|  |  |
| --- | --- |
| 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:**

The simplest monoalphabetic cipher is the additive cipher. This cipher is sometimes called a shift cipher and sometimes a Caesar cipher

**Brute Force Attack in Caesar Cipher:**

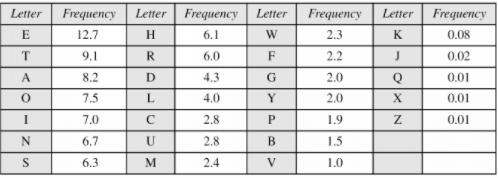
 The technique of trying every possible decryption key is called a **brute-force attack**

**Frequency Analysis Attack in Caesar Cipher:**

Frequency Analysis attack is performed based on the **frequency** of letters or groups of letters in a ciphertext.

The letter 'e' is the most frequent (about 12% of all letters in the book), followed by 't', and so on

**Frequency of Character in English:**



|  |
| --- |
| **ALGORITHM:** |

1. Start.
2. Define 5 methods : encrypt,decrypt,maxoccuringchar,bruteforce, frequencyanalysis.
3. Main method receives input and key for encryption and decryption.
4. Encryption is done shifting characters using the key and vice versa for decryption.
5. The position equivalence for the plain text is added with key value and the corresponding position equivalent character is displayed as cipher text.
6. The key is subtracted from the position equivalence of the cipher text and the corresponding position equivalent character is displayed as plain text.
7. Brute force : The input is cipher text to be decrypted and the decryption method is used with all possible keys and displays all output.
8. Frequency analysis: The input is cipher text and the most occurring cipher letter is found. Its position equivalence is subtracted from the position equivalence of the most occurring letter in alphabets to get the key. Using the key, the plain text is obtained , if it is not meaningful then the next most occurring letter in alphabet is taken for consideration.
9. End.

|  |
| --- |
| **CODING:** |

**Ceaser Cipher:**

package com.yajith.cipher;  
  
import java.util.Dictionary;  
import java.util.Hashtable;  
import java.util.Scanner;  
  
  
public class Main {  
 private static String *cipher*;  
 private static int *choice*,*j*;  
 public static void main(String[] args) {  
 Scanner s=new Scanner(System.*in*);  
 System.*out*.println("1.encrypt\n2.decrypt");  
 *choice*=s.nextInt();  
 System.*out*.println("Enter the key");  
 *j*=s.nextInt();  
 if(*choice*==1)  
 {  
 Scanner s1=new Scanner(System.*in*);  
 System.*out*.println("Enter The Plain Text:");  
 *cipher*=s1.nextLine();  
 *cipher*=*cipher*.toLowerCase();  
 for (int i = 0; i < *cipher*.length(); i++) {  
 int a = (int) *cipher*.charAt(i);  
 if (a == 32) {  
 System.*out*.print(" ");  
 continue;  
 }  
 char c = (char) ( (( (a + *j*) - 97) % 26) + 97);  
 c = Character.*toUpperCase*(c);  
 System.*out*.print(c);  
 }  
 System.*out*.println();  
 }  
 else if(*choice*==2) {  
 Scanner s2=new Scanner(System.*in*);  
 System.*out*.println("Enter The Cipher Text:");  
 *cipher*=s2.nextLine();  
 *cipher*=*cipher*.toLowerCase();  
 System.*out*.println("The Plain Text are");  
 for (int i = 0; i < *cipher*.length(); i++) {  
 int a = (int) *cipher*.charAt(i);  
 if (a == 32) {  
 System.*out*.print(" ");  
 continue;  
 }  
 char c = (char) ( ( ((a - *j*) - 97) % 26) + 97);  
 System.*out*.print(c);  
 }  
 System.*out*.println();  
 }  
 else  
 {  
 System.*out*.println("Wrong Choice");  
 }  
 }  
}

**Brute Force Attack:**

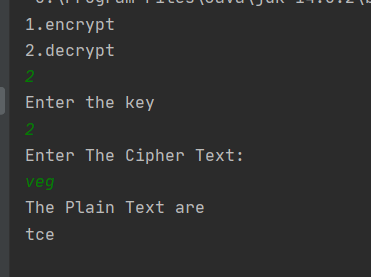
package com.yajith.cipher;  
  
import java.util.Dictionary;  
import java.util.Hashtable;  
import java.util.Scanner;  
  
  
public class Main {  
 private static String *cipher*;  
 private static int *choice*;  
 public static void main(String[] args) {  
 Scanner s=new Scanner(System.*in*);  
 System.*out*.println("1.encrypt\n2.decrypt");  
 *choice*=s.nextInt();  
 if(*choice*==1)  
 {  
 Scanner s1=new Scanner(System.*in*);  
 System.*out*.println("Enter The Plain Text:");  
 *cipher*=s1.nextLine();  
 *cipher*=*cipher*.toLowerCase();  
 for(int j=0;j<26;j++) {  
 System.*out*.print(j+" ");  
 for (int i = 0; i < *cipher*.length(); i++) {  
 int a = (int) *cipher*.charAt(i);  
 if (a == 32) {  
 System.*out*.print(" ");  
 continue;  
 }  
 char c = (char) ( (( (a + j) - 97) % 26) + 97);  
 c = Character.*toUpperCase*(c);  
 System.*out*.print(c);  
 }  
 System.*out*.println();  
 }  
 }  
 else if(*choice*==2) {  
 Scanner s2=new Scanner(System.*in*);  
 System.*out*.println("Enter The Cipher Text:");  
 *cipher*=s2.nextLine();  
 *cipher*=*cipher*.toLowerCase();  
 System.*out*.println("The Plain Text are");  
 for(int j=0;j<26;j++) {  
 System.*out*.print(j+" ");  
 for (int i = 0; i < *cipher*.length(); i++) {  
 int a = (int) *cipher*.charAt(i);  
 if (a == 32) {  
 System.*out*.print(" ");  
 continue;  
 }  
 char c = (char) ( ( ((a - j) - 97) % 26) + 97);  
 System.*out*.print(c);  
 }  
 System.*out*.println();  
  
 }  
 }  
 else  
 {  
 System.*out*.println("Wrong Choice");  
 }  
 }  
}

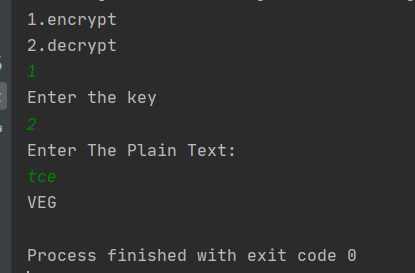
**Frequency Analysis:**

package com.yajith.frequency;  
  
import java.util.Scanner;  
  
public class Main {  
 private static String *input*;  
 public static void main(String[] args) {  
 Scanner scanner=new Scanner(System.*in*);  
 *input*=scanner.nextLine();  
 String cipherText = "";  
 String ALPHABET = "abcdefghijklmnopqrstuvwxyz";  
 cipherText = *input*;  
 char freq = *getMaxOccuringChar*(cipherText);  
 System.*out*.println("The most occurring cipher letter is:"+ freq);  
 System.*out*.println("Enter the plain letter corresponding to " + freq + ": ");  
 char p = scanner.next().charAt(0);  
 int cpos = ALPHABET.indexOf(freq);  
 int ppos = ALPHABET.indexOf(p);  
 int keyVal = (cpos - ppos) % 26;  
 System.*out*.println("Key value: " + (keyVal\*-1));  
 *decrypt*(cipherText);  
 }  
 static void decrypt(String text){  
 System.*out*.println("Key: ");  
 int key = new Scanner(System.*in*).nextInt();  
 char chars[] = text.toCharArray();  
 int i = 0;  
 while(i<chars.length){  
 chars[i] = Character.*toLowerCase*(chars[i]);  
 int val = ((int)chars[i]-key);  
 val = (val >= 97)?val:121-(97-val);  
 chars[i] = (char)val;  
 System.*out*.print(Character.*toLowerCase*(chars[i]));  
 i++;  
 }  
 System.*out*.println('\n');  
 }  
 static char getMaxOccuringChar(String str) {  
 // Create array to keep the count of individual  
 // characters and initialize the array as 0  
 int count[] = new int[256];  
  
 // Construct character count array from the input  
 // string.  
 int len = str.length();  
 for (int i = 0; i < len; i++)  
 count[str.charAt(i)]++;  
  
 int max = -1; // Initialize max count  
 char result = ' '; // Initialize result  
  
 // Traversing through the string and maintaining  
 // the count of each character  
 for (int i = 0; i < len; i++) {  
 if (max < count[str.charAt(i)]) {  
 max = count[str.charAt(i)];  
 result = str.charAt(i);  
 }  
 }  
  
 return result;  
 }  
  
}

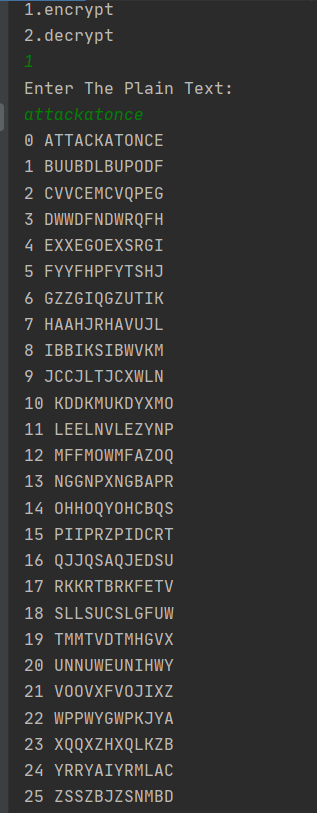
|  |
| --- |
| **SCREEN SHOTS:** |

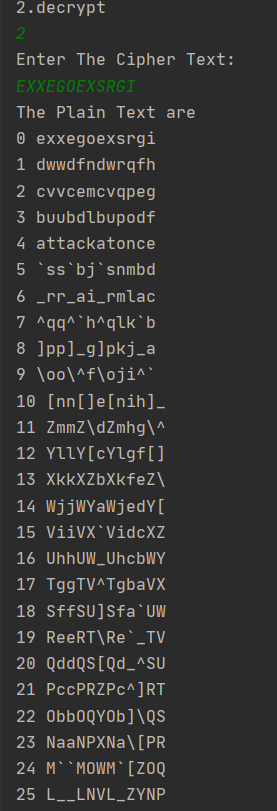
**Ceaser Cipher:**



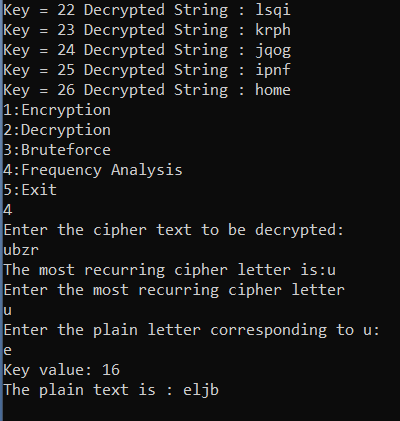


**Brute Force attack:**





**Frequency Analysis:**



|  |
| --- |
| **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**

To encrypt a message using the Hill Cipher we must first turn our keyword into a key matrix (a 2 x 2 matrix for working with digraphs, a 3 x 3 matrix for working with trigraphs, etc). We also turn the plaintext into digraphs (or trigraphs) and each of these into a column vector.

**Decryption**

To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26).

**Constraints for selecting Key Matrix in Hill Cipher**

If we work modulo 26 as above, the determinant must be nonzero, and must not be divisible by 2 or 13. If the determinant is 0, or has common factors with the modular base, then the matrix cannot be used in the Hill cipher, and another matrix must be chosen (otherwise it will not be possible to decrypt).

**Steps for calculation of inverse of Key Matrix in Modulo arithmetic**

1. First, compute the determinant of the matrix,detA. If detA is coprime to m, then you can be sure that A is invertible mod m.
2. Next, compute the matrix of cofactors of A, call this B.
3. The matrix (detA)−1×B is an inverse to A modulo m.

**Euclidean and Extended Euclidean algorithm**

The Euclidean algorithm is a way to find the greatest common divisor of two positive integers, a and b

Extended Euclidean algorithm also refers to a very similar algorithm for computing the polynomial greatest common divisor and the coefficients of Bézout's identity of two univariate polynomials. The extended Euclidean algorithm is particularly useful when a and b are coprime.

**Known Plaintext and Cipher text attack with an example**

During known-plaintext attacks, the attacker has an access to the ciphertext and its corresponding plaintext. His goal is to guess the secret key (or a number of secret keys) or to develop an algorithm which would allow him to decrypt any further messages. This gives the attacker much bigger possibilities to break the cipher than just by performing ciphertext only attacks. However, he is no able to actively provide customized data or secret keys which would be processed by the cipher.

|  |
| --- |
| **ALGORITHM:** |

**Encryption:**

Step1: Start

Step 2: Get the order of the key matrix.

Step 3: Get the elements of the key matrix.

Step 4: Get the Plain text to be encoded

Step 5:Perform number encoding for given plain text and give it as input for matrix P.

Step 6:Perform matrix multiplication between K (key) and P.

Step 7:Do modulo m with all element of matrix.

Step 8:Perform character encoding and it results in cipher text.

**Decryption:**

Step1: Start

Step 2: Get the order of the key matrix.

Step 3: Get the elements of the key matrix.

Step 4: Get the Cipher text to be decoded

Step 5:Perform number encoding for given cipher text and give it as input for matrix C.

Step 6:Find determinant of the key matrix and then do modulo.

Step 7:Next find the determinant inverse and calculate the adjoint of the key (K) matrix.

Step 8:Calculate inverse of key (K) matrix.

Step 9:Perform matrix multiplication between inverse of key (K) and C.Step 10:Do modulo m with all element of matrix.

Step 11:Perform character encoding and it results in plain text.

**Known Plaintext attack and Cipher text attack**

Step 1:Get plain text and cipher text as inputs

Step2:Perform number encoding.

Step 3:Get their order of matrix respectively.

Step 4:Perform matrix multiplication between inverse of C and P.

Step 5:Do modulo m with all element of matrix.

Step 6:The resultant matrix is the key inverse.

|  |
| --- |
| **DESIGN** |

**List of classes:**

1. HillCipher
2. Main

**List of methods:**

1. Determinant(array,size) - to find the determinant of matrix.
2. inverse(key,size) -to find inverse of the matrix.
3. chartonum(pt, key) - to convert characters to it’sasciiform
4. decode(array, order) - to convert numbers to characters.
5. getCofactor(array , order) - to get the cofactor of the respective matrix.
6. adjoint(array,size) - to calculate adjoint of the matrix.
7. Determinant(array,size) - to find the determinant of matrix.
8. matmul(array) - to do matrix multiplication between two matrices.
9. inverse(key,size) -to find inverse of the matrix.
10. encrypt(array,size) - to encrypt plain text to cipher text.
11. decrpypt(array,size) - to decrypt cipher text to plain text.
12. knownptct(array,size) - to perform known pt ct.

|  |
| --- |
| **TASK DISTRIBUTION** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Reg No** | **Name** | **Task** | **Support from Group Members** | **Learning** |
| 18IT079 | G.L. Sankara Subramaniyan | Determinant, Inverse of matrix, encrypt, decrypt | Cross validated code | Out of bound exception avoidance and findings |
| 18IT116 | S. Yajith Vishwa | Matrix Multiplication, cofactor, adjoint | Cross validated code | Splitting matrix for adjoint calculation |

|  |
| --- |
| **MODULE WISE CODING AND UNIT TESTING** |

**1)Determinant:**

public static void main(String args[])

{

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

{

int D = 0;

if (n == 1)

return mat[0][0];

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;

}

**2)Inverse of matrix**

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];

}

}

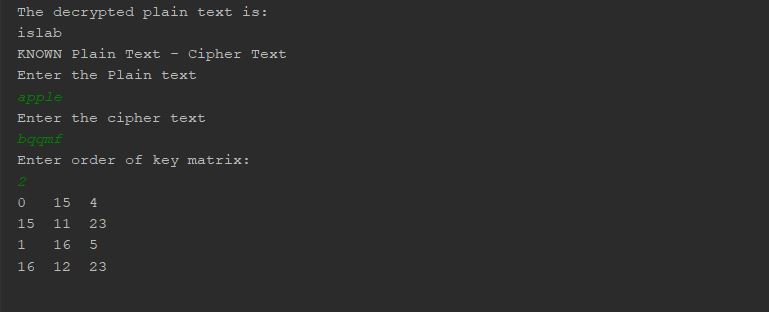
}

return k;

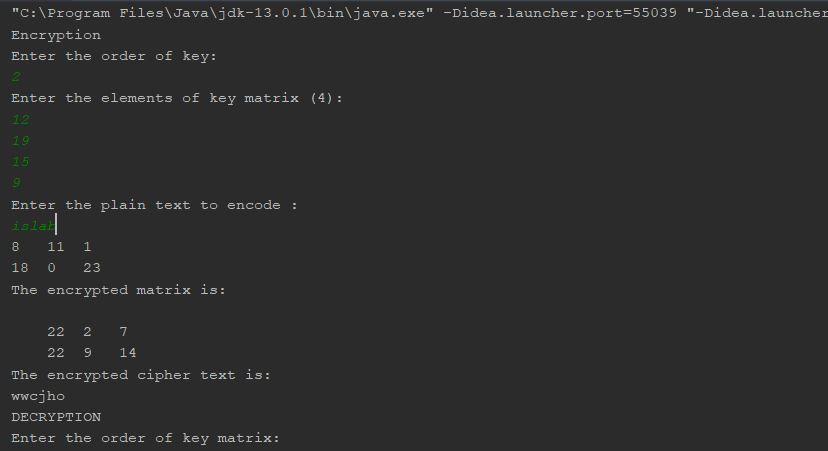
}

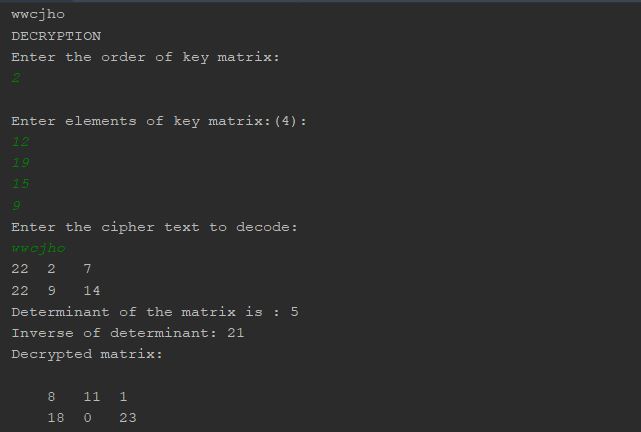
|  |
| --- |
| **INTEGRATED CODE AND TESTING** |

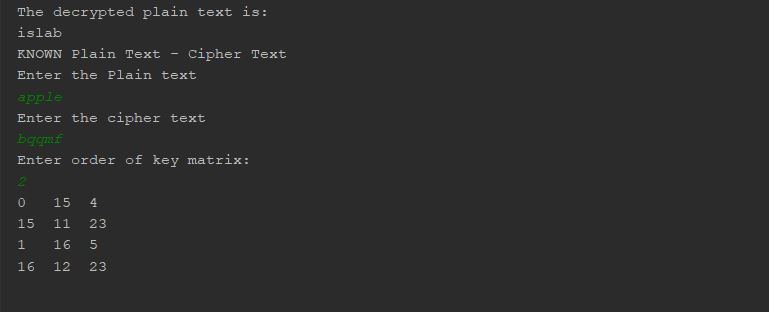
import java.util.\*;  
class Hillcipher  
{  
 static final String *ALPHABET*="abcdefghijklmnopqrstuvwxyz";  
 int[][] numencode(String ptext,int key)  
 {  
 int row=0;  
 int col=0;  
 int res=0;  
 int extracol=0;  
 int k=0;  
 int i=0;  
 int j=0;  
 int col1=0;  
 String alpha="abcdefghijklmnopqrstuvwxyz";  
 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="";  
  
 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*.print(p);  
 }  
 void decode1(int c[][],int len)  
 {  
 String p="";  
 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++;  
 }  
 }  
 }  
 }  
 }  
  
 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;  
 }  
}  
  
  
class Execution{  
  
 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("\nDECRYPTION");  
 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);  
 }  
}

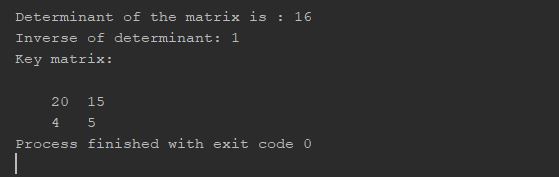
****

|  |
| --- |
| **SCREEN SHOTS:** |

****

****

****

****

|  |
| --- |
| **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, unambiguous naming). | 3.0 pts  Good  • Includes name, date, and assignment title. • Good use of white space.  • Organized work.  • Good use of variables (no global variables, unambiguous naming) | 2.0 pts  Satisfactory  • Completed between 70-80% of the requirements.  • Delivered on time, and in correct format (disk, email, etc.) global variables, unambiguous naming). | 0.0 pts  Unsatisfactory  • Completed less than 70% of the requirements.  • Not delivered on time or not in correct format (disk, email, etc.) | |  |
| 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. | |  |
| 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 | **DEMONSTRATION OF TRIPLE DES AND BLOCK CIPHER MODES OF OPERATION** |

|  |
| --- |
| **AIM:** |

To demonstrate the working principle of Triple DES and Block cipher modes of operation

|  |
| --- |
| **THEORY:** |

**Meet in the middle attack**

The meet-in-the-middle attack (MITM), a known plaintext attack, is a generic space–time trade off cryptographic attack against encryption schemes that rely on performing multiple encryption operations in sequence. The MITM attack is the primary reason why Double Des is not used and why a Triple Des key (168-bit) can be brute forced by an attacker with 256 space and 2112 operations.

**Triple DES Architecture**

The Triple Data Encryption Algorithm is a symmetric key block cipher, which applies the DES cipher algorithm three times to each data block. The Data Encryption Standard's (DES) 56-bit key is no longer considered adequate in the face of modern cryptanalytic techniques and supercomputing power. However, an adapted version of DES, Triple DES (3DES), uses the same algorithm to produce a more secure encryption.

