Course Outcome
CO2

DELTA Pg No. 1
Date 07/09/21

PRACTICAL 1

Aim: To implement firewall through App to login into bank-site; to implement E-commerce, debit card transaction through payment galeway.

Language used: HTML, MySQL.

Theory: A firewall is a dedicated hardware, or software or a combination of both, which inspects network traffic passing through it, and denies or permits passage based on set of rules.

Characteristics of firewall →

Capabilities:
- A firewall defines a single choke point that keeps unauthorized users out the protected network.
- A firewall provides a location for monitoring security-related events. Audits and alarms can be implemented on the firewall system.
- A firewall is a convenient platform for several Internet functions that are not security related.
- A firewall can serve as the platform for IPSec. Using the tunnel mode capability, the firewall can be used to implement virtual private nws.

Limitations:
- The firewall cannot protect against attacks that bypass the firewall.
- The firewall doesnot protect against internal threats.
- The firewall cannot protect against the transfer of virus-infected programs or files.

Methods of control in firewall

- User control → Only authorized users are having access to the other side.
- Access control → The access over the firewall is restricted to certain services. A service is characterized e.g. by IP address & port number.
- Behaviour control → For an application, the allowed usage scenarios are known. E.g. filters for e-mail attachments (virus removing).
- Direction control → Different rules for traffic into the Intranet & outgoing traffic to the Internet can be defined.

Types of Firewall

→ Packet filtering → It is the simplest packet screening method. A packet filtering firewall does exactly what its name implies – it filters packets. The most common implementation is on a router or dual-homed gateway.

→ Application Gateways / Proxies → An application gateway / proxy is considered by many to be the most complex packet screening method. This type of firewall is usually implemented on a secure host system configured with two network interfaces.

→ Circuit-level Gateway → Unlike a packet filtering firewall, a circuit-level gateway doesnot examine individual packets. Instead, circuit-level gateways monitor TCP or UDP sessions. Once a session has been established, it leaves the port open to allow all other packets belonging to that session to pass.

SOURCE CODE:

1) For the database

```
  define('DB_SERVER', 'localhost:3036');
  define('DB_USERNAME', 'root');
  define('DB_PASSWORD', 'rootpassword');
  define('DB_DATABASE', 'database');
  $db = mysqli_connect(DB_SERVER,DB_USERNAME,DB_PASSWORD,DB_DATABASE);
?>
<?php
  include("config.php");
  session_start();

  if($_SERVER["REQUEST_METHOD"] == "POST") {
    // username and password sent from form

    $result = mysqli_query($db,$sql);
    $row = mysqli_fetch_array($result,MYSQLI_ASSOC);
    $active = $row['active'];

    $count = mysqli_num_rows($result);

    // If result matched $myusername and $mypassword, table row must be 1 row

    if($count == 1) {
      session_register("myusername");
      $_SESSION['login_user'] = $myusername;

      header("location: welcome.php");
    }else {
      $error = "Your Login Name or Password is invalid";
    }
```

b) For the login page:

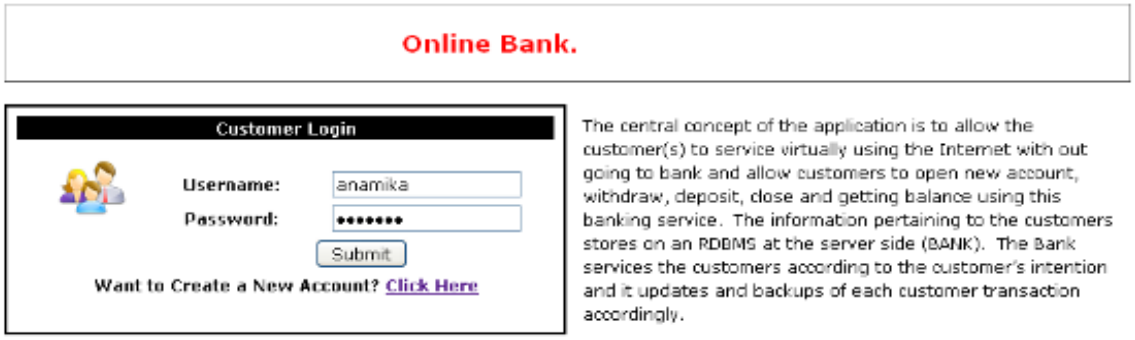
```
    <div align = "center">
      <div style = "width:300px; border: solid 1px #333333; " align = "left">
        <div style = "background-color:#333333; color:#FFFFFF;
padding:3px;"><b>Login</b></div>

        <div style = "margin:30px">

          <form action = "" method = "post">
            <label>UserName  :</label><input type = "text" name = "username" class =
"box"/><br /><br />
```

```
            <label>Password  :</label><input type = "password" name = "password" class = "box" /><br/><br />
            <input type = "submit" value = " Submit "/><br />
          </form>
```

c) Session:

```
  $ses_sql = mysqli_query($db,"select username from admin where username = '$user_check' ");

  $row = mysqli_fetch_array($ses_sql,MYSQLI_ASSOC);

  $login_session = $row['username'];

  if(!isset($_SESSION['login_user'])){
    header("location:login.php");
  }
?>
```

OUTPUT:



RESULT: Successfully studied and implemented Firewall.

