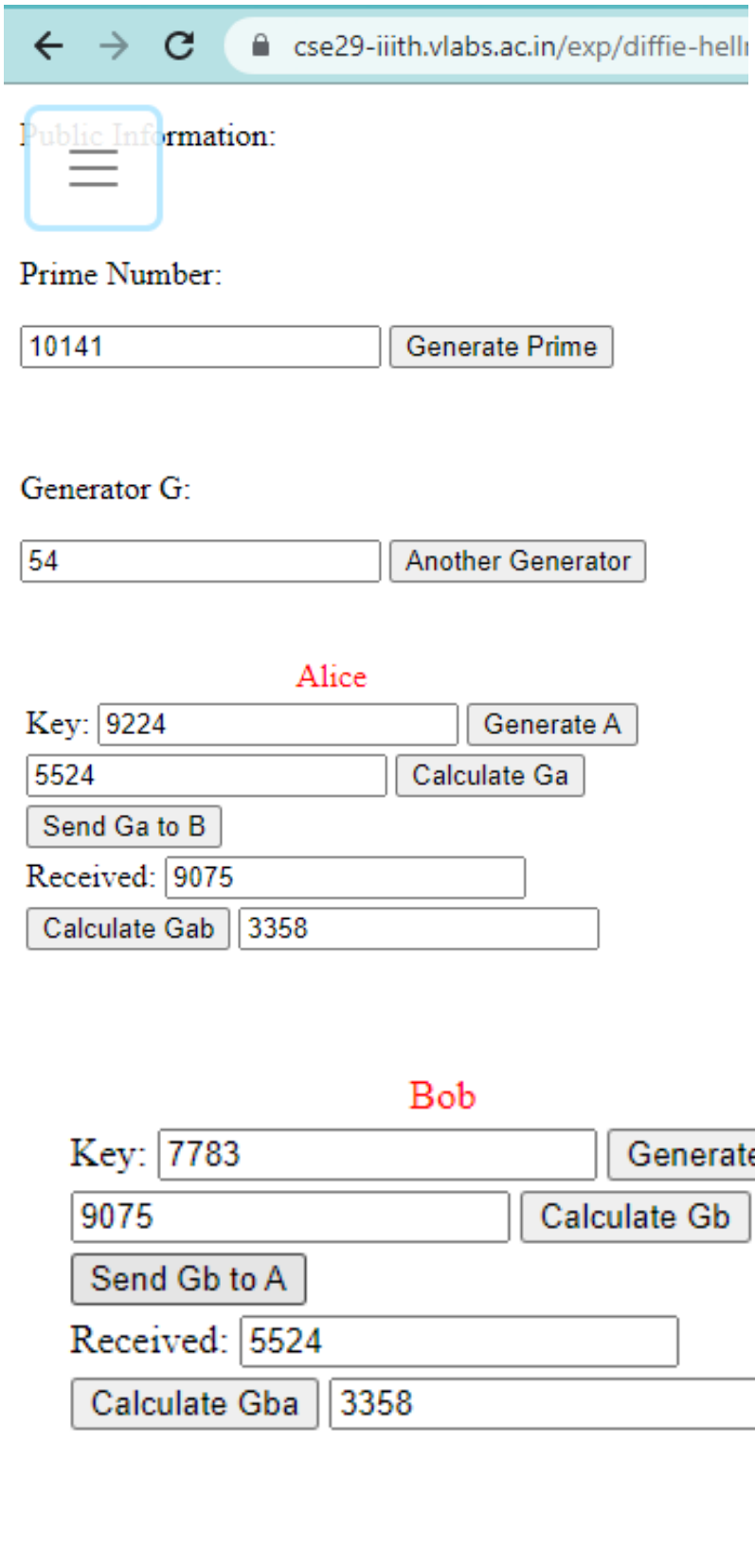
1. Start.
2. We initialize port and private number of B. We create an instance for the class Server Socket and receive inputs for IP address and ports and connect to the client.
3. We create an instance for the class DataStream input and receive inputs for P,G from client to compute the discrete log function.
4. Computing the discrete log function, the final key is sent to the client.
5. And, the symmetric key is calculated using the final key and private number of B and displayed.
6. End.

Process 2 (Client):

1. Start.
2. We initialize values like P, G, private number of A. We create instance of the Socket class to connect to the server using the IP address and port.
3. We calculate the discrete log function for the final key using the values and send it to the server.
4. Once the final key is received, the client finds the symmetric key using its private number.
5. End.

Screen Shots of simulation in Virtual labs

STEP 1: Choose a large prime number p and a generator g for that prime.
STEP 2: Both Alice and Bob generate their respective keys A and B. And (ga, gb) for their keys respectively.
STEP 3: Both Alice and bob send exchange their ga, gb.
STEP 4: Both calculate their public keys gab and gba respectively.
STEP 5: If both gab and gba are equal then Diffie-Hellman key exchange is verified


cse29-iiith.vlabs.ac.in/exp/diffie-hell
Public Information:
Prime Number:
10141
Generate Prime
Generator G:
54
Another Generator
Alice
Key: 9224
Generate A
5524
Calculate Ga
Send Ga to B
Received: 9075
Calculate Gab
3358
Bob
Key: 7783
Generate B
9075
Calculate Gb
Send Gb to A
Received: 5524
Calculate Gba
3358

Coding

Client_21.java

```
import java.net.*;
import java.io.*;
import java.math.BigInteger;
import java.util.*;
class Client_21 {

   public static BigInteger discreteKey(BigInteger y,BigInteger a,BigInteger p){

      BigInteger i;
      System.out.println("i\tx\ty");
      for(i=BigInteger.ONE;i.compareTo(p)<0;i=i.add(BigInteger.ONE)){
         BigInteger x=a.modPow(i, p);
         System.out.println(i+"\t"+x+"\t"+y);
         if(x.equals(y)){
            break;
         }
      }
      return i;
   }
   public static BigInteger ManInTheMiddle(BigInteger Xc,BigInteger a,BigInteger p){
      BigInteger Ya1=a.modPow(Xc,p);
      return Ya1;
   }

   public static void writeBigInteger(BigInteger integer, DataOutputStream out) throws
IOException {

      if (integer.signum() == -1) {
         throw new IllegalStateException("Negative BigInteger!");
      }
      byte[] buf = integer.toByteArray();
      if (buf.length > Short.MAX_VALUE) {
         throw new IllegalStateException("Too long: " + buf.length);
      }
      out.writeShort((short) buf.length);
```

```
    out.write(buf);
}
public static BigInteger readBigInteger(DataInputStream dis) throws IOException {
    short i = dis.readShort();
    if (i < 0) {
        throw new IOException("Invalid BigInteger length: " + i);
    }
    byte[] buf = new byte[i];
    dis.readFully(buf);
    return new BigInteger(1, buf);
}

public static void main(String args[]) throws Exception {
    Scanner sc=new Scanner(System.in);
    System.out.println("USER A\n");
    Socket s = new Socket("localhost", 3000);
    DataInputStream in = new DataInputStream(s.getInputStream());
    DataOutputStream out = new DataOutputStream(s.getOutputStream());
    System.out.print("KEY EXCHANGE\n");
        System.out.print("Enter the Prime Number : ");
        BigInteger p=sc.nextBigInteger();
        writeBigInteger(p,out);
        System.out.print("Enter the Generator : ");
        BigInteger a=sc.nextBigInteger();
        writeBigInteger(a,out);
        System.out.print("Enter Secret Key Xa : ");
        BigInteger Xa=sc.nextBigInteger();
        BigInteger Ya=a.modPow(Xa,p);
        writeBigInteger(Ya,out);
        BigInteger Yb=readBigInteger(in);
        System.out.println("\nThe value of Yb :"+Yb);
        BigInteger key=Yb.modPow(Xa,p);
        System.out.println("\nKey of User A : "+key);

    System.out.print("\nDISCRETE KEY PROBLEM\n");
        long start = System.currentTimeMillis();
        BigInteger x=discreteKey(Ya,a,p);
```

```
        long end = System.currentTimeMillis();
        System.out.println("The function takes " +(end - start) + "ms");
        System.out.println("\nPrivate key of User A (Xa): "+x);

      System.out.print("\nMAN IN THE MIDDLE\n");
        System.out.print("Enter Xc :");
        BigInteger Xc=sc.nextBigInteger();
        BigInteger Ya1=ManInTheMiddle(Xc,a,p);
        writeBigInteger(Ya1,out);
        BigInteger Yb1=readBigInteger(in);
        System.out.println("\nThe value of Yb' is :"+Yb1);
        BigInteger k=Yb1.modPow(Xa, p);
        System.out.println("\nThe key generated using Yb' is: "+k);
        System.out.println("");
      out.close();
      s.close();
   }
}
```

Server_21.java

```
import java.net.*;
import java.io.*;
import java.util.*;
import java.lang.Math;
import java.math.BigInteger;

class Server_21 {

   public static BigInteger discreteKey(BigInteger y,BigInteger a,BigInteger p){

      BigInteger i;
      System.out.println("i\tx\ty");
      for(i=BigInteger.ONE;i.compareTo(p)<0;i=i.add(BigInteger.ONE)){
        BigInteger x=a.modPow(i, p);
        System.out.println(i+"\t"+x+"\t"+y);
        if(x.equals(y)){
          break;
```

```
        }
      }
      return i;
   }

   public static BigInteger ManInTheMiddle(BigInteger Xd,BigInteger a,BigInteger p){
      BigInteger Yb1=a.modPow(Xd, p);
      return Yb1;
   }

   public static void writeBigInteger(BigInteger integer, DataOutputStream out) throws
IOException {

      if (integer.signum() == -1) {
         throw new IllegalStateException("Negative BigInteger!");
      }
      byte[] buf = integer.toByteArray();
      if (buf.length > Short.MAX_VALUE) {
         throw new IllegalStateException("Too long: " + buf.length);
      }
      out.writeShort((short) buf.length);
      out.write(buf);
   }
   public static BigInteger readBigInteger(DataInputStream dis) throws IOException {
      short i = dis.readShort();
      if (i < 0) {
         throw new IOException("Invalid BigInteger length: " + i);
      }
      byte[] buf = new byte[i];
      dis.readFully(buf);
      return new BigInteger(1, buf);
   }

   public static void main(String args[]) throws IOException{
   Scanner sc=new Scanner(System.in);
   System.out.println("USER B\n");
   ServerSocket ss = new ServerSocket(3000);
```

```
Socket s = ss.accept();

DataInputStream in = new DataInputStream(s.getInputStream());
DataOutputStream out= new DataOutputStream(s.getOutputStream());

System.out.println("KEY EXCHANGE");
   BigInteger p=readBigInteger(in);
   BigInteger a=readBigInteger(in);
   System.out.println("The value of Prime Number is :"+p);
   System.out.println("The value of Generator is :"+a);
   System.out.print("Enter Secret Key Xb : ");
   BigInteger Xb=sc.nextBigInteger();
   BigInteger Yb=a.modPow(Xb,p);
   writeBigInteger(Yb,out);
   BigInteger Ya=readBigInteger(in);
   System.out.println("\nThe value of Ya : "+Ya);
   BigInteger key=Ya.modPow(Xb, p);
   System.out.println("\nKey of User B : "+key);

System.out.println("\nDISCRETE KEY PROBLEM");
   long start = System.currentTimeMillis();
   BigInteger x=discreteKey(Yb,a,p);
   long end = System.currentTimeMillis();
   System.out.println("The function takes " +(end - start) + "ms");
   System.out.println("\nPrivate key of User B (Xb): "+x);

System.out.println("\nMAN IN THE MIDDLE");
   System.out.print("Enter Xd :");
   BigInteger Xd=sc.nextBigInteger();
   BigInteger Yb1=ManInTheMiddle(Xd,a,p);
   writeBigInteger(Yb1,out);
   BigInteger Ya1=readBigInteger(in);
   System.out.println("\nThe value of Ya' is :"+Ya1);
   BigInteger k=Ya1.modPow(Xb,p);
   System.out.println("\nThe key generated using Yb' is: "+k);

in.close();
```

```
    s.close();
    ss.close();
    }
}
```

SCREEN SHOTS:

Command Prompt

```
E:\19IT021>java Client_21
USER A

KEY EXCHANGE
Enter the Prime Number : 7
Enter the Generator : 3
Enter Secret Key Xa : 6

The value of Yb :2

Key of User A : 1

DISCRETE KEY PROBLEM
i       x       y
1       3       1
2       2       1
3       6       1
4       4       1
5       5       1
6       1       1
The function takes 33ms

Private key of User A (Xa): 6

MAN IN THE MIDDLE
Enter Xc :3

The value of Yb' is :1

The key generated using Yb' is: 1


E:\19IT021>
```

Command Prompt

```
E:\19IT021>java Server_21
USER B

KEY EXCHANGE
The value of Prime Number is :7
The value of Generator is :3
Enter Secret Key Xb : 2

The value of Ya : 1

Key of User B : 1

DISCRETE KEY PROBLEM
i       x       y
1       3       2
2       2       2
The function takes 66ms

Private key of User B (Xb): 2

MAN IN THE MIDDLE
Enter Xd :6

The value of Ya' is :6

The key generated using Yb' is: 1

E:\19IT021>
```

RESULT:

Thus, the simulation of Diffie Hellman in Virtual lab environment and the implementation of the same in Client/ Server model using Java is executed.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	15	
Completion of experiment on time	5	
Documentation	5	
Simulation in Vlabs	5	
Total	30	
Signature of the faculty with Date		

Ex.No.5	**Exploring Standard Cryptographic libraries for Secure Communication - OpenSSL**

AIM:

To gain experience in using OpenSSL for

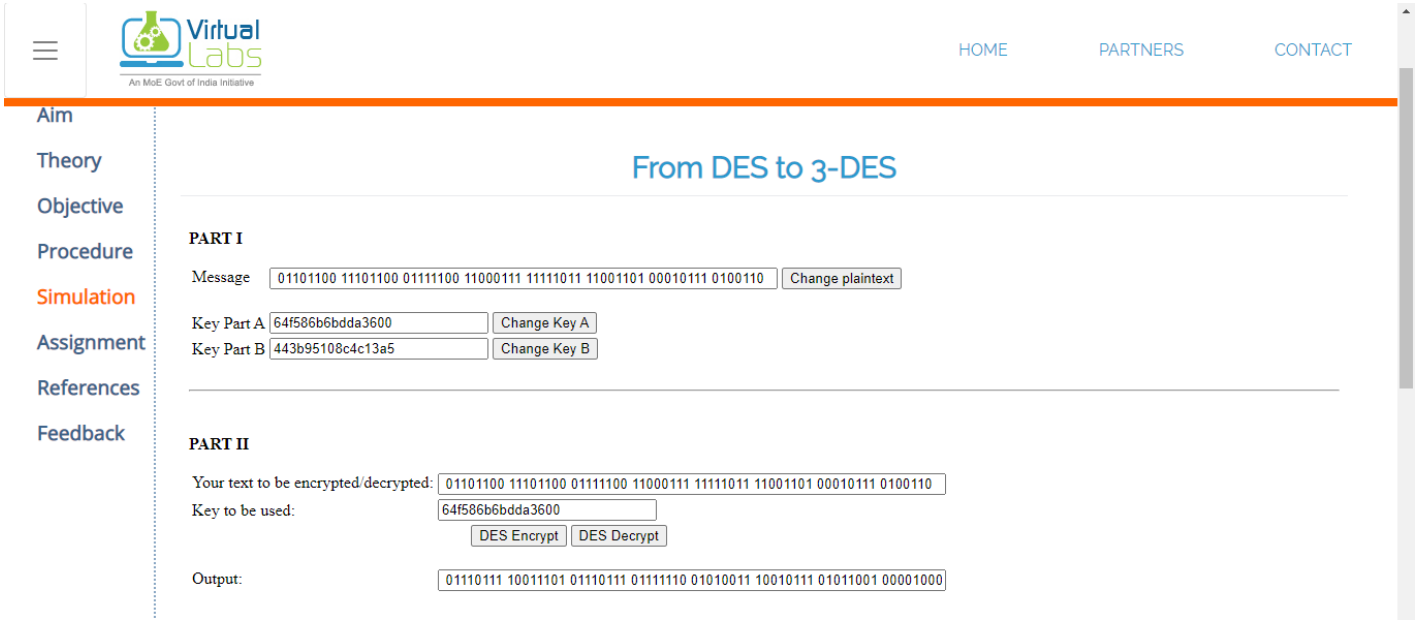
- Symmetric and Asymmetric encryption,
- Message digest and Hash,
- Digital signature – generation and verification

THEORY:

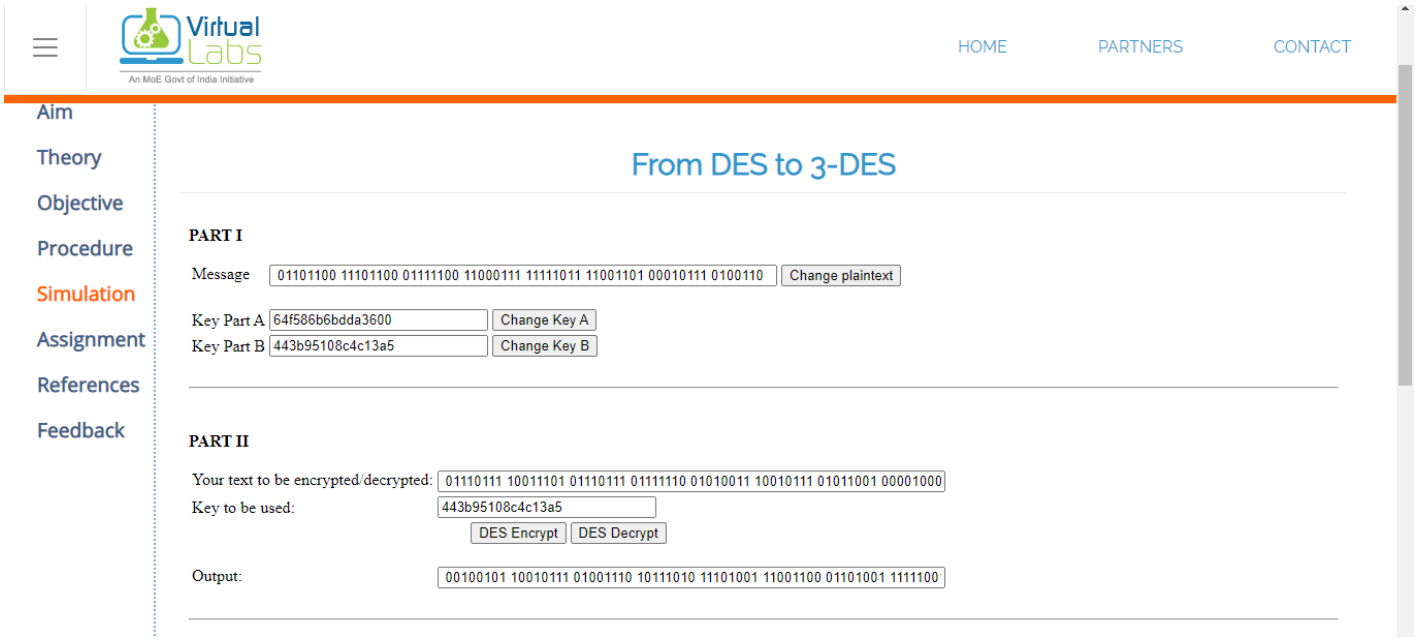
About OpenSSL

- OpenSSL is a software library for applications that secure communications over computer networks against eavesdropping or need to identify the party at the other end. It is widely used by Internet servers, including the majority of HTTPS websites.
- OpenSSL contains an open-source implementation of the SSL and TLS protocols. The core library, written in the C programming language, implements basic cryptographic functions and provides various utility functions. Wrappers allowing the use of the OpenSSL library in a variety of computer languages are available.
- The OpenSSL Software Foundation (OSF) represents the OpenSSL project in most legal capacities including contributor license agreements, managing donations, and so on. OpenSSL Software Services (OSS) also represents the OpenSSL project, for Support Contracts.