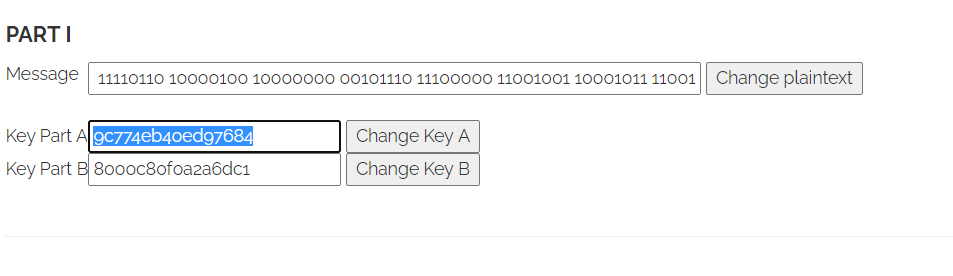
**Block Cipher Modes of Operation**

A block cipher is a deterministic algorithm operating on fixed-length groups of bits, called blocks. It uses an unvarying transformation, that is, it uses a symmetric key. They are specified elementary components in the design of many cryptographic protocol and are widely used to implement the encryption of large amounts of data, including data exchange protocols.

|  |
| --- |
| **DEMONSTRATION IN VIRTUAL LABS** |

**Triple DES:**

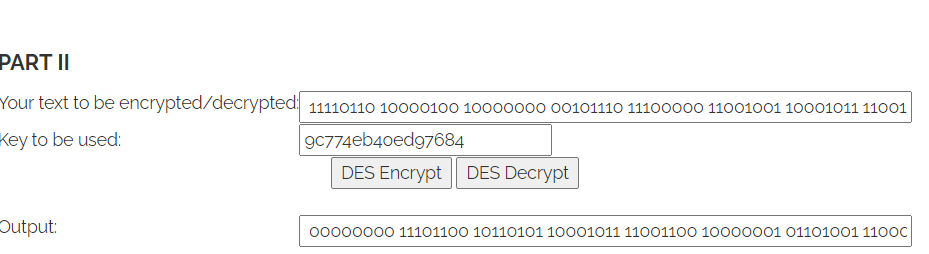
1. Generate Plain text, key1 and key2



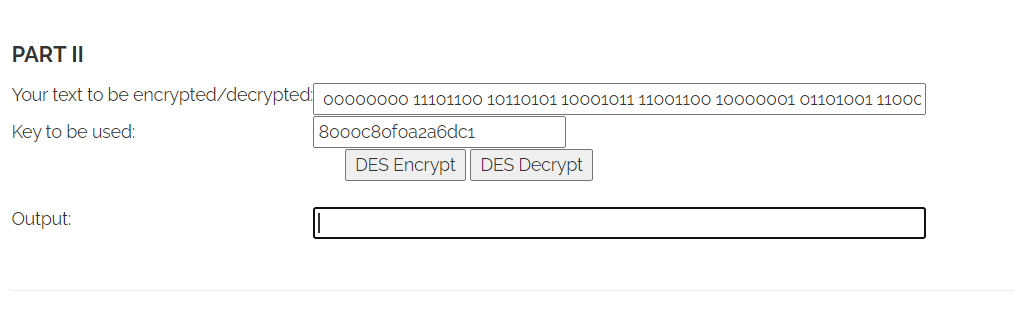
1. Use key1 and do encrypt we get



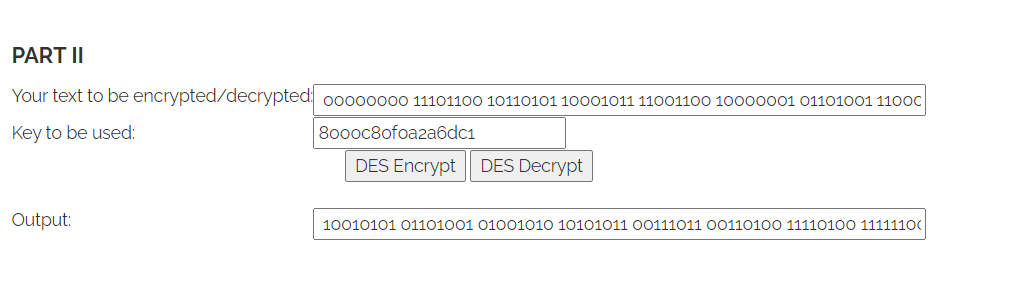
1. Click encrypt



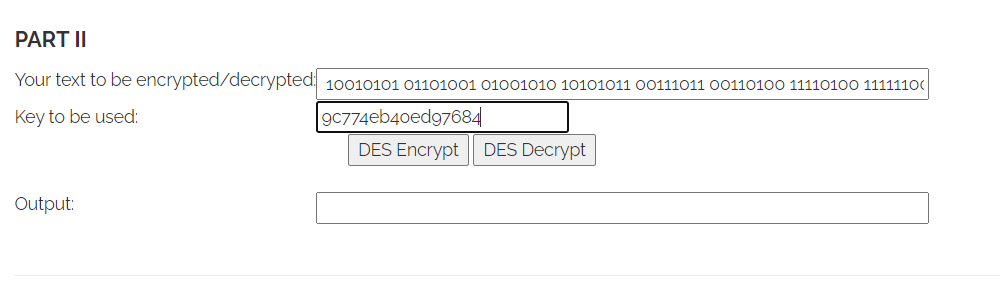
1. Enter cipher text 1 in your text to be encrypted/decrypted textbox and key 2 is used



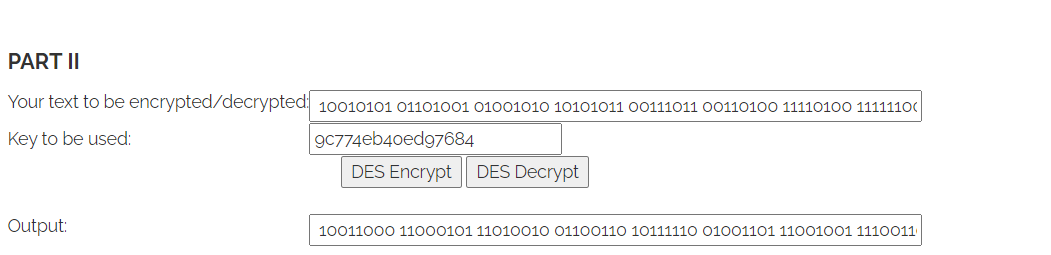
1. Click decrypt



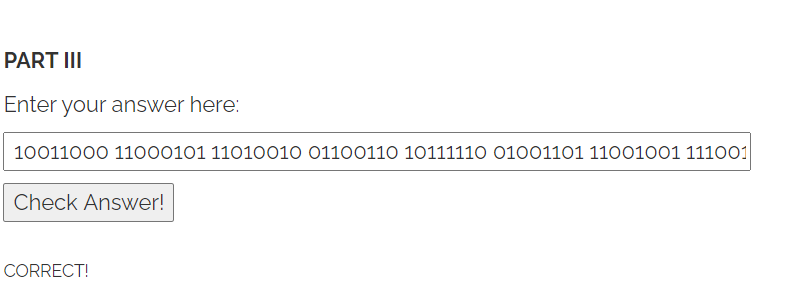
1. Enter cipher text 2 and key 1



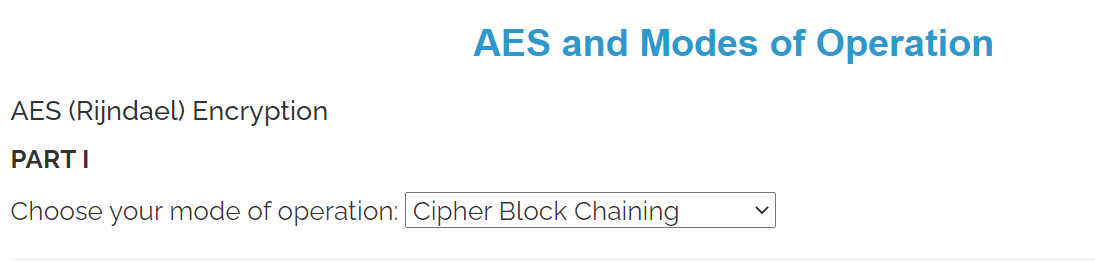
1. Click Encrypt



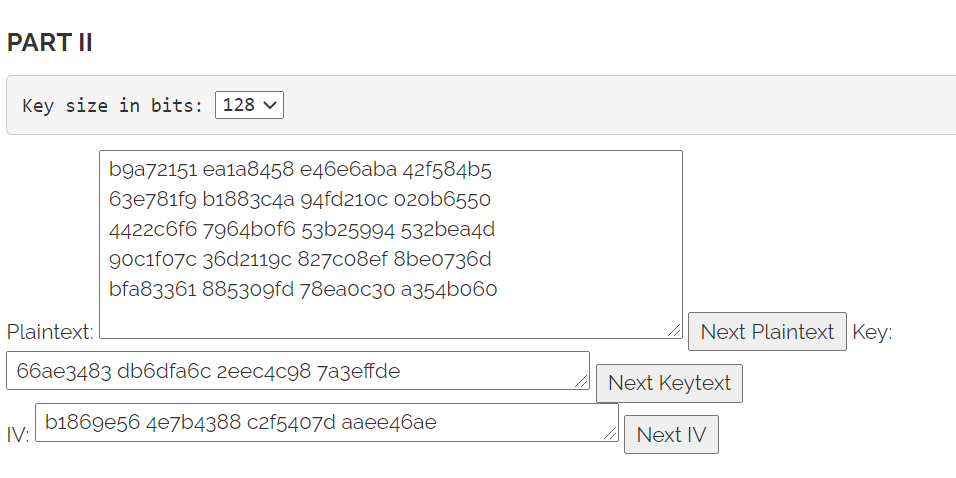
1. Check the answer



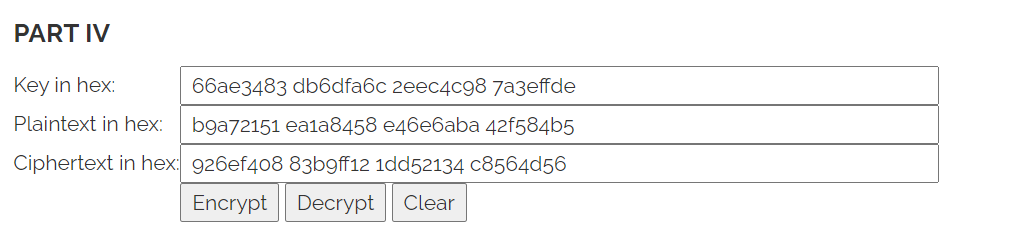
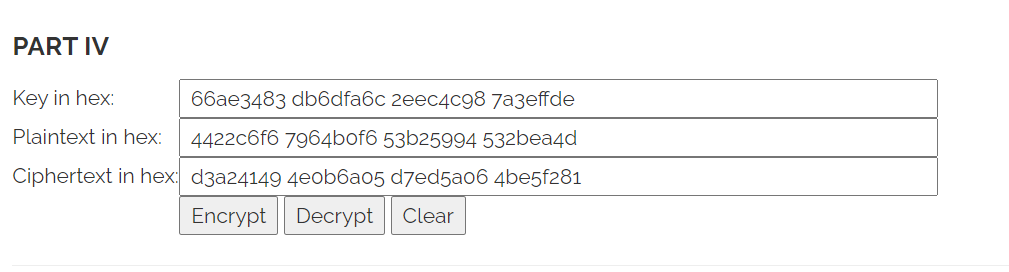
**Block Cipher:**

****

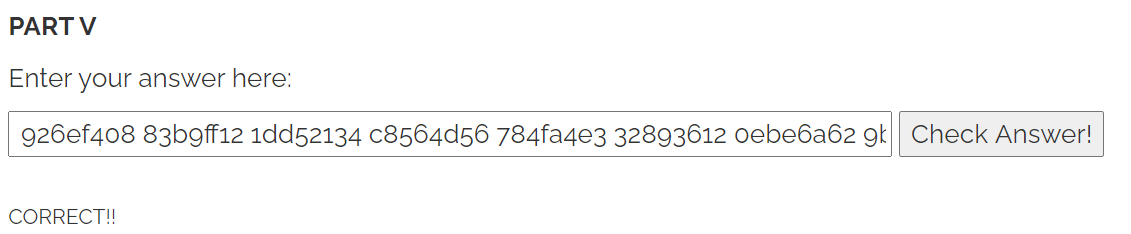
**Step 2 :**

****

**Step 3 :**

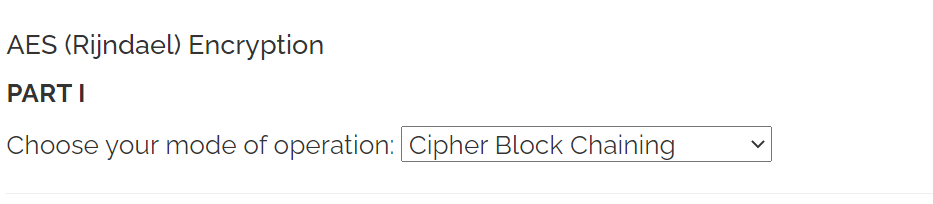
**    **

**Step 4 :**

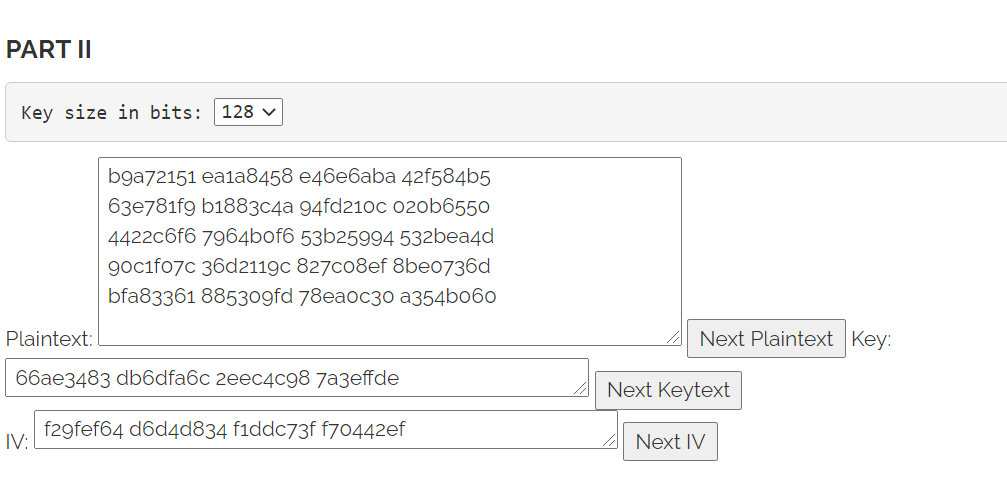
****

**Code Block Chaining :**

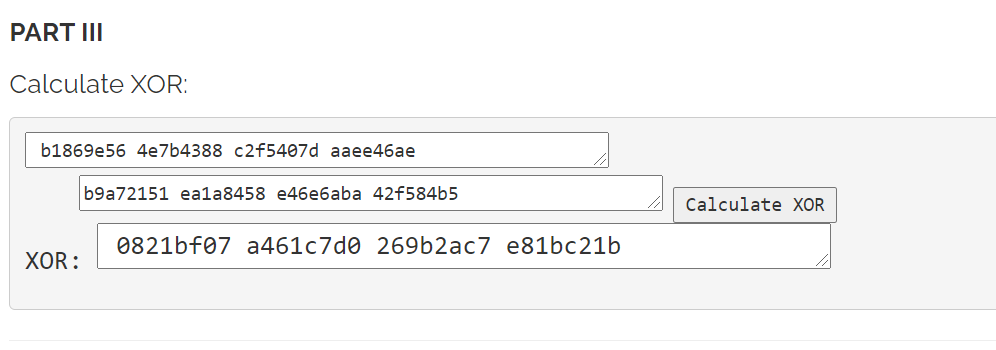
**Step 1:**

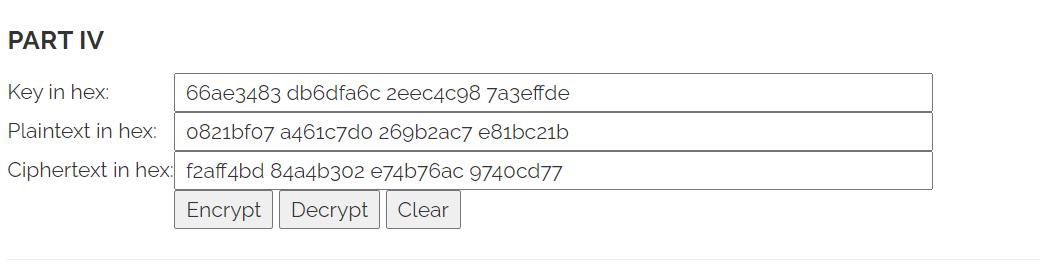
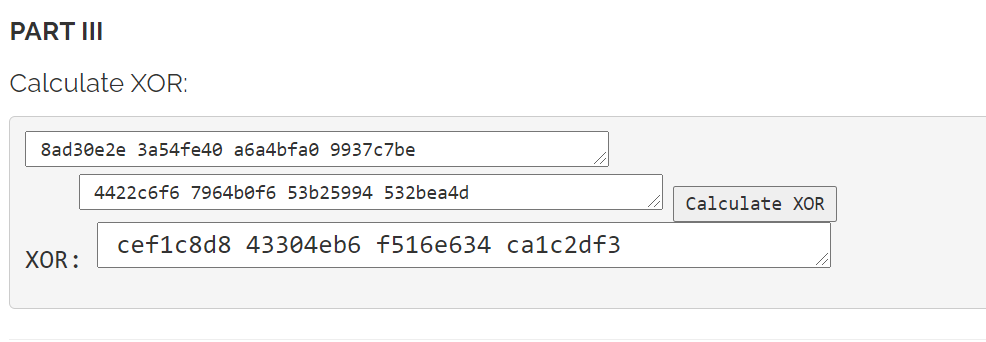
****