EXPECTED OUTPUT ATTAINED: YES

Course Outcome
CO3

DELTA Pg No. 3
Date 14/09/21

PRACTICAL 2

Aim: To implement Transposition cipher.

Software used: VSCode

Theory: In cryptography, a transposition cipher is a method of encryption by which the positions held by units of plaintext (which are commonly characters or groups of characters) are shifted according to a regular system, so that the ciphertext constitutes a permutation of the plaintext. That is, the order of the units is changed. Mathematically a bijective function is used on the characters' positions to encrypt and an inverse function to decrypt.

Rail fence cipher

The Rail fence cipher is a form of transposition cipher that gets its name from the way in which it is encoded. In the rail fence cipher, the plaintext is written downwards on successive "rails" of an imaginary fence, then moving up when we get to the bottom. The message is then read off in rows. For example, using three "rails" and a message of 'WE ARE DISCOVERED. FLEE AT ONCE', the cipher writes out:

```
W . . . E . . . C . . . R . . . L . . . T . . . E
. E . R . D . S . O . E . E . F . E . A . O . C .
. . A . . . I . . . V . . . D . . . E . . . N . .
```

Then reads off:

WECRL TEERD SOEEF EAOCA IVDEN

Route cipher

In a route cipher the plain text is first written out in grid of given dimensions, then read off in a pattern given in the key. For example, using the same plaintext that we used for rail fence:

```
WRIORFEOE
EESVELANJ
ADCEDETCX
```

The key might specify "spiral inwards, clockwise, starting from the top right". That would give a cipher text:

EJXCTEDECDAEWRIORFEONALEVSE

Route ciphers have many more keys than the rail fence. In fact, for messages of reasonable length, the no. of possible keys is potentially too great to be enumerated even by modern machinery. However, not all keys are good equally. Badly chosen routes will leave excessive chunks of plaintext, or text simply reversed, & this will give cryptanalysts a clue as to the routes.

Columnar Transposition

In a columnar transposition, the message is written out in rows of a fixed length, & then read out again column by column, & the columns are chosen in some scrambled order. Both the width of the rows & the permutation of the columns are usually defined by a keyword. For example, the word ZEBRAS is of length 6 (so the rows of length 6), & the permutation is defined by the alphabetical order of the letters in the keyword. In this case, the order

would be "6 3 2 4 1 5".

In a regular columnar transposition cipher, any spare spaces are filled with nulls; in an irregular columnar transposition cipher, the spaces are left blank. Finally, the message is read off in columns, in the order specified by the keyword. For the message: WE ARE DISCOVERED. FLEE AT ONCE.

Regular columnar transposition

6	3	2	4	1	5
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	F	L	E
E	A	T	O	N	C
E	Q	K	J	E	U

Providing 5 nulls (QKJEU) at the end.

→ EVLNE ACDTK ESEAQ ROFOJ DEECU WIREE

Irregular transposition

6	3	2	4	1	5
W	E	A	R	E	D
I	S	C	O	V	E
R	E	D	F	L	E
E	A	T	O	N	C
E					

columns are not completed by nulls

→ EVLNA CDTES EAROF ODEEC WIREE

Columnar transposition is used for serious purposes as a component of more complex cipher atleast into the 1920's.

Detection & cryptanalysis

Since transposition does not affect the frequency of individual symbols, simple transposition can be easily detected by the cryptanalyst by doing a frequent count. If the ciphertext exhibits a frequency distribution very similar to plaintext, it is mostly a transposition. Once such anagrams have been found, they reveal information about the transposition pattern, and can consequently be extended.