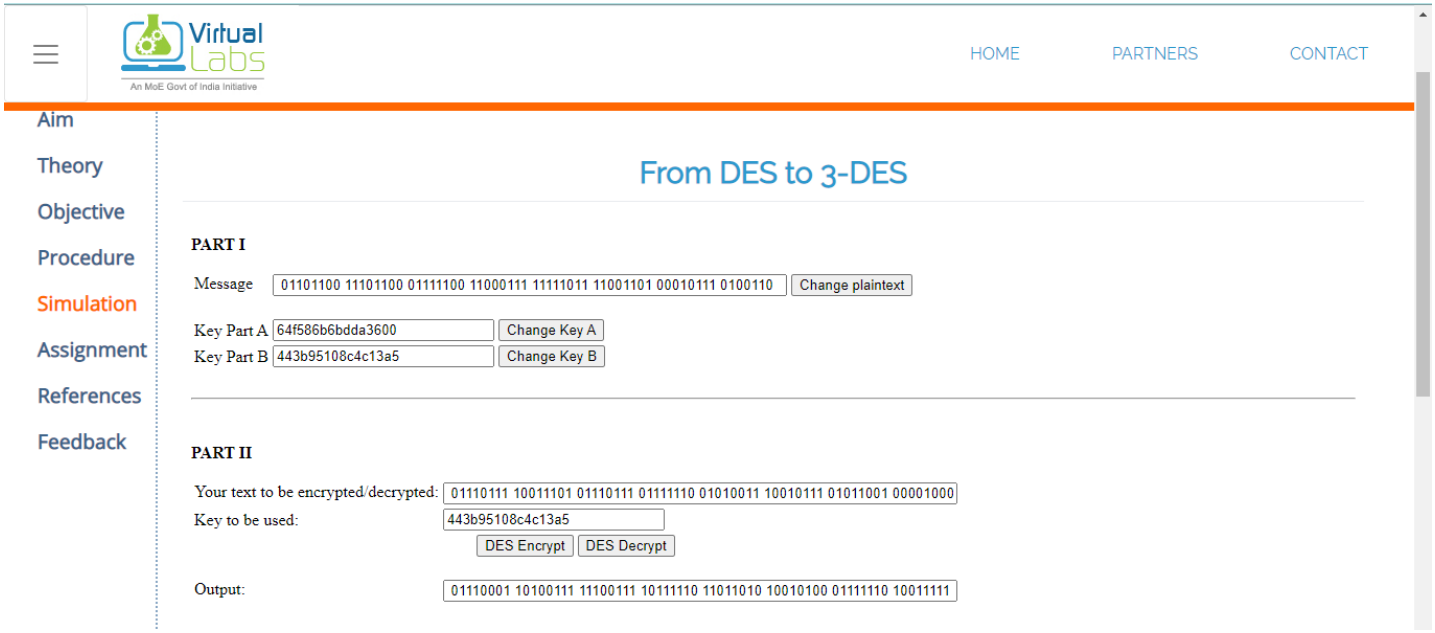
VIRTUAL LAB:



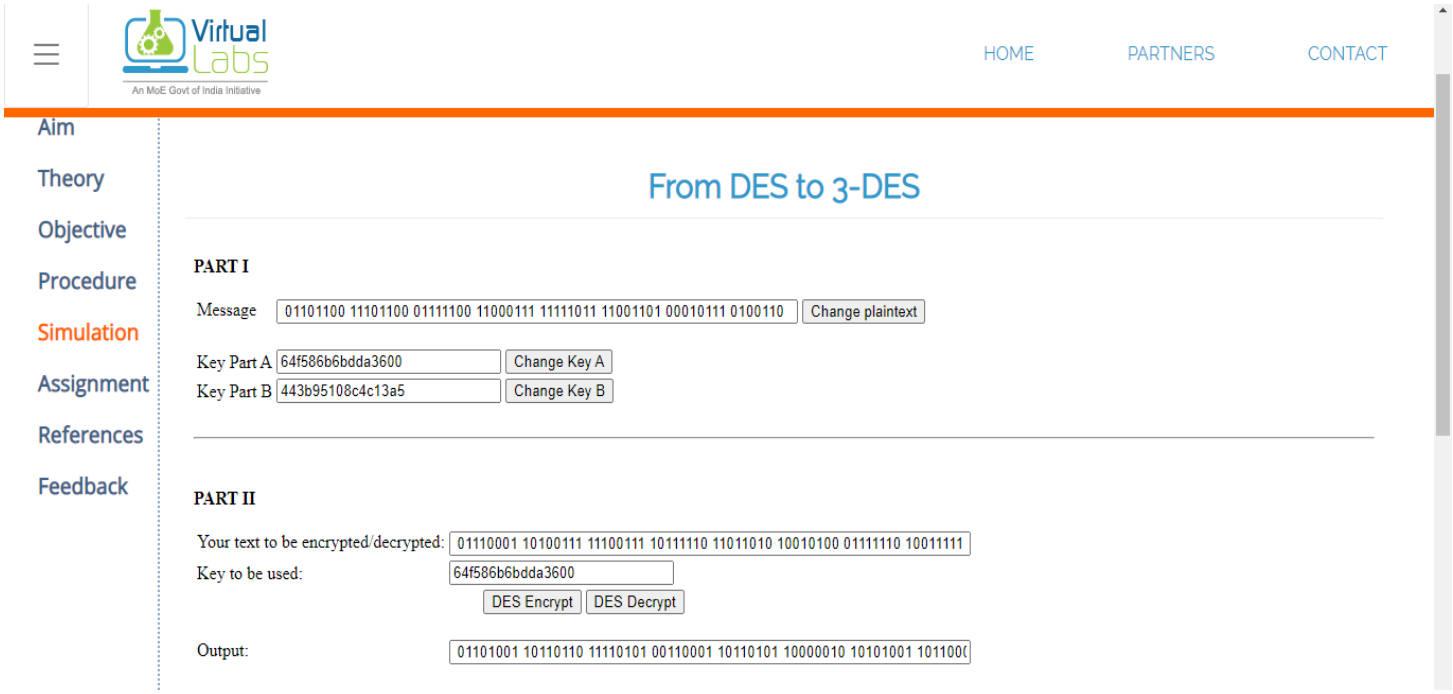
ENCRYPT USING KEY PART B



DECRYPT USING KEY PART B



ENCRYPT USING KEY PART A



VALIDATION OF OUTPUT:

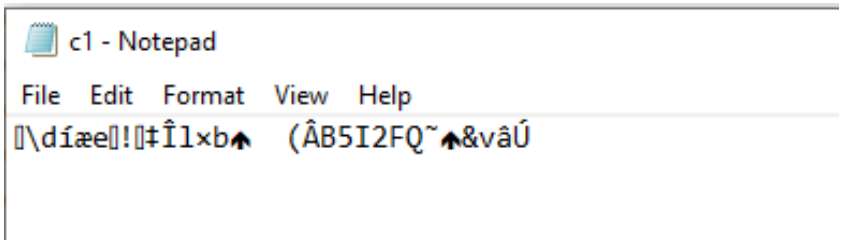
Worksheet (Answer the following ques and paste the relevant outputs)

1. Create a plaintext. txt file with your name and regno.

2. Encrypt using aes in all block cipher modes of operation.

openssl aes-128-cbc -e -in p.txt -out c1.bin -k "password" –nosalt

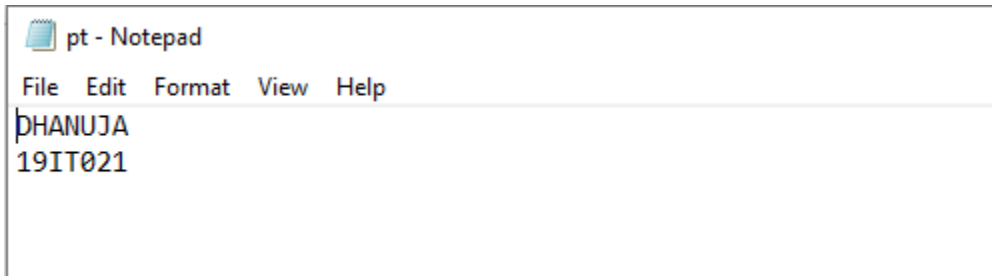
3. Display the contents of cipher.bin



4. Decrypt the contents of cipher.bin

openssl-1.1\x64\bin>openssl aes-128-cbc -d -in c1.bin -out pt.txt -k "password" –nosalt

5. Display the contents of pt.txt



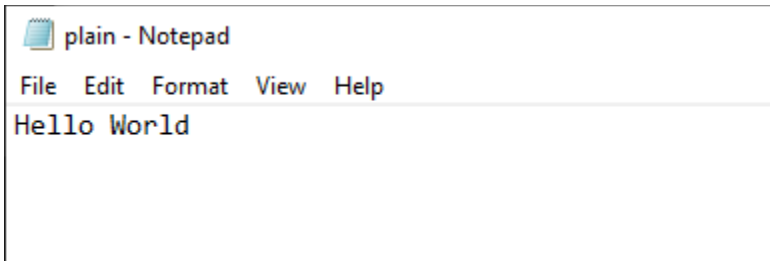
6. Examine Avalanche effect by changing one bit/charater in your plain text file

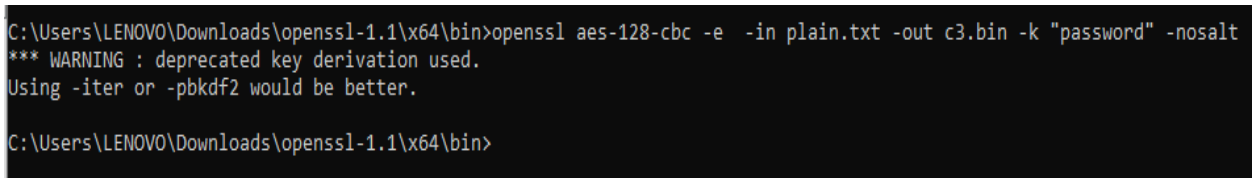
Avalanche effect:

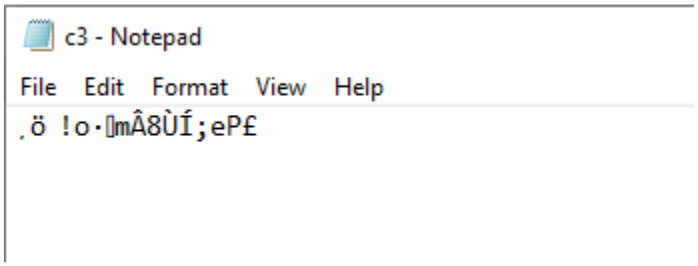
The Avalanche Effect refers to the fact that for a good cipher, changes in the plaintext affect the ciphertext. The algorithm produces a completely

different output for a minimally changed input. For example, the SHA-2 checksum algorithm or the AES encryption algorithm show a strong avalanche effect.

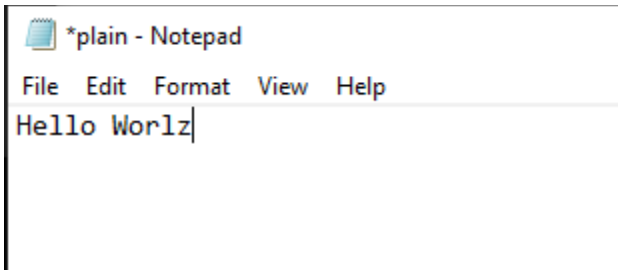
- In cryptography, the avalanche effect is a term associated with a specific behavior of mathematical functions used for encryption. Avalanche effect is considered as one of the desirable property of any encryption algorithm. A slight change in either the key or the plain-text should result in a significant change in the cipher-text. This property is termed as avalanche effect.
- In simple words, it quantifies the effect on the cipher-text with respect to the small change made in plaint text or the key.



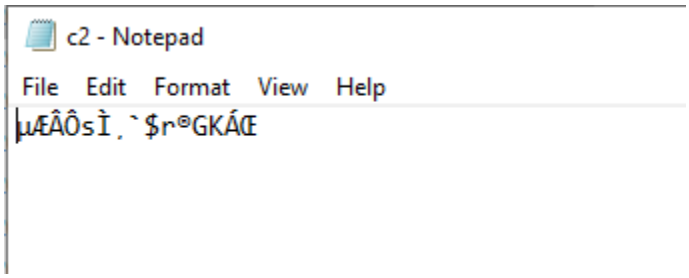




One letter is edited:



```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl aes-128-cbc -e  -in plain.txt -out c2.bin -k "password" -nosalt
*** WARNING : deprecated key derivation used.
Using -iter or -pbkdf2 would be better.

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```



7. Generate private and public key for RSA
   openssl genrsa –out pvtkey.pem

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl genrsa -out pvtkey.pem
Generating RSA private key, 2048 bit long modulus (2 primes)
.........................+++++
..............................+++++
e is 65537 (0x010001)

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

pvtkey - Notepad

File Edit Format View Help

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA3d+Of6Ffp0lzhv3FRMIB3rTX8ba7PPbDBTGYgMYp3fWaOlKB
8mV7+zlOVwbh8p7MgIXQ5/yze/9laDShxqb0+MgVVWTdLVsDflo+kBlHQrybKM0n
AlR78rKxgGBWYBS8vxYfr50tcNzZGnFpd2HwRR0WmSSbGfiQrS2yRwZpqPFNW3T7
6eiqMRmtO/h1GoLfeOHQj4G9Qoknl5mHnWnjlr7JgWazZ5oYakR8gDr/hzLsz13F
vCOUSs3O7QZvXYdRXFVISN/4Yp+dV5Ia4LSvs/9aD1zg9sHXVrJIzliV4jroGPwR
+WHkUQZkyI5b3nkgFZFYfCgL/zTzPoM2WkX8uQIDAQABAoIBADoSJhjPotqFBgDB
8NzTLmwcxxxTqGQHbfmxBmpF83VNsAUtzSc1WrLhR9jNS1wWJg5Jd+kQIeYFbYQ9
qk31Ks7mh3bnRGB3ns12T0Ol6hnUhYPDVj9s31ghXwXrsQ/KvbAB2V88c0VD+GrM
Xj9J2fos59pDU3kTrsI04KFoU0VNDe/LsQCoLvQn3qfo9sZ9/moagQ6Drs7xA7v7
r0+MMAzuybU25l+riUoryCyrqtLOZbyatOT9cGj1CcNyqC8E1JBLAqvXWl1WzhFe
7ykUhs0KSy84DkJiN30biTpY3tLUxX1UBpg6ZOAu4pQ2zpwhv//b54cIxRbtPTOj
Abca//UCgYEA/AHyg3ouSMu5hTKChLSV3AK4w0yDJPmJQrAvDK0Ksf3w+8GN3eF2
gIO9pt+3EIAVcIkYw+UYMygANQsOjNJ5ONS0WuApSZtAArL+/mYSM+mTzH0uRF5E
X87cbOVeAM4wtgryfKvY8S1R0CSDFr/+aoOqWb4ZeG0azv7v/g3z6ncCgYEA4WN1
LToG46bpZ6nUhTQcfWDdQ0XwQzTHjxpgefcj3/VDfSCQSs6g8eJ7KIpxw2GbqctQ
nGrpVKc05dOD3gpzThtfZPiwnsycAyvlJK/zH1cB7c+9+CajjmAL2dKH6jJ6GDvk
irARExlhlsnXjykSAwyCMHEhsWElFE+SL1Oq7k8CgYB3pUkJiRmucCLdjVSQQtif
gx9LgekFxf+YDOPZpdJdHU6riIKdZG0JqQKC1tYoXFdRfZWnc/5gU0594IkA7PCm
vb3/I0wW6UGjxm3wRG/B+9SzEn/D7+mQfzsB7IgL49kWpO3pqQEvyC9+nccrJu5Y
jbh0MSAQn+RJgJpayS8UEwKBgQCPtMo77fWajT/gor4oODO6FVq+nAc3Mp3StmSx
H0y2bTEcxPh7WgfwchFIpg0nIEtkB4hcUyZ2nS3zI8u+dTdTo8xNGuuyNI98zi4K
GE2Ru+407PIZGLnL+51SwxN51blXXki3zM72OIaqmANi+jLL0tvnexqaeHAl5R42
IsinrQKBgAw57Xv+RjnA1QssKzsoUEPdXDd/3IrhvxMnEXiWGQXZAz3Vq0pszBwc
arGq94uN3l4bYfelyzbX9lzbN+nNDqT7e2Iy4plMS6CSL7Tsv5lWJT/lgcdxSaQQ
+Wqr1ElFpEkA1TN5jGhDME6JghnosyoPf2jQDYS+kWa+uTF4ZkRx
-----END RSA PRIVATE KEY-----
```

openssl rsa -pubout -in pvtkey.pem -out pubkey.pem

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl rsa -pubout -in pvtkey.pem -out pubkey.pem
writing RSA key

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

pubkey - Notepad

File Edit Format View Help

```
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA3d+Of6Ffp0lzhv3FRMIB
3rTX8ba7PPbDBTGYgMYp3fWaOlKB8mV7+zlOVwbh8p7MgIXQ5/yze/9laDShxqb0
+MgVVWTdLVsDflo+kBlHQrybKM0nAlR78rKxgGBWYBS8vxYfr50tcNzZGnFpd2Hw
RR0WmSSbGfiQrS2yRwZpqPFNW3T76eiqMRmtO/h1GoLfeOHQj4G9Qokn15mHnWnj
lr7JgWazZ5oYakR8gDr/hzLsz13FvCOUSs3O7QZvXYdRXFVISN/4Yp+dV5Ia4LSv
s/9aD1zg9sHXVrJIzliV4jroGPwR+WHkUQZkyI5b3nkgFZFYfCgL/zTzPoM2WkX8
uQIDAQAB
-----END PUBLIC KEY-----
```

8. Display the private key in hexadecimal
   openssl rsa -text -in pvtkey.pem

Command Prompt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl rsa -text -in pvtkey.pem
RSA Private-Key: (2048 bit, 2 primes)
modulus:
    00:dd:df:8e:7f:a1:5f:a7:49:73:86:fd:c5:44:c2:
    01:de:b4:d7:f1:b6:bb:3c:f6:c3:05:31:98:80:c6:
    29:dd:f5:9a:3a:52:81:f2:65:7b:fb:39:4e:57:06:
    e1:f2:9e:cc:80:85:d0:e7:fc:b3:7b:ff:65:68:34:
    a1:c6:a6:f4:f8:c8:15:55:64:dd:2d:5b:03:7e:5a:
    3e:90:19:47:42:bc:9b:28:cd:27:02:54:7b:f2:b2:
    b1:80:60:56:60:14:bc:bf:16:1f:af:9d:2d:70:dc:
    d9:1a:71:69:77:61:f0:45:1d:16:99:24:9b:19:f8:
    90:ad:2d:b2:47:06:69:a8:f1:4d:5b:74:fb:e9:e8:
    aa:31:19:ad:3b:f8:75:1a:82:df:78:e1:d0:8f:81:
    bd:42:89:27:97:99:87:9d:69:e3:96:be:c9:81:66:
    b3:67:9a:18:6a:44:7c:80:3a:ff:87:32:ec:cf:5d:
    c5:bc:23:94:4a:cd:ce:ed:06:6f:5d:87:51:5c:55:
    48:48:df:f8:62:9f:9d:57:92:1a:e0:b4:af:b3:ff:
    5a:0f:5c:e0:f6:c1:d7:56:b2:48:ce:58:95:e2:3a:
    e8:18:fc:11:f9:61:e4:51:06:64:c8:8e:5b:de:79:
    20:15:91:58:7c:28:0b:ff:34:f3:3e:83:36:5a:45:
    fc:b9
publicExponent: 65537 (0x10001)
privateExponent:
    3a:12:26:18:cf:a2:da:85:06:00:c1:f0:dc:d3:2e:
    6c:1c:c7:1c:53:a8:64:07:6d:f9:b1:06:6a:45:f3:
    75:4d:b0:05:2d:cd:27:25:5a:b2:e1:47:d8:cd:4b:
    5c:16:26:0e:49:77:e9:10:21:e6:05:6d:84:3d:aa:
    4d:f5:2a:ce:e6:87:76:e7:44:60:77:9e:cd:76:4f:
    43:a5:ea:19:d4:85:83:c3:56:3f:6c:df:58:21:5f:
    05:eb:b1:0f:ca:bd:b0:01:d9:5f:3c:73:45:43:f8:
    6a:cc:5e:3f:49:d9:fa:2c:e7:da:43:53:79:13:ae:
    c2:34:e0:a1:68:53:45:4d:0d:ef:cb:b1:00:a8:2e:
    f4:27:de:a7:e8:f6:c6:7d:fe:6a:1a:81:0e:83:ae:
    ce:f1:03:bb:fb:af:4f:8c:30:0c:ee:c9:b5:36:e6:
    5f:ab:89:4a:2b:c8:2c:ab:aa:d2:ce:65:bc:9a:b4:
    e4:fd:70:68:f5:09:c3:72:a8:2f:04:d4:90:4b:02:
    ab:d7:5a:5d:56:ce:11:5e:ef:29:14:86:cd:0a:4b:
    2f:38:0e:42:62:37:7d:1b:89:3a:58:de:d2:d4:c5:
    7d:54:06:98:3a:64:e0:2e:e2:94:36:ce:9c:21:bf:
    ff:db:e7:87:08:c5:16:ed:3d:33:a3:01:b7:1a:ff:
    f5
prime1:
    00:fc:01:f2:83:7a:2e:48:cb:b9:85:32:82:84:b4:
```

Command Prompt

```
    0a:b1:fd:f0:fb:c1:8d:dd:e1:76:80:83:bd:a6:df:
    b7:10:80:15:70:89:18:c3:e5:18:33:28:00:35:0b:
    0e:8c:d2:79:38:d4:b4:5a:e0:29:49:9b:40:02:b2:
    fe:fe:66:12:33:e9:93:cc:7d:2e:44:5e:44:5f:ce:
    dc:6c:e5:5e:00:ce:30:b6:0a:f2:7c:ab:d8:f1:2d:
    51:d0:24:83:16:bf:fe:6a:83:aa:59:be:19:78:6d:
    1a:ce:fe:ef:fe:0d:f3:ea:77
prime2:
    00:e1:63:65:2d:3a:06:e3:a6:e9:67:a9:d4:85:34:
    1c:7d:60:dd:43:45:f0:43:34:c7:8f:1a:60:79:f7:
    23:df:f5:43:7d:20:90:4a:ce:a0:f1:e2:7b:28:8a:
    71:c3:61:a8:a9:cb:50:9c:6a:e9:54:a7:34:e5:d3:
    83:de:0a:73:4e:1b:5f:64:f8:b0:9e:cc:9c:03:2b:
    e5:24:af:f3:1f:57:01:ed:cf:bd:f8:26:a3:8e:60:
    0b:d9:d2:87:ea:32:7a:18:3b:e4:8a:b0:11:13:19:
    61:96:c9:d7:8f:29:12:03:0c:82:30:71:21:b1:61:
    25:14:4f:92:2f:53:aa:ee:4f
exponent1:
    77:a5:49:09:89:19:ae:70:22:dd:8d:54:90:42:d8:
    9f:83:1f:4b:81:e9:05:c5:ff:98:0c:e3:d9:a5:d2:
    5d:1d:4e:ab:88:82:9d:64:6d:09:a9:02:82:d6:d6:
    28:5c:57:51:7d:95:a7:73:fe:60:53:4e:7d:e0:89:
    00:ec:f0:a6:bd:bd:ff:23:4c:16:e9:41:a3:c6:6d:
    f0:44:6f:c1:fb:d4:b3:12:7f:c3:ef:e9:90:7f:3b:
    01:ec:88:0b:e3:d9:16:a4:ed:e9:a9:01:2f:c8:2f:
    7e:9d:c7:2b:26:ee:58:8d:b8:74:31:20:10:9f:e4:
    49:80:9a:5a:c9:2f:14:13
exponent2:
    00:8f:b4:ca:3b:ed:f5:9a:8d:3f:e0:a2:be:28:38:
    33:ba:15:5a:be:9c:07:37:32:9d:d2:b6:64:b1:1f:
    4c:b6:6d:31:1c:c4:f8:7b:5a:07:f0:72:11:48:a6:
    0d:27:20:4b:64:07:88:5c:53:26:76:9d:2d:f3:23:
    cb:be:75:37:53:a3:cc:4d:1a:eb:b2:34:8f:7c:ce:
    2e:0a:18:4d:91:bb:ee:34:ec:f2:19:18:b9:cb:fb:
    9d:52:c3:13:79:d5:b9:57:5e:48:b7:cc:ce:f6:38:
    86:aa:98:03:62:fa:32:cb:d2:db:e7:7b:1a:9a:78:
    70:25:e5:1e:36:22:c8:a7:ad
coefficient:
    0c:39:ed:7b:fe:46:39:c0:d5:0b:2c:2b:3b:28:50:
    43:dd:5c:37:7f:dc:8a:e1:bf:13:27:11:78:96:19:
    05:d9:03:3d:d5:ab:4a:6c:cc:1c:1c:6a:b1:aa:f7:
    8b:8d:de:5e:1b:61:f7:a5:cb:36:d7:f6:5c:db:37:
    e9:cd:0e:a4:fb:7b:62:32:e2:99:4c:4b:a0:92:2f:
    b4:ec:bf:99:56:25:3f:e5:81:c7:71:49:a4:10:f9:
```

Command Prompt