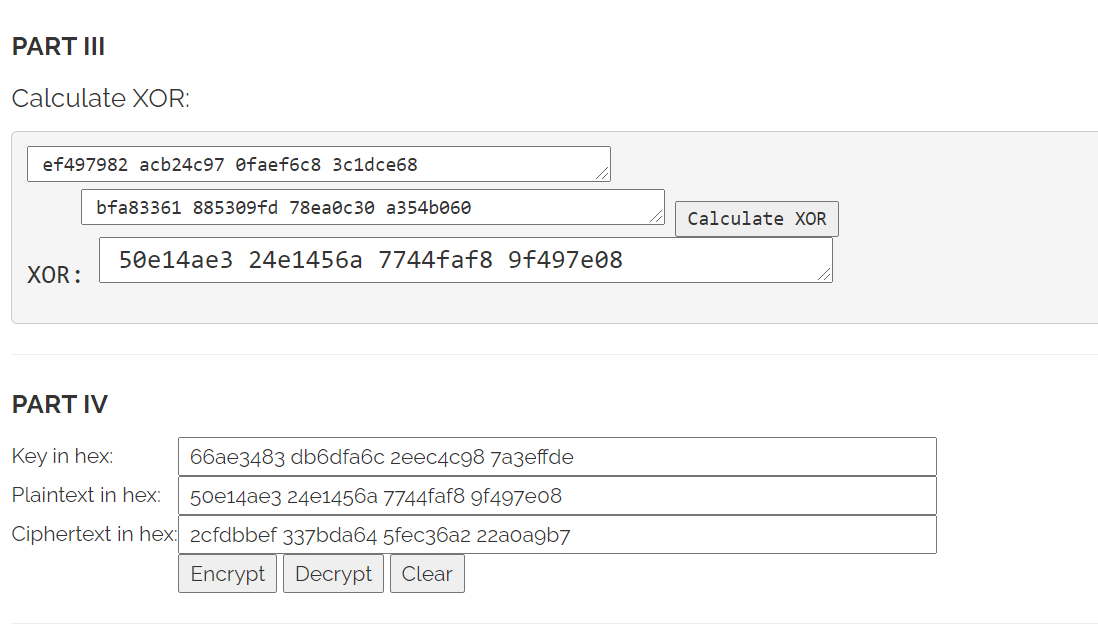
**Step 2 :**

****

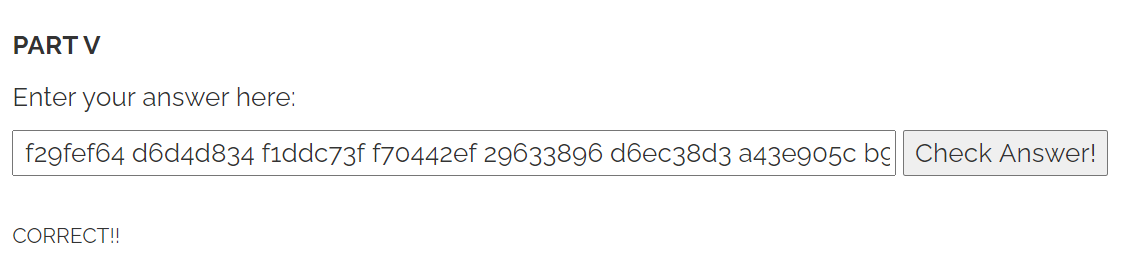
**Step 3:**

****

**    **

****

**Step 4 :**

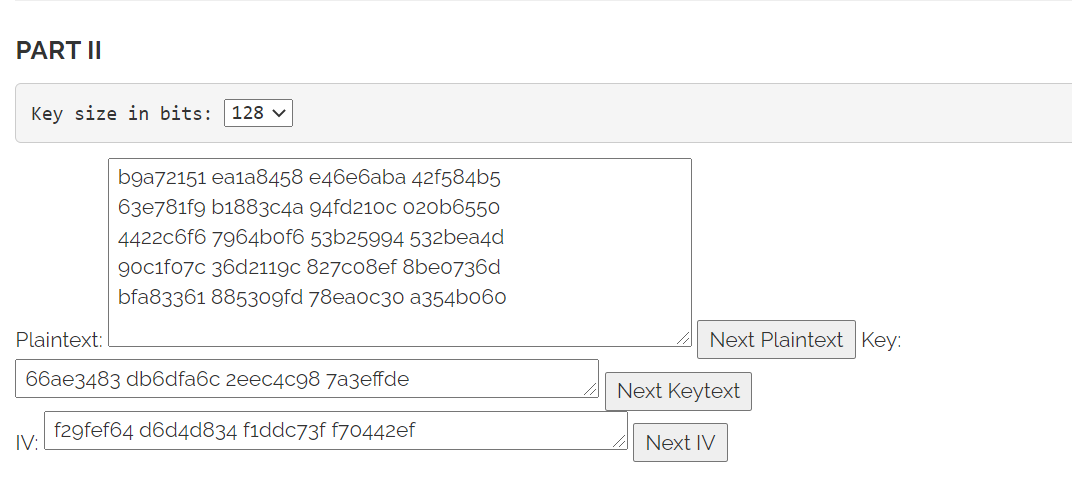
****

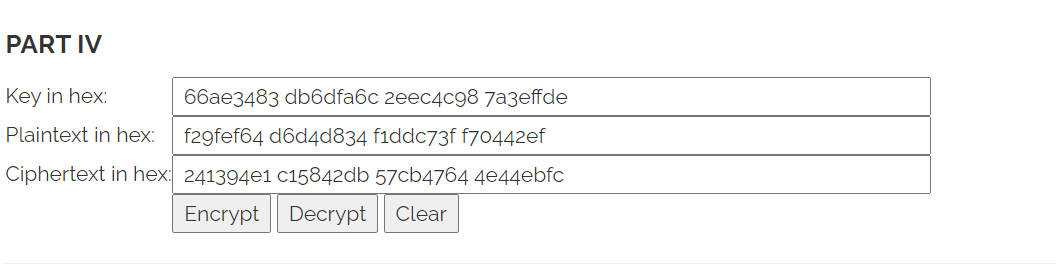
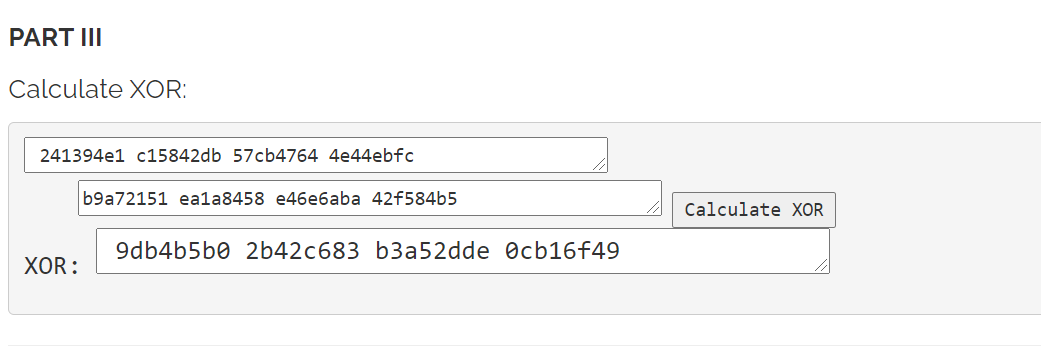
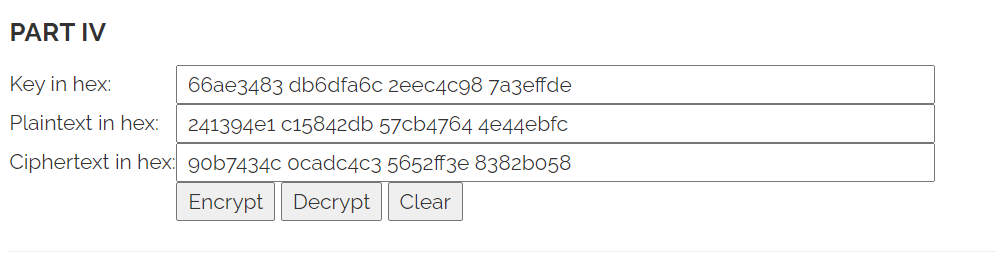
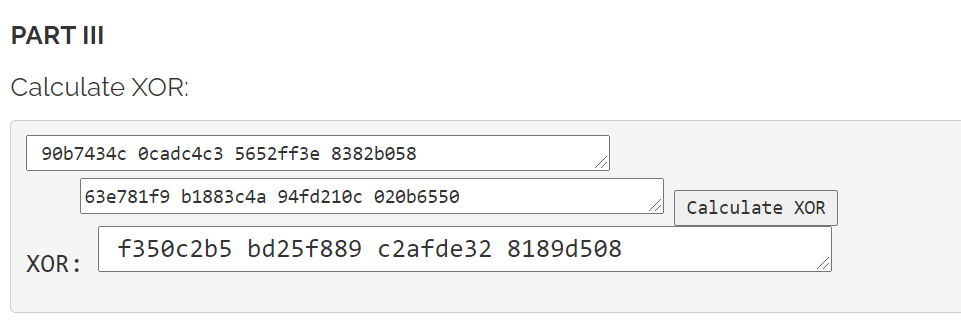
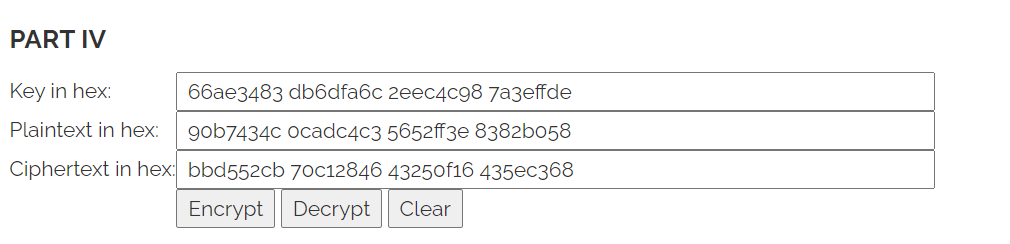
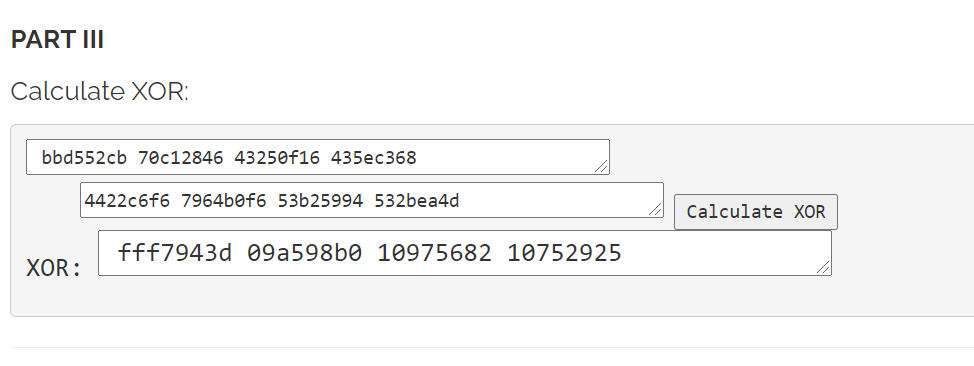
**Output Feedback :**

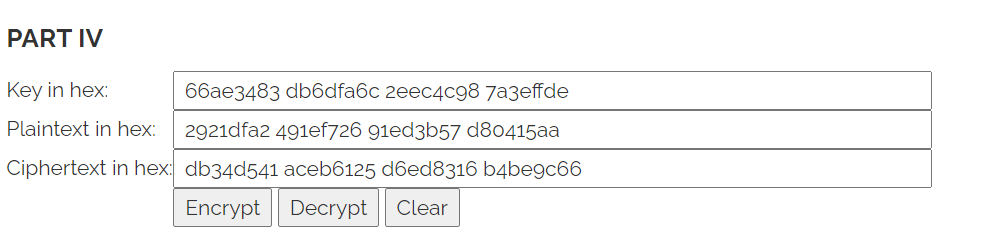
**Step 1 :**

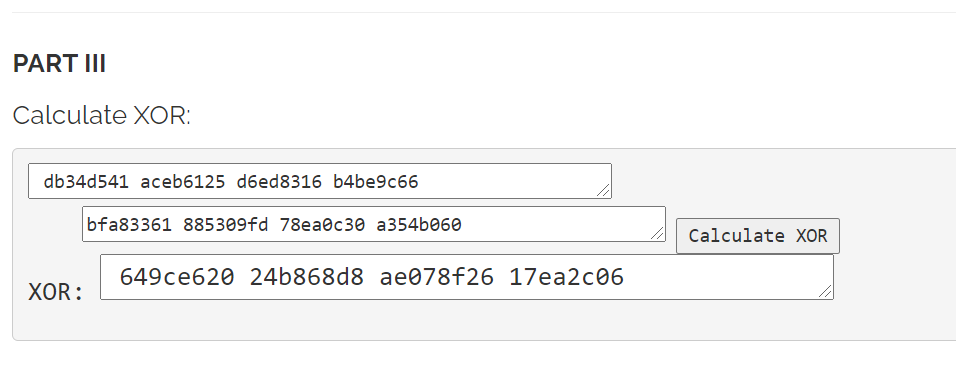
****

**Step 2 :**

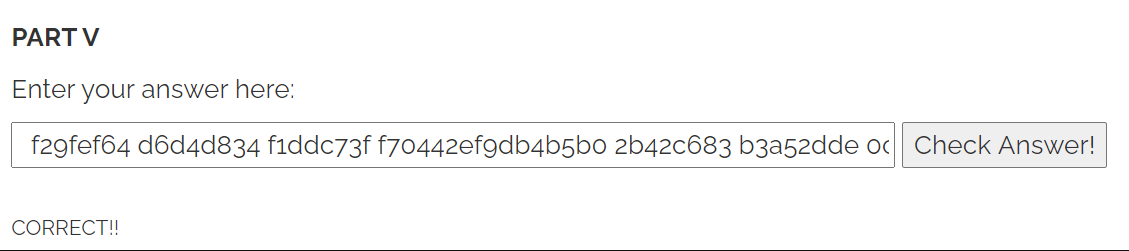
****

**Step 3 :     **

****

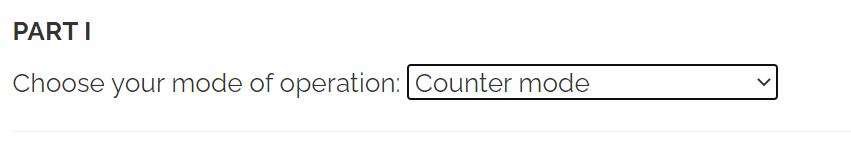
****

**Step 4:**

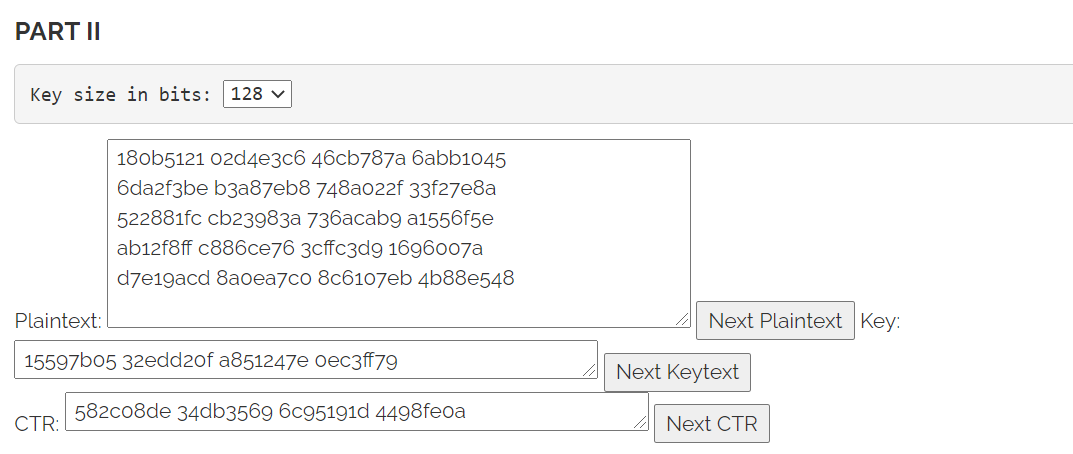
****

**Counter Mode :**

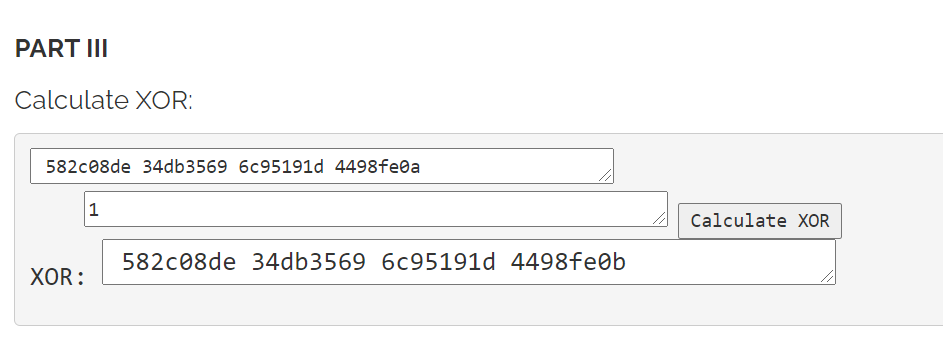
**Step 1 :**

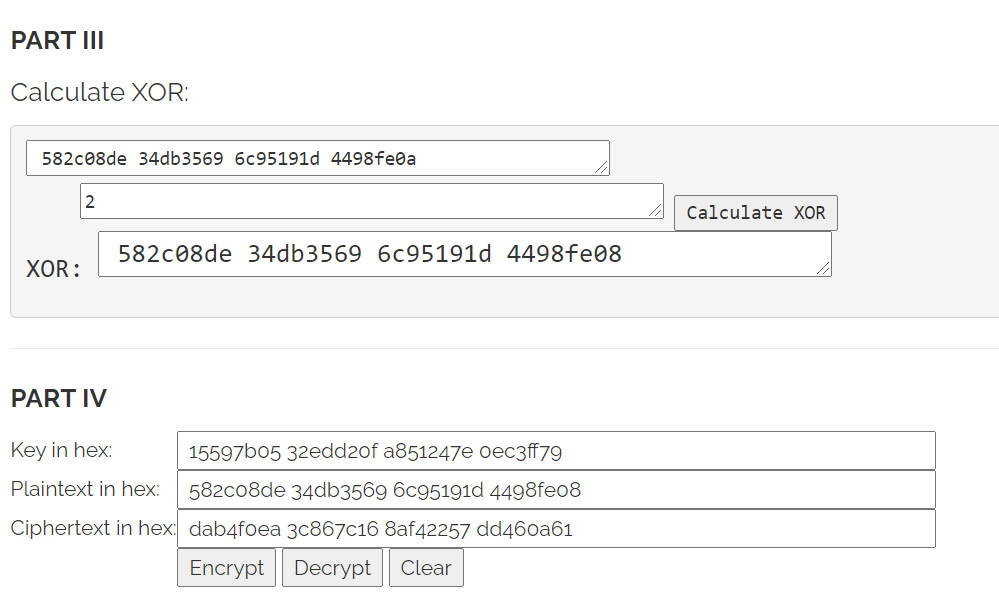
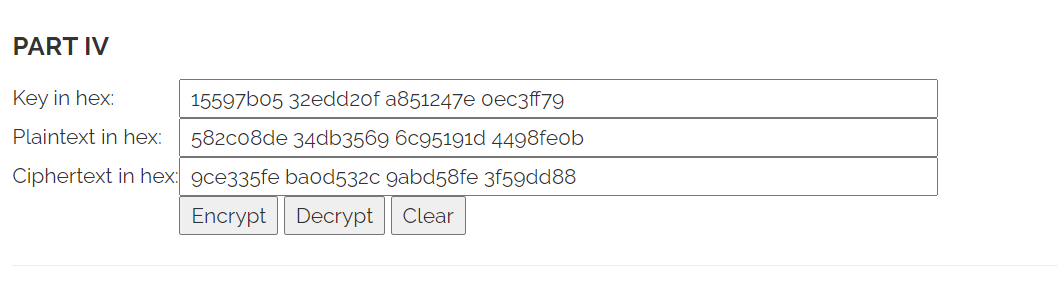
****

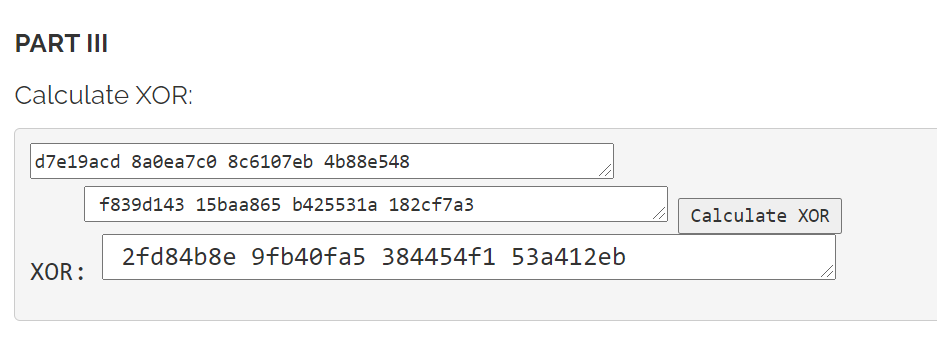
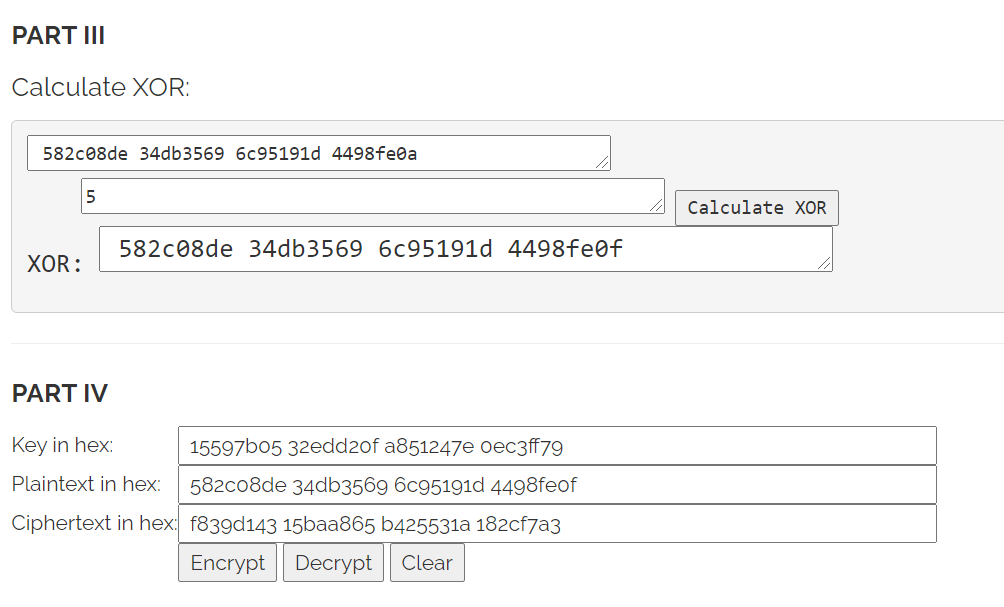
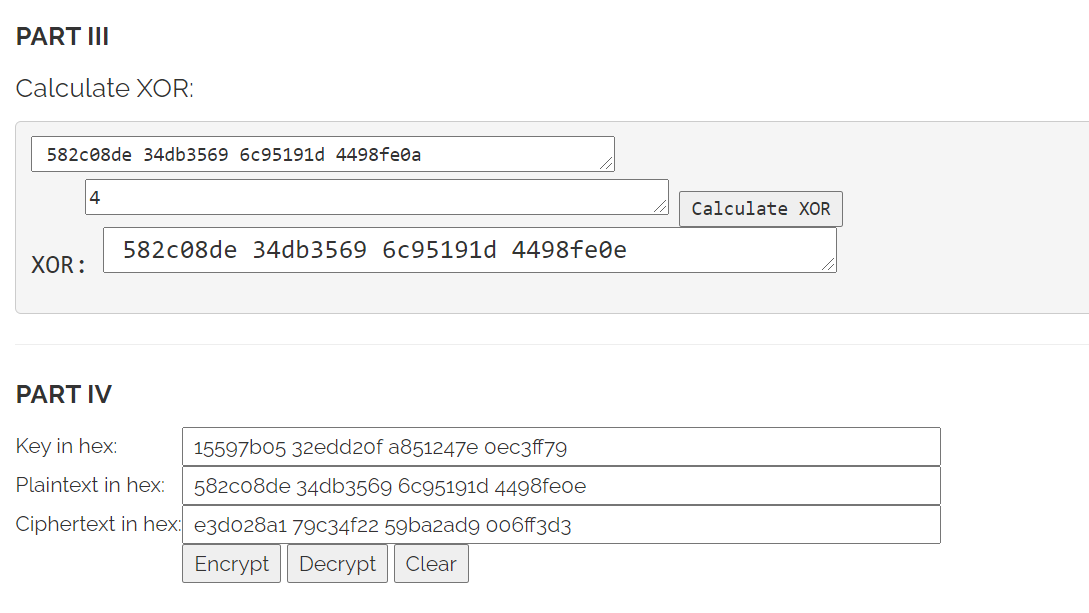
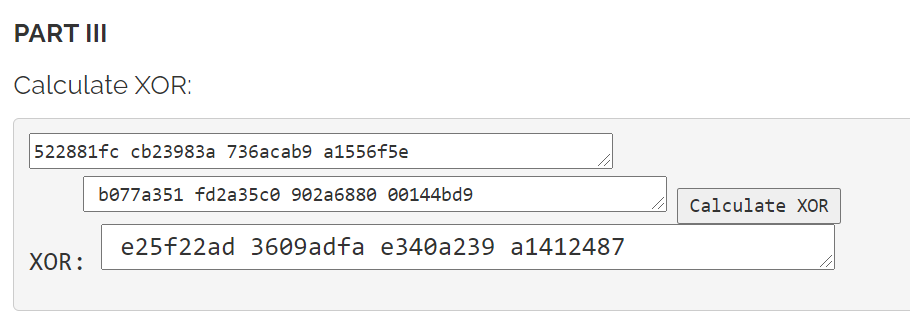
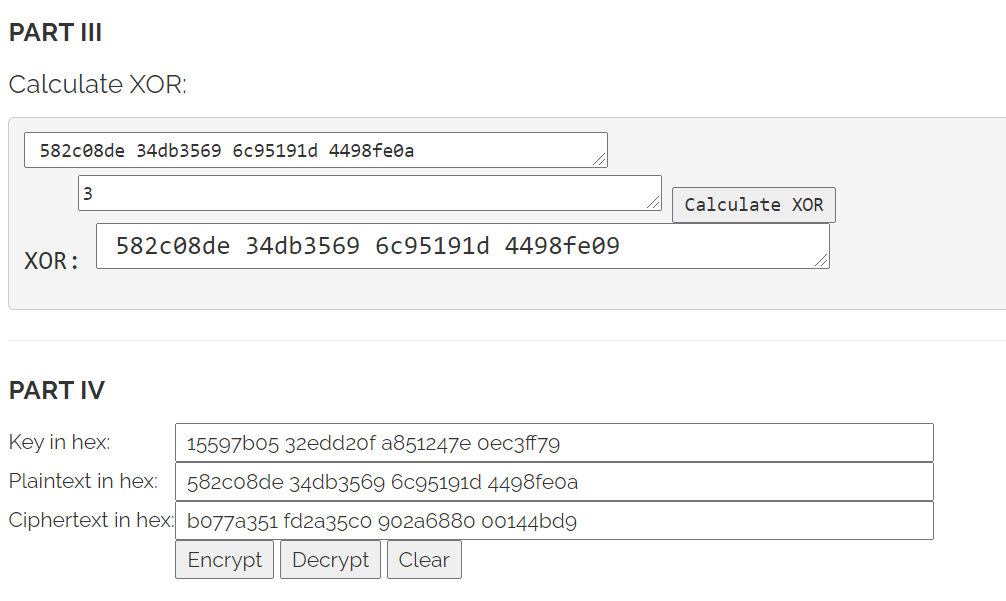
**Step 2 :**