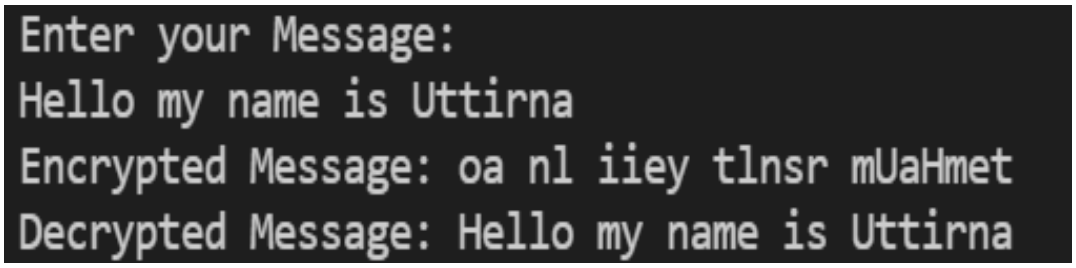
SOURCE CODE:

```
#include<bits/stdc++.h>
#include<string>
using namespace std;

string const key = "ZEBRAS";
map<int,int> keyMap;

void setPermutationOrder()
{
	for(int i=0; i < key.length(); i++)
	{
		keyMap[key[i]] = i;
	}
}
string encryptMessage(string msg)
{
	int row, col, j;
	string cipher = "";
	col = key.length();
	row = msg.length()/col;
	if (msg.length() % col)
		row += 1;
	char matrix[row][col];

	for (int i=0,k=0; i < row; i++)
	{
		for (int j=0; j<col; )
		{
			if(msg[k] == '\0')
			{
				matrix[i][j] = '_';
				j++;
			}
			if( isalpha(msg[k]) || msg[k]==' ')
			{
				matrix[i][j] = msg[k];
				j++;
			}
			k++;
		}
	}
	for (map<int,int>::iterator ii = keyMap.begin(); ii!=keyMap.end(); ++ii)
```

```
	{
		j=ii->second;
		for (int i=0; i<row; i++)
		{
			if( isalpha(matrix[i][j]) || matrix[i][j]==' ' || matrix[i][j]=='_')
				cipher += matrix[i][j];
		}
	}
	return cipher;
}
string decryptMessage(string cipher)
{
	int col = key.length();
	int row = cipher.length()/col;
	char cipherMat[row][col];

	for (int j=0,k=0; j<col; j++)
		for (int i=0; i<row; i++)
			cipherMat[i][j] = cipher[k++];

	int index = 0;
	for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
		ii->second = index++;

	char decCipher[row][col];
	map<int,int>::iterator ii=keyMap.begin();
	int k = 0;
	for (int l=0,j; key[l]!='\0'; k++)
	{
		j = keyMap[key[l++]];
		for (int i=0; i<row; i++)
		{
			decCipher[i][k]=cipherMat[i][j];
		}
	}
	string msg = "";
	for (int i=0; i<row; i++)
	{
		for(int j=0; j<col; j++)
		{
			if(decCipher[i][j] != '_')
				msg += decCipher[i][j];
		}
	}
```

```
        return msg;
}

int main(void)
{
cout << "Encrypted Message: " << cipher << endl;
string msg;
getline(cin, msg);
        setPermutationOrder();
        string cipher = encryptMessage(msg);
        cout << "Encrypted Message: " << cipher << endl;
        cout << "Decrypted Message: " << decryptMessage(cipher) << endl;
        return 0;
}
```

OUTPUT:



RESULT: Successfully studied and implemented Transposition Cipher.

EXPECTED OUTPUT ATTAINED: YES

Course Outcome
CO5

DELTA Pg No. 6
Date / /

PRACTICAL 5

Aim: To implement RSA Algorithm

Software used: VS Code

Theory: The RSA Algorithm is named after Ron Rivest, Adi Shamir & Len Adleman, who invited it in 1977. The basic technique was first ~~discov~~ discovered in 1973 by Clifford Cocks of CESG (part of the British GCHQ) but this was a secret until 1997. The patent taken out by RSA Labs has expired.

The RSA algorithm can be used for both public key encryption & digital signatures. Its security is based on the difficulty of factoring large integers.

Key Generation Algorithm

1) Generate two large random primes, p & q, of approx. equal size such that their product n = pq is of the required bit length, e.g. 1024 bits.
2) Compute n = pq and (phi) $\phi = (p-1)(q-1)$
3) Choose an integer e, $1 < e < phi$, such that: $gcd(e, phi) = 1$.
4) Compute the secret exponent d, $1 < d \leq phi$, such that $ed \equiv 1 \pmod{phi}$.
5) The public key is (n, e) & the private key (d, p, q). Keep all the values d, p, q & phi secret.

→ n is known as the modules.
→ e is known as the public exponent or encryption exponent.
→ d is known as the secret exponent or decryption exponent.

Encryption: Sender A does the following:-

1) Obtain the recipient B's public key (n, e).
2) Represents the plain text message as a positive integer m, $1 < m < n$.
3) Computes the cipher text $c = m^e \bmod n$.
4) Sends the cipher text c to B.