```
    4e:89:82:19:e8:b3:2a:0f:7f:68:d0:0d:84:be:91:
    66:be:b9:31:78:66:44:71
writing RSA key
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA3d+Of6Ffp0lzhv3FRMIB3rTX8ba7PPbDBTGYgMYp3fWaOlKB
8mV7+zlOVwbh8p7MgIXQ5/yze/9laDShxqb0+MgVVWTdLVsDflo+kBlHQrybKM0n
AlR78rKxgGBWYBS8vxYfr50tcNzZGnFpd2HwRR0WmSSbGfiQrS2yRwZpqPFNW3T7
6eiqMRmtO/h1GoLfeOHQj4G9Qoknl5mHnWnjlr7JgWazZ5oYakR8gDr/hzLsz13F
vCOUSs3O7QZvXYdRXFVISN/4Yp+dV5Ia4LSvs/9aD1zg9sHXVrJIzliV4jroGPwR
+WHkUQZkyI5b3nkgFZFYfCgL/zTzPoM2WkX8uQIDAQABAoIBADoSJhjPotqFBgDB
8NzTLmwcxxxTqGQHbfmxBmpF83VNsAUtzSclWrLhR9jNS1wWJg5Jd+kQIeYFbYQ9
qk31Ks7mh3bnRGB3ns12T0Ol6hnUhYPDVj9s31ghXwXrsQ/KvbAB2V88c0VD+GrM
Xj9J2fos59pDU3kTrsI04KFoU0VNDe/LsQCoLvQn3qfo9sZ9/moagQ6Drs7xA7v7
r0+MMAzuybU25l+riUoryCyrqtLOZbyatOT9cGj1CcNyqC8E1JBLAqvXWl1WzhFe
7ykUhs0KSy84DkJiN30biTpY3tLUxX1UBpg6ZOAu4pQ2zpwhv//b54cIxRbtPTOj
Abca//UCgYEA/AHyg3ouSMu5hTKChLSV3AK4w0yDJPmJQrAvDK0Ksf3w+8GN3eF2
gIO9pt+3EIAVcIkYw+UYMygANQsOjNJ5ONS0WuApSZtAArL+/mYSM+mTzH0uRF5E
X87cbOVeAM4wtgryfKvY8S1R0CSDFr/+aoOqWb4ZeG0azv7v/g3z6ncCgYEA4WNl
LToG46bpZ6nUhTQcfWDdQ0XwQzTHjxpgefcj3/VDfSCQSs6g8eJ7KIpxw2GoqctQ
nGrpVKc05dOD3gpzThtfZPiwnsycAyvlJK/zH1cB7c+9+CajjmAL2dKH6jJ6GDvk
irARExlhlsnXjykSAwyCMHEhsWElFE+SL1Oq7k8CgYB3pUkJiRmucCLdjVSQQtif
gx9LgekFxf+YDOPZpdJdHU6riIKdZG0JqQKC1tYoXFdRfZWnc/5gU0594IkA7PCm
vb3/I0wW6UGjxm3wRG/B+9SzEn/D7+mQfzsB7IgL49kWpO3pqQEvyC9+nccrJu5Y
jbh0MSAQn+RJgJpayS8UEwKBgQCPtMo77fWajT/gor4oODO6FVq+nAc3Mp3StmSx
H0y2bTEcxPh7WgfwchFIpg0nIEtkB4hcUyZ2nS3zI8u+dTdTo8xNGuuyNI98zi4K
GE2Ru+407PIZGLnL+51SwxN51blXXki3zM72OIaqmANi+jLL0tvnexqaeHAl5R42
IsinrQKBgAw57Xv+RjnA1QssKzsoUEPdXDd/3IrhvxMnEXiWGQXZAz3Vq0pszBwc
arGq94uN3l4bYfelyzbX9lzbN+nNDqT7e2Iy4plMS6CSL7Tsv5lWJT/lgcdxSaQQ
+Wqr1ElFpEkA1TN5jGhDME6JghnosyoPf2jQDYS+kWa+uTF4ZkRx
-----END RSA PRIVATE KEY-----

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

9. Perform encryption using RSA public key

openssl rsautl -encrypt -in plain.txt -pubin -inkey pubkey.pem -out c4.bin

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl rsautl -encrypt -in plain.txt -pubin -inkey pubkey.pem -out c4.bin

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

c4 - Notepad

File Edit Format View Help

10. Display the contents of encrypted file

c4 - Notepad

File Edit Format View Help

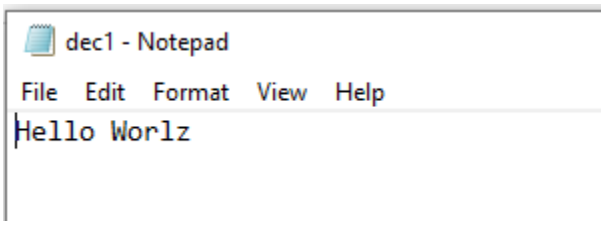
11. Decrypt the result using RSA private key

openssl rsautl -decrypt -in c4.bin -inkey pvtkey.pem -out dec1.txt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl rsautl -decrypt -in c4.bin  -inkey pvtkey.pem -out dec1.txt

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

12. Display the contents of decrypted file

openssl rsautl -decrypt -in c4.bin -inkey pvtkey.pem -out dec1.txt



13. Generate the hash of a file using MD5

openssl md5 plain.txt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl md5 plain.txt
MD5(plain.txt)= 4316cb922892a7127a7561dbe24092fc

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

14. Generate the hash of a file using SHA256

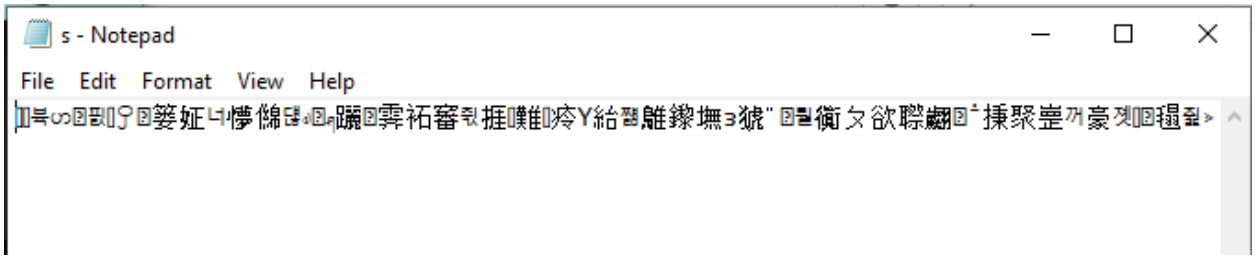
Openssl SHA256 plain.txt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>Openssl SHA256 plain.txt
SHA256(plain.txt)= 66d6c76cd10d21d6986552b42342317b95c37b19406023111c562e7ede2383a2

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

15. Generate Signature using SHA and RSA

openssl dgst -sha1 -sign pvtkey.pem -out s.bin plain.txt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl dgst -sha1 -sign pvtkey.pem -out s.bin plain.txt

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```



16. *Verify the signature*

openssl dgst -sha1 -verify pubkey.pem -signature s.bin plain.txt

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl dgst -sha1 -verify pubkey.pem -signature s.bin plain.txt
Verified OK

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

```
C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>openssl dgst -sha1 -verify pubkey.pem -signature s.bin p.txt
Verification Failure

C:\Users\LENOVO\Downloads\openssl-1.1\x64\bin>
```

RESULT:

Thus the libraries for

- Symmetric and Asymmetric encryption,
- Message digest and Hash,
- Digital signature – generation and verification

are utilized for securing the communication

Evaluation

Parameter	Max Marks	Marks Obtained
Originality of the work	30	
Completion of experiment on time	10	
Documentation	10	
Total	50	
Signature of the faculty with Date		

Ex.No.6	**Perform password extraction, cracking and recovery from target system**

AIM:

To use open source software tools for extraction, cracking and recovery from target system.
To implement Dictionary attack in Java/Python

THEORY:

Tool Selected:

John The Ripper password cracker

About the Tool:

John the Ripper is a free password cracking software tool. It is among the most frequently used password testing and breaking programs as it combines a number of password crackers into one package, auto detects password hash types, and includes a customizable cracker. It can be run against various encrypted password formats including several crypt password hash types.

Software Requirements:

Software runs On Windows XP and higher. Operates on networks with Windows NT, 2000, XP, Server 2003 R1/R2, Server 2008 R1/R2, on 32- and 64-bit environments, as well as most BSD and Linux variants with an SSH daemon.

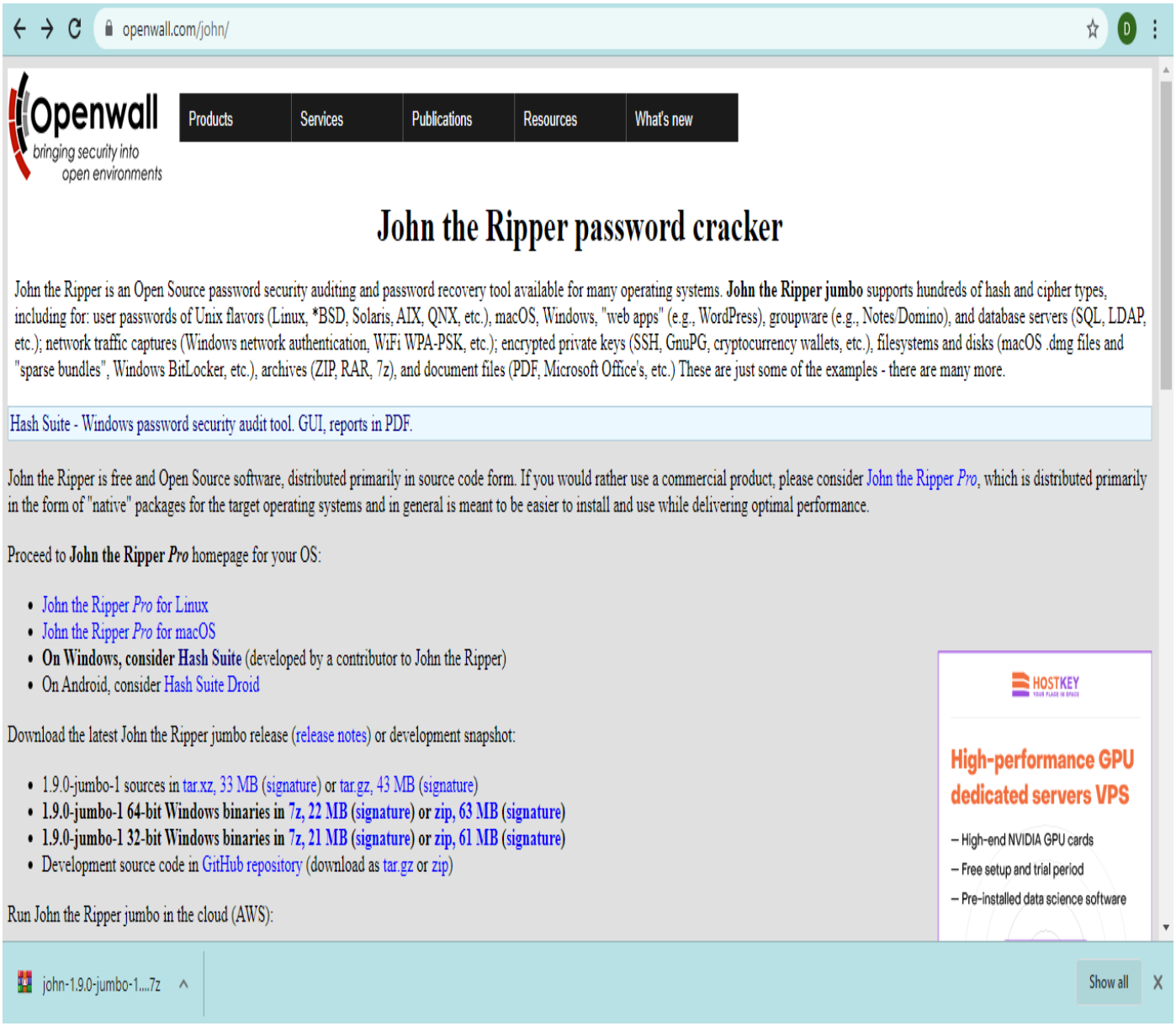
Dictionary attack:

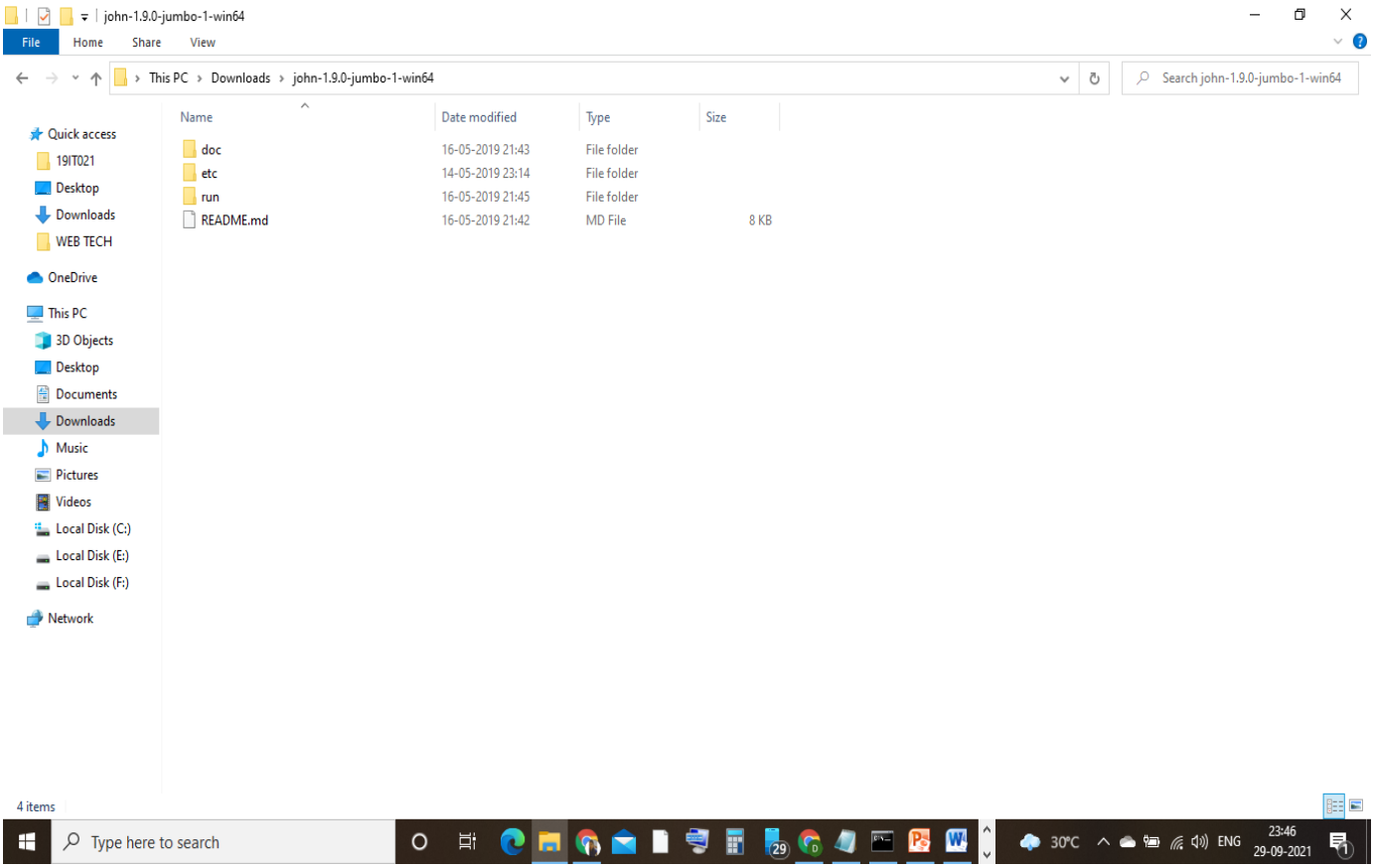
A dictionary attack is an attack that tries to guess at the key of a ciphertext by attempting many different common passwords and possible passwords that are likely to be used by humans

Demonstrations (With Screen shots)

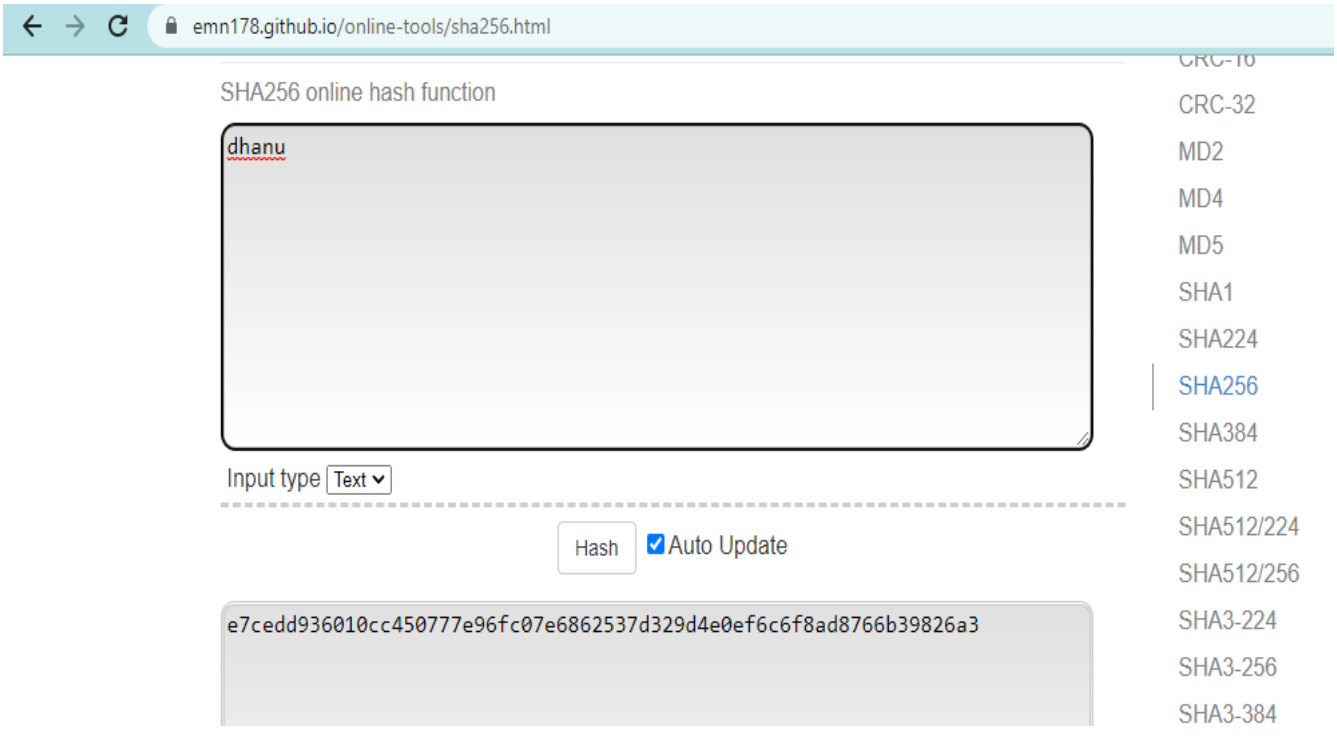
Installation Procedure

1. Go to https://www.openwall.com/john/ to download john the ripper
2. Download the executable file and install the same.
3. Proceed with free trial





Password cracking (Dictionary attack)



19IT021 - Notepad

File Edit Format View Help

```
e7cedd936010cc450777e96fc07e6862537d329d4e0ef6c6f8ad8766b39826a3
```

Command Prompt

```
Microsoft Windows [Version 10.0.18363.1556]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\LENOVO>cd C:\Users\LENOVO\Downloads\john-1.9.0-jumbo-1-win64\run

C:\Users\LENOVO\Downloads\john-1.9.0-jumbo-1-win64\run>john
John the Ripper 1.9.0-jumbo-1 OMP [cygwin 64-bit x86_64 SSE4.1 AC]
Copyright (c) 1996-2019 by Solar Designer and others
Homepage: http://www.openwall.com/john/

Usage: john [OPTIONS] [PASSWORD-FILES]
--single[=SECTION[,..]]    "single crack" mode, using default or named rules
--single=:rule[,..]        same, using "immediate" rule(s)
--wordlist[=FILE] --stdin  wordlist mode, read words from FILE or stdin
                  --pipe   like --stdin, but bulk reads, and allows rules
--loopback[=FILE]          like --wordlist, but extract words from a .pot file
--dupe-suppression         suppress all dupes in wordlist (and force preload)
--prince[=FILE]            PRINCE mode, read words from FILE
--encoding=NAME            input encoding (eg. UTF-8, ISO-8859-1). See also
                           doc/ENCODINGS and --list=hidden-options.
--rules[=SECTION[,..]]     enable word mangling rules (for wordlist or PRINCE
                           modes), using default or named rules
--rules=:rule[;..]]        same, using "immediate" rule(s)
--rules-stack=SECTION[,..] stacked rules, applied after regular rules or to
                           modes that otherwise don't support rules
--rules-stack=:rule[;..]   same, using "immediate" rule(s)
--incremental[=MODE]       "incremental" mode [using section MODE]
--mask[=MASK]              mask mode using MASK (or default from john.conf)
--markov[=OPTIONS]         "Markov" mode (see doc/MARKOV)
--external=MODE            external mode or word filter
--subsets[=CHARSET]        "subsets" mode (see doc/SUBSETS)
--stdout[=LENGTH]          just output candidate passwords [cut at LENGTH]
--restore[=NAME]           restore an interrupted session [called NAME]
--session=NAME             give a new session the NAME
--status[=NAME]            print status of a session [called NAME]
--make-charset=FILE        make a charset file. It will be overwritten
--show[=left]              show cracked passwords [if =left, then uncracked]
--test[=TIME]              run tests and benchmarks for TIME seconds each
--users=[-]LOGIN|UID[,..]  [do not] load this (these) user(s) only
--groups=[-]GID[,..]       load users [not] of this (these) group(s) only
--shells=[-]SHELL[,..]     load users with[out] this (these) shell(s) only
--salts=[-]COUNT[:MAX]     load salts with[out] COUNT [to MAX] hashes
--costs=[-]C[:M][,...]     load salts with[out] cost value Cn [to Mn]. For
                           tunable cost parameters, see doc/OPTIONS
```