****

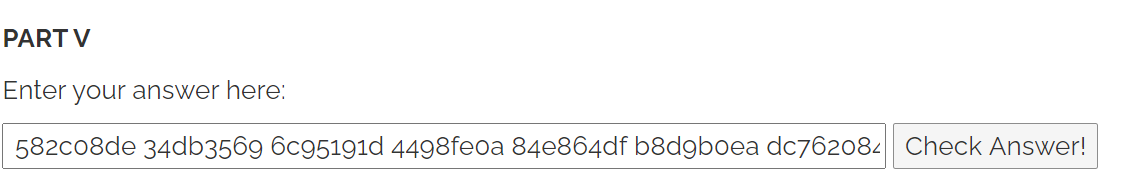
**Step 3 :**

****

****

****

**Step 4 :**

****

|  |
| --- |
| **RESULT:** |

The Electronic Code Block(ECB),Cipher Block Chaining(CBC),Output Feedback, Counter Mode are verified.

|  |
| --- |
| **Evaluation** |

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Originality of the Work | 15 |  |
| Documentation | 15 |  |
| Total |  |  |
| 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:** |

Diffie-Hellman protocol allows two communicating parties, say Alice and Bob, to create a symmetric session key with out the need of a KDC (Key Distribution Center)

**Diffie-Hellman Protocol :**

Alice and Bob chose two numbers p and g which are public.

‘p’ is a large prime of the order of 1024 bits. ‘g’ is a generator of order p-1 in the group Z p \*

Alice chooses a large random number ‘x’ in the range 0 to p-1 and calculates R1 = gx mod p

Bob chooses a large random number ‘y’ in the range 0 to p-1 and calculates R2 = gy mod p

Alice sends R1 to Bob and Bob sends R2 to Alice Alice Calculates K = (R2)x mod p

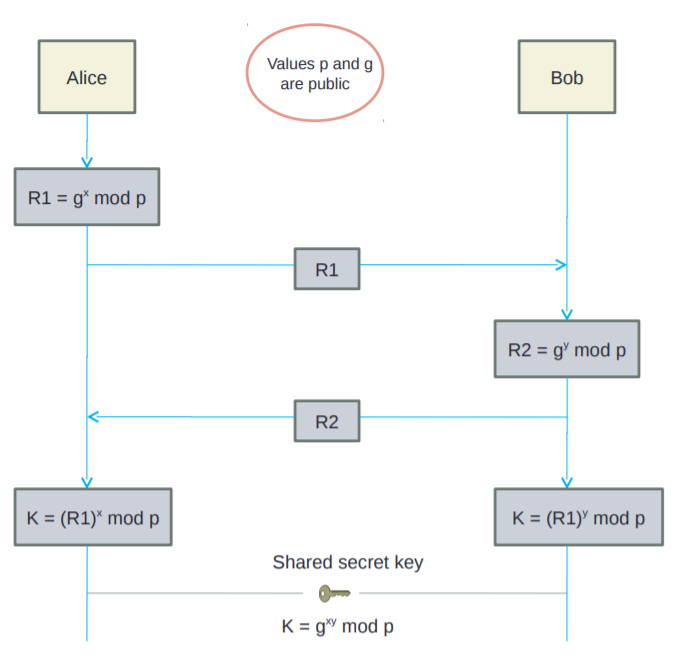
Bob Calculates K = (R1)y mod p

K = (gx mod p)y mod p = (gy mod p)x mod p = gxy mod p

K is the symmetric key for the session

P is a large prime of the order of 1024 bits

g is a generator of order p-1 in the group Zp\*

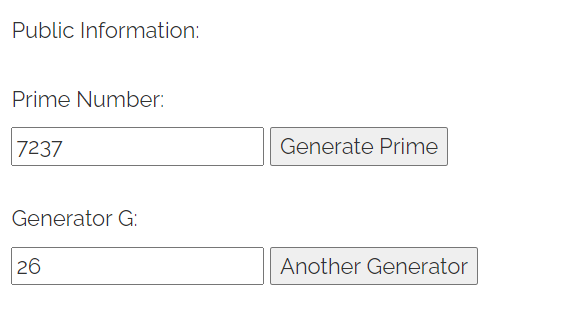
****

|  |
| --- |
| **ALGORITHM:** |

**STEP 1 :** Firslty,choose a large prime number **p** and a generator **g** for that prime.  
  
**STEP 2 :** Secondly,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 :** Finally,both calculate their public keys **gab** and **gba** repectively.  
  
**STEP 5 :** If both **gab** and **gba** are equal then Deffie-Hellman key exchange is verified.

|  |
| --- |
| **Screen Shots of simulation in Virtual labs** |

**STEP 1 :**



**STEP 2 :**



**STEP 3 :**



|  |
| --- |
| **Coding** |

**Alice:**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Alice{

public static void main(String[] args) throws IOException

{

try {

int port = 8000;

Scanner Sc= new Scanner (System.in);

System.out.print("enter a Private key B of Bob:");

int b =Sc.nextInt();

double clientP, clientG, clientA, B, Bdash;

String Bstr;

ServerSocket serverSocket = new ServerSocket(port);

Socket server = serverSocket.accept();

System.out.println("Connected to port ");

System.out.println("The private key of Bob = " + b);

DataInputStream in = new DataInputStream(server.getInputStream());

clientP = Integer.parseInt(in.readUTF());System.out.println("From Alice: P = " + clientP);

clientG = Integer.parseInt(in.readUTF());

System.out.println("From Alice : G = " + clientG);

clientA = Double.parseDouble(in.readUTF());

System.out.println("From Alice : Public Key = " + clientA);

B = ((Math.pow(clientG, b)) % clientP);

Bstr = Double.toString(B);

OutputStream outToclient = server.getOutputStream();

DataOutputStream out = new DataOutputStream(outToclient);

out.writeUTF(Bstr);

Bdash = ((Math.pow(clientA, b)) % clientP); System.out.println("Secret key of Alice "+ Bdash);

server.close();

}

catch (SocketTimeoutException s) {

System.out.println("Timed out!");

}

catch (IOException e) {

}

}

}

**Bob:**

import java.net.\*;

import java.io.\*;

import java.util.\*;

public class Bob{

static void Pr (int num)

{

boolean flag = false;

for(int i = 2; i <= num/2; ++i)

{

if(num % i == 0)

{

flag = true;

break;

}

}

if (!flag)

{

System.out.println(num + " is a prime number.");

}

else

{

System.out.println(num + " is not a prime number.");

}

for(int i=2;i < num;i++)

{

int p=0;

int[] a = new int[200];

System.out.println(" ");

for(int j=1;j < num;j++)

{

p = (int)Math.pow(i, j);

int x1=p%num;

System.out.println(+i+" ^"+ j +" mod "+num +"="+x1);

a[j]=x1;

}

find(a,i,num);

}

}

static void find ( int [] a, int i, int num)

{

int[] b = new int[200];

for(int z=1;z<num ;z++)

{

int count=0;

for(int e=1;e<num;e++)

{

b=a;

if(b[e]==a[z]){

count++;

}

}

if(count ==1 && z == num-1)

System.out.println(+i+" is primitive root of "+num);

}

}

public static void main(String[] args)

{

try {

String pstr, gstr, Astr;

String serverName = "localhost";

int port = 8000;

System.out.print("enter prime:");

Scanner S1= new Scanner (System.in);

int p = S1.nextInt();

Pr(p);

System.out.print("Enter private key of Alice:");

int a = S1.nextInt();

System.out.print("Enter Gb:");

int g = S1.nextInt();

double Adash, serverB;

Socket client = new Socket(serverName, port);

System.out.println("Connected to port ");

OutputStream outToServer = client.getOutputStream();

DataOutputStream out = new DataOutputStream(outToServer);

pstr = Integer.toString(p);

out.writeUTF(pstr);

gstr = Integer.toString(g);

out.writeUTF(gstr);

double A = ((Math.pow(g, a)) % p);

Astr = Double.toString(A);

out.writeUTF(Astr);

System.out.println("From Alice (server): Private Key = " + a);

DataInputStream in = new DataInputStream(client.getInputStream());

serverB = Double.parseDouble(in.readUTF());

System.out.println("From Bob(client) : Public Key = " + serverB);

Adash = ((Math.pow(serverB, a)) % p);

System.out.println("Secret Key of Bob "+ Adash);

client.close();

}

catch (Exception e) {

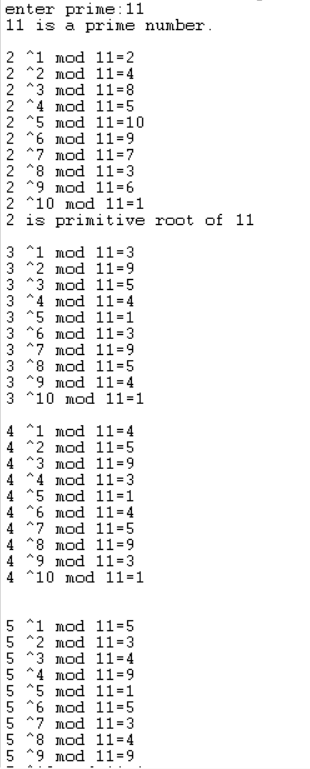
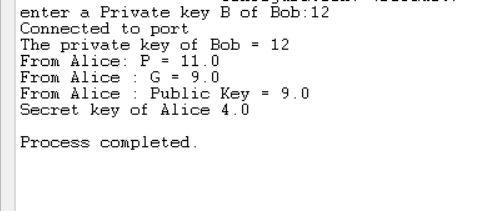
e.printStackTrace();

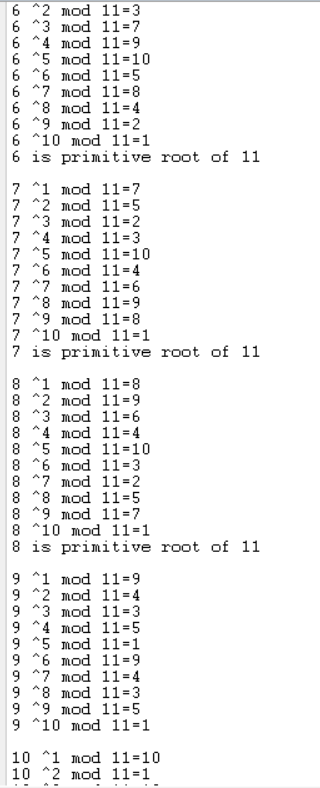
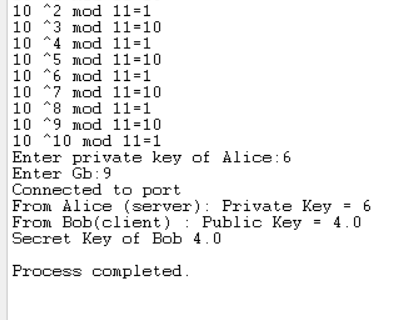
}

}

}

|  |
| --- |
| **SCREEN SHOTS:** |



|  |
| --- |
| **RESULT:** |

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

|  |
| --- |
| **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 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 Trapdoor one-way function is a one-way function which might be described as a function for which evaluation in one direction is straightforward, while computation in the reverse direction is far more difficult.

**Key generation**

The difficulty of determining a private key from an RSA public key is equivalent to factoring the modulus n. An attacker thus cannot use knowledge of an RSA public key to determine an RSA private key unless he can factor n. It is also a one-way function, going from p & q values to modulus n is easy but reverse is not possible.

Select two large primes, p and q.

Calculate n=p\*q.

Select number e which must be greater than 1 and less than (p − 1)(q − 1) and such that e and (p – 1)(q – 1) are coprime.

(n, e) - RSA public key

d ≡ e-1 mod (p − 1)(q − 1) – RSA private key

**RSA Encryption**

The sender whose public key is (n, e) represents the plaintext as a series of numbers less than n.

To encrypt the first plaintext P, which is a number modulo n. The encryption process is simple mathematical step as

**c = pe mod n**

**RSA Decryption**

The receiver of public-key pair (n, e) with a ciphertext c raises c to the power of his private key d. The result modulo n will be the plaintext P

**p = cd mod n**

|  |
| --- |
| **ALGORITHM:** |

**Key generation**

1. Select two large primes, p and q.
2. Calculate n=p\*q.
3. Select number e which must be greater than 1 and less than (p − 1)(q − 1) and such that e and (p – 1)(q – 1) are coprime.
4. (n, e) gives the public key pair.
5. d ≡ e-1 mod (p − 1)(q − 1) gives the private key of the recipient.

**RSA Encryption**

1. Make the plain text as p.
2. With the public key pair (n,e) calculate cipher text where c is

**c = pe mod n**

1. Send the cipher text to the receiver.

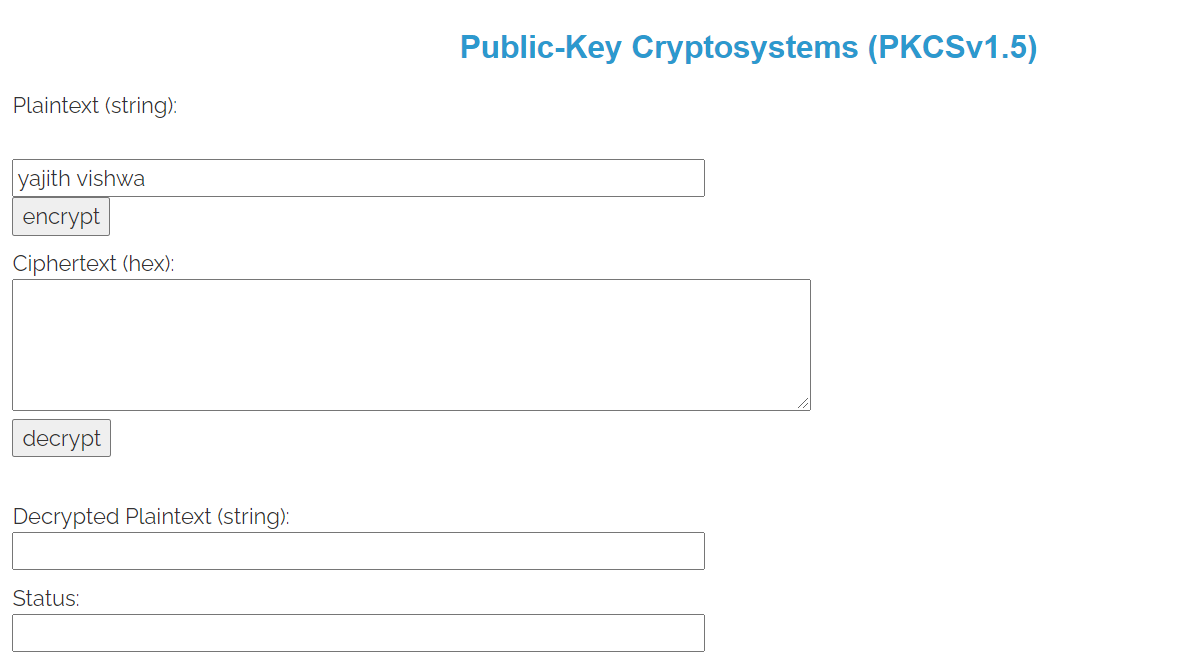
**RSA Decryption**

1. Receive the cipher text as c.
2. With the public key pair (n,e) and with private key d calculate p

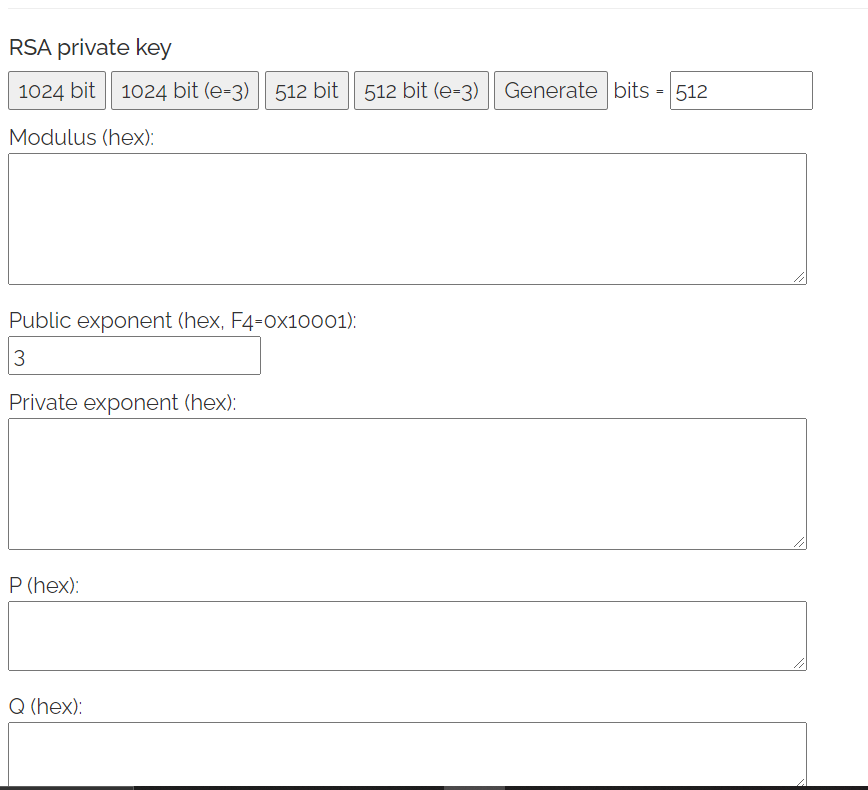
**p = cd mod n**

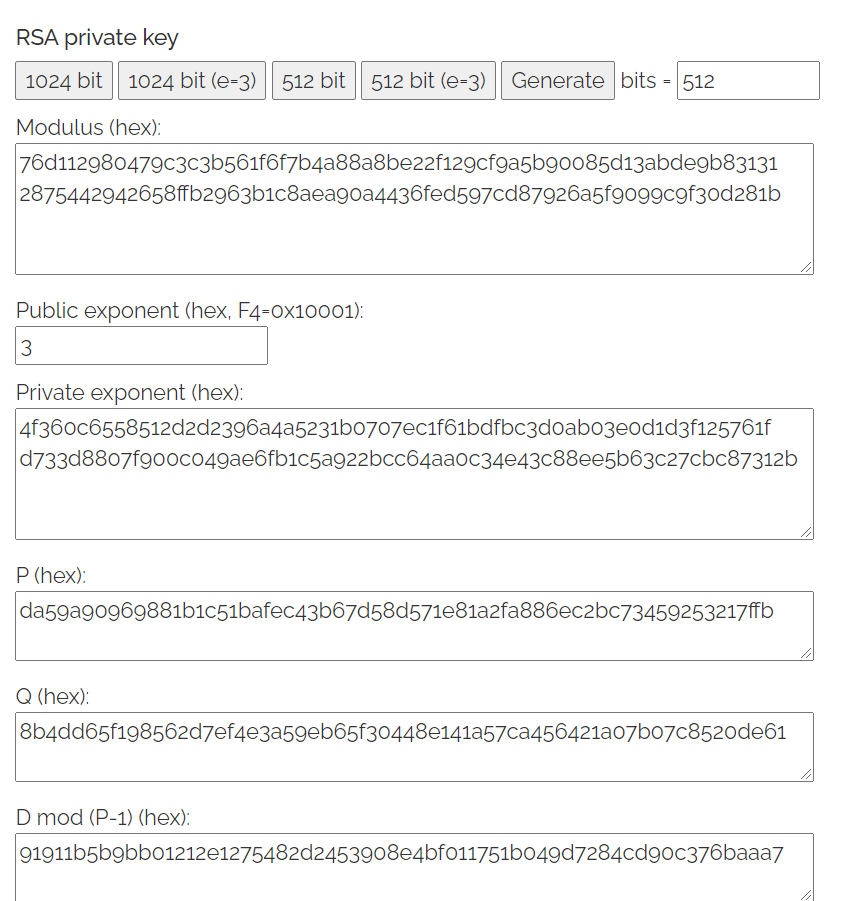
|  |
| --- |
| **Screen Shots of simulation in Virtual labs** |

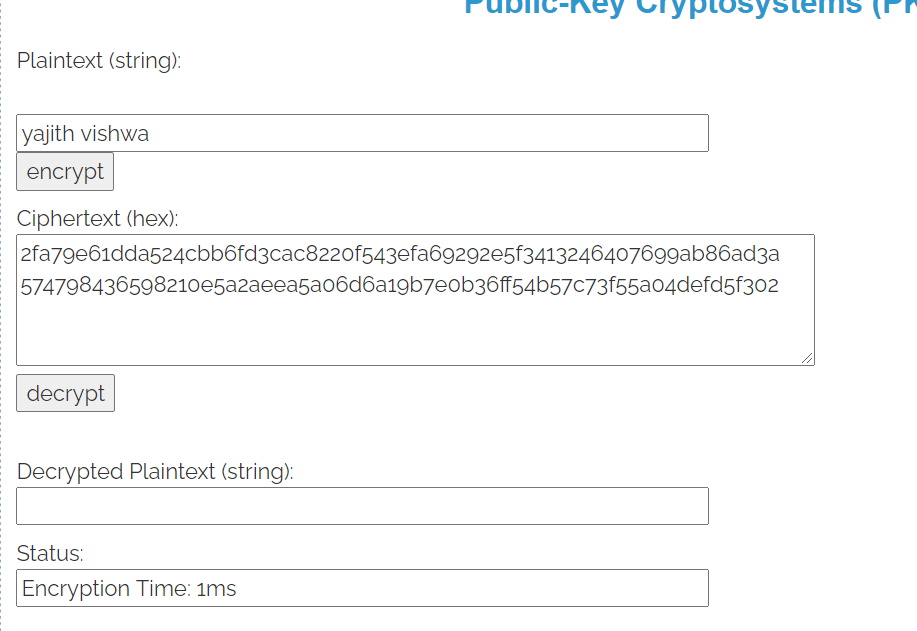
1. Enter the plain text .