Decryption: Recipient B does the following:-

1) Uses his private key (n, d) to compute $m = c^d \bmod n$.
2) Extracts the plaintext from the message representative m.

Sign. verification: Recipient B does the following:-

1) Uses sender A's public key (n, e) to compute integer $v = s^e \bmod n$.
2) Extracts the message digest from this integer.
3) Independently computes the message digest of the information that has been signed.
4) If both message digests are identical, the signature is valid.

Digital signing: Sender A does the following:-

1) Creates a message digest of the information to be sent.
2) Represents this digest as an integer m between 1 and $n-1$.
3) Uses their private key (n, d) to compute the signature $s = m^d \bmod n$.
4) Sends this signature s to the recipient, B.

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;

long int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100], en[100], i;
char msg[1000];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();

int prime(long int pr)
{
int i;
   j = sqrt(pr);
for (i = 2; i <= j; i++)
   {
if (pr % i == 0)
return 0;
   }
return 1;
}

int main()
{
cout<< "ENTER FIRST PRIME NUMBER: "<<endl;
cin>> p;
flag = prime(p);
if (flag == 0)
   {
cout<< "WRONG INPUT!"<<endl;
exit(1);
   }
cout<< "ENTER ANOTHER PRIME NUMBER: "<<endl;
cin>> q;
flag = prime(q);
if (flag == 0 || p == q)
   {
cout<< "WRONG INPUT!"<<endl;
exit(1);
   }
cout<< "ENTER MESSAGE: "<<endl;
fflush(stdin);
cin>>msg;
for (i = 0; msg[i] != NULL; i++)
```

```
m[i] = msg[i];
   n = p * q;
   t = (p - 1) * (q - 1);
ce();
cout<< "POSSIBLE VALUES OF en AND de ARE: "<<endl;
for (i = 0; i < j - 1; i++)
cout<< e[i] << "\t" << d[i] << "\n";
encrypt();
decrypt();
return 0;
}

void ce()
{
int k;
   k = 0;
for (i = 2; i < t; i++)
   {
if (t % i == 0)
continue;
flag = prime(i);
if (flag == 1 && i != p && i != q)
      {
e[k] = i;
flag = cd(e[k]);
if (flag > 0)
         {
d[k] = flag;
k++;
         }
if (k == 99)
break;
      }
   }
}
long int cd(long int x)
{
long int k = 1;
while (1)
   {
      k = k + t;
if (k % x == 0)
return (k / x);
   }
}
```

```
void encrypt()
{
long int pt, ct, key = e[0], k, len;
   i = 0;
len = strlen(msg);
while (i != len)
   {
pt = m[i];
pt = pt - 96;
      k = 1;
for (j = 0; j < key; j++)
      {
         k = k * pt;
         k = k % n;
      }
temp[i] = k;
ct = k + 96;
en[i] = ct;
i++;
   }
en[i] = -1;
cout<< "THE ENCRYPTED MESSAGE IS: "<<endl;
for (i = 0; en[i] != -1; i++)
printf("%c", en[i]);
}
void decrypt()
{
long int pt, ct, key = d[0], k;
   i = 0;
while (en[i] != -1)
   {
ct = temp[i];
      k = 1;
for (j = 0; j < key; j++)
      {
         k = k * ct;
         k = k % n;
      }
pt = k + 96;
m[i] = pt;
i++;
   }
m[i] = -1;
cout<< "\nTHE DECRYPTED MESSAGE IS: "<<endl;
for (i = 0; m[i] != -1; i++)
printf("%c", m[i]);
}
```

OUTPUT:

```
ENTER FIRST PRIME NUMBER:
73
ENTER ANOTHER PRIME NUMBER:
31
ENTER MESSAGE:
Uttirna
POSSIBLE VALUES OF en AND de ARE:
7       1543
11      1571
13      997
17      2033
19      1819
23      1127
29      149
37      1693
41      1001
43      1507
47      1103
53      1997
59      659
61      1381
67      1483
71      791
79      1039
83      1067
89      1529
97      913
101     941
103     1447
THE ENCRYPTED MESSAGE IS:
é⌠⌠Bû↔a
THE DECRYPTED MESSAGE IS:
Uttirna
```

OUTPUT: Successfully studied and implemented RSA Algorithm.

EXPECTED OUTPUT ATTAINED: YES

Course Outcome CO4

DELTA Pg No. 8
Date 28/09/21

PRACTICAL 4

Aim: To implement DES Algorithm.

Software used: VS Code

Theory: DES (Data Encryption Standard) put forward by NIST. This is based on Feistel cipher (Product cipher). DES uses 64 bit plaintext and 56 bit key (48 bit sub key).

DES is the archetypal block cipher - an algorithm that takes a fixed-length string of plaintext bits & transforms it through a series of complicated operations into another ciphertext bitstring of the same length. In the case of DES, the block size is 64 bits.

DES encrypts and decrypts data in 64 bits blocks, using a 64-bit key. It takes a 64-bit block of plaintext as input and outputs a 64-bit block of ciphertext. Since it always operates on blocks of equal size & it uses both permutations & substitutions in the algorithm, DES is both a block cipher & a product cipher.

DES has 16 rounds, meaning the main algorithm is repeated 16 times to produce the ciphertext. It has been found that the no. of rounds is exponentially proportional to the amount of time required to find a key using a brute-force attack. So as the no. of rounds increases, the security of the algorithm increases exponentially.

SOURCE CODE:

```
#include <bits/stdc++.h>
using namespace std;
string hex2bin(string s)
{
	// hexadecimal to binary conversion
	unordered_map<char, string> mp;
	mp['0'] = "0000";
	mp['1'] = "0001";
	mp['2'] = "0010";
	mp['3'] = "0011";
	mp['4'] = "0100";
	mp['5'] = "0101";
	mp['6'] = "0110";
	mp['7'] = "0111";
	mp['8'] = "1000";
	mp['9'] = "1001";
	mp['A'] = "1010";
	mp['B'] = "1011";
	mp['C'] = "1100";
	mp['D'] = "1101";
	mp['E'] = "1110";
	mp['F'] = "1111";
	string bin = "";
	for (int i = 0; i < s.size(); i++) {
		bin += mp[s[i]];
	}
	return bin;
}
string bin2hex(string s)
{
	// binary to hexadecimal conversion
	unordered_map<string, string> mp;
	mp["0000"] = "0";
	mp["0001"] = "1";
	mp["0010"] = "2";
	mp["0011"] = "3";
	mp["0100"] = "4";
	mp["0101"] = "5";
	mp["0110"] = "6";
	mp["0111"] = "7";
	mp["1000"] = "8";
	mp["1001"] = "9";
	mp["1010"] = "A";
	mp["1011"] = "B";
	mp["1100"] = "C";
```

```
	mp["1101"] = "D";
	mp["1110"] = "E";
	mp["1111"] = "F";
	string hex = "";
	for (int i = 0; i < s.length(); i += 4) {
		string ch = "";
		ch += s[i];
		ch += s[i + 1];
		ch += s[i + 2];
		ch += s[i + 3];
		hex += mp[ch];
	}
	return hex;
}

string permute(string k, int* arr, int n)
{
	string per = "";
	for (int i = 0; i < n; i++) {
		per += k[arr[i] - 1];
	}
	return per;
}

string shift_left(string k, int shifts)
{
	string s = "";
	for (int i = 0; i < shifts; i++) {
		for (int j = 1; j < 28; j++) {
			s += k[j];
		}
		s += k[0];
		k = s;
		s = "";
	}
	return k;
}

string xor_(string a, string b)
{
	string ans = "";
	for (int i = 0; i < a.size(); i++) {
		if (a[i] == b[i]) {
			ans += "0";
		}
		else {
			ans += "1";
```

```
                }
        }
        return ans;
}
string encrypt(string pt, vector<string> rkb, vector<string> rk)
{
        // Hexadecimal to binary
        pt = hex2bin(pt);

        // Initial Permutation Table
        int initial_perm[64] = { 58, 50, 42, 34, 26, 18, 10, 2,
                                                        60, 52, 44, 36, 28, 20, 12, 4,
                                                        62, 54, 46, 38, 30, 22, 14, 6,
                                                        64, 56, 48, 40, 32, 24, 16, 8,
                                                        57, 49, 41, 33, 25, 17, 9, 1,
                                                        59, 51, 43, 35, 27, 19, 11, 3,
                                                        61, 53, 45, 37, 29, 21, 13, 5,
                                                        63, 55, 47, 39, 31, 23, 15, 7 };
        // Initial Permutation
        pt = permute(pt, initial_perm, 64);
        cout << "After initial permutation: " << bin2hex(pt) << endl;

        // Splitting
        string left = pt.substr(0, 32);
        string right = pt.substr(32, 32);
        cout << "After splitting: L0=" << bin2hex(left)
                << " R0=" << bin2hex(right) << endl;