Command Prompt

```
C:\Users\LENOVO\Downloads\john-1.9.0-jumbo-1-win64\run>

C:\Users\LENOVO\Downloads\john-1.9.0-jumbo-1-win64\run>john --list=formats
descrypt, bsdicrypt, md5crypt, md5crypt-long, bcrypt, scrypt, LM, AFS,
tripcode, AndroidBackup, adxcrypt, agilekeychain, aix-ssha1, aix-ssha256,
aix-ssha512, andOTP, ansible, argon2, as400-des, as400-ssha1, asa-md5,
AxCrypt, AzureAD, BestCrypt, bfegg, Bitcoin, BitLocker, bitshares, Bitwarden,
BKS, Blackberry-ES10, WoWSRP, Blockchain, chap, Clipperz, cloudkeychain,
dynamic_n, cq, CRC32, sha1crypt, sha256crypt, sha512crypt, Citrix_NS10,
dahua, dashlane, diskcryptor, Django, django-scrypt, dmd5, dmg, dominosec,
dominosec8, DPAPImk, dragonfly3-32, dragonfly3-64, dragonfly4-32,
dragonfly4-64, Drupal7, eCryptfs, eigrp, electrum, EncFS, enpass, EPI,
EPiServer, ethereum, fde, Fortigate256, Fortigate, FormSpring, FVDE, geli,
gost, gpg, HAVAL-128-4, HAVAL-256-3, hdaa, hMailServer, hsrp, IKE, ipb2,
itunes-backup, iwork, KeePass, keychain, keyring, keystore, known_hosts,
krb4, krb5, krb5asrep, krb5pa-sha1, krb5tgs, krb5-17, krb5-18, krb5-3,
kwallet, lp, lpcli, leet, lotus5, lotus85, LUKS, MD2, mdc2, MediaWiki,
monero, money, MongoDB, scram, Mozilla, mscash, mscash2, MSCHAPv2,
mschapv2-naive, krb5pa-md5, mssql, mssql05, mssql12, multibit, mysqlna,
mysql-sha1, mysql, net-ah, nethalflm, netlm, netlmv2, net-md5, netntlmv2,
netntlm, netntlm-naive, net-sha1, nk, notes, md5ns, nsec3, NT, o10glogon,
o3logon, o5logon, ODF, Office, oldoffice, OpenBSD-SoftRAID, openssl-enc,
oracle, oracle11, Oracle12C, osc, ospf, Padlock, Palshop, Panama,
PBKDF2-HMAC-MD4, PBKDF2-HMAC-MD5, PBKDF2-HMAC-SHA1, PBKDF2-HMAC-SHA256,
PBKDF2-HMAC-SHA512, PDF, PEM, pfx, pgpdisk, pgpsda, pgpwde, phpass, PHPS,
PHPS2, pix-md5, PKZIP, po, postgres, PST, PuTTY, pwsafe, qnx, RACF,
RACF-KDFAES, radius, RAdmin, RAKP, rar, RAR5, Raw-SHA512, Raw-Blake2,
Raw-Keccak, Raw-Keccak-256, Raw-MD4, Raw-MD5, Raw-MD5u, Raw-SHA1,
Raw-SHA1-AxCrypt, Raw-SHA1-Linkedin, Raw-SHA224, Raw-SHA256, Raw-SHA3,
Raw-SHA384, ripemd-128, ripemd-160, rsvp, Siemens-S7, Salted-SHA1, SSHA512,
sapb, sapg, saph, sappse, securezip, 7z, Signal, SIP, skein-256, skein-512,
skey, SL3, Snefru-128, Snefru-256, LastPass, SNMP, solarwinds, SSH, sspr,
Stribog-256, Stribog-512, STRIP, SunMD5, SybaseASE, Sybase-PROP, tacacs-plus,
tcp-md5, telegram, tezos, Tiger, tc_aes_xts, tc_ripemd160, tc_ripemd160boot,
tc_sha512, tc_whirlpool, vdi, OpenVMS, vmx, VNC, vtp, wbb3, whirlpool,
whirlpool0, whirlpool1, wpapsk, wpapsk-pmk, xmpp-scram, xsha, xsha512, ZIP,
ZipMonster, plaintext, has-160, HMAC-MD5, HMAC-SHA1, HMAC-SHA224,
HMAC-SHA256, HMAC-SHA384, HMAC-SHA512, sha1crypt-opencl, KeePass-opencl,
oldoffice-opencl, PBKDF2-HMAC-MD4-opencl, PBKDF2-HMAC-MD5-opencl,
PBKDF2-HMAC-SHA1-opencl, rar-opencl, RAR5-opencl, TrueCrypt-opencl,
lotus5-opencl, AndroidBackup-opencl, agilekeychain-opencl, ansible-opencl,
axcrypt-opencl, axcrypt2-opencl, bcrypt-opencl, BitLocker-opencl,
bitwarden-opencl, blockchain-opencl, cloudkeychain-opencl, md5crypt-opencl,
```

Select C:\Windows\System32\cmd.exe

```
PBKDF2-HMAC-SHA1-opencl, rar-opencl, RAR5-opencl, TrueCrypt-opencl,
lotus5-opencl, AndroidBackup-opencl, agilekeychain-opencl, ansible-opencl,
axcrypt-opencl, axcrypt2-opencl, bcrypt-opencl, BitLocker-opencl,
bitwarden-opencl, blockchain-opencl, cloudkeychain-opencl, md5crypt-opencl,
sha256crypt-opencl, sha512crypt-opencl, dashlane-opencl, descrypt-opencl,
diskcryptor-opencl, diskcryptor-aes-opencl, dmg-opencl,
electrum-modern-opencl, EncFS-opencl, enpass-opencl, ethereum-opencl,
ethereum-presale-opencl, FVDE-opencl, geli-opencl, gpg-opencl, iwork-opencl,
keychain-opencl, keyring-opencl, keystore-opencl, krb5pa-md5-opencl,
krb5pa-sha1-opencl, krb5asrep-aes-opencl, lp-opencl, lpcli-opencl, LM-opencl,
mscash-opencl, mscash2-opencl, mysql-sha1-opencl, notes-opencl, NT-opencl,
ntlmv2-opencl, o5logon-opencl, ODF-opencl, office-opencl,
OpenBSD-SoftRAID-opencl, PBKDF2-HMAC-SHA256-opencl,
PBKDF2-HMAC-SHA512-opencl, pem-opencl, pfx-opencl, pgpdisk-opencl,
pgpsda-opencl, pgpwde-opencl, PHPass-opencl, pwsafe-opencl, RAKP-opencl,
raw-MD4-opencl, raw-MD5-opencl, raw-SHA1-opencl, raw-SHA256-opencl,
raw-SHA512-free-opencl, raw-SHA512-opencl, salted-SHA1-opencl, sappse-opencl,
7z-opencl, SL3-opencl, solarwinds-opencl, ssh-opencl, sspr-opencl,
strip-opencl, telegram-opencl, tezos-opencl, vmx-opencl, wpapsk-opencl,
wpapsk-pmk-opencl, XSHA512-free-opencl, XSHA512-opencl, ZIP-opencl, dummy,
crypt
```

```
C:\Users\admin\Downloads\john-1.9.0-jumbo-1-win64\john-1.9.0-jumbo-1-win64\run>john --format=raw-sha256 D.txt
Using default input encoding: UTF-8
Loaded 1 password hash (Raw-SHA256 [SHA256 256/256 AVX2 8x])
Warning: poor OpenMP scalability for this hash type, consider --fork=4
Will run 4 OpenMP threads
Proceeding with single, rules:Single
Press 'q' or Ctrl-C to abort, almost any other key for status
Almost done: Processing the remaining buffered candidate passwords, if any.
Proceeding with wordlist:password.lst, rules:Wordlist
Proceeding with incremental:ASCII
dhanu            (?)
1g 0:00:00:10 DONE 3/3 (2021-09-30 11:52) 0.09544g/s 1122Kp/s 1122Kc/s 1122KC/s clubly01..peq2s
Use the "--show --format=Raw-SHA256" options to display all of the cracked passwords reliably
Session completed

C:\Users\admin\Downloads\john-1.9.0-jumbo-1-win64\john-1.9.0-jumbo-1-win64\run>
```

Algorithm for Dictionary Attack

1. Start.
2. Find all possible pairs of alphabet.
3. Store the hash value of the string alphabet pair as key and the corresponding alphabet string as value in a dictionary.
4. Get the password from the user.
5. Print the corresponding hash value using the hash function.
6. Get the hash value from the user , d[hash value] is the corresponding password.
7. End.

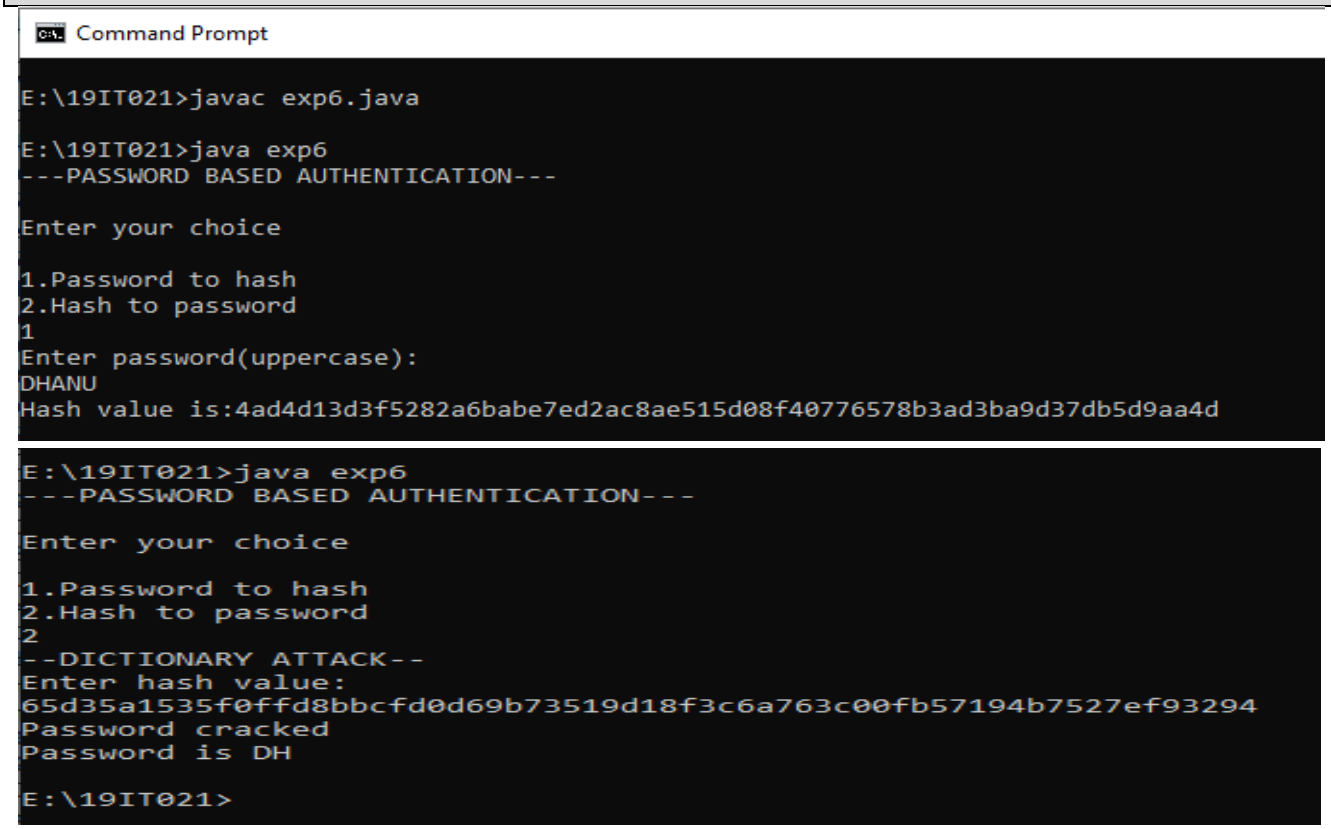
Coding

```
import java.io.*;
import java.math.BigInteger;
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;
public class exp6 {
  public static byte[] getSHA(String input) throws NoSuchAlgorithmException {
    MessageDigest md = MessageDigest.getInstance("SHA-256");
    return md.digest(input.getBytes(StandardCharsets.UTF_8));
  }
  public static String toHexString(byte[] hash) {
    BigInteger number = new BigInteger(1, hash);
    StringBuilder hexString = new StringBuilder(number.toString(16));
    while (hexString.length() < 32) {
      hexString.insert(0, '0');
    }
    return hexString.toString();
  }
  public static void printAllKLength(char[] alpha, int k) {
    int n = alpha.length;
    printAllKLengthRec(alpha, "", n, k);
  }
  public static void printAllKLengthRec(char[] alpha, String prefix, int n, int k) {
    if (k == 0) {
      try {
        String hash_word = "";
        try {
          hash_word = toHexString(getSHA(prefix));
        } catch (Exception e) {
          System.out.println(e);
        }
        FileWriter fw = new FileWriter(
            "E:\\19it021\\passwords.txt", true);
        fw.write(prefix + " " + hash_word + "\n");
```

```
            fw.close();
        } catch (IOException e) {
            System.out.println(e);
        }
        return;
    }
    for (int i = 0; i < n; ++i) {
        String newPrefix = prefix + alpha[i];
        printAllKLengthRec(alpha, newPrefix, n, k - 1);

    }
}
public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    char[] alpha = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S',
            'T', 'U', 'V', 'W', 'X', 'Y', 'Z' };
    int k = 2;
    printAllKLength(alpha, k);
    System.out.println("---PASSWORD BASED AUTHENTICATION---\n");
    System.out.println("Enter your choice\n");
    System.out.println("1.Password to hash \n2.Hash to password");
    int c = sc.nextInt();
    if (c == 1) {
        System.out.println("Enter password(uppercase):");
        String pwd = sc.next();
        try {
            String hash = toHexString(getSHA(pwd.toUpperCase()));
            System.out.println("Hash value is:" + hash);
        } catch (NoSuchAlgorithmException e) {
            System.out.println(e);
        }
    }
    if (c == 2) {
        System.out.println("--DICTIONARY ATTACK--");
        System.out.println("Enter hash value:");
        String hash = sc.next();
```

```
        try {
            File f = new File("E:\\19it021\\passwords.txt");
            Scanner scan = new Scanner(f);
            while (scan.hasNextLine()) {
                String word = scan.nextLine();
                if (word.contains(hash)) {
                    System.out.println("Password cracked");
                    String[] word1 = word.split(" ");
                    System.out.println("Password is " + word1[0]);
                    break;
                }
            }
        } catch (Exception e) {
            System.out.println(e);
        }
    }
  }
}
```

Output



passwords - Notepad

File Edit Format View Help

```
AA 58bb119c35513a451d24dc20ef0e9031ec85b35bfc919d263e7e5d9868909cb5
AB 38164fbd17603d73f696b8b4d72664d735bb6a7c88577687fd2ae33fd6964153
AC 472e73d796e20aa8ff9059e6316f218e0322548f661ec4dc267507ed66317404
AD c7bf4bbdbcd88d9d7f7c7b299c94e9e52091af2fd2888ecf85a9d6a4160b4184
AE bb1c202965ca241975a90c4d4db43001bad7ee64ba9b4411be5d2010ac8db164
AF 9b31d4edf386cb24248c71da624c02f71e3565d3a3c4e565f921fb851d5b58bc
AG 9af9eef31e66682c7a4c3053e2f28007fd5f6ea4f2b668263ed2716fe4c793d8
AH 52198b80bf68e8139a9be342771e2085d658d71ccc132fe04945ab902283a446
AI 11fb682be0a0233d5fb899721ecfc1827d20f0d2ff2e093310efa61efca8af1c
AJ a976e628e98a6dce44e87fec6ee8f557a9191663992da550899fb39085ed841a
AK 3c4e58eff203b042713969d15d21e5a80fcc43b7ce3c90517c58641dd98887c2
AL b7aea05ac84d296afc2f35daec1541dda23ec7466ea0544d19270877e2a2c41a
AM c8f48a68649d1cae147fb8afd54801e3e82622465c98ff1ee29075da2c201d78
AN 5ac4d5d4c2d31e896d7d75569c3386ac6154b70a494fa618a8975974269ef290
AO f786874742181e8c921c12e7ed5329c5587cbfa9fca8a1cde922c390b40caa45
AP 84810bfea2375dab1b1c8417b8eed248488f55b002f091f4d4ef37dffca3b307
AQ 408009a0a0b7a159e64b0a9778b7f74c3818a0696ca81dce44fa56bc089032e5
AR b1b7afa76db271451a1f3fff738ff21ed53cca91f7f580bb294193e9d2da31b4
AS de148153b07d429235e6324fa7eff44d0e873484cafdb9742080213b1d340545
AT eb5442705969dc8da5dfac8f9658dc68acdab5bb7635ba7cc551e389416ac34e
AU 86936315fce40c126916c0c980e24be16cd8fd390243c6740f58c62d08cea336
AV 10203ae4fc14a963f9d30ab2a1d0e52f2d824b30911118a9cc5089e1e0e6c969
AW 1f65c7e62e189ccf1ef96293b95b500c812a701358fd76707bda578dd9dfff2d
AX bbe96fa9f44ea0fcf38b0ec729822f37f5cc11e625753b2a250a90a3b59dd098
AY afe8c678b4fc83c7987c7e1aca3aed644887528bec25fd679e3151d54412c93a
AZ 38318c00d873bcdbf1dd521e4c68bcf16945d7e63a97f0f9d92b1c9fcbb2338e
BA 296d71a7f66e75b751c597094536329dcf2cf484f83e475d91f7aea1ff4c9738
BB fc686c314491e1f68bf1899fc54b2327353c44dd1ab4ed56538ef623edd1e866
BC 768921a22b8e190c2cfafeb0688f0d58a5f76ee4c7fb369758a208c7ba5e9acb
BD 24349c8054862cb8cbd4d857d096943e21434041ea744799074e2e28e753c14c
BE d5a79707c0bb522e0de814a42fc80baa78e3bec1267720418c8c4c019d53a1fa
BF 567d6942cf5211b4321ec51929f02f900a56260d92e6df24be91c909dedd8adb
BG 736b57465bc098745b079bbf59b7645dc4548bc5e23e4805c92fa6a35eb0e3a9
```

passwords - Notepad

File Edit Format View Help

```
YU 4b017fd4a04100462efdf7b4b21675b8655eb3e35f5a5859674e2b656cd3e6c8
YV 204f9209e73d1de4b49f59b07fd44f6e6d84c00bfca346801f4f5b480c8a7629
YW 5cb5c765a11d2215d0ce375a678a3537f915a5d06555ff2366cf58e9d3275527
YX 96d5c2e9349201e3a6adc06614678133de408b0314fe9742b9c2fb30db31add6
YY c1de18d33b8f569af81c3696b6118f28f249e99bf374d01c754de0247f34614d
YZ b7198772da96fcef4ced89a98012153597aa661d40e8cb2d38982f57af7b299b
ZA 9a8bf3ebbe6f826e39772c4763bea0722b18c0e4b5b7ba2fe6813f02f7f0a06a
ZB 2b48d13fef8d832129006ccb82737dfff20d5f4d06d6d40beb4eecb02afd3d8
ZC 1afadd3fd25008a72e359e0b35a0578c1ecfb54403aa562faf6bd5ddc03b25dd
ZD a8f5feed41a03000ce4048608db1795219ee7f544968a481c97c449b34fc0710
ZE da2969b098a8b89b82e4b159449f9dc597d38f78cfe8cbad13944c4a0e49f795
ZF d78b39badc314015ca402b1b3be5cba662e7b6f2b654092ea8fe1c57a9a15250
ZG fa5552145107fb4caf3ff0cce3621223099724d736366384b6ba72fba1a394b4
ZH e0f5e60c89070efab4b523a5b1f5e257abfab93428fd268129adf1b1d00413a5
ZI c0f9080ce75406997b07e7adb23dc218ca0bc7a810df83f9b59b98132696409c
ZJ 694f85cc4911408d30dfc5270a7a792780fe106b38718e91fa24f7ba581d5412
ZK b21eefc9fb43d7eb09a7df2f759e4fd005679a4c1cb4fcd7ddf90d8a53f714ba
ZL 4374e9564b253d09b9d628a697f2c8e6ec5b01cfd237fd20bb242078baf915
ZM 837e0bab02bfe616e5ae46c4bd4c093bbcfa44d62a33dc0c6dbc08488686dbdb
ZN d83d23aa0d3f185c1660be3eb925e10a5ecacd5fa2aa88fb69e67bc473897ef7
ZO 429e6c4f8fed9d226f77214e69c5c97fff568bbc6f24f9bd152207a73db1d581
ZP cc9929ed8247025795216e61f45e4cb9b9b688b999ec699237b2cecadbe1c556
ZQ 6528dd40892e801a7988c32307f5c953a6f1ec9008fdf3aabe0fd84d8c2ab9d5
ZR 4a0bed9babebe4eb417d611ababf6cb6c699026486d64f6d48849557fdae5b82
ZS 7dd589016eebc634490784b44ae8dae4bd91d20ce1b1fe01184dc58d7349caf6
ZT 4881e0e39e2e29ed0b5d9359112d61604d89c405ec83f7d241ef4730c4463acb
ZU 8d807b2c8677eb2d531856f7c238520ef3e108506a5906c10caa64c12610d15b
ZV a83ce6a7c082ad60e8edc14d61955e002c590235410a477d567c0ae7c5c78308
ZW 22f78469636967d0d4d49fd3ef2edbf6060ee702ad8eab9a649330bc7df6ffc5
ZX 8ddd6efd66437e0aba243767492810e279f58b7b9cab27bb66928dff051c91c1
ZY fd83ad606fcbce2b01fab6bac0530c45b4a81993a7cc2c1a6522d69ddedeeeb2
ZZ 99987a5188a32ab07b68b4219a824bb83bfcc10aca0fd4f58e41c99b37f09f9
```

RESULT:

The password cracking tool has been installed and password recovery has been done.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Tool (Installation and Exploration of Functionalities)	20	
Uniqueness of Code for Dictionary Attack	15	
Completion of experiment on time	5	
Documentation	10	
Total	50	
Signature of the faculty with Date		

Ex.No.7	ANALYSIS OF SECURITY PROTOCOLS USING WIRESHARK

AIM:

To study the working principle of security protocols like SSL/TLS using wireshark.

THEORY:

Wireshark

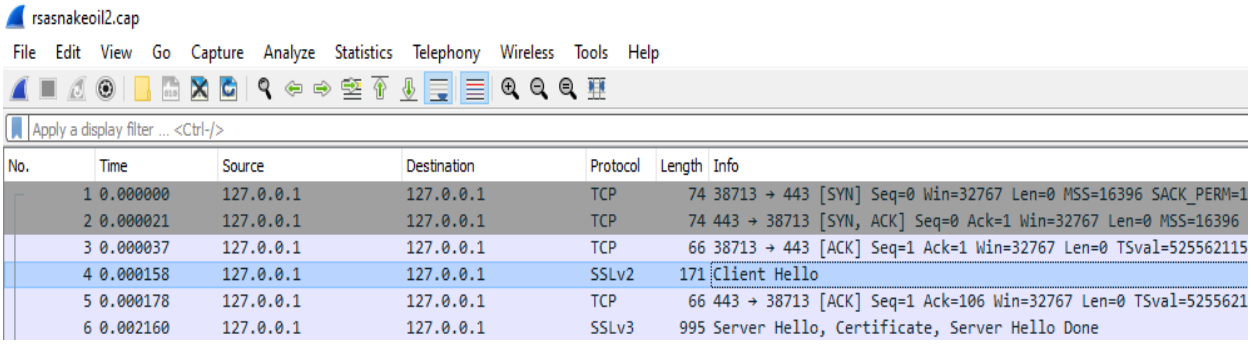
Wireshark is a packet sniffer and analysis tool. It captures network traffic on the local network and stores that data for offline analysis. Wireshark captures network traffic from Ethernet, Bluetooth, Wireless (IEEE.802.11), Token Ring, Frame Relay connections, and more. Wireshark allows you to filter the log either before the capture starts or during analysis, so you can narrow down and zero into what you are looking for in the network trace.