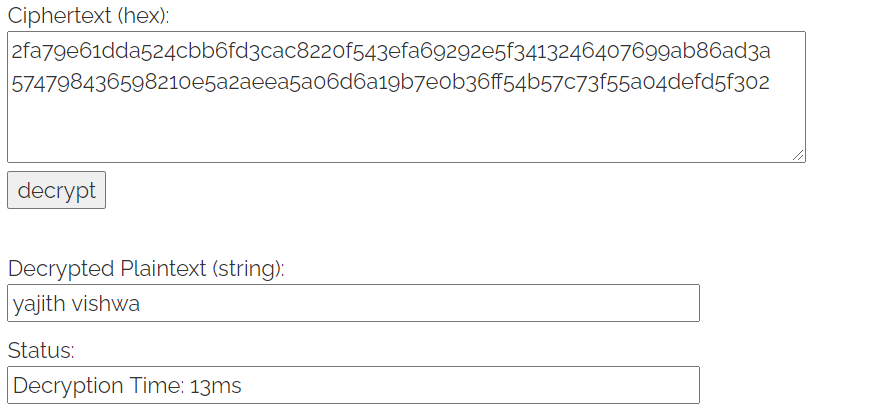


1. Generate RSA private key by clicking any one bit key



3. Generate private key.

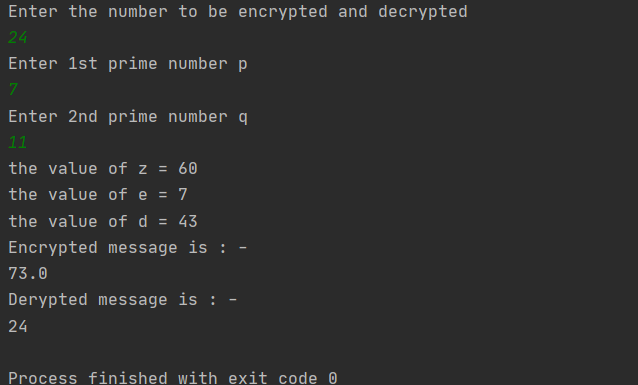
4. Click Encrypt and will get cipher text.

5. Click Decrypt to get back the plain text.

|  |
| --- |
| **Coding** |

package com.yajith.hillcipher;  
import java.math.BigDecimal;  
import java.math.BigInteger;  
import java.util.Scanner;  
public class Main {  
 public static void main(String[] args) {  
 Scanner sc=new Scanner(System.*in*);  
 int p,q,n,z,d=0,e,i;  
 System.*out*.println("Enter the number to be encrypted and decrypted");  
 int msg=sc.nextInt();  
 double c;  
 BigInteger msgback;  
 System.*out*.println("Enter 1st prime number p");  
 p=sc.nextInt();  
 System.*out*.println("Enter 2nd prime number q");  
 q=sc.nextInt();  
  
 n=p\*q;  
 z=(p-1)\*(q-1);  
 System.*out*.println("the value of z = "+z);  
 for(e=2;e<z;e++)  
 {  
 if(*gcd*(e,z)==1) // e is from public key pair  
 {  
 break;  
 }  
 }  
 System.*out*.println("the value of e = "+e);  
 for(i=0;i<=9;i++)  
 {  
 int x=1+(i\*z);  
 if(x%e==0) //d is for private key  
 {  
 d=x/e;  
 break;  
 }  
 }  
 System.*out*.println("the value of d = "+d);  
 c=(Math.*pow*(msg,e))%n;  
 System.*out*.println("Encrypted message is : -");  
 System.*out*.println(c);  
 BigInteger N = BigInteger.*valueOf*(n);  
 BigInteger C = BigDecimal.*valueOf*(c).toBigInteger();  
 msgback = (C.pow(d)).mod(N);  
 System.*out*.println("Derypted message is : -");  
 System.*out*.println(msgback);  
  
 }  
 static int gcd(int e, int z)  
 {  
 if(e==0)  
 return z;  
 else  
 return *gcd*(z%e,e);  
 }  
}

|  |
| --- |
| **SCREEN SHOTS:** |



|  |
| --- |
| **RESULT:** |

Thus, the simulation of RSA algorithm in done in virtual lab and the Java program for the algorithm is also implemented and the results are obtained.

|  |
| --- |
| **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.6 | **SECURITY ANALYSIS OF NETWORK TRAFFIC USING WIRESHARK** |

|  |
| --- |
| **AIM:** |

To analyse the network traffic for security analysis 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 etc.

**SSL/TLS**

**SSL** and its successor, **TLS** are protocols for establishing authenticated and encrypted links between networked computers. Although the SSL protocol was deprecated with the release of TLS 1.0 in 1999, it is still common to refer to these related technologies as “SSL” or “SSL/TLS.” The most current version is **TLS 1.3**, defined in [RFC 8446](https://tools.ietf.org/html/rfc8446)**.**

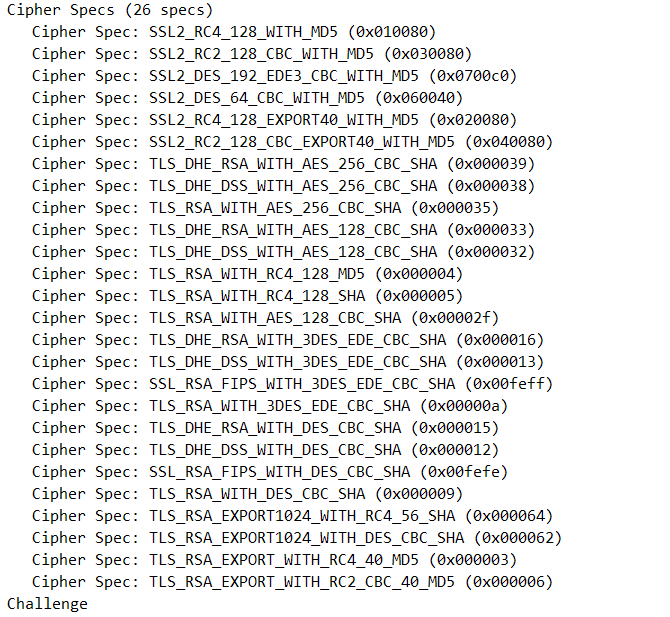
|  |
| --- |
| **WORKSHEET (using sample.pcap)** |

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

SSL version 2 is used by the client.



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



1. List the various parameters present in the public key certificate of the server.



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

30818902818100a46e53140ade2ce360559af242a6af47122f17cefabadc4e635634b9ba734b78443dc66c69a425b361029d09043f723dd827d3b05a4577b736e42623cc12b8aedea7b63a823c7c24590af896438ba329363f917f5dc72394297f0ace0abd8d9b2f1917aad58eec66a237eb3f57533cf2aabb79194b907ea7a399fe844c89f03d0203010001

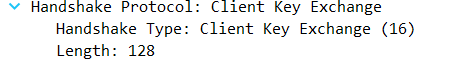


Yes we can trust because they have used RDN Sequence



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

The length of the key exchanged by the client is 128.



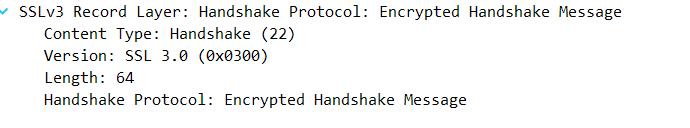
1. What algorithm is used for encrypting the session key?

The Algorithm used for encrypting the session key is RSA Encryption algorithm.



1. List the various parameters specified in the Encrypted handshake message.

The parameters are Content Type, Version, Length, Handshake protocol.



1. Calculate the time taken for completion of the entire handshake protocol.

End



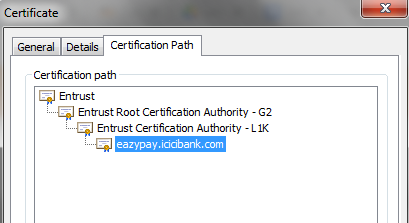
Start



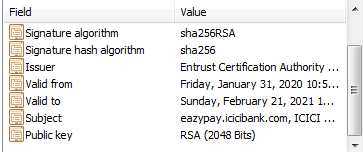
2.943406-0.000158=2.94328

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.

Website Chosen : <https://eazypay.icicibank.com/homePage>



Algorithms used :



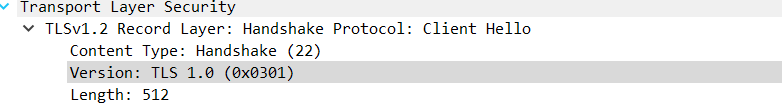
10. Enlist the differences between HTTP and HTTPS.

|  |  |
| --- | --- |
| **HTTP** | **HTTPS** |
| HTTP is hypertext transfer protocol. | HTTPS stands for Hyper Text Transfer Protocol Secure. |
| HTTP uses port 80 for Data Communication. | HTTPS uses the port no. 443 for Data Communication. |
| HTTP website do not need SSL. | HTTPS is a combination of SSL/TLS protocol and HTTP. |
| HTTP does not improve search rankings. | HTTPS helps to improve search ranking. |
| HTTP URLs begin with http:// | HTTPs URLs begin with https:// |
| HTTP is less secure as the data can be vulnerable to hackers. | HTTPS is designed to prevent hackers from accessing critical information. It is secure against such attacks |

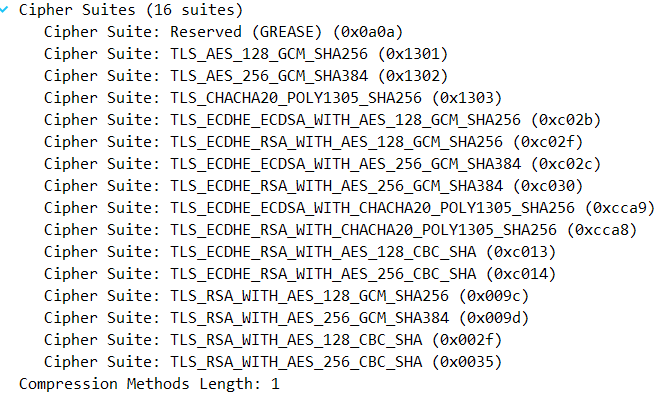
|  |
| --- |
| **WORKSHEET (For packet captured for real time data** |

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

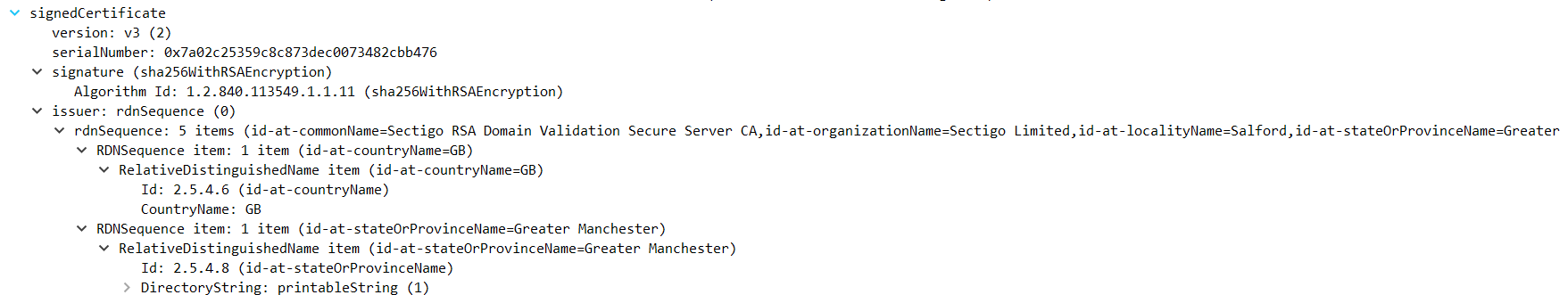
TLS version used here is 1.0.



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



1. List the various parameters present in the public key certificate of the server.



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

Yes we can trust the same because digital signature is used here.



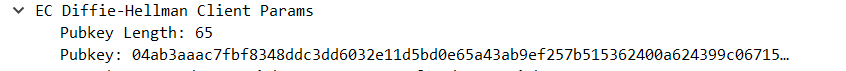
1. Identify the length of key exchanged by the Client?

The length of the client key exchange is 66.



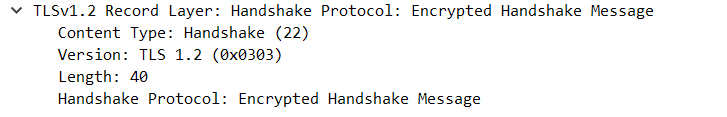
1. What algorithm is used for encrypting the session key?

The algorithm used here is EC Diffie Hellman.



1. List the various parameters specified in the Encrypted handshake message.

The parameters are Content Type, Version, Length, Handshake protocol.



1. Calculate the time taken for completion of the entire handshake protocol.

Start



End



6.940876-6.493908=0.446968

|  |
| --- |
| **RESULT:** |

The analyse the network traffic for security analysis using wireshark 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.7 | **Perform password extraction, cracking and recovery from target system** |

|  |
| --- |
| **AIM:** |

To use open source software tools for extraction, cracking and recovery from target system.

|  |
| --- |
| **THEORY:** |

**Tool Selected:**

John The Ripper

Office2john.py .This file helped to create hash file for the word file. This has support up to Microsoft office 2013.

**About the Tool:**

The tool used for

**1. Cracking Linux User Password**  
**2.Cracking Password Protected ZIP/RAR Files**  
**3.Decrypting MD5 Hash**  
**4.Using Wordlists To Crack Passwords**

**Software Requirements:**

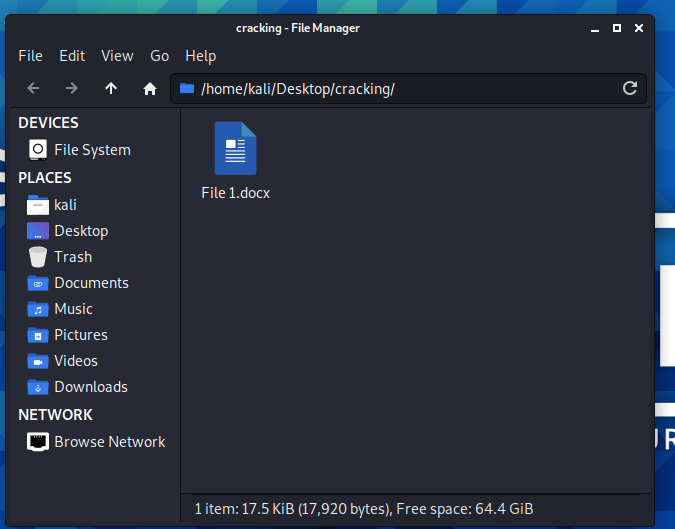
Unix based System

**Dictionary attack:**

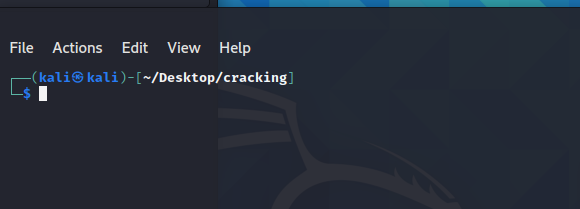
A dictionary attack is a method of breaking into a password-protected computer or server by systematically entering every word in a dictionary as a password. A dictionary attack can also be used in an attempt to find the key necessary to decrypt an encrypted message or document

|  |
| --- |
| **Demonstrations (With Screen shots)** |

1. Create a folder and store the encrypted file in the folder.



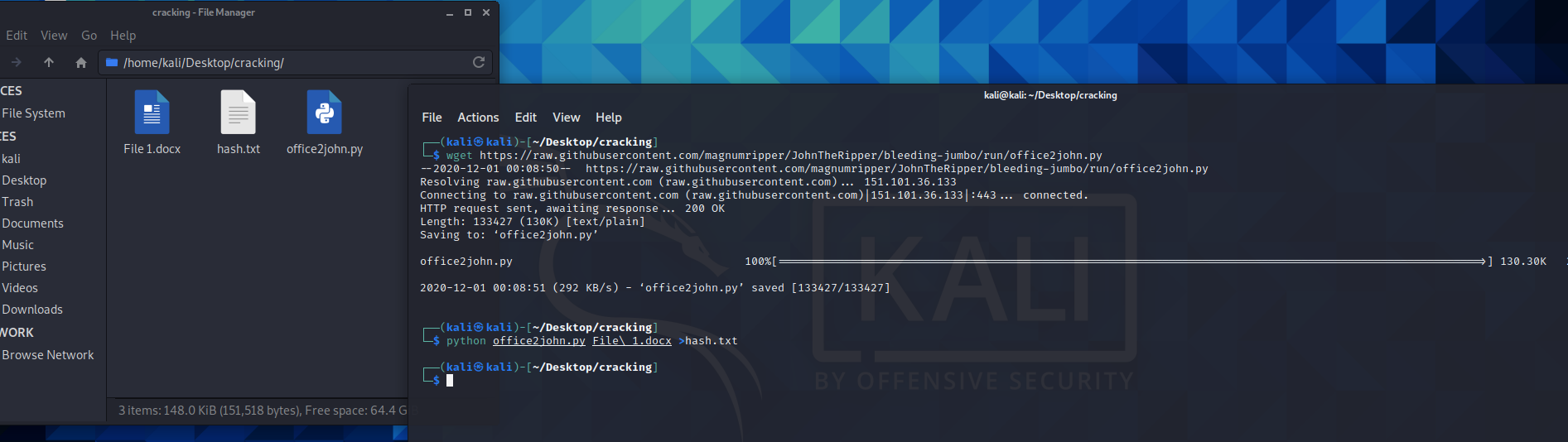
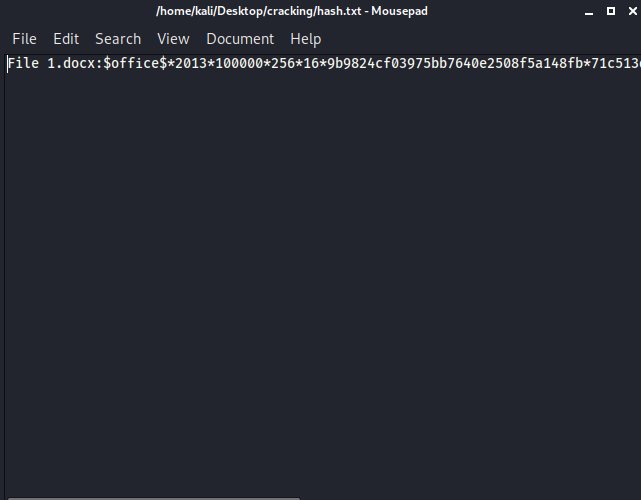
1. Open the terminal in the folder.