        // Expansion D-box Table
        int exp_d[48] = { 32, 1, 2, 3, 4, 5, 4, 5,
                                        6, 7, 8, 9, 8, 9, 10, 11,
                                        12, 13, 12, 13, 14, 15, 16, 17,
                                        16, 17, 18, 19, 20, 21, 20, 21,
                                        22, 23, 24, 25, 24, 25, 26, 27,
                                        28, 29, 28, 29, 30, 31, 32, 1 };

        // S-box Table
        int s[8][4][16] = { { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7,
                                0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8,
                                4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0,
                                15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 },
                            { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10,
                                3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5,
                                0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15,
                                13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 },
                            { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8,
                                13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1,
```

```
                                13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7,
                                 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 },
                               { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15,
                                13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9,
                                10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4,
                                3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 },
                               { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9,
                                14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6,
                                4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14,
                                11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 },
                               { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11,
                                10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8,
                                9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6,
                                4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 },
                              { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1,
                                13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6,
                                1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2,
                                6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 },
                              { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7,
                                1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2,
                                7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8,
                                2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

        // Straight Permutation Table
        int per[32] = { 16, 7, 20, 21,
                                        29, 12, 28, 17,
                                        1, 15, 23, 26,
                                        5, 18, 31, 10,
                                        2, 8, 24, 14,
                                        32, 27, 3, 9,
                                        19, 13, 30, 6,
                                        22, 11, 4, 25 };

        cout << endl;
        for (int i = 0; i < 16; i++) {
                // Expansion D-box
                string right_expanded = permute(right, exp_d, 48);

                // XOR RoundKey[i] and right_expanded
                string x = xor_(rkb[i], right_expanded);

                // S-boxes
                string op = "";
                for (int i = 0; i < 8; i++) {
                        int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');
                        int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] - '0') + 2 * int(x[i *
6 + 3] - '0') + int(x[i * 6 + 4] - '0');
```

```
                int val = s[i][row][col];
                op += char(val / 8 + '0');
                val = val % 8;
                op += char(val / 4 + '0');
                val = val % 4;
                op += char(val / 2 + '0');
                val = val % 2;
                op += char(val + '0');
            }
            // Straight D-box
            op = permute(op, per, 32);

            // XOR left and op
            x = xor_(op, left);

            left = x;

            // Swapper
            if (i != 15) {
                swap(left, right);
            }
            cout << "Round " << i + 1 << " " << bin2hex(left) << " "
                << bin2hex(right) << " " << rk[i] << endl;
        }

        // Combination
        string combine = left + right;

        // Final Permutation Table
        int final_perm[64] = { 40, 8, 48, 16, 56, 24, 64, 32,
                                39, 7, 47, 15, 55, 23, 63, 31,
                                38, 6, 46, 14, 54, 22, 62, 30,
                                37, 5, 45, 13, 53, 21, 61, 29,
                                36, 4, 44, 12, 52, 20, 60, 28,
                                35, 3, 43, 11, 51, 19, 59, 27,
                                34, 2, 42, 10, 50, 18, 58, 26,
                                33, 1, 41, 9, 49, 17, 57, 25 };

        // Final Permutation
        string cipher = bin2hex(permute(combine, final_perm, 64));
        return cipher;
}
int main()
{
        // pt is plain text
        string pt, key;
        /*cout<<"Enter plain text(in hexadecimal): ";
```

```
cin>>pt;
cout<<"Enter key(in hexadecimal): ";
cin>>key;*/

pt = "123456ABCD132536";
key = "AABB09182736CCDD";
// Key Generation

// Hex to binary
key = hex2bin(key);

// Parity bit drop table
int keyp[56] = { 57, 49, 41, 33, 25, 17, 9,
                 1, 58, 50, 42, 34, 26, 18,
                 10, 2, 59, 51, 43, 35, 27,
                 19, 11, 3, 60, 52, 44, 36,
                 63, 55, 47, 39, 31, 23, 15,
                 7, 62, 54, 46, 38, 30, 22,
                 14, 6, 61, 53, 45, 37, 29,
                 21, 13, 5, 28, 20, 12, 4 };

// getting 56 bit key from 64 bit using the parity bits
key = permute(key, keyp, 56); // key without parity

// Number of bit shifts
int shift_table[16] = { 1, 1, 2, 2,
                        2, 2, 2, 2,
                        1, 2, 2, 2,
                        2, 2, 2, 1 };

// Key- Compression Table
int key_comp[48] = { 14, 17, 11, 24, 1, 5,
                     3, 28, 15, 6, 21, 10,
                     23, 19, 12, 4, 26, 8,
                     16, 7, 27, 20, 13, 2,
                     41, 52, 31, 37, 47, 55,
                     30, 40, 51, 45, 33, 48,
                     44, 49, 39, 56, 34, 53,
                     46, 42, 50, 36, 29, 32 };