SSL/TLS

Secure Sockets Layer (SSL) and Transport Layer Security (TLS) are cryptographic security protocols. They are used to make sure that network communication is secure. Their main goals are to provide data integrity and communication privacy. The SSL protocol was the first protocol designed for this purpose and TLS is its successor. SSL is now considered obsolete and insecure (even its latest version), so modern browsers such as Chrome or Firefox use TLS instead. SSL and TLS are frameworks that use a lot of
different cryptographic algorithms
for example, RSA and various Diffie–Hellman algorithms.

WORKSHEET (using sample.pcap)

1. What version of SSL is supported by the client?
   Version of SSL used is version 2 (SSLv2)



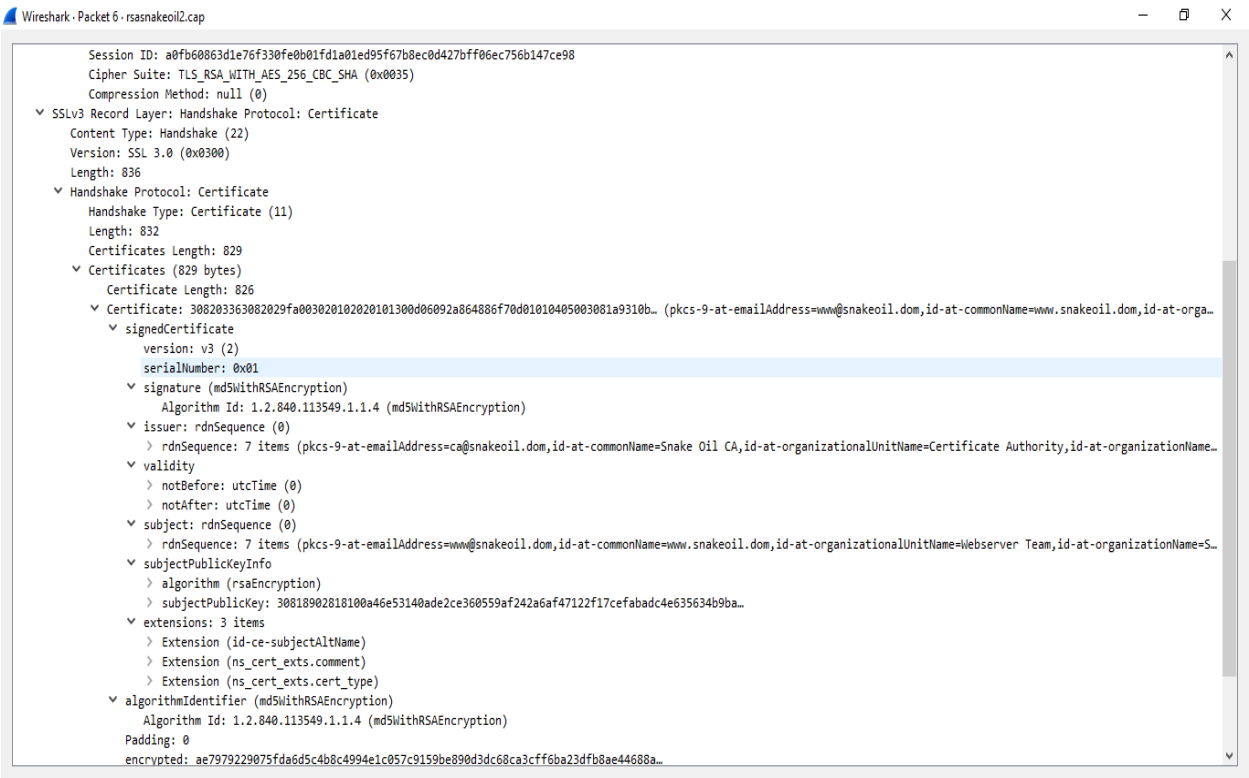
2. List the cryptographic algorithms supported by the client in Client Hello message.
Client hello --> exploring the details --> transport layer security -- > cipher specs.

Wireshark · Packet 4 · rsasnakeoil2.cap

```
        [Version: SSL 2.0 (0x0002)]
        Length: 103
        Handshake Message Type: Client Hello (1)
        Version: SSL 3.0 (0x0300)
        Cipher Spec Length: 78
        Session ID Length: 0
        Challenge Length: 16
      ▾ Cipher Specs (26 specs)
          Cipher Spec: SSL2_RC4_128_WITH_MD5 (0x010080)
          Cipher Spec: SSL2_RC2_128_CBC_WITH_MD5 (0x030080)
          Cipher Spec: SSL2_DES_192_EDE3_CBC_WITH_MD5 (0x0700c0)
          Cipher Spec: SSL2_DES_64_CBC_WITH_MD5 (0x060040)
          Cipher Spec: SSL2_RC4_128_EXPORT40_WITH_MD5 (0x020080)
          Cipher Spec: SSL2_RC2_128_CBC_EXPORT40_WITH_MD5 (0x040080)
          Cipher Spec: TLS_DHE_RSA_WITH_AES_256_CBC_SHA (0x000039)
          Cipher Spec: TLS_DHE_DSS_WITH_AES_256_CBC_SHA (0x000038)
          Cipher Spec: TLS_RSA_WITH_AES_256_CBC_SHA (0x000035)
          Cipher Spec: TLS_DHE_RSA_WITH_AES_128_CBC_SHA (0x000033)
          Cipher Spec: TLS_DHE_DSS_WITH_AES_128_CBC_SHA (0x000032)
          Cipher Spec: TLS_RSA_WITH_RC4_128_MD5 (0x000004)
          Cipher Spec: TLS_RSA_WITH_RC4_128_SHA (0x000005)
          Cipher Spec: TLS_RSA_WITH_AES_128_CBC_SHA (0x00002f)
          Cipher Spec: TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA (0x000016)
          Cipher Spec: TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA (0x000013)
          Cipher Spec: SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA (0x00feff)
          Cipher Spec: TLS_RSA_WITH_3DES_EDE_CBC_SHA (0x00000a)
          Cipher Spec: TLS_DHE_RSA_WITH_DES_CBC_SHA (0x000015)
          Cipher Spec: TLS_DHE_DSS_WITH_DES_CBC_SHA (0x000012)
          Cipher Spec: SSL_RSA_FIPS_WITH_DES_CBC_SHA (0x00fefe)
          Cipher Spec: TLS_RSA_WITH_DES_CBC_SHA (0x000009)
          Cipher Spec: TLS_RSA_EXPORT1024_WITH_RC4_56_SHA (0x000064)
          Cipher Spec: TLS_RSA_EXPORT1024_WITH_DES_CBC_SHA (0x000062)
          Cipher Spec: TLS_RSA_EXPORT_WITH_RC4_40_MD5 (0x000003)
          Cipher Spec: TLS_RSA_EXPORT_WITH_RC2_CBC_40_MD5 (0x000006)
        Challenge
```

```
0000  00 00 00 00 00 00 00 00  00 00 00 00 08 00 45 00   ........ ......E·
```

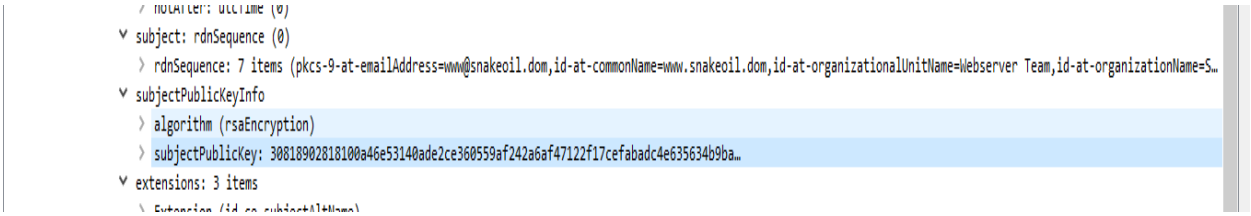
3. List the various parameters present in the public key certificate of the server.
Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate.



4. Identify the public key of the server? Can you trust the same?

   Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate. → subject public key info.

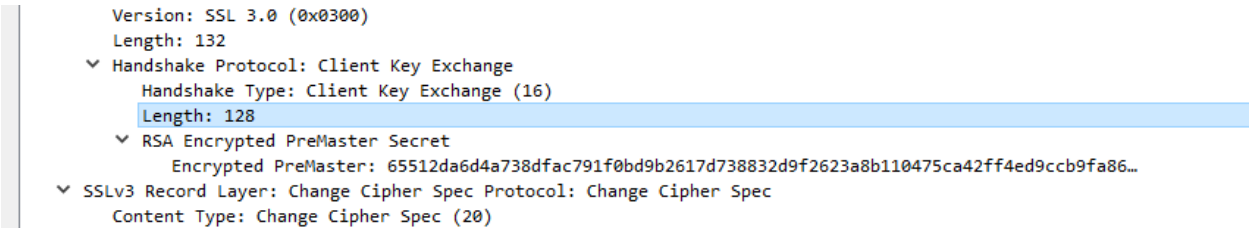
   Yes we can trust the same, because it is signed by a certificate authority(rdnSequence).



5. Identify the length of key exchanged by the Client?

   128 bits

   Client key exchange →exploring the details → transport layer security → SSLv3 Record Layer: Handshake Protocol: Client Key Exchange → Handshake Protocol: Client Key Exchange → length.



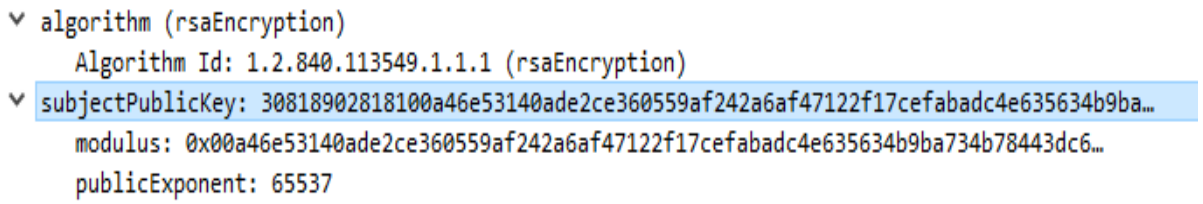
6. What algorithm is used for encrypting the session key?

RSA algorithm is used

Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate. → subject public key info.

PUBLIC KEY:

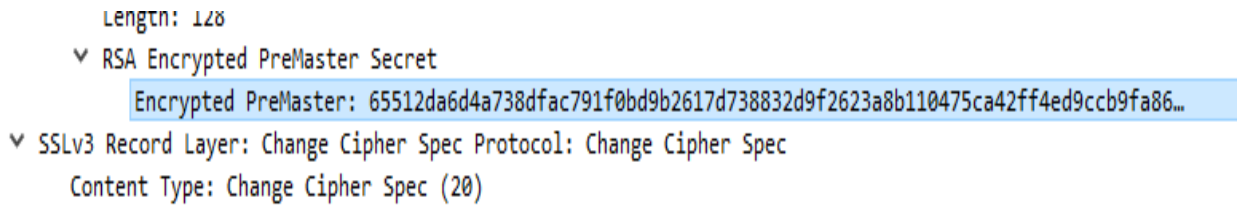
30818902818100a46e53140ade2ce360559af242a6af47122f17cefabadc4e635634b9ba734b78443dc66c69a425b361029d09043f723dd827d3b05a4577b736e42623cc12b8aedea7b63a823c7c24590af896438ba329363f917f5dc72394297f0ace0abd8d9b2f1917aad58eec66a237eb3f57533cf2aabb79194b907ea7a399fe844c89f03d0203010001



Client key exchange →exploring the details → transport layer security → SSLv3 Record Layer: Handshake Protocol: Client Key Exchange → Handshake Protocol: Client Key Exchange → RSA Encrypted PreMaster Secret

ENCRYPTED KEY:

65512da6d4a738dfac791f0bd9b2617d738832d9f2623a8b110475ca42ff4ed9ccb9fa86f3162f09735166aa29cd80610fe813ce5b8e0a23f8915e5f5470808e7b28efb669b259857498e27ed8cc7680e1b6454dc7cd84ceb4527974cde6d7d19cadef636c0ff705e44d1ad3cb9cd251b561cbff7ceec7bc5e15a3f2520fbb32



7. List the various parameters specified in the Encrypted handshake message.
   Encrypted handshake message → transport layer security → SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message → exploring the details.

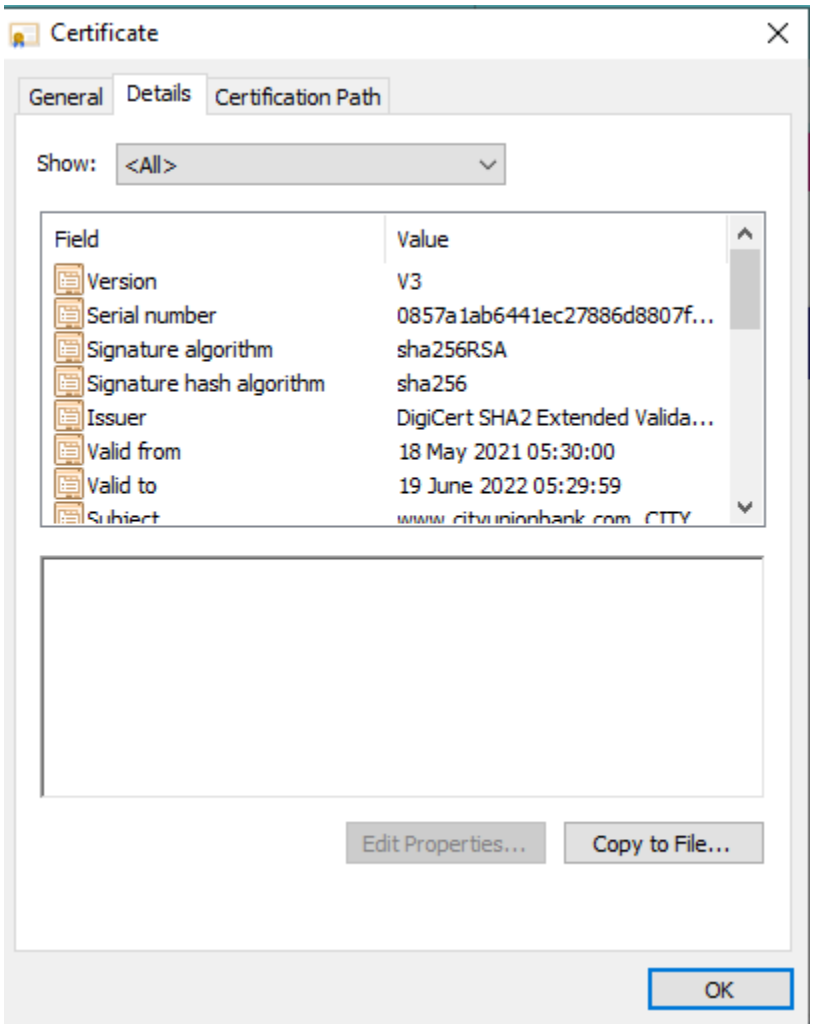
Wireshark · Packet 8 · rsasnakeoil2.cap

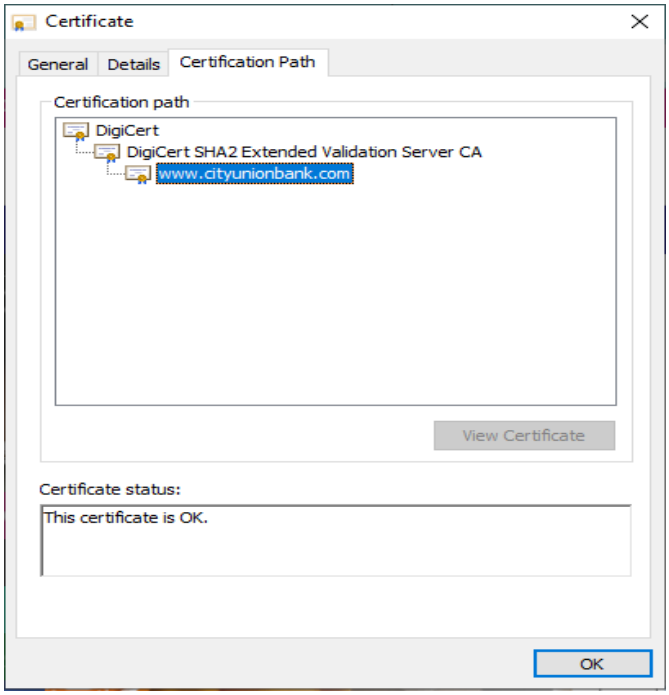
```
> Frame 8: 278 bytes on wire (2224 bits), 278 bytes captured (2224 bits)
> Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
> Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
> Transmission Control Protocol, Src Port: 38713, Dst Port: 443, Seq: 106, Ack: 930, Len: 212
v Transport Layer Security
  > SSLv3 Record Layer: Handshake Protocol: Client Key Exchange
  > SSLv3 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
  v SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: SSL 3.0 (0x0300)
      Length: 64
      Handshake Protocol: Encrypted Handshake Message
```

8. Calculate the time taken for completion of the entire handshake protocol.
   time taken for completion = End of SSL communication - Start of SSL communication.
   time taken for completion = 2.943248

3	0.000037	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1 Ack=1 Win=32767 Len=0 TSval=525562115 TSecr=525562115
4	0.000158	127.0.0.1	127.0.0.1	SSLv2	171	Client Hello
5	0.000178	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1 Ack=106 Win=32767 Len=0 TSval=525562115 TSecr=525562115
6	0.002160	127.0.0.1	127.0.0.1	SSLv3	995	Server Hello, Certificate, Server Hello Done
7	0.002609	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=106 Ack=930 Win=32767 Len=0 TSval=525562117 TSecr=525562117
8	2.808933	127.0.0.1	127.0.0.1	SSLv3	278	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
9	2.822770	127.0.0.1	127.0.0.1	SSLv3	141	Change Cipher Spec, Encrypted Handshake Message
10	2.822809	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=318 Ack=1005 Win=32767 Len=0 TSval=525564938 TSecr=525564938
11	2.833071	127.0.0.1	127.0.0.1	SSLv3	503	Application Data
12	2.873275	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1005 Ack=755 Win=32767 Len=0 TSval=525564989 TSecr=525564948
13	2.938485	127.0.0.1	127.0.0.1	SSLv3	103	Encrypted Handshake Message
14	2.938750	127.0.0.1	127.0.0.1	SSLv3	183	Encrypted Handshake Message
15	2.938761	127.0.0.1	127.0.0.1	TCP	66	443 → 38713 [ACK] Seq=1042 Ack=872 Win=32767 Len=0 TSval=525565054 TSecr=525565054
16	2.938999	127.0.0.1	127.0.0.1	SSLv3	1073	Encrypted Handshake Message, Encrypted Handshake Message, Encrypted Handshake Message
17	2.940026	127.0.0.1	127.0.0.1	SSLv3	337	Encrypted Handshake Message, Change Cipher Spec, Encrypted Handshake Message
18	2.943406	127.0.0.1	127.0.0.1	SSLv3	172	Change Cipher Spec, Encrypted Handshake Message
19	2.944825	127.0.0.1	127.0.0.1	SSLv3	5756	Application Data, Application Data
20	2.944864	127.0.0.1	127.0.0.1	TCP	66	38713 → 443 [ACK] Seq=1143 Ack=7845 Win=32767 Len=0 TSval=525565060 TSecr=525565059

9. Examine the certificates of gmail server, any bank server. Identify the Cerification Authority, Hierarchy of Certification Authority, Validity date, crypto algorithms used for signing etc.
   bank server: city union bank
   Certificate authority: DigiCert SHA2 Secure Server CA
   Validity: september 18-05-2021 to, November 19-06-2022
   Signing Alogorithm: sha256RSA
   Hash algorithm: sha256
   Public key : RSA(2048 bits)





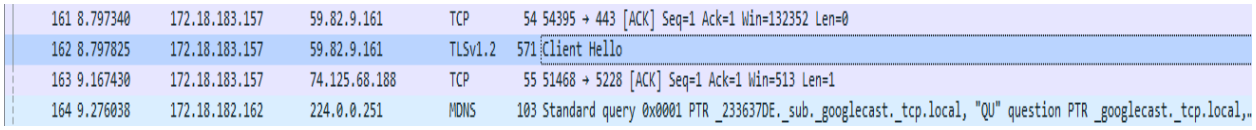
Hierarchy of certificate authority

10. Enlist the differences between HTTP and HTTPS.

HTTP	**HTTPS**
1) Hypertext Transfer Protocol is unsecured when compared to HTTP.	1) Hypertext Transfer Protocol Secure is more secure.
2) HTTP is the foundation of the data communication for the web. It is TCP/IP based protocol and things like text, audio, videos, images can be transmitted through it	2) HTTPS is nothing but the HTTP working in tandem with **SSL (Secure Socket Layer)** that is the "S" in HTTPS. SSL takes care of transferring data securely over the internet.
3) Hypertext Transfer Protocol operates at the application layer.	3) Hypertext Transfer Protocol Secure functions at Transport Layer.
4) HTTP requires no certificates to verify the identity of websites.	4) HTTPS requires certificates to verify the identity of the websites.
5) No encryption is used in HTTP.	5) Both encryption and decryption are used in HTTPS
6) HTTP is prone to man-in-the-middle and eavesdropping attacks.	6) HTTPS is designed to resist such attacks
7) Examples: Educational sites, internet forums etc	7) Examples: Payment gateways, shopping websites etc.
WORKSHEET (For packet captured for real time data)	

1. What version of SSL is supported by the client?

   Version of SSL used is version 1.2 (TLSv1.2)



2. List the cryptographic algorithms supported by the client in Client Hello message.

   Client hello --> exploring the details --> transport layer security -- > cipher specs.

Wireshark · Packet 162 · WiFi

```
Session ID: c5d818662f68d22e4652e853db5f3e7d1586022a90442904a68aca59d1ad72d1
Cipher Suites Length: 32
Cipher Suites (16 suites)
    Cipher Suite: Reserved (GREASE) (0x1a1a)
    Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
    Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
    Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca9)
    Cipher Suite: TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256 (0xcca8)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256 (0xc02b)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
    Cipher Suite: TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384 (0xc02c)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA (0xc013)
    Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA (0xc014)
    Cipher Suite: TLS_RSA_WITH_AES_128_GCM_SHA256 (0x009c)
    Cipher Suite: TLS_RSA_WITH_AES_256_GCM_SHA384 (0x009d)
    Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA (0x002f)
    Cipher Suite: TLS_RSA_WITH_AES_256_CBC_SHA (0x0035)
Compression Methods Length: 1
```

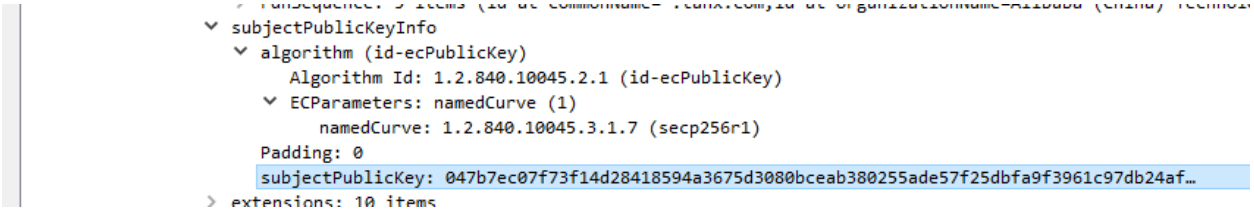
3. List the various parameters present in the public key certificate of the server.
   Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate.