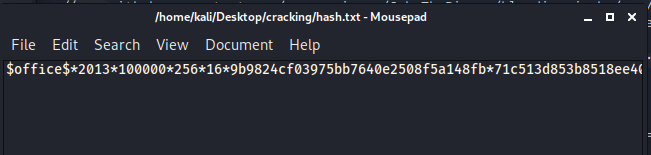


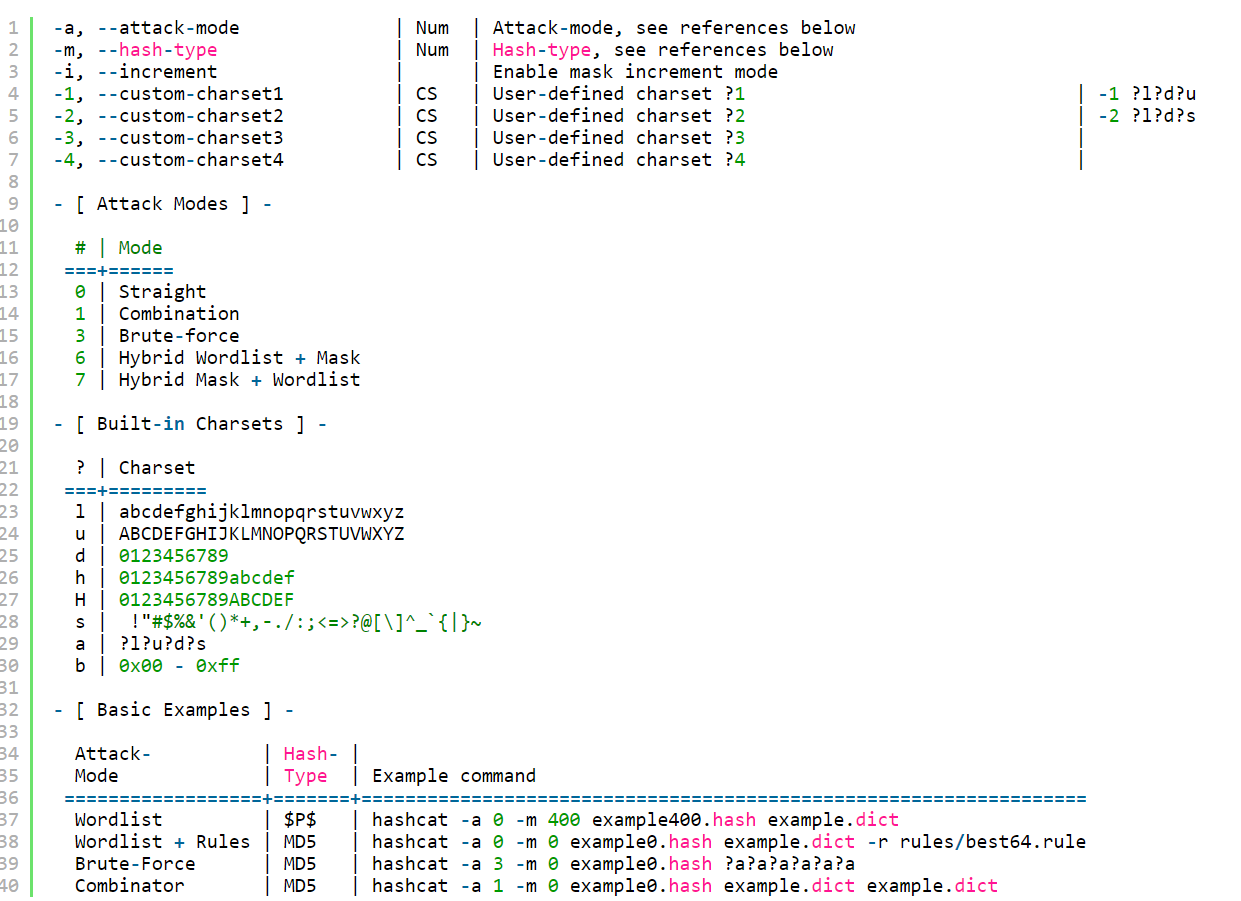
1. Download the python file using the command

wget https://raw.githubusercontent.com/magnumripper/JohnTheRipper/bleeding-jumbo/run/office2john.py



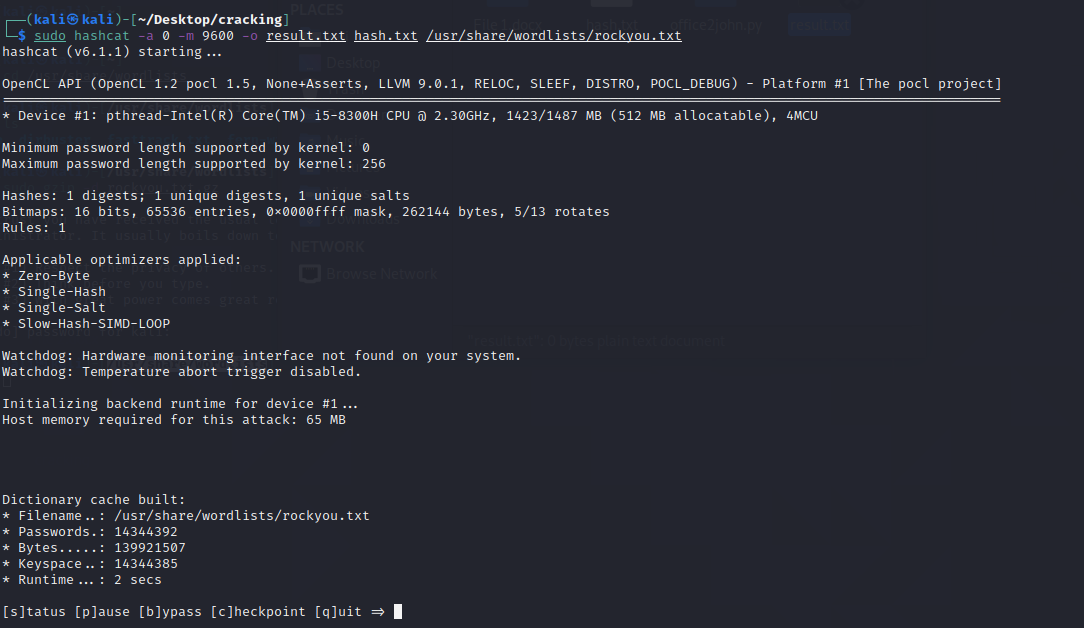
1. Run the python and pass the docx file to get hash file. 
2. Open the hash file and delete upto $ office. 
3. Finally hash file looks like this.



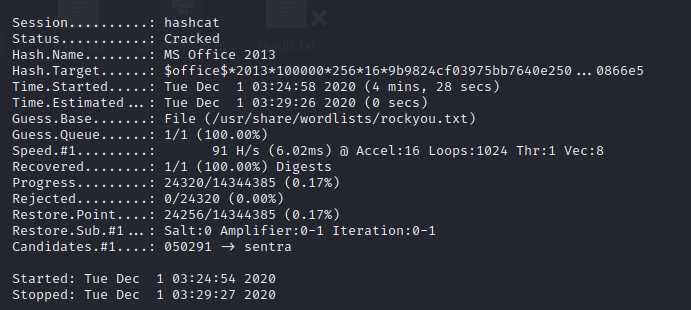


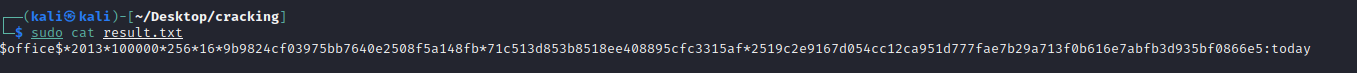
1. Start cracking with hashcat function.

sudo hashcat -a 0 -m 9600 -o result.txt hash.txt /usr/share/wordlists/rockyou.txt



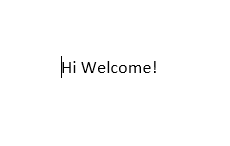
**After 4 mins**





**Password for File 1:**

today



|  |
| --- |
| **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) | 30 |  |
| Completion of experiment on time | 10 |  |
| Documentation | 10 |  |
| Total | 50 |  |
| Signature of the faculty with Date |  |  |

|  |  |
| --- | --- |
| Ex.No.8 | **HASHING TECHNIQUES** |

|  |
| --- |
| **AIM:** |

To implement SHA-1 for integrity check

To explore the usage of online calculators’ for hash computation

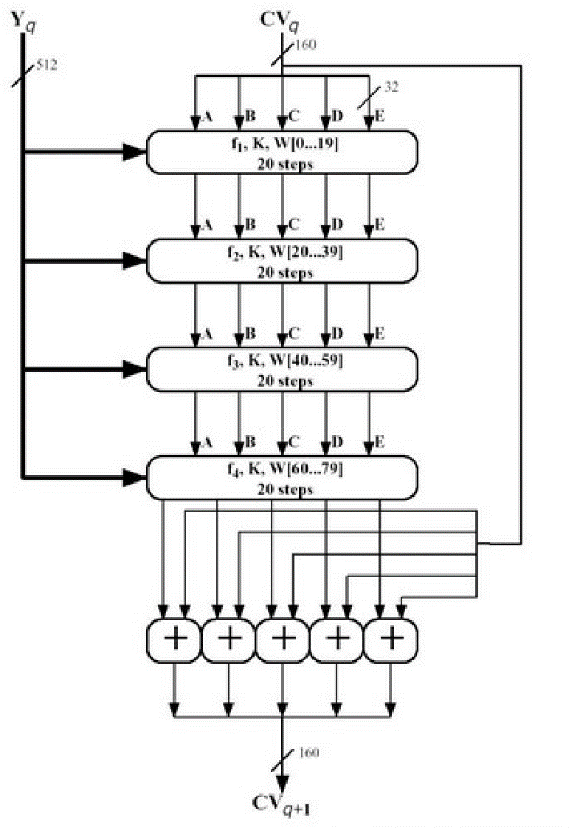
|  |
| --- |
| **THEORY:** |

SHA-1 takes an input of any size and produces a 160-bit hash value known as a message digest.

**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** |

1. Initialize the buffer values
2. The message is then padded by appending a 1, followed by enough 0s until the message is 448 bits.
3. The length of the message represented by 64 bits is then added to the end, producing a message that is 512 bits long.
4. The padded message is then divided into 16 words each of size 32 bits
5. Start the 80 iterations of sha-1.
6. Calculate W as W(i)=S1(W(i−3)⊕W(i−8)⊕W(i−14)⊕W(i−16))
7. Assign

B = A

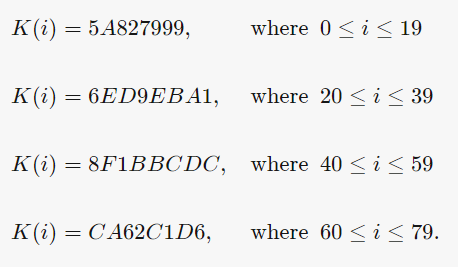
C = S30(B)

D = C

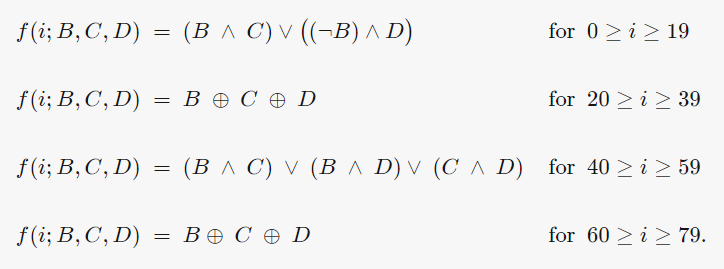
E = D

A =F ⊕ E ⊕ S5(A) ⊕ Wt ⊕ Kt

Where K is



​ and F is

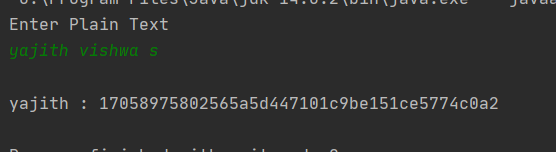


1. Repeat above for 79 times and the final result is calculated as logically OR with the buffer values. This is the message digest

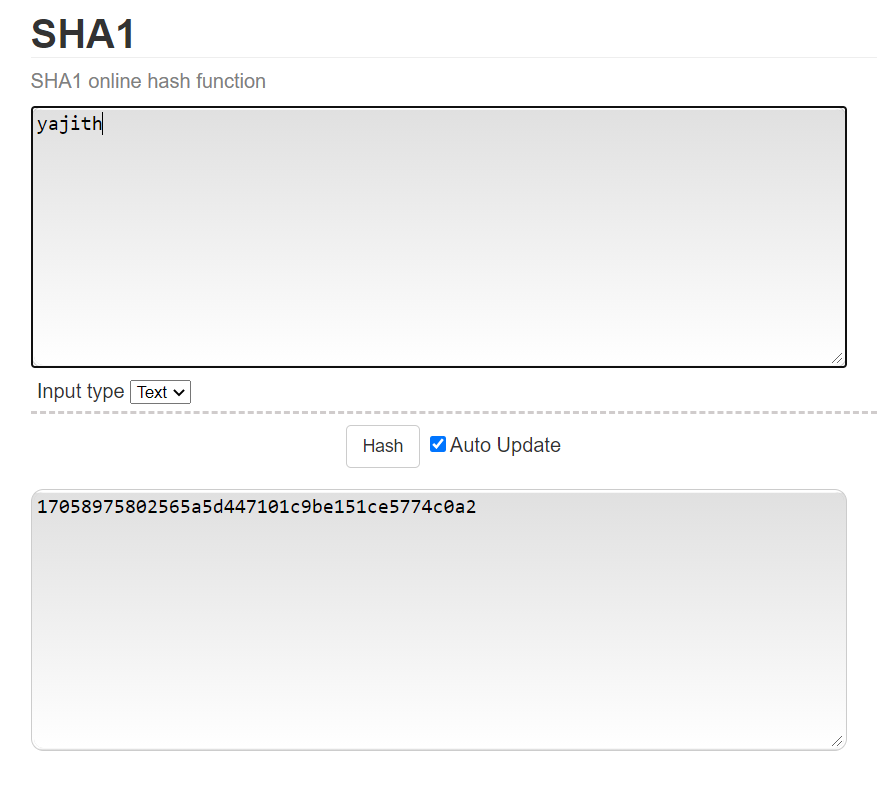
|  |
| --- |
| **Coding** |

package com.yajith.sha1;  
  
import java.math.BigInteger;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.util.Scanner;  
public class Main {  
 public static String encryptThisString(String input)  
 {  
 try {  
 MessageDigest md = MessageDigest.*getInstance*("SHA-1");  
 byte[] messageDigest = md.digest(input.getBytes());  
 BigInteger no = new BigInteger(1, messageDigest);  
 String hashtext = no.toString(16);  
 while (hashtext.length() < 32) {  
 hashtext = "0" + hashtext;  
 }  
 return hashtext;  
 }  
 catch (NoSuchAlgorithmException e) {  
 throw new RuntimeException(e);  
 }  
 }  
 public static void main(String args[])  
 {  
  
 System.*out*.println("Enter Plain Text");  
 Scanner scanner=new Scanner(System.*in*);  
 String s1 = scanner.next();  
 System.*out*.println("\n" + s1 + " : " + *encryptThisString*(s1));  
 }  
}

|  |
| --- |
| **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 |  |  |

|  |  |
| --- | --- |
| Ex.No.9 | **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

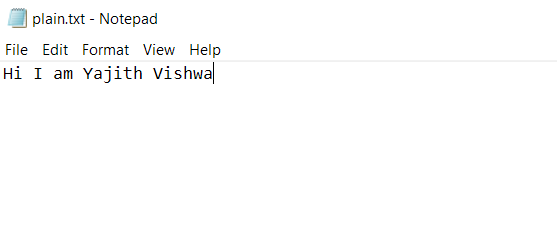
|  |
| --- |
| **THEORY:** |

**About OpenSSL**

OpenSSL is a full-featured toolkit for the Transport Layer Security (TLS) and Secure Sockets Layer (SSL) protocols. It is also a general-purpose cryptography library which can be used to perform AES, RSA and hashing algorithm like SHA, MD5 and DSS. It is widely used by Internet servers, including the majority of HTTPS websites.

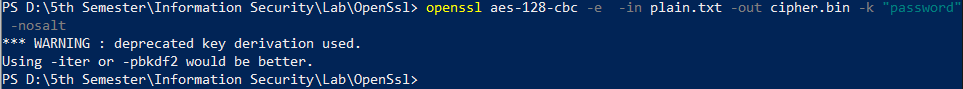
|  |
| --- |
| **Worksheet (Answer the following ques and paste the relevant outputs)** |

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

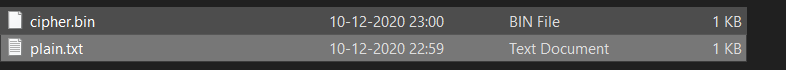


1. Encrypt using aes in all block cipher modes of operation.

*openssl aes-128-cbc -e -in plain.txt -out cipher.bin -k "password" -nosalt*



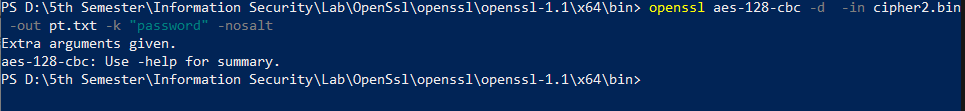
1. Display the contents of cipher.bin



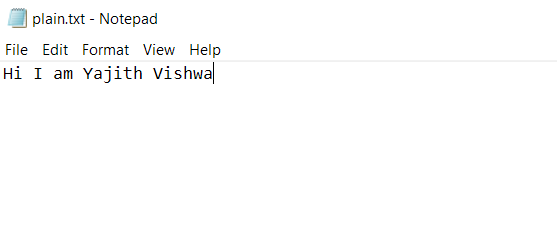


1. Decrypt the contents of cipher.bin

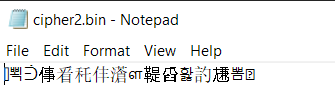
*openssl-1.1\x64\bin>openssl aes-128-cbc -d -in cipher1.bin -out pt.txt -k "password" –nosalt*



1. Display the contents of pt.txt



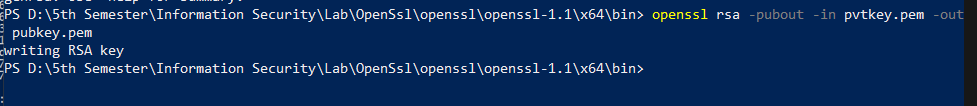
1. Examine Avalanche effect by changing one bit/charater in your plain text file



1. Generate private and public key for RSA

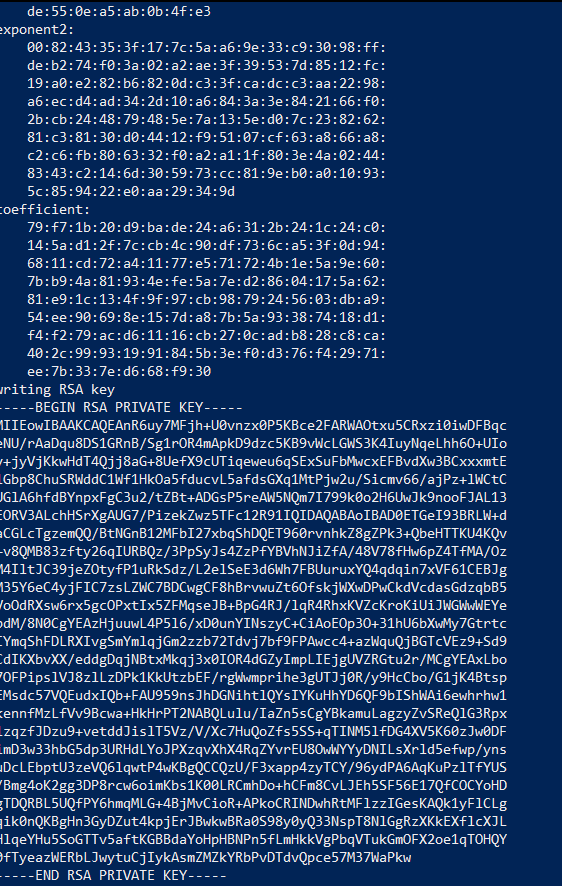
*openssl genrsa –out pvtkey.pem*

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



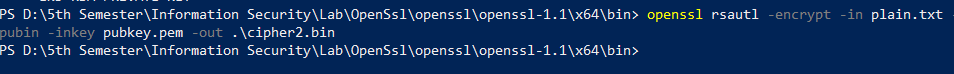
1. Display the private key in hexadecimal

*openssl rsa -text -in pvtkey.pem*

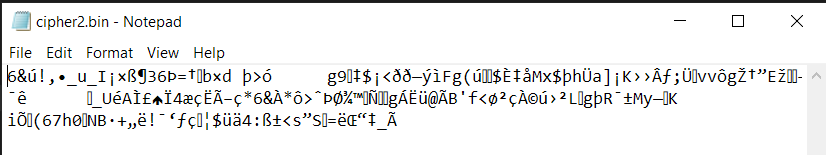


1. Perform encryption using RSA public key

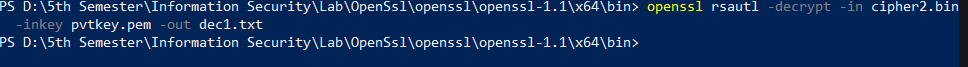
*openssl rsautl -encrypt -in plain.txt -pubin -inkey pubkey.pem -out c1.bin*



1. Display the contents of encrypted file

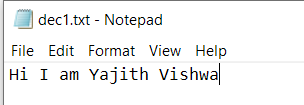


1. Decrypt the result using RSA private key



1. Display the contents of decrypted file

*openssl rsautl -decrypt -in c1.bin -inkey pvtkey.pem -out dec1.txt*



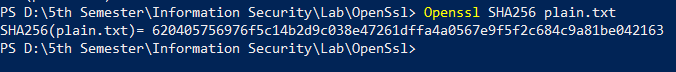
1. Generate the hash of a file using MD5

*openssl md5 plain.txt*



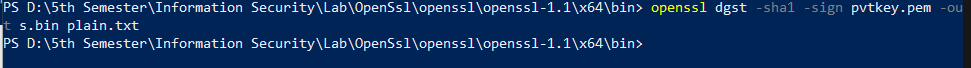
1. Generate the hash of a file using SHA256

*Openssl SHA256 plain.txt*



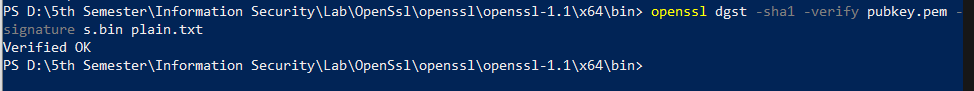
1. *Generate Signature using SHA and RSA*

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



1. *Verify the signature*

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



|  |
| --- |
| **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.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.

# DVWA is a PHP/MySQL web application, whose main goal is to be an aid for security professionals to test their skills and tools in a legal environment.