// Splitting
string left = key.substr(0, 28);
string right = key.substr(28, 28);

vector<string> rkb; // rkb for RoundKeys in binary
vector<string> rk; // rk for RoundKeys in hexadecimal
for (int i = 0; i < 16; i++) {
```

```
            // Shifting
            left = shift_left(left, shift_table[i]);
            right = shift_left(right, shift_table[i]);

            // Combining
            string combine = left + right;

            // Key Compression
            string RoundKey = permute(combine, key_comp, 48);

            rkb.push_back(RoundKey);
            rk.push_back(bin2hex(RoundKey));
        }

        cout << "\nEncryption:\n\n";
        string cipher = encrypt(pt, rkb, rk);
        cout << "\nCipher Text: " << cipher << endl;

        cout << "\nDecryption\n\n";
        reverse(rkb.begin(), rkb.end());
        reverse(rk.begin(), rk.end());
        string text = encrypt(cipher, rkb, rk);
        cout << "\nPlain Text: " << text << endl;
}
```

OUTPUT:

Encryption:

After initial permutation: 14A7D67818CA18AD
After splitting: L0=14A7D678 R0=18CA18AD

Round 1 18CA18AD 5A78E394 194CD072DE8C
Round 2 5A78E394 4A1210F6 4568581ABCCE
Round 3 4A1210F6 B8089591 06EDA4ACF5B5
Round 4 B8089591 236779C2 DA2D032B6EE3
Round 5 236779C2 A15A4B87 69A629FEC913
Round 6 A15A4B87 2E8F9C65 C1948E87475E
Round 7 2E8F9C65 A9FC20A3 708AD2DDB3C0
Round 8 A9FC20A3 308BEE97 34F822F0C66D
Round 9 308BEE97 10AF9D37 84BB4473DCCC
Round 10 10AF9D37 6CA6CB20 02765708B5BF
Round 11 6CA6CB20 FF3C485F 6D5560AF7CA5
Round 12 FF3C485F 22A5963B C2C1E96A4BF3
Round 13 22A5963B 387CCDAA 99C31397C91F
Round 14 387CCDAA BD2DD2AB 251B8BC717D0
Round 15 BD2DD2AB CF26B472 3330C5D9A36D

Round 16 19BA9212 CF26B472 181C5D75C66D

Cipher Text: C0B7A8D05F3A829C

Decryption

After initial permutation: 19BA9212CF26B472
After splitting: L0=19BA9212 R0=CF26B472

Round 1 CF26B472 BD2DD2AB 181C5D75C66D
Round 2 BD2DD2AB 387CCDAA 3330C5D9A36D
Round 3 387CCDAA 22A5963B 251B8BC717D0
Round 4 22A5963B FF3C485F 99C31397C91F
Round 5 FF3C485F 6CA6CB20 C2C1E96A4BF3
Round 6 6CA6CB20 10AF9D37 6D5560AF7CA5
Round 7 10AF9D37 308BEE97 02765708B5BF
Round 8 308BEE97 A9FC20A3 84BB4473DCCC
Round 9 A9FC20A3 2E8F9C65 34F822F0C66D
Round 10 2E8F9C65 A15A4B87 708AD2DDB3C0
Round 11 A15A4B87 236779C2 C1948E87475E
Round 12 236779C2 B8089591 69A629FEC913
Round 13 B8089591 4A1210F6 DA2D032B6EE3
Round 14 4A1210F6 5A78E394 06EDA4ACF5B5
Round 15 5A78E394 18CA18AD 4568581ABCCE
Round 16 14A7D678 18CA18AD 194CD072DE8C

Plain Text: 123456ABCD132536

RESULT: Successfully implemented DES Algorithm.

EXPECTED OUTPUT ATTAINED: YES

Source outcome: CO4, CO5

DELTA Pg No. 9
Date 28/09/2021

PRACTICAL 5

Aim: To implement Diffie - Hellman Algorithm.

Software used: VS Code

Theory: Diffie Hellman key exchange algorithm uses asymmetric key principles for the distribution of symmetric keys to both parties in a communication network. Key distribution is an important aspect of conventional algorithm and the entire safety is dependent on the distribution of key using secured channel. Diffie Hellman utilizes the public and private key of asymmetric key cryptography to exchange the secret key.

The steps for Diffie Hellman key exchange algorithm are:

Step 1 → GLOBAL PUBLIC ELEMENTS

Select any prime no.: 'q'

Calculate the primitive root of q: 'a' such that $a < q$.

Step 2 → ASYMMETRIC KEY GENERATION BY USER 'A'

Select a random no. as the private key X_A where $X_A < q$.

Calculate the public key Y_A where $Y_A = a X_A \bmod q$

Step 3 → KEY GENERATION BY USER 'B'.

Select a random no. as the private key X_B where $X_B < q$.

Calculate the public key Y_B where $Y_B = a X_B \bmod q$.

Step 4 → Exchange the values of public key between A & B.

Step 5 → SYMMETRIC KEY (K) GENERATION BY USER 'A'

$K = Y_B X_A \bmod q$.

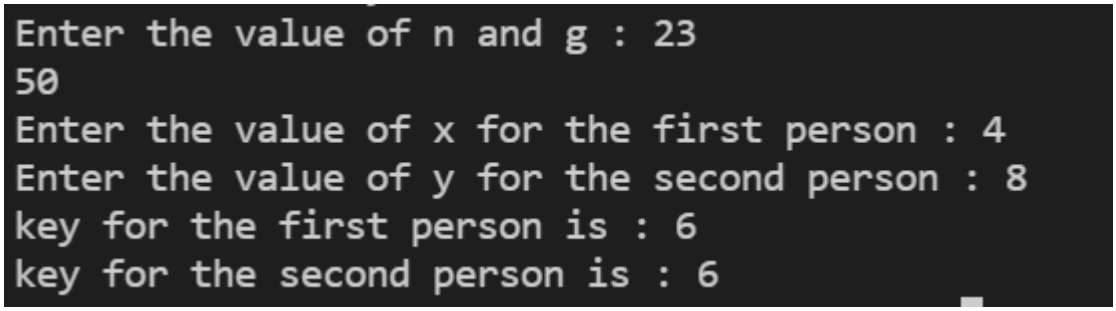
Step 6 → SYMMETRIC KEY (K) GENERATION BY USER 'B'.

$K = Y_A X_B \bmod q$.

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;

long int power(int a,int b,int mod)
 {
long long int t;
if(b==1)
return a;

t=power(a,b/2,mod);
if(b%2==0)
return (t*t)%mod;
else
return (((t*t)%mod)*a)%mod;
 }

long long int calculateKey(int a,int x,int n)
 {
return power(a,x,n);
 }

int main()
 {
int n,g,x,a,y,b;

// both the persons will be agreed upon the common n and g

cout<<"Enter the value of n and g : "<<endl;
cin>>n>>g;

// first person will choose the x

cout<<"Enter the value of x for the first person : "<<endl;
cin>>x;
a=power(g,x,n);

// second person will choose the y

cout<<"Enter the value of y for the second person : "<<endl;
cin>>y;
b=power(g,y,n);
cout<<"key for the first person is : "<<power(b,x,n)<<endl;
cout<<"key for the second person is : "<<power(a,y,n)<<endl;
return 0;
 }
```