Wireshark · Packet 191 · WiFi

```
Handshake Protocol: Certificate
    Handshake Type: Certificate (11)
    Length: 12823
    Certificates Length: 12820
  Certificates (12820 bytes)
      Certificate Length: 11681
    Certificate: 30822d9d30822c85a003020102020c59f55bf1b5497340a828b4dc300d06092a864886f7… (id-at-commonName=*.tanx.com,id-at-organizationName=Alibaba (China) Technology Co., Ltd
      signedCertificate
          version: v3 (2)
          serialNumber: 0x59f55bf1b5497340a828b4dc
        signature (sha256WithRSAEncryption)
            Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
        issuer: rdnSequence (0)
          rdnSequence: 3 items (id-at-commonName=GlobalSign Organization Validation CA - SHA256,id-at-organizationName=GlobalSign nv-sa,id-at-countryName=BE)
            RDNSequence item: 1 item (id-at-countryName=BE)
            RDNSequence item: 1 item (id-at-organizationName=GlobalSign nv-sa)
            RDNSequence item: 1 item (id-at-commonName=GlobalSign Organization Validation CA - SHA256)
        validity
            notBefore: utcTime (0)
            notAfter: utcTime (0)
        subject: rdnSequence (0)
            rdnSequence: 5 items (id-at-commonName=*.tanx.com,id-at-organizationName=Alibaba (China) Technology Co., Ltd.,id-at-localityName=HangZhou,id-at-stateOrProvinceName=Zh
        subjectPublicKeyInfo
          algorithm (id-ecPublicKey)
              Algorithm Id: 1.2.840.10045.2.1 (id-ecPublicKey)
            ECParameters: namedCurve (1)
                namedCurve: 1.2.840.10045.3.1.7 (secp256r1)
            Padding: 0
            subjectPublicKey: 047b7ec07f73f14d28418594a3675d3080bceab380255ade57f25dbfa9f3961c97db24af…
        extensions: 10 items
      algorithmIdentifier (sha256WithRSAEncryption)
          Algorithm Id: 1.2.840.113549.1.1.11 (sha256WithRSAEncryption)
        Padding: 0
        encrypted: 1cb9aa138ca7c49f72d74dbfd1db41ecb418162ac6b9d8a59540e46c4403da2b1666ac11…
```

4. Identify the public key of the server? Can you trust the same?
   Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate. → subject public key info.

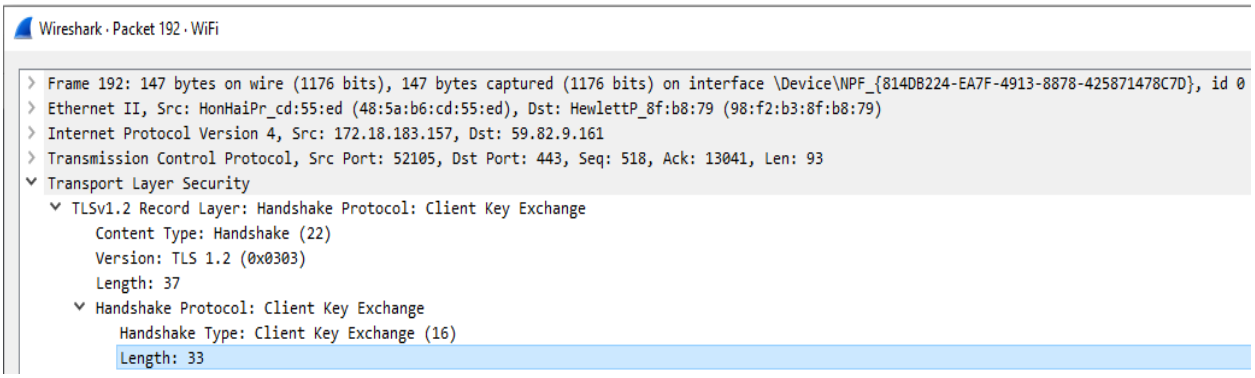
Yes we can trust the same, because it is signed by a certificate authority(rdnSequence).



5. Identify the length of key exchanged by the Client?

33 bits

Client key exchange →exploring the details → transport layer security → SSLv3 Record Layer: Handshake Protocol: Client Key Exchange → Handshake Protocol: Client Key Exchange → length.



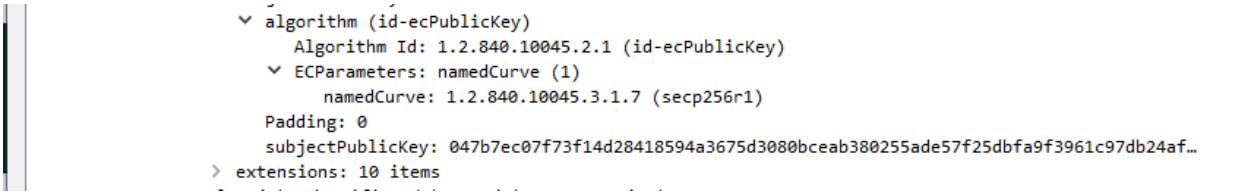
6. What algorithm is used for encrypting the session key?

RSA algorithm is used

Server hello certificate → exploring the details → transport layers security →SSLv3 Record Layer: Handshake Protocol: Certificate. → subject public key info.

PUBLIC KEY:

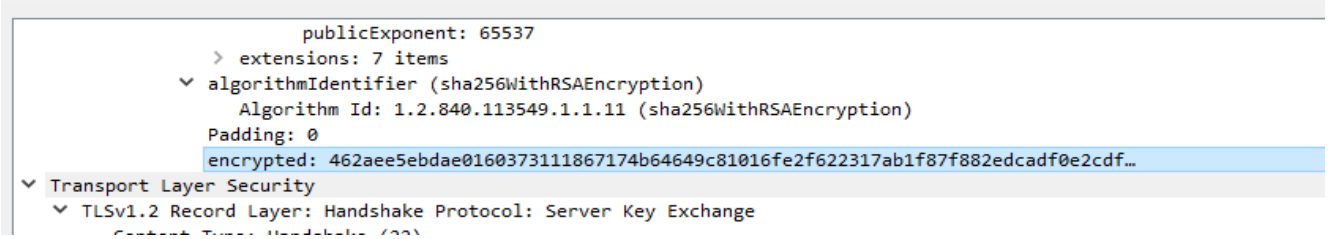
047b7ec07f73f14d28418594a3675d3080bceab380255ade57f25dbfa9f3961c97db24af894ff0cb7630f5d47cf64206c411a8f66e1116a1260df30d4815a0ebac….



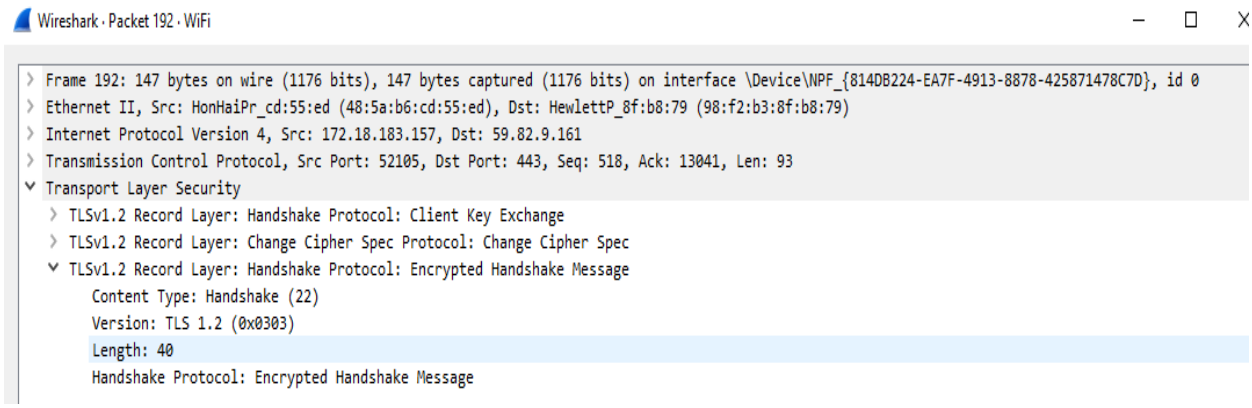
Client key exchange →exploring the details → transport layer security → SSLv3 Record Layer: Handshake Protocol: Client Key Exchange → Handshake Protocol: Client Key Exchange → RSA Encrypted PreMaster Secret

ENCRYPTED KEY:

462aee5ebdae0160373111867174b64649c81016fe2f622317ab1f87f882edcadf0e2cdf647
58ee51872a78c3a8bc9aca57750f7ef9ea4e0a08f1457a32a5fec7e6d10e6ba8db00887760e
4cb2d951bb1102f25cdd1cbdf355960fd406c0fce2238a2470d3bbf0791aa76170838aaf06c
520d8a163d06cae4f32d7ae7c184575052977df4240646486be2a7609316f1d24f499d085f
ef22108f9c6f6f1d059edd6563c08280367baf0f9f1901647ae67e6bc8048e9427634975569
240e83d6a02db4f5f3798a4928741a41a1c2d324883530609417b4e10422313d3b2f1706
b2b89d862b5a69ef83f54bc4aab42af87ca1b185948cf40c870cf4ac40f8594998

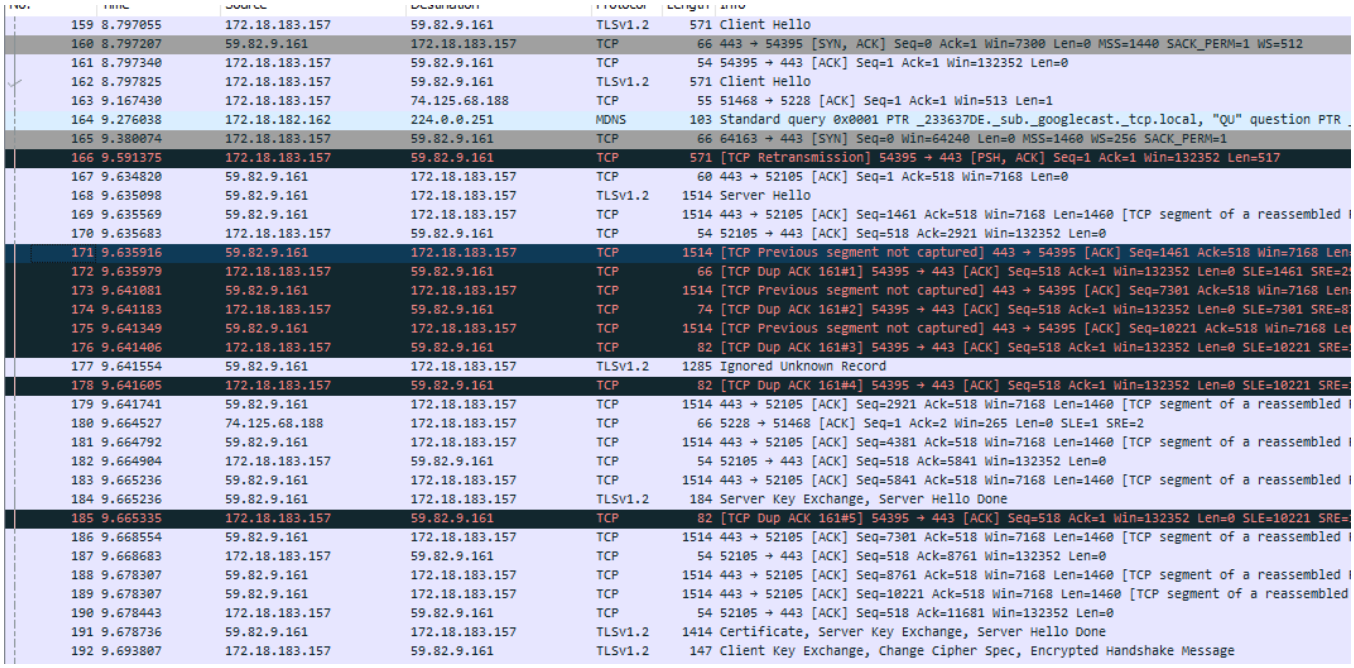


7. List the various parameters specified in the Encrypted handshake message.
   Encrypted handshake message → transport layer security → SSLv3 Record Layer: Handshake Protocol: Encrypted Handshake Message → exploring the details.



8. Calculate the time taken for completion of the entire handshake protocol.
   time taken for completion = End of SSL communication - Start of SSL communication.

time taken for completion = 0.896752



No.	Time	Source	Destination	Protocol	Length	Info
159	8.797055	172.18.183.157	59.82.9.161	TLSv1.2	571	Client Hello
160	8.797207	59.82.9.161	172.18.183.157	TCP	66	443 → 54395 [SYN, ACK] Seq=0 Ack=1 Win=7300 Len=0 MSS=1440 SACK_PERM=1 WS=512
161	8.797340	172.18.183.157	59.82.9.161	TCP	54	54395 → 443 [ACK] Seq=1 Ack=1 Win=132352 Len=0
162	8.797825	172.18.183.157	59.82.9.161	TLSv1.2	571	Client Hello
163	9.167430	172.18.183.157	74.125.68.188	TCP	55	51468 → 5228 [ACK] Seq=1 Ack=1 Win=513 Len=1
164	9.276038	172.18.182.162	224.0.0.251	MDNS	103	Standard query 0x0001 PTR _233637DE._sub._googlecast._tcp.local, "QU" question PTR
165	9.380074	172.18.183.157	59.82.9.161	TCP	66	64163 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
166	9.591375	172.18.183.157	59.82.9.161	TCP	571	[TCP Retransmission] 54395 → 443 [PSH, ACK] Seq=1 Ack=1 Win=132352 Len=517
167	9.634820	59.82.9.161	172.18.183.157	TCP	60	443 → 52105 [ACK] Seq=1 Ack=518 Win=7168 Len=0
168	9.635098	59.82.9.161	172.18.183.157	TLSv1.2	1514	Server Hello
169	9.635569	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=1461 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
170	9.635683	172.18.183.157	59.82.9.161	TCP	54	52105 → 443 [ACK] Seq=518 Ack=2921 Win=132352 Len=0
171	9.635916	59.82.9.161	172.18.183.157	TCP	1514	[TCP Previous segment not captured] 443 → 54395 [ACK] Seq=1461 Ack=518 Win=7168 Len
172	9.635979	172.18.183.157	59.82.9.161	TCP	66	[TCP Dup ACK 161#1] 54395 → 443 [ACK] Seq=518 Ack=1 Win=132352 Len=0 SLE=1461 SRE=2
173	9.641081	59.82.9.161	172.18.183.157	TCP	1514	[TCP Previous segment not captured] 443 → 54395 [ACK] Seq=7301 Ack=518 Win=7168 Len
174	9.641183	172.18.183.157	59.82.9.161	TCP	74	[TCP Dup ACK 161#2] 54395 → 443 [ACK] Seq=518 Ack=1 Win=132352 Len=0 SLE=7301 SRE=8
175	9.641349	59.82.9.161	172.18.183.157	TCP	1514	[TCP Previous segment not captured] 443 → 54395 [ACK] Seq=10221 Ack=518 Win=7168 Le
176	9.641406	172.18.183.157	59.82.9.161	TCP	82	[TCP Dup ACK 161#3] 54395 → 443 [ACK] Seq=518 Ack=1 Win=132352 Len=0 SLE=10221 SRE=
177	9.641554	59.82.9.161	172.18.183.157	TLSv1.2	1285	Ignored Unknown Record
178	9.641605	172.18.183.157	59.82.9.161	TCP	82	[TCP Dup ACK 161#4] 54395 → 443 [ACK] Seq=518 Ack=1 Win=132352 Len=0 SLE=10221 SRE=
179	9.641741	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=2921 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
180	9.664527	74.125.68.188	172.18.183.157	TCP	66	5228 → 51468 [ACK] Seq=1 Ack=2 Win=265 Len=0 SLE=1 SRE=2
181	9.664792	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=4381 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
182	9.664904	172.18.183.157	59.82.9.161	TCP	54	52105 → 443 [ACK] Seq=518 Ack=5841 Win=132352 Len=0
183	9.665236	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=5841 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
184	9.665236	59.82.9.161	172.18.183.157	TLSv1.2	184	Server Key Exchange, Server Hello Done
185	9.665335	172.18.183.157	59.82.9.161	TCP	82	[TCP Dup ACK 161#5] 54395 → 443 [ACK] Seq=518 Ack=1 Win=132352 Len=0 SLE=10221 SRE=
186	9.668554	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=7301 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
187	9.668683	172.18.183.157	59.82.9.161	TCP	54	52105 → 443 [ACK] Seq=518 Ack=8761 Win=132352 Len=0
188	9.678307	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=8761 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
189	9.678307	59.82.9.161	172.18.183.157	TCP	1514	443 → 52105 [ACK] Seq=10221 Ack=518 Win=7168 Len=1460 [TCP segment of a reassembled
190	9.678443	172.18.183.157	59.82.9.161	TCP	54	52105 → 443 [ACK] Seq=518 Ack=11681 Win=132352 Len=0
191	9.678736	59.82.9.161	172.18.183.157	TLSv1.2	1414	Certificate, Server Key Exchange, Server Hello Done
192	9.693807	172.18.183.157	59.82.9.161	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message

RESULT:

Thus the network traffic is analyzed for security analysis using Wireshark.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Work	15	
Completion of experiment on time	10	
Documentation	5	
Total	30	
Signature of the faculty with Date		

Ex.No.8	**IMPLEMENTATION OF KEY GENERATION IN ADVANCED ENCRYPTION STANDARD**

AIM:

To implement key generation in Advanced Encryption Standard using Java/Python

ALGORITHM:

- Start
- Create a sbox for AES and rcon matrix for substitute bytes and key generationprocess.

Substitutestate:

- Loop through he state matrix
- substitute the corresponding values in

sbox.Print:

- loop through the given matrix and print the values in the

matrix.Rotword:

- implement left circular shift for the given matrix by looping through the

matrix.Getcol:

- Get the input matrix and column number
- And create column matrix
- And return the

matrixXor:

- Get two input matrix and loop through the matrix.
- And XOR the element of the matrix.

G:

- Get the word and call the rotword
- And get the result with substitutebytes.
- Xor the result with rcon matrix created.

Fill:

- Get two matrix and 2 integers.
- Refill the first matrix with the condition the elements should between given integers.

Main:

- Create the main key and word
- First call the fill method with the created word and main key
- Then print the main key.
- Then loop from 4 to 44
- If the num%4 is 0 then call g with the word and xor with temp then fill function
- Else getcol for the word and xor between them
- Then fill function with the result and word
- If num%4 ==3 then print the same word.
- End.

CODING:

AES.java

```
public class AES {

public static final int[][] sbox = {
    {0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe,
0xd7,0xab, 0x76},
    {0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c,
0xa4, 0x72,0xc0},
    {0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71,
0xd8, 0x31,0x15},
    {0x04, 0xc7, 0x23, 0xc3, 0x18, 0x96, 0x05, 0x9a, 0x07, 0x12, 0x80, 0xe2,
0xeb, 0x27,0xb2, 0x75},
    {0x09, 0x83, 0x2c, 0x1a, 0x1b, 0x6e, 0x5a, 0xa0, 0x52, 0x3b, 0xd6, 0xb3,
0x29, 0xe3,0x2f, 0x84},
    {0x53, 0xd1, 0x00, 0xed, 0x20, 0xfc, 0xb1, 0x5b, 0x6a, 0xcb, 0xbe, 0x39,
0x4a, 0x4c,0x58, 0xcf},
    {0xd0, 0xef, 0xaa, 0xfb, 0x43, 0x4d, 0x33, 0x85, 0x45, 0xf9, 0x02, 0x7f, 0x50,
0x3c, 0x9f,0xa8},
    {0x51, 0xa3, 0x40, 0x8f, 0x92, 0x9d, 0x38, 0xf5, 0xbc, 0xb6, 0xda, 0x21,
0x10, 0xff, 0xf3,0xd2},
    {0xcd, 0x0c, 0x13, 0xec, 0x5f, 0x97, 0x44, 0x17, 0xc4, 0xa7, 0x7e, 0x3d,
0x64, 0x5d,0x19, 0x73},
```

```
    {0x60, 0x81, 0x4f, 0xdc, 0x22, 0x2a, 0x90, 0x88, 0x46, 0xee, 0xb8, 0x14, 0xde,
0x5e,0x0b, 0xdb},
    {0xe0, 0x32, 0x3a, 0x0a, 0x49, 0x06, 0x24, 0x5c, 0xc2, 0xd3, 0xac, 0x62, 0x91,
0x95,0xe4, 0x79},
    {0xe7, 0xc8, 0x37, 0x6d, 0x8d, 0xd5, 0x4e, 0xa9, 0x6c, 0x56, 0xf4, 0xea, 0x65,
0x7a,0xae, 0x08},
    {0xba, 0x78, 0x25, 0x2e, 0x1c, 0xa6, 0xb4, 0xc6, 0xe8, 0xdd, 0x74, 0x1f, 0x4b,
0xbd,0x8b, 0x8a},
{0x70, 0x3e, 0xb5, 0x66, 0x48, 0x03, 0xf6, 0x0e, 0x61, 0x35, 0x57, 0xb9, 0x86,
0xc1,
0x1d, 0x9e},
{0xe1, 0xf8, 0x98, 0x11, 0x69, 0xd9, 0x8e, 0x94, 0x9b, 0x1e, 0x87, 0xe9, 0xce, 0x55,
0x28, 0xdf},
{0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54,
0xbb, 0x16}
};

public static final int[][] rcon = {
{0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0x1b,0x36},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
{0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00}
};

public static int[][] substituteState(int[][] state){
for(int i=0;i<state.length;i++){
for(int j=0;j<state[0].length;j++){
int hex=state[i][j];
state[i][j]=sbox[hex/16][hex%16];
}
}
return state;
}

public static void print(int
mat[][]){for(int
i=0;i<mat.length;i++){
for(int j=0;j<mat[0].length;j++){
```

```
        System.out.print(mat[i][j]+" ");

System.out.println();
        }
    }

    public static int[][] rotword(int mat[][]){
        int temp=mat[0][0];
        for(int i=1;i<mat.length;i++){
            mat[i-1][0]=mat[i][0];
        }
        mat[mat.length-1][0]=temp;
        return mat;
    }

    public static int[][] getCol(int mat[][],int j){
        int col[][]=new int[mat.length][1];
        for(int i=0;i<mat.length;i++){
            col[i][0]=mat[i][j];
        }
        return col;
          }

          public static int[][] xor(int[][] mat1,int[][] mat2){
            int[][] temp=new int[mat1.length][1];
          for(int i=0;i<mat1.length;i++){
            temp[i][0]=(mat1[i][0] ^
            mat2[i][0]);
          }
          return temp;
          }

          public static int[][] g(int[][] word,int col,int i){
          int[][] tempword=substituteState(rotword(getCol(word,col)));
            return xor(tempword,getCol(rcon,i));
          }

          public static int[][] fill(int[][] mat1,int[][] mat2,int j1,int j2){
```

```
  for(int i=0;i<mat1.length;i++){
for(int j=j1,k=0;(j<j2) && (k<mat2[0].length);j++,k++){
  mat1[i][j]=mat2[i][k];
}
}
return mat1;
}

public static void main(String[]
  args){int mainkey[][]={
{0x2b,0x28,0xab,0x9},
{0x7e,0xae,0xf7,0xcf},
{0x15,0xd2,0x15,0x4f},
{0x16,0xa6,0x88,0x3c}
};
int word[][]=new int[4][44];
  fill(word,mainkey,0,4);
  System.out.println("MAIN KEY
  0");print(word);

int[][]
  temp;int
  count=0;
for(int
  i=4;i<44;i++){
  if(i%4==0){
temp=g(word,i-1,count);
  temp=xor(getCol(word,i-4),temp);
  fill(word,temp,i,i+1);
count++;
}
else{
temp=xor(getCol(word,i-4),getCol(word,i-1));
  fill(word,temp,i,i+1);
} if(i%4==3){
System.out.println("ROUND "+((i/4)));
  print(word);
}
```

```
}
}
}
```