**SQL injection Attack:-**

SQL injection is a web security vulnerability that allows an attacker to interfere with the queries that an application makes to its database. It generally allows an attacker to view data that they are not normally able to retrieve. This might include data belonging to other users, or any other data that the application itself is able to access. In many cases, an attacker can modify or delete this data, causing persistent changes to the application's content or behavior.

# Ways to prevent SQL injection:

1. **Input validation:-** The validation process is aimed at verifying whether or not the type of input submitted by a user is allowed. Input validation makes sure it is the accepted type, length, format, and so on. Only the value which passes the validation can be processed. It helps counteract any commands inserted in the input string. In a way, it is similar to looking to see who is knocking before opening the door.

# Parametrized queries

Parameterized queries are a means of pre-compiling an SQL statement so that you can then supply the parameters in order for the statement to be executed. This method makes it possible for the database to recognize the code and distinguish it from input data.

The user input is automatically quoted and the supplied input will not cause the change of the intent, so this coding style helps mitigate an SQL injection attack.

# Stored procedures

Stored procedures (SP) require the developer to group one or more SQL statements into a logical unit to create an execution plan. Subsequent executions allow statements to be automatically parameterized. Simply put, it is a type of code that can be stored for later and used many times.

So, whenever you need to execute the query, instead of writing it over and over, you can just call the stored procedure.

# Escaping

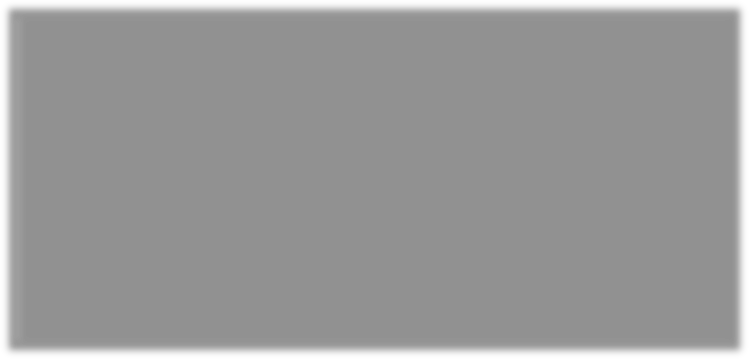
Always use character-escaping functions for user-supplied input provided by each database management system (DBMS). This is done to make sure the DBMS never confuses it with the SQL statement provided by the developer.

For example, use the mysql\_real\_escape\_string() in PHP to avoid characters that could lead to an unintended SQL command.

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

# Installation procedure for WAMP and DVWA:-

Download Wamp and click next and install it.



# Download DVWA and extract it:-



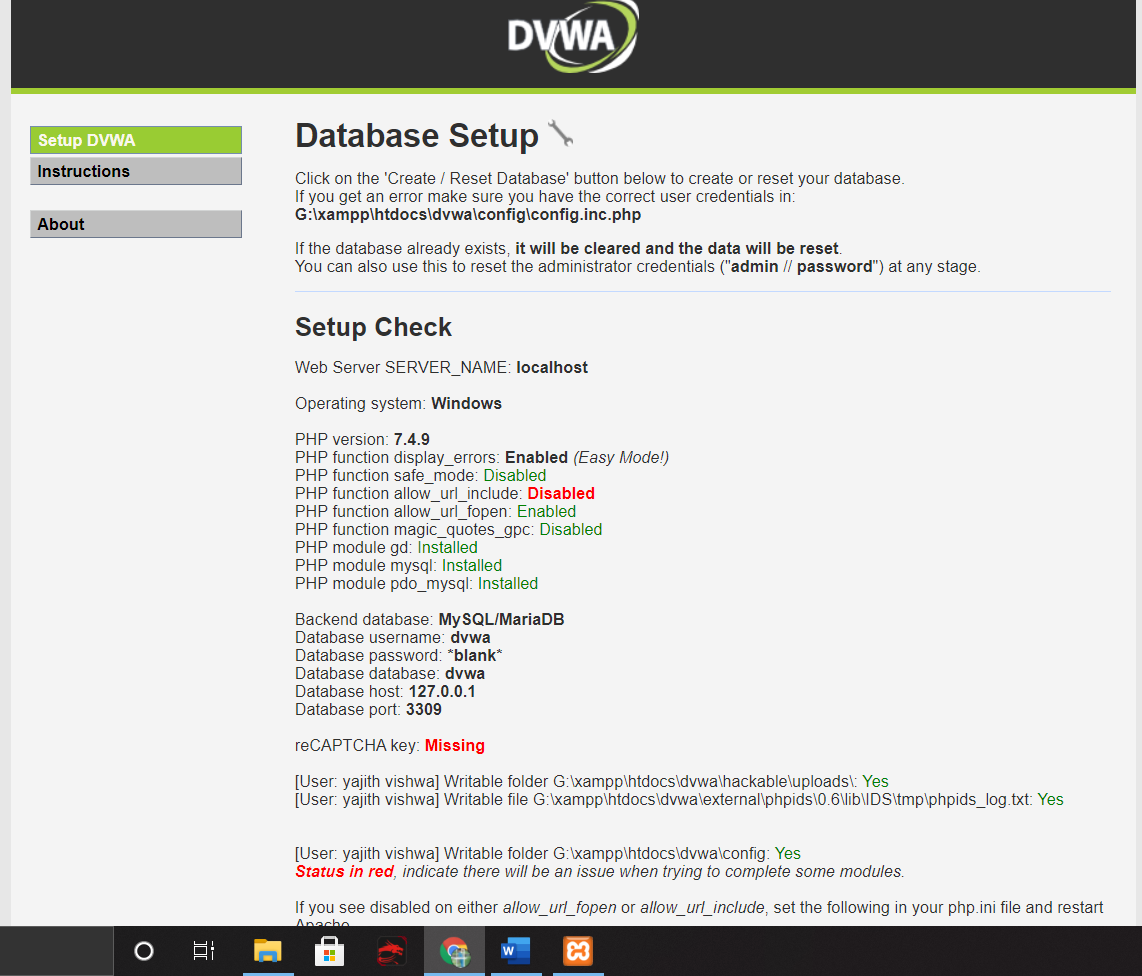
* 1. **Copy DVWA Folder to WAMP/**[**www.**](http://www/)



* 1. **Open Wamp-Localhost.**



* 1. **In any browser type localhost/dvwa.**



**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:

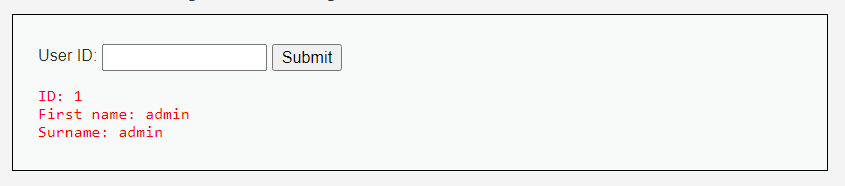
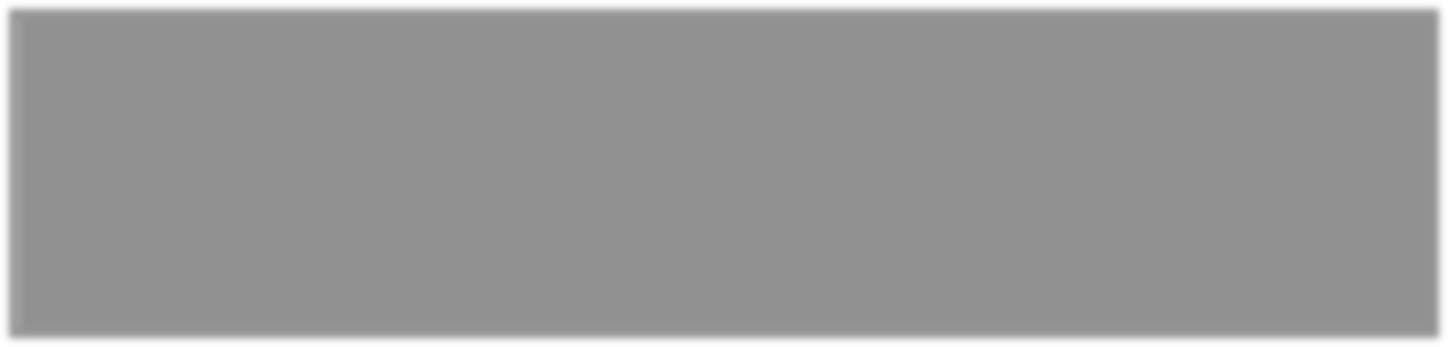
# Username : admin

* + **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.

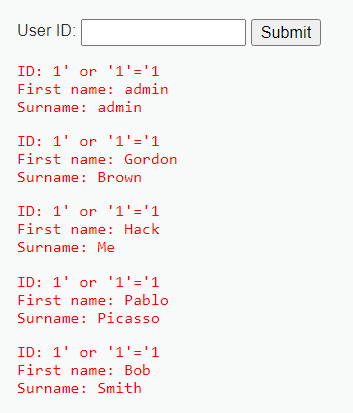
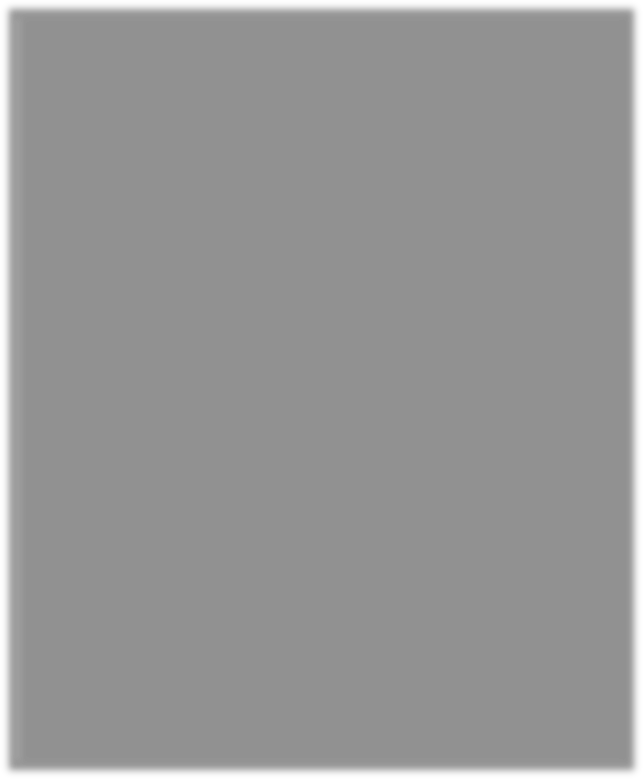
# Check expected results:

SELECT first\_name, last\_name FROM users WHERE user\_id = '1'



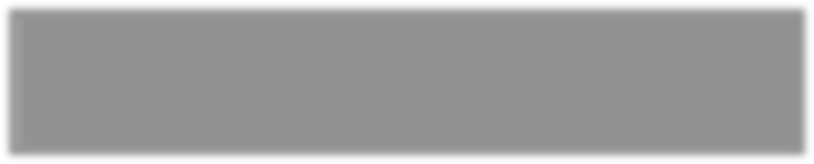
# Check the results of an OR True statement

SELECT first\_name, last\_name FROM users WHERE user\_id = ‘1' or '1'='1’



# Find the number of columns in table:

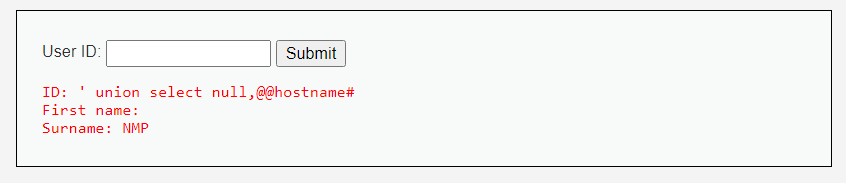
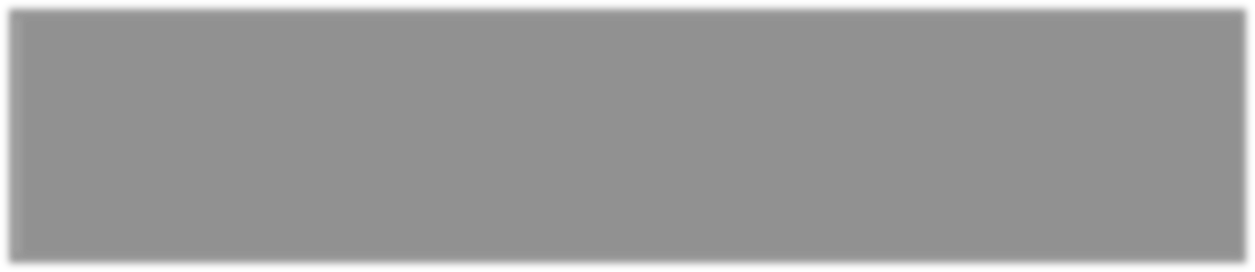
SELECT first\_name, last\_name FROM users WHERE user\_id = 'a' ORDER BY 3;#'



So No of column is 2.

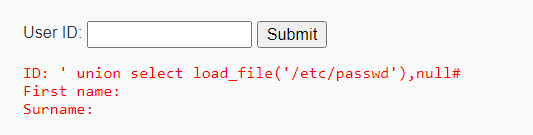
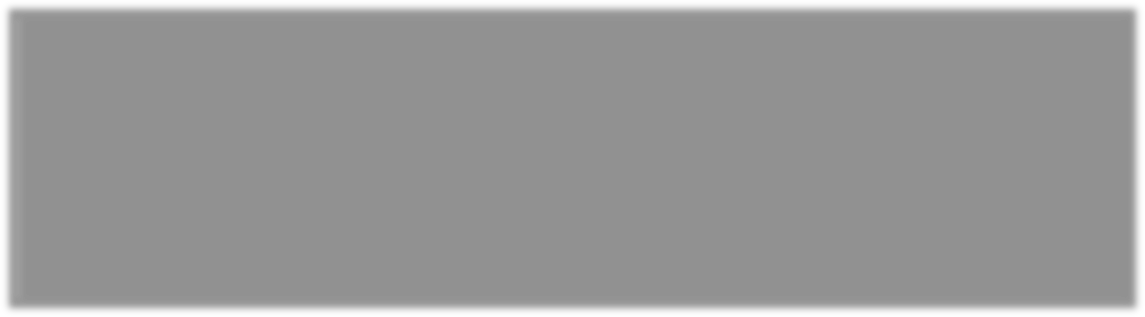
# Find Hostname:

SELECT first\_name, last\_name FROM users WHERE user\_id = ' ' union select null,@@hostname#'



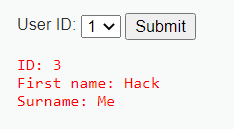
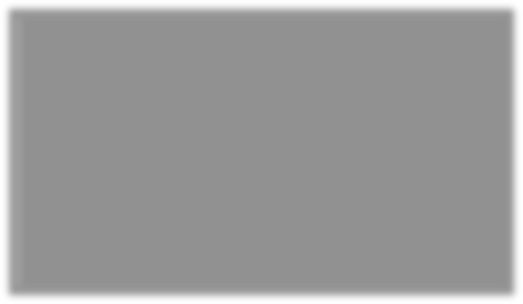
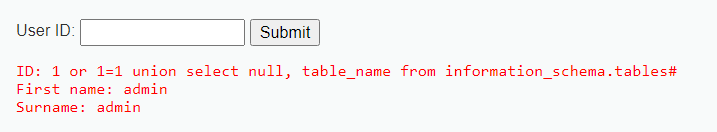
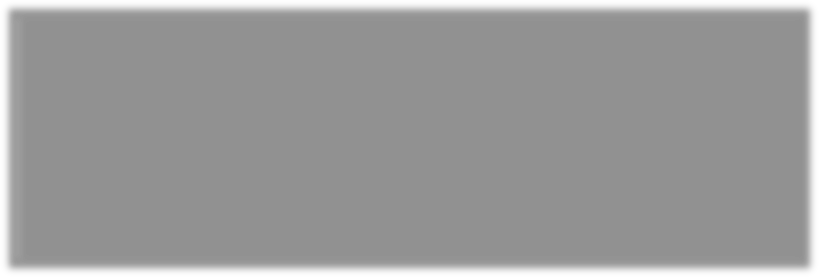
# Display File:

SELECT first\_name, last\_name FROM users WHERE user\_id = ' ' union select load\_file('/etc/passwd'),null#'



# Try to do SQL Injection on DVWA with security level set to “low/Medium” and find out all schema name from database.

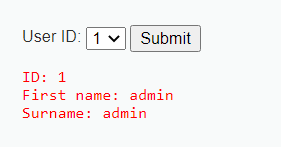
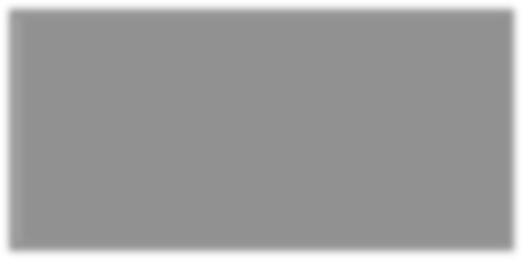
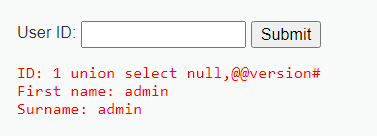
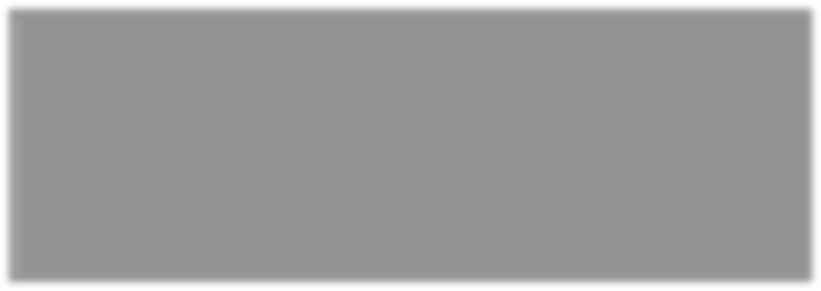
1 or 1=1 union select null, table\_name from information\_schema.tables#



For medium it is restricted to select only the User ID.

# Try to do SQL Injection on DVWA with security level set to “low/Medium” and find out mysql version.

1 union select null,@@version#



For medium it is restricted to select only the User ID.

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

**Ans:-**

Every vulnerability has four different security levels, low, medium, high and impossible.

The security levels give a challenge to the ‘attacker’ and also shows how each vulnerability can be counter measured by secure coding.

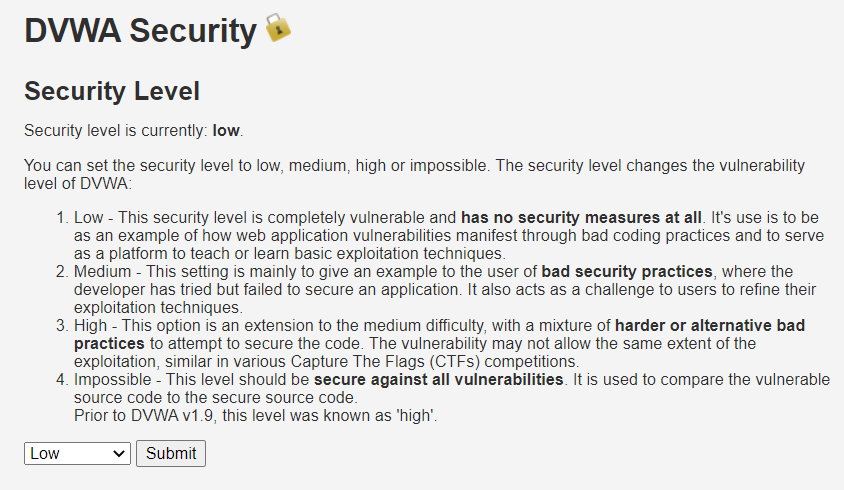
# Medium:-

This security level’s purpose is to give the ‘attacker’ a challenge in exploitation and also serve as an example of bad coding/security practices.This security level is basically to a case for the user having awful coding practices, where the developer is attempt but neglect to secure an application.It is also used to test the skills of client to refine their vulnerable techniques.

# Low:-

This security level is meant to simulate a website with no security at all implemented in their coding. It gives the ‘attacker’ the chance to refine their exploitation skills.

This level contain no security i.e most vulnerable level. Programmer gives bad coding practice.



Compared to low; **medium level security** protects against various vulnerabilities.

**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.11 | **Buffer Overflow** |

|  |
| --- |
| **AIM:** |

To demonstrate Buffer flow attack

|  |
| --- |
| **THEORY:** |

**Buffer overflow Attack**

A buffer overflow condition exists when a program attempts to put more data in a buffer than it can hold or when a program attempts to put data in a memory area past a buffer. In this case, a buffer is a sequential section of memory allocated to contain anything from a character string to an array of integers. Writing outside the bounds of a block of allocated memory can corrupt data, crash the program, or cause the execution of malicious code.

**Stack-based** buffer overflows, which are more common among attackers, exploit applications and programs by using what is known as a stack: memory space used to store user input.

**Heap-based** attacks are harder to carry out and involve flooding the memory space allocated for a program beyond memory used for current runtime operations.