SCREEN SHOTS:

```
D:\java>javac AES.java

D:\java>java AES
MAIN KEY 0
43 40 171 9 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
126 174 247 207 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21 210 21 79 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22 166 136 60 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROUND 1
43 40 171 9 160 136 35 42 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROUND 2
43 40 171 9 160 136 35 42 242 122 89 115 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROUND 3
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROUND 4
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 0 0 0 0 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 0 0 0 0 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 0 0 0 0 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ROUND 5
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 0 0 0 0 0 0 0 0
ROUND 6
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 109 17 219 202 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 136 11 249 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 163 62 134 147 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 122 253 65 253 0 0 0 0 0 0 0 0
ROUND 7
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 109 17 219 202 78 95 132 78 0 0 0 0 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 136 11 249 0 84 95 166 166 0 0 0 0 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 163 62 134 147 247 201 79 220 0 0 0 0 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 122 253 65 253 14 243 178 79 0 0 0 0 0 0 0 0
ROUND 8
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 109 17 219 202 78 95 132 78 234 181 49 127 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 136 11 249 0 84 95 166 166 210 141 43 141 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 163 62 134 147 247 201 79 220 115 186 245 41 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 122 253 65 253 14 243 178 79 33 210 96 47 0 0 0 0
ROUND 9
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 109 17 219 202 78 95 132 78 234 181 49 127 172 25 40 87 0 0 0 0
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 136 11 249 0 84 95 166 166 210 141 43 141 119 250 209 92 0 0 0 0
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 163 62 134 147 247 201 79 220 115 186 245 41 102 220 41 0 0 0 0
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 122 253 65 253 14 243 178 79 33 210 96 47 243 33 65 110 0 0 0 0
ROUND 10
43 40 171 9 160 136 35 42 242 122 89 115 61 71 30 109 239 168 182 219 212 124 202 17 109 17 219 202 78 95 132 78 234 181 49 127 172 25 40 87 208 201 225 182
126 174 247 207 250 84 163 108 194 150 53 89 128 22 35 122 68 82 113 11 209 131 242 249 136 11 249 0 84 95 166 166 210 141 43 141 119 250 209 92 20 238 63 99
21 210 21 79 254 44 57 118 149 185 128 246 71 254 126 136 165 91 37 173 198 157 184 21 163 62 134 147 247 201 79 220 115 186 245 41 102 220 41 0 249 37 12 12
22 166 136 60 23 177 57 5 242 67 122 127 125 62 68 59 65 127 59 0 248 135 188 188 122 253 65 253 14 243 178 79 33 210 96 47 243 33 65 110 168 137 200 166
```

RESULT:

Key generation in AES using Java is implemented.

Evaluation

Parameter	Max Marks	Marks Obtained
Uniqueness of the Code	15	
Completion of experiment on time	10	
Documentation	5	
Total	30	
Signature of the faculty with Date		

Ex.No.10	SQL Injection – Damn Vulnerable Web Application

AIM:

To demonstrate SQL injection attack using Damn Vulnerable Web Application (DVWA).

THEORY:

About DVWA

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

SQL injection Attack

SQL injection is a code injection technique, used to attack data-driven applications, in which malicious SQL statements are inserted into an entry field for execution (e.g. to dump the database contents to the attacker). SQL injection must exploit a security vulnerability in an application's software, for example, when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and unexpectedly executed. SQL injection is mostly known as an attack vector for websites but can be used to attack any type of SQL database.

Ways to prevent SQL injection

- **Continuous Scanning and Penetration Testing**

The automated web application scanner has been the best choice to point out vulnerabilities within the web applications for quite some time now. Now, with SQL injections getting smarter in exploiting logical flaws, website security professionals should explore manual testing with the help of a security vendor.

- **Restrict Privileges**

It is more of a database management function, but enforcing specific privileges to specific accounts helps prevent blind SQL injection attacks. Begin with no privileges account and move on to ‘read-only’, ‘edit’, ‘delete’ and similar privilege levels.

- **Use Query Parameters**

Dynamic queries create a lot of troubles for security professionals. They have to deal with variable vulnerabilities in each application, which only gets graver with updates and changes. It is recommended that you prepare parameterized queries.

- **Instant Protection**

A majority of organizations fail the problems like outdated code, scarcity of resources to test and make changes, no knowledge of application security, and frequent updates in the application. For these, web application protection is the best solution.

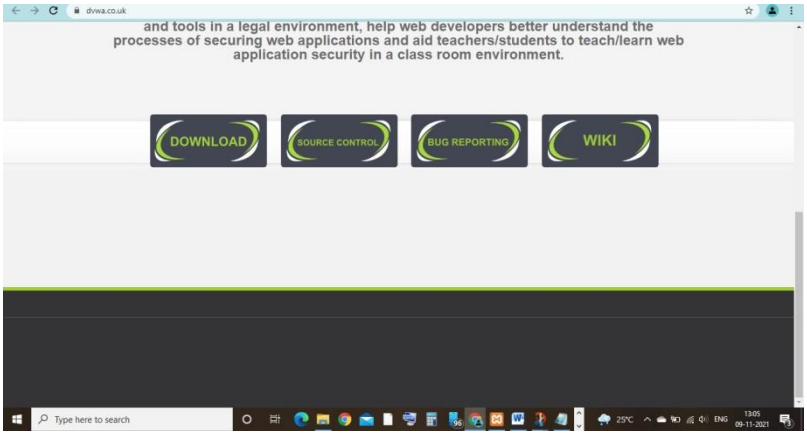
Worksheet (Answer the following ques and paste the relevant outputs)

Installation procedure for XAMPP and DVWA

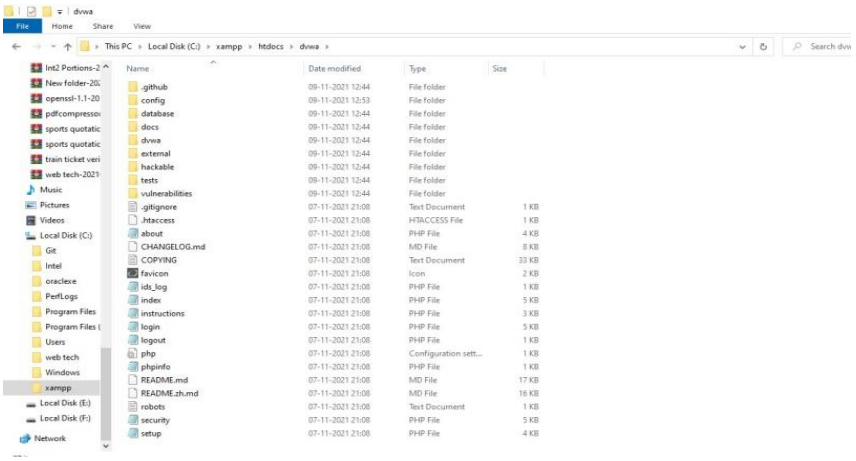
1) What is **DVWA** ?

Damn Vulnerable Web App (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goals are to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and aid teachers/students to teach/learn web application security in a class room environment.

Download Link : http://www.dvwa.co.uk/

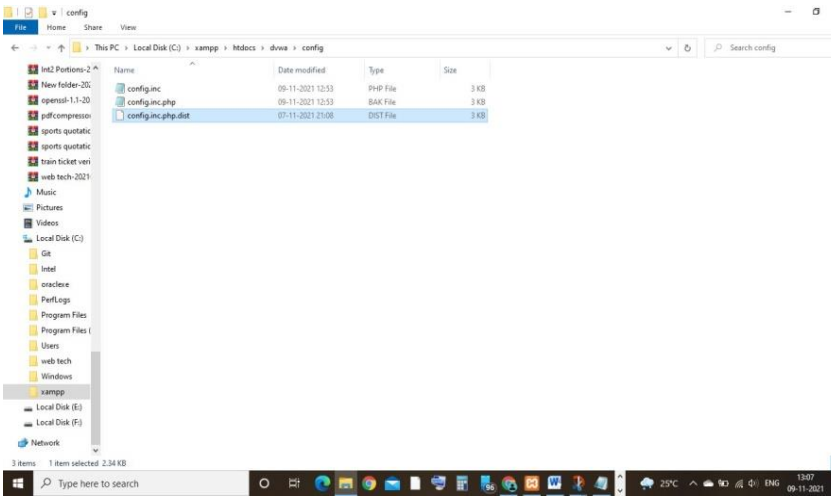


3) Download the zip file and extract it and rename it to "**dvwa**".



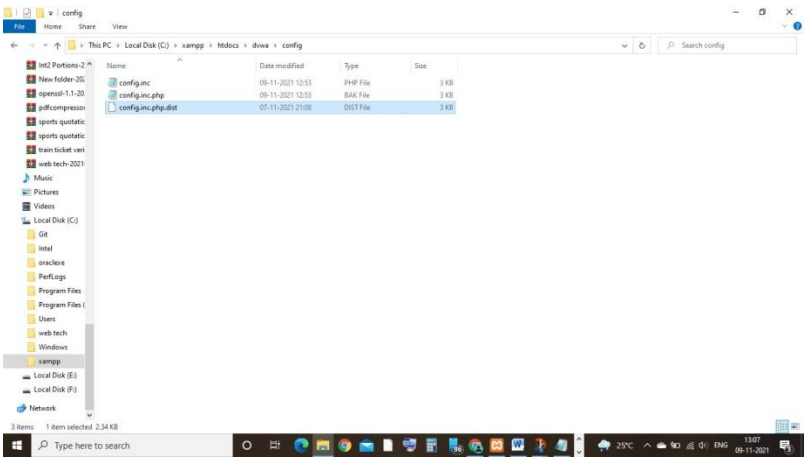
Extract the downloaded dvwa zip and rename it to dvwa.

4) Now, Open "**dvwa**" folder, "**DVWA-MASTER**". There will be a folder named "**config**". Open that folder and you can see a file named "**config.inc.php.dist**".



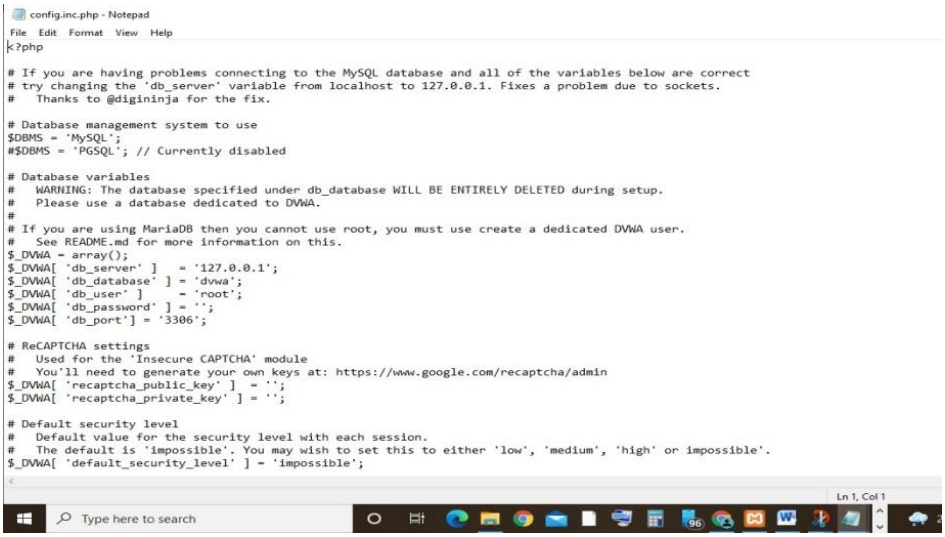
Open config folder

5) Copy "**config.inc.php.dist**" file and paste it in same folder. and rename it to .php extension "**config.inc.php**".



6) Now, Edit "**config.inc.php**" file with notepad and remove "**p@ssw0rd**" from **$_DVWA['db_password'] = 'p@ssw0rd';**

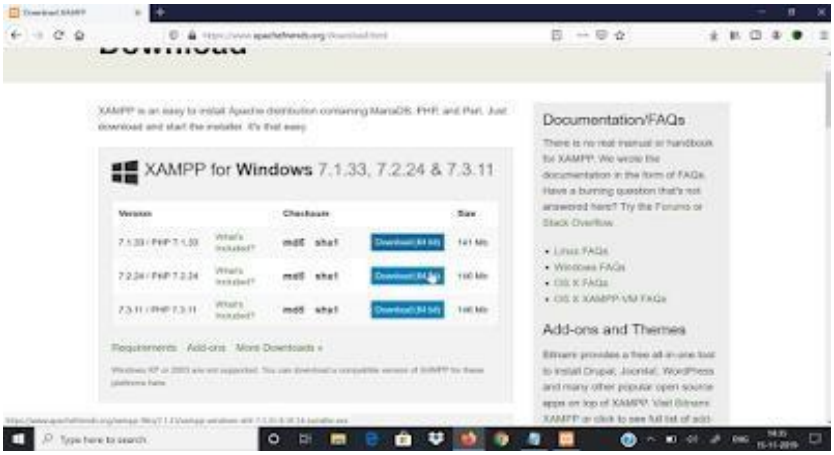
So, the command would be : **$_DVWA['db_password'] = '';**
and save the file.



7) For installing dvwa on your local server, You will have to install **XAMPP** on your windows machine and if you are a linux user, You could use **LAMPP.**

8) Download Xampp and install it.

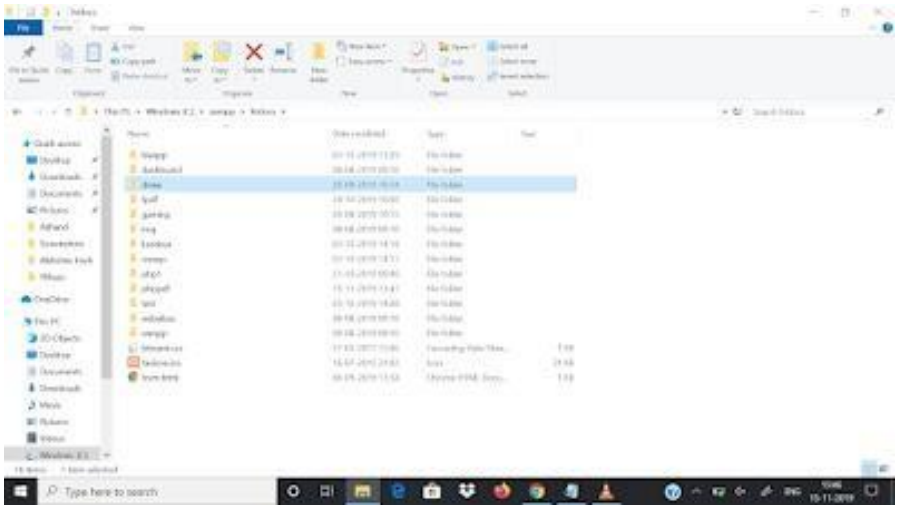
Download Link : https://www.apachefriends.org/download.html



Download Xampp Image screenshot.

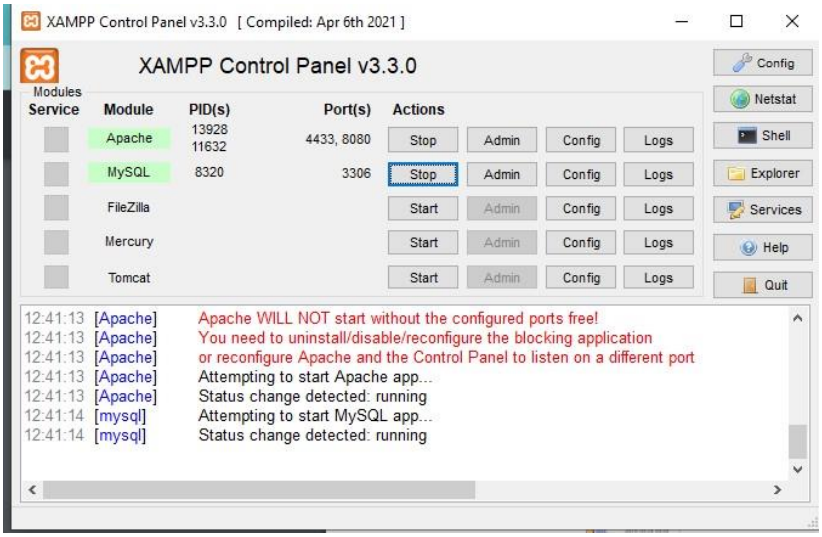
9) After completing the installation of **XAMPP**, copy the extracted **DVWA** folder and paste it into :

C:\xampp\htdocs



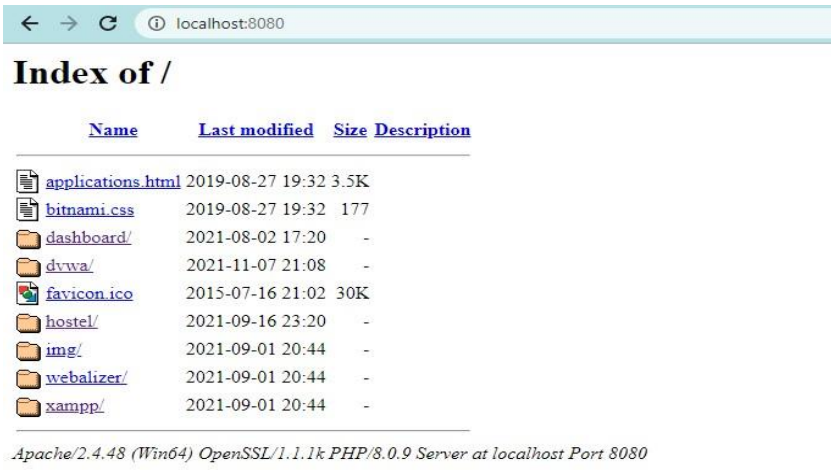
Move that folder in xampp\htdocs

10) Now, Run **XAMPP** software and Start **Apache** and **MySQL** service and Minimize it.



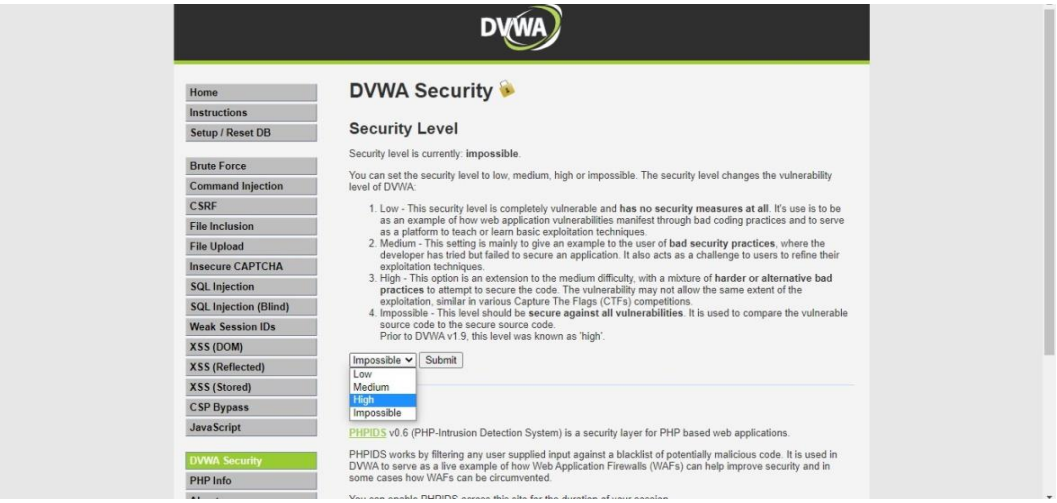
Start Apache and MySQL service in XAMPP

11) Open your browser, and type in the url : "**localhos**t" and hit enter.

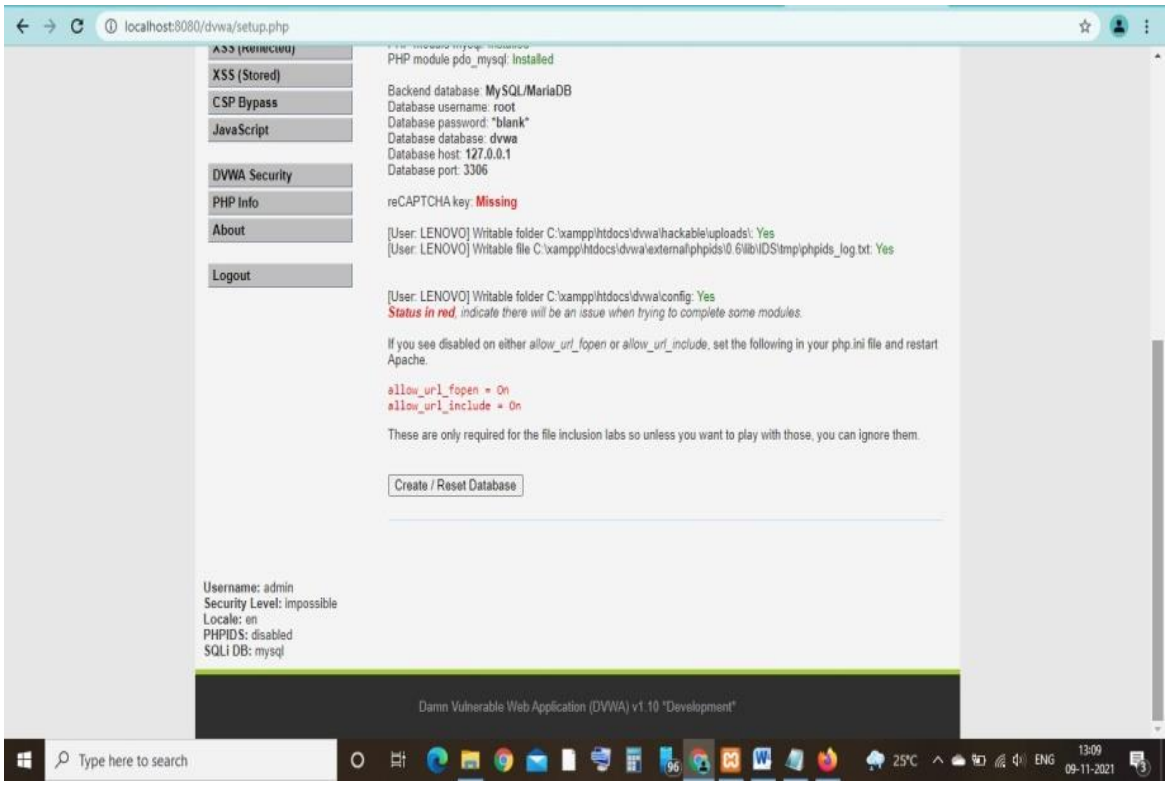


Open browser and type localhost in the URL

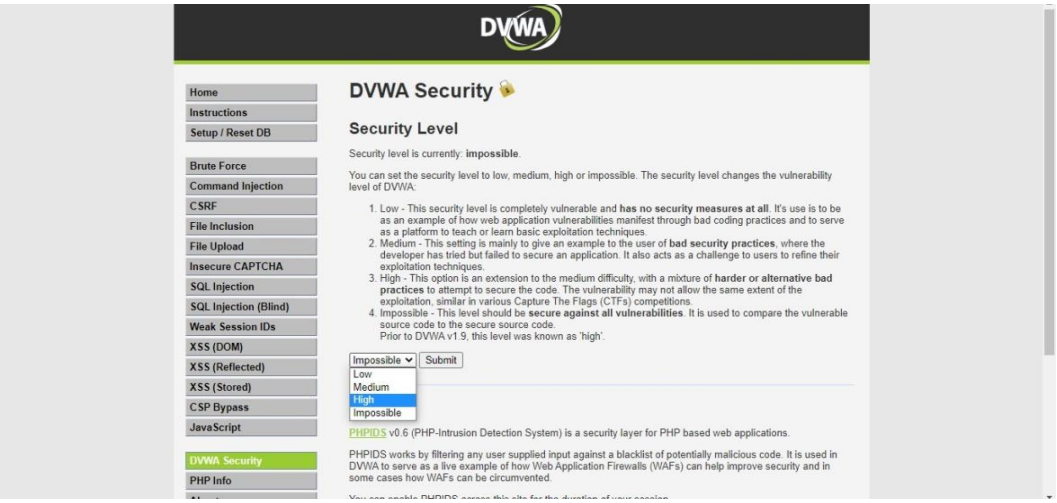
12) Now, You can see the folders and files those are inside your **htdocs folder**. Click on the **dvwa**. and you will the **dvwa setup page** as shown below :



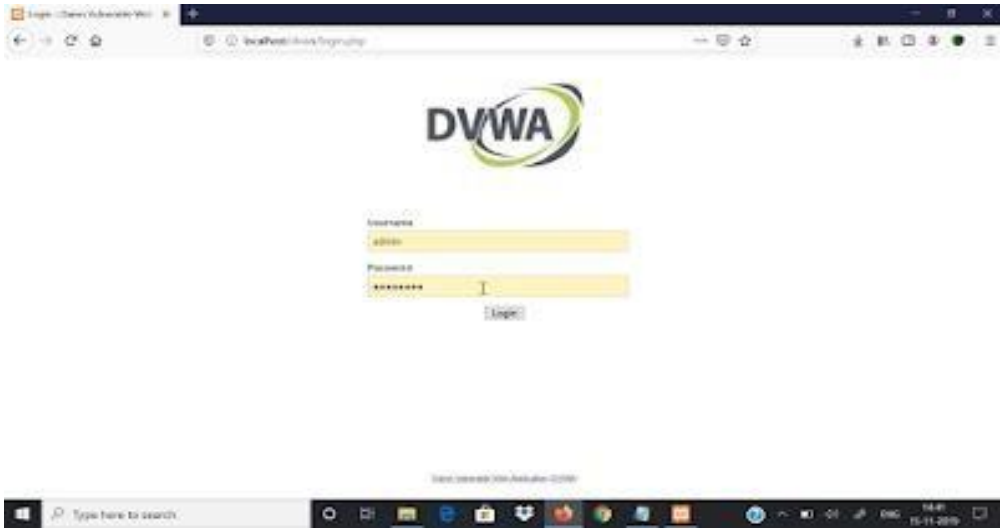
13) Scroll down the screen, You will a button **Create/Reset database.** Click on that button.



14) After clicking on that button, **Your database would be created**. Now, scroll down your screen You will see, **Setup successful. Please Login.** Click on **Login**.

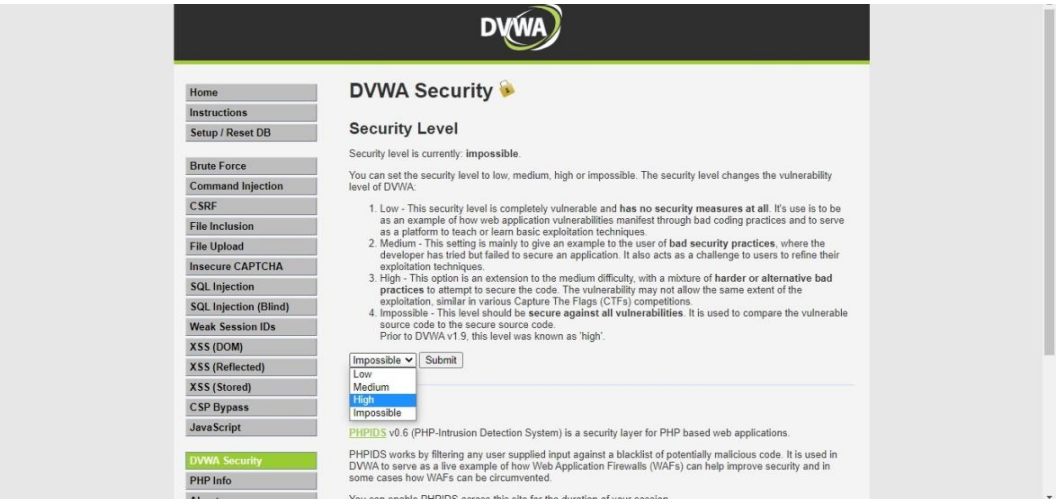


15) Now, You will see the **login panel** of **dvwa**. The default **username** is **admin** and **password** is **password**.



Congratulations. **DVWA** is successfully setup. Now You can test your hacking skills and attacks on your local server in which most vulnerabilities are very famous. So, you can test those vulnerabilities on your localhost in four security levels :

1) Low
2) Medium
3) High
4) Impossible

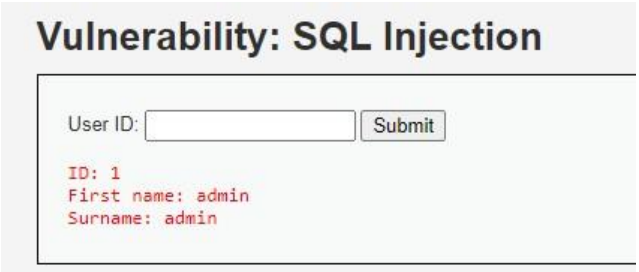


Instructions:

- Open Mozilla Firefox in you system.
- In URL bar, type localhost/dvwa/setup.php and then click on "Create/Reset Database"
- Again type localhost/dvwa . Press enter key.
- Now you should see DVWA login page. Credentials for login page are:
    - Username : admin
    - Password : password
- After logging in, go to “DVWA security” section. There is a drop down box on that page.
- You can set the security level of this web application accordingly. Set it “low” which is the least secure mode.
- Now move to SQL injection section, by clicking “sql injection” button on page. And try SQL injection there.

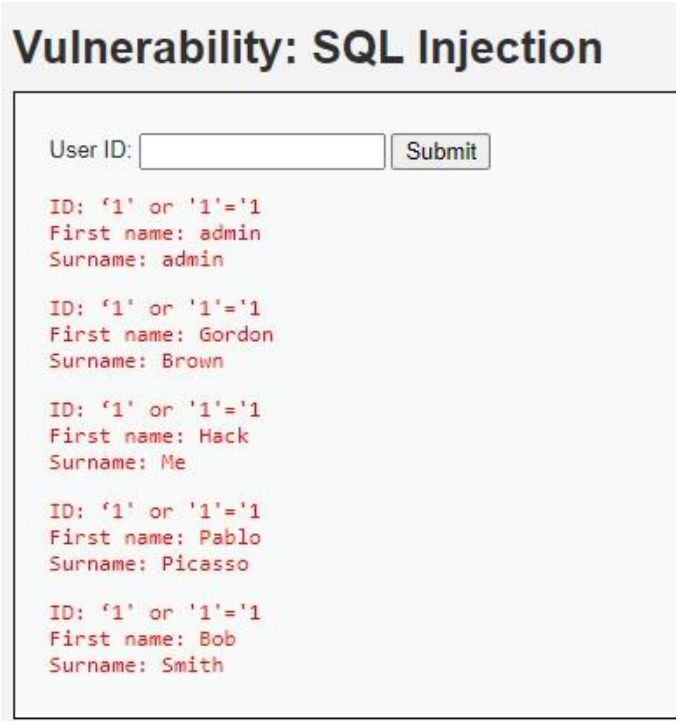
1. Check expected results:

SELECT first_name, last_name FROM users WHERE user_id = '1'



2. Check the results of an OR True statement

SELECT first_name, last_name FROM users WHERE user_id = ‘1' or '1'='1’



3. Find the number of columns in table:

SELECT first_name, last_name FROM users WHERE user_id = 'a' ORDER BY 1;#'

4. Find Hostname:

SELECT first_name, last_name FROM users WHERE user_id = ' ' union select null,@@hostname#'

5. Display File:

SELECT first_name, last_name FROM users WHERE user_id = ' ' union select

load_file('/etc/passwd'),null#'

6. Try to do SQL Injection on DVWA with security level set to "Medium" and find out all schemaname from database.

1 or 1=1 union select null, table_name from information_schema.tables#

This command does not work if we set Security level to "Medium". Since for medium level can accept only the values ranging from 1-5.

7. Try to do SQL Injection on DVWA with security level set to "Medium" and find out mysql version.

1 union select null,@@version#

This command does not work if we set Security level to "Medium". Since for medium level can accept only the values ranging from 1-5.

8. Analyze the source code for security on "low" and "medium" level. Comment on the ways by which SQL injection is prevented.

In low level, the vulnerable line is:

```
$query = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
```

Since the user input is jest concatenated to the command without being checked or sanitized, it
allows the user to pass arbitrary commands. But in case of medium level, the client-side is protected. Here passing arbitrary value is being prevented and a drop-down list is added. This certainly prevents the common user from entering arbitrary data and serves as good userexperience. However, an attacker knows how to interrupt and modify requests and can bypass this kind of protection easily.

RESULT:

Thus SQL injection attack has been demonstrated using Damn Vulnerable Web Application (DVWA).

Evaluation

Parameter	Max Marks	Marks Obtained
Originality of the work	30	
Completion of experiment on time	10	
Documentation	10	
Total	50	
Signature of the faculty with Date		

Ex.No.12	**Hashing Techniques**

AIM:

To implement SHA-1 for integrity check

To explore the usage of online calculators' for hash computation

THEORY:

Requirements of Hash Functions

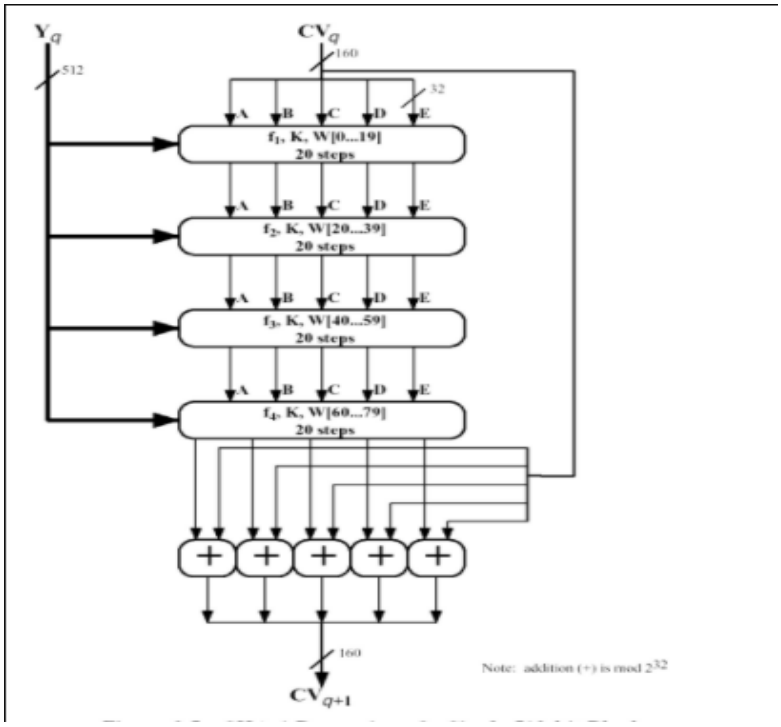
1. The input can be of any length.

2. The output has a fixed length of 160 bits.

3. H(x) is relatively easy to compute for any given x,

4. H(x) should be one-way function.

Block Diagram of One step of SHA-1:



Algorithm

- Initialize the buffer values.
- The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits.
- The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long.
- The padded message is then divided into 16 words each of size 32 bits
- Start the 80 iterations of sha-1.
- Calculate Was W(i)=S1(W(i−3)⊕W(i−8)⊕W(i−14)⊕W(i−16))
- Assign B = A
- C = S30(B)
- D = C
- E = D
- A =F ⊕ E ⊕ S5(A) ⊕Wt⊕ Kt

  Where K is

$$K(i) = 5A827999, \quad \text{where } 0 \le i \le 19$$
$$K(i) = 6ED9EBA1, \quad \text{where } 20 \le i \le 39$$
$$K(i) = 8F1BBCDC, \quad \text{where } 40 \le i \le 59$$
$$K(i) = CA62C1D6, \quad \text{where } 60 \le i \le 79.$$

  and F is

$$f(i;B,C,D) = (B \wedge C) \vee ((\neg B) \wedge D) \quad \text{for } 0 \ge i \ge 19$$
$$f(i;B,C,D) = B \oplus C \oplus D \quad \text{for } 20 \ge i \ge 39$$
$$f(i;B,C,D) = (B \wedge C) \vee (B \wedge D) \vee (C \wedge D) \quad \text{for } 40 \ge i \ge 59$$
$$f(i;B,C,D) = B \oplus C \oplus D \quad \text{for } 60 \ge i \ge 79.$$

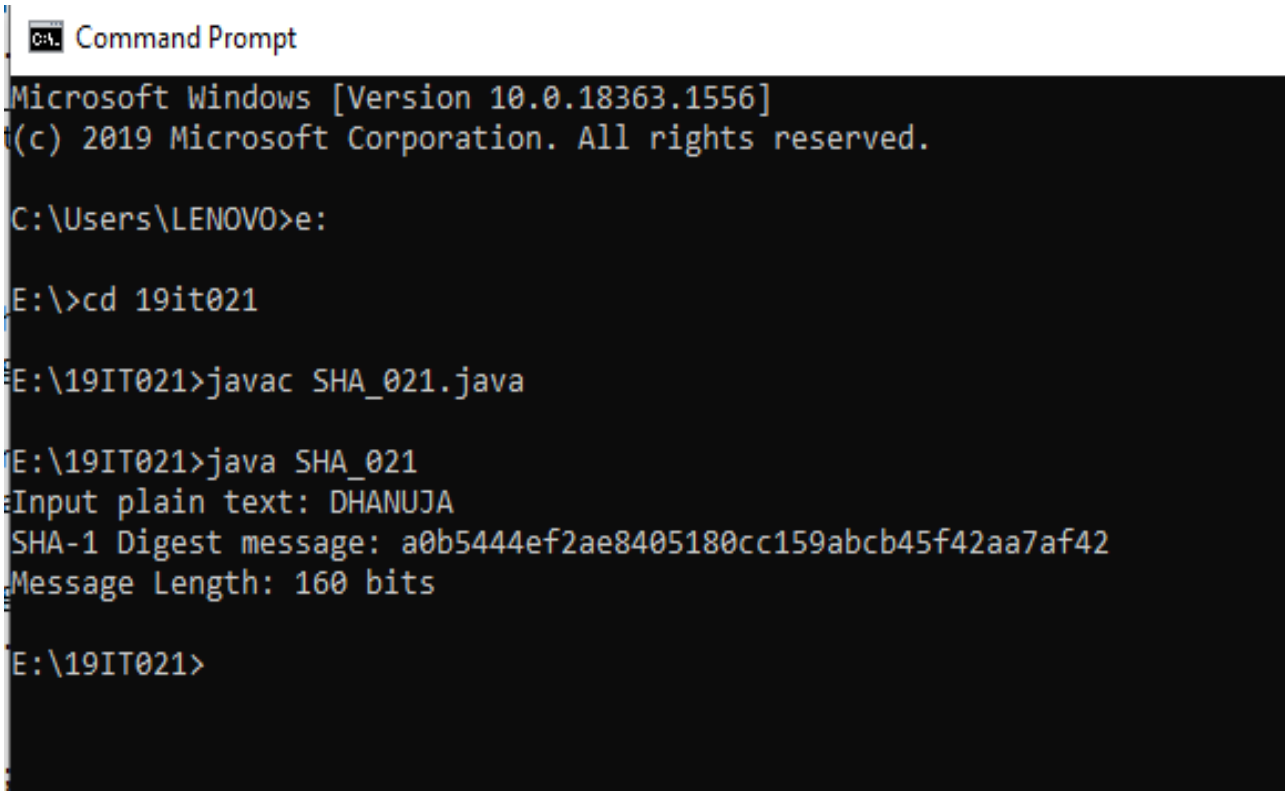
- Repeat above for 79 times and the final result is calculated as logically OR with the buffer values. This is the message digest

Coding

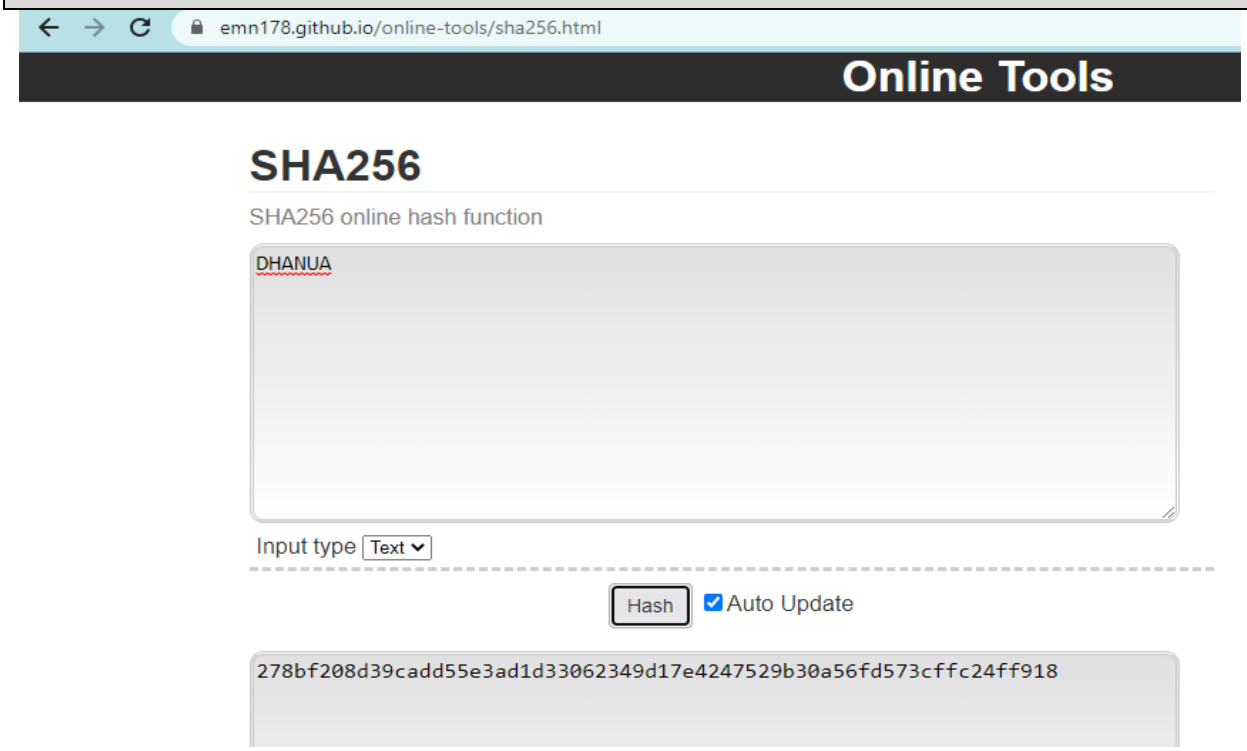
```
import java.math.BigInteger;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.*;
public class SHA_021 {
 public static String encryptThisString(String input)
 {
 try {
MessageDigest md = MessageDigest.getInstance("SHA-1");
 // digest() method is called to calculate message digest of the input string
byte[] messageDigest = md.digest(input.getBytes());
 // Convert byte array into signum representation
BigInteger no = new BigInteger(1, messageDigest);
 // Convert message digest into hex value
 String hashtext = no.toString(16);
 // Add preceding 0s to make it 32 bit
 while (hashtext.length() < 32) {
hashtext = "0" + hashtext;
 }
 return hashtext;
 }
 catch (NoSuchAlgorithmException e) {
 throw new RuntimeException(e);
 }
 }
```

```
 public static void main(String args[]) throws
NoSuchAlgorithmException
 {
 Scanner sc= new Scanner(System.in);
System.out.print("Input plain text: ");
 String s1= sc.nextLine();
 String res = encryptThisString(s1);
System.out.println("SHA-1 Digest message: "+res);
System.out.println("Message Length: " + res.length()*4 + " bits");
 }
}
```

Output



Hash Computation – Use of Online Calculator



RESULT:

Thus the program for one step of Secure Hash Algorithm -1 is implemented in Java and the results are verified.

Evaluation

Parameter	Max Marks	Marks Obtained
Originality of the code	25	
Hash Computation using Online Calculators	5	
Completion of experiment on time	10	
Documentation	10	
Total	50	
Signature of the faculty with Date		