**Ways to prevent Buffer overflow**

* **Address space randomization** (ASLR) randomly moves around the address space locations of data regions. Typically, buffer overflow attacks need to know the locality of executable code, and randomizing address spaces makes this virtually impossible.
* **Data execution prevention** flags certain areas of memory as non-executable or executable, which stops an attack from running code in a non-executable region.
* **Structured exception handler overwrite protection (SEHOP)** helps stop malicious code from attacking Structured Exception Handling (SEH), a built-in system for managing hardware and software exceptions. It thus prevents an attacker from being able to make use of the SEH overwrite exploitation technique.

|  |
| --- |
| **Demonstration** |

**Normal Case:**

#include <stdio.h>

#include <stdlib.h>

void main()

{

char \*name;

char \*command;

name=(char \* ) malloc (10);

command=(char \*) malloc (128);

printf("Ente r your name:");

gets(name );

printf("Hello %s\n",name);

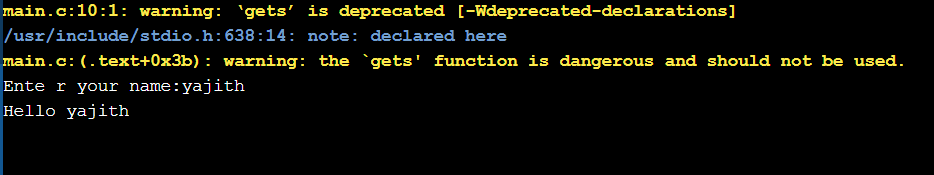
system(command);

}

**Input and Output:**

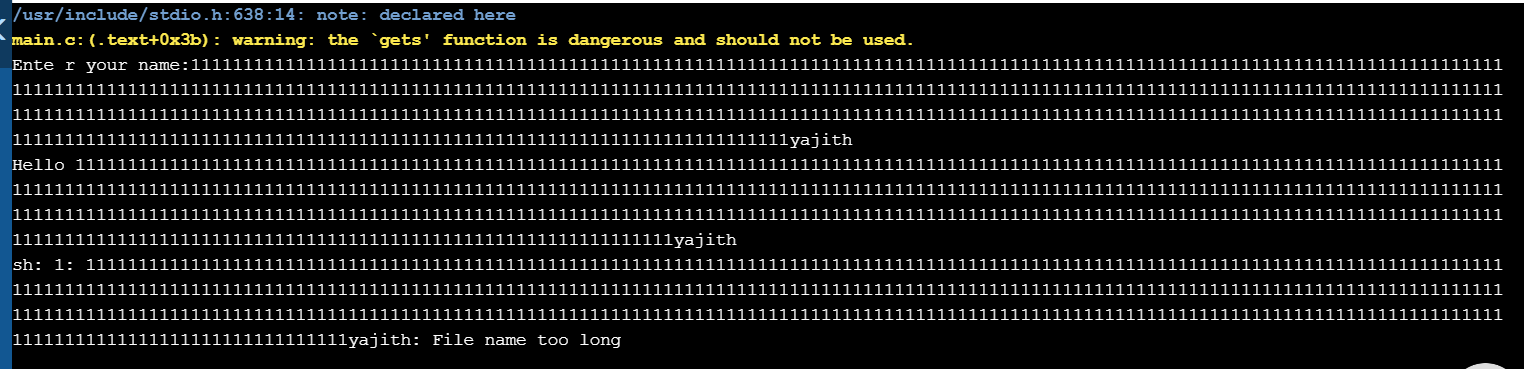
Input: yajith

Output: Hello yajith

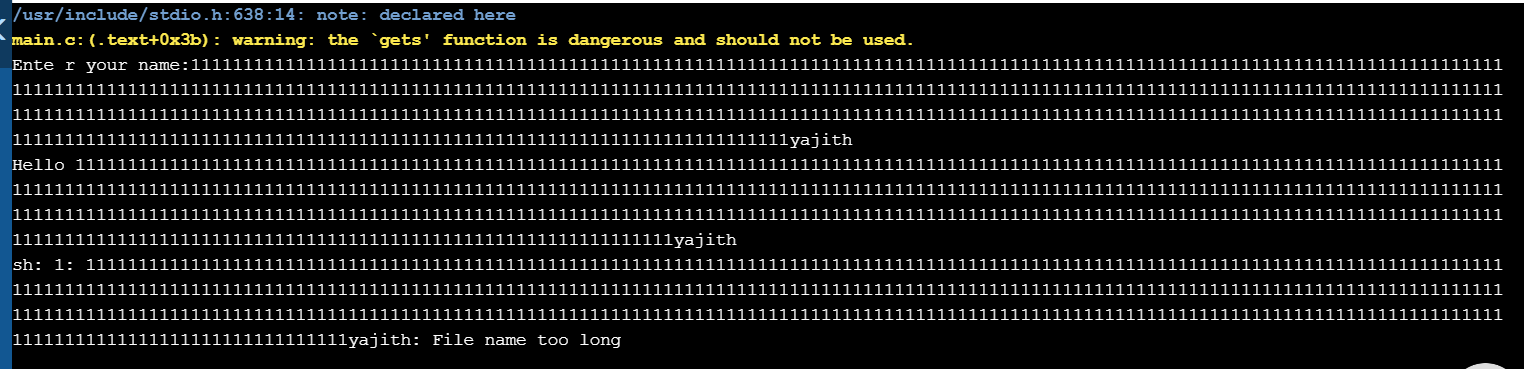


**Buffer Overflow :**

Input:



Output:



Inferences:

The input string executed the linux command in a shell because the input exceeded the size and overflows the buffer.

**Stack based Overflow:**

#include <stdio.h>

#include <stdlib.h>

void fun(int x)

{

if (x == 1)

return;

x = 2;

fun(x);

}

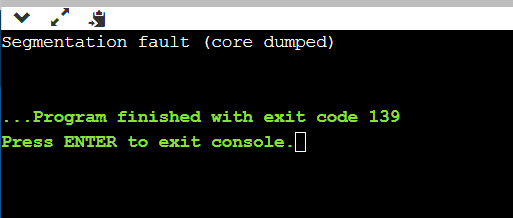
int main()

{

fun(25);

return 0;

}



**Heap based Overflow:**

#include <stdio.h>

#include <stdlib.h>

int main()

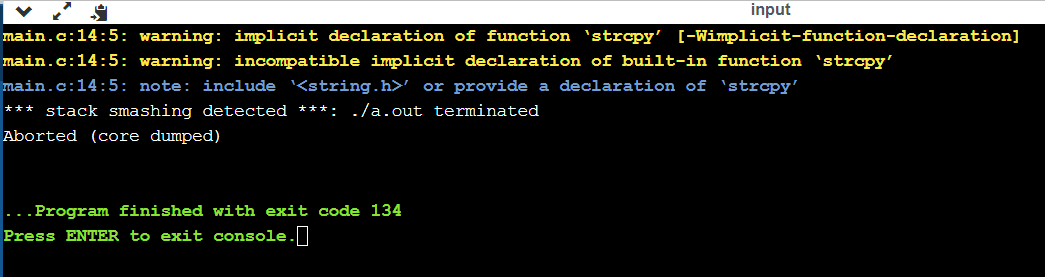
{

char buff[10] = {0};

strcpy(buff, "IS Lab buffer overflow attack example");

return 0;

}



|  |
| --- |
| **RESULT:** |

Thus, Bufferoverflow has been demonstrated with a sample program

|  |
| --- |
| **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 | **Configuration of Firewalls in system environment** |

|  |
| --- |
| **AIM:** |

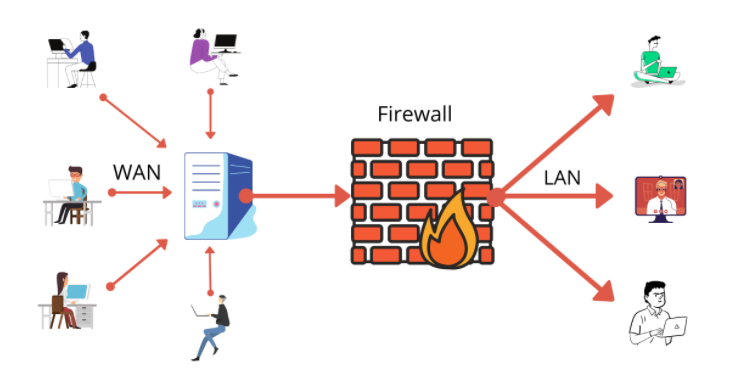
To configure the firewall in Windows operating system environment.

|  |
| --- |
| **THEORY:** |

**Firewall**

A **firewall** is a network security system that monitors and controls incoming and outgoing network traffic based on predetermined security rules.

A **firewall** typically establishes a barrier between a trusted network and an untrusted network, such as the Internet.



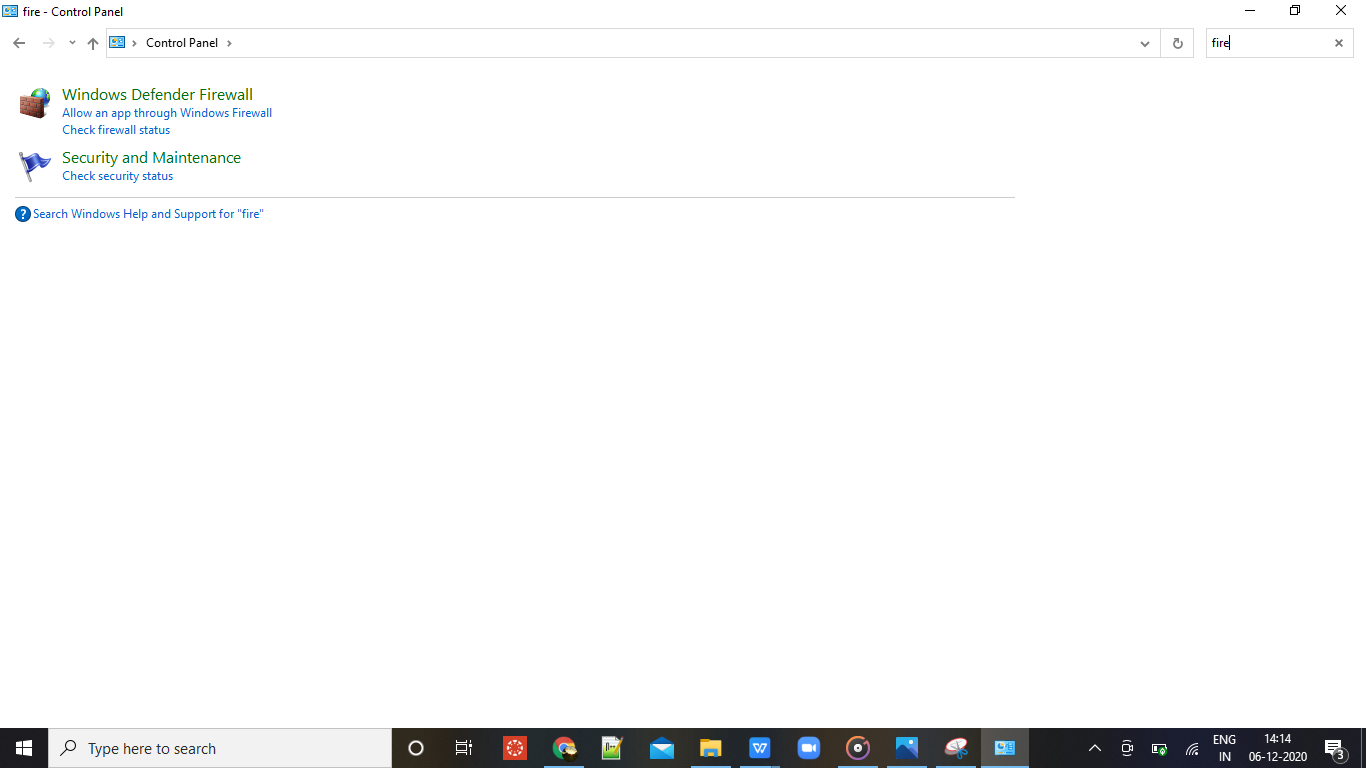
**Types of Firewall**

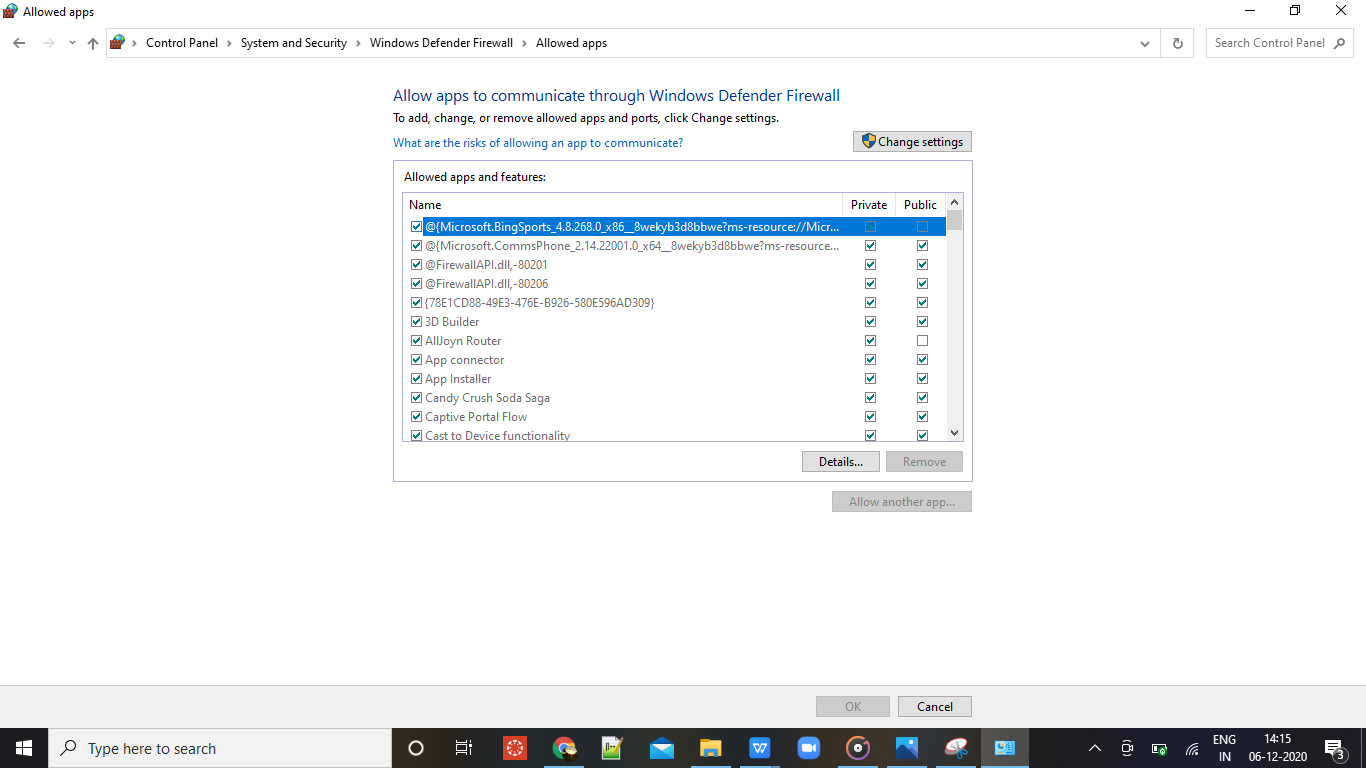
* Packet-filtering firewalls.
* Circuit-level gateways.
* Stateful inspection firewalls.
* Application-level gateways (a.k.a. proxy firewalls)
* Next-gen firewalls.
* Software firewalls.
* Hardware firewalls.
* Cloud firewalls.

|  |
| --- |
| **Screenshots of Configuration Steps** |

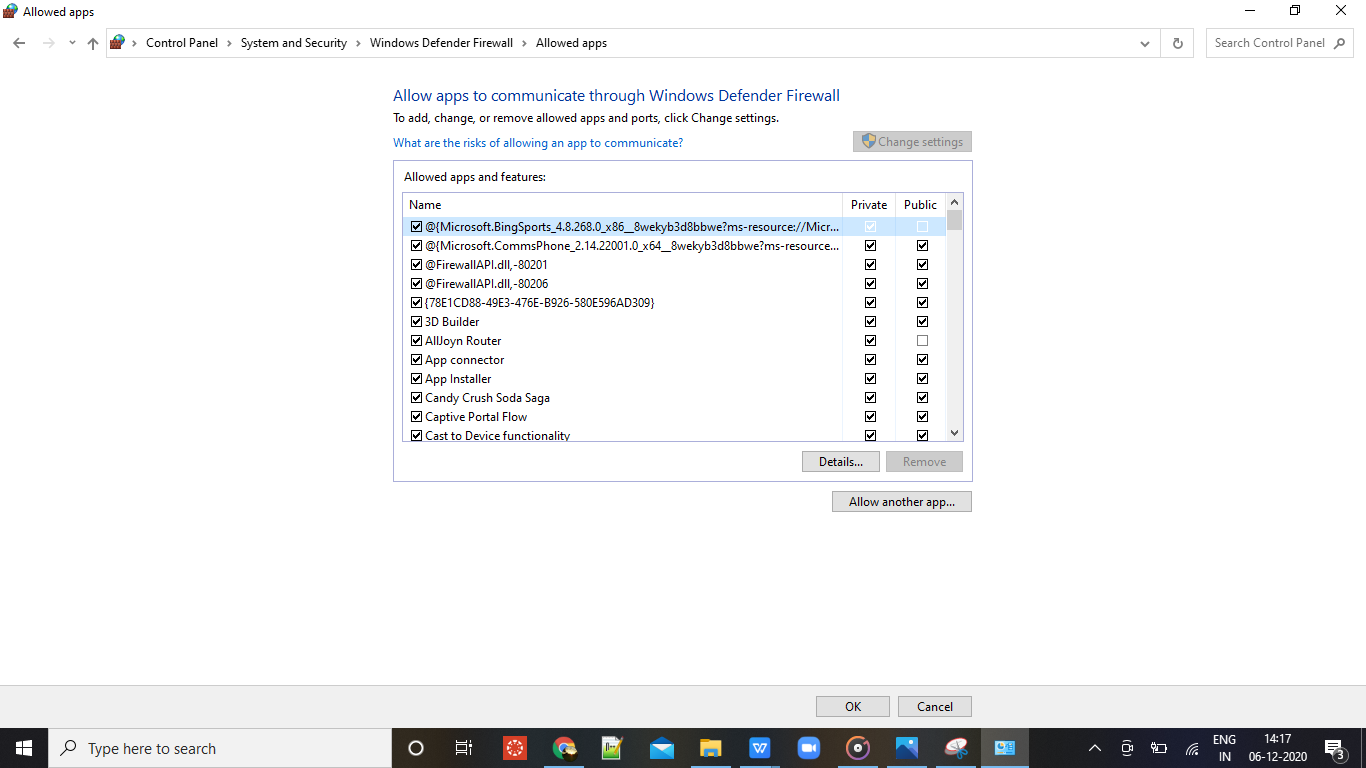
**To open Windows Firewall we can go to Start > Control Panel > Windows Firewall.**

**Step 1: Control Panel**

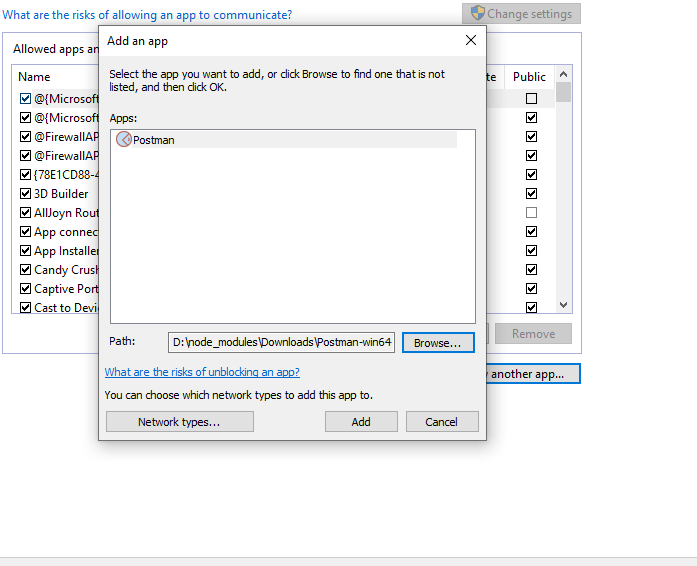




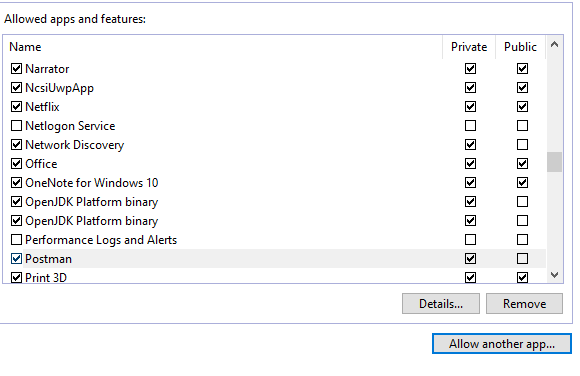
**Step 2: Click Change Settings**

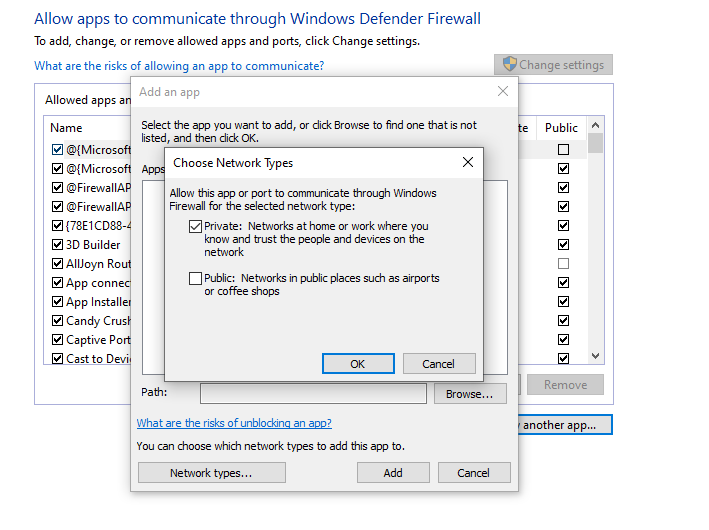


**Step 3: add programs to configure**

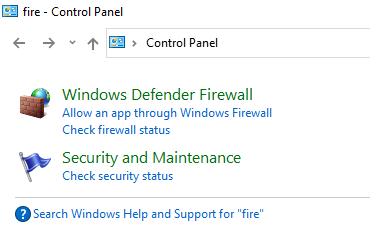


**Postman Is Added**





**Step 4: Click the Status of firewall**



**Step 5:**



|  |
| --- |
| **RESULT:** |

Thus Firewall has been configured for Windows operating systems environment.

|  |
| --- |
| **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 |  |  |