OUTPUT:



RESULT: Successfully implemented Diffie Hellman Algorithm.

EXPECTED OUTPUT ATTAINED: YES

Course Outcome
CO4, CO5

DELTA Pg No. 10
Date 04/10/21

PRACTICAL 6

Aim: Make a study of any one simulation tool based on parameters of IS.

Theory: Nmap (Network mapper) is a security scanner originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich) used to discover hosts and services on a computer network, thus creating a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host & then analyzes the responses.
Nmap was originally a Linux-only utility, but it was ported to Windows, Solaris, HP-UX, BSD variants (including OS X), AmigaOS and IRIX. Linux is the most popular platform, followed closely by Windows.

Nmap features include:

1) Host discovery - Identifying hosts on a network. For example, listing the hosts that respond to TCP and/or ICMP requests.
2) Port scanning - Enumerating the open ports on target hosts.
3) Version detection - Interrogating the operating system and hardware characteristics of network devices.
4) OS detection - Determining the operating system and hardware characteristics of network devices.
5) Scriptable interaction with the target - using Nmap Scripting Engine (NSE) and Lua programming language.
6) Nmap can provide further information on targets, including reverse DNS names, device types and MAC addresses.

NMAP (ZENMAP)

Fig1: Nmap Logo

Graphical Interfaces: NmapFE, originally written by Zach Smith, was Nmap's official GUI for Nmap versions 2.2 to 4.22. For Nmap 4.50 (originally in the 4.22SOC development series) NmapFE was replacedwith Zenmap, a new official graphical user interface based on UMIT, developed by Adriano Monteiro Marques.

Various web-based interfaces allow controlling Nmap or analysing Nmap results from a web browser. Theseinclude LOCALSCAN,nmap-web, and Nmap-CGI.

Microsoft Windows specific GUIs exist, including NMapWin, which has not been updated since June 2003(v1.4.0), and NmapW by Syhunt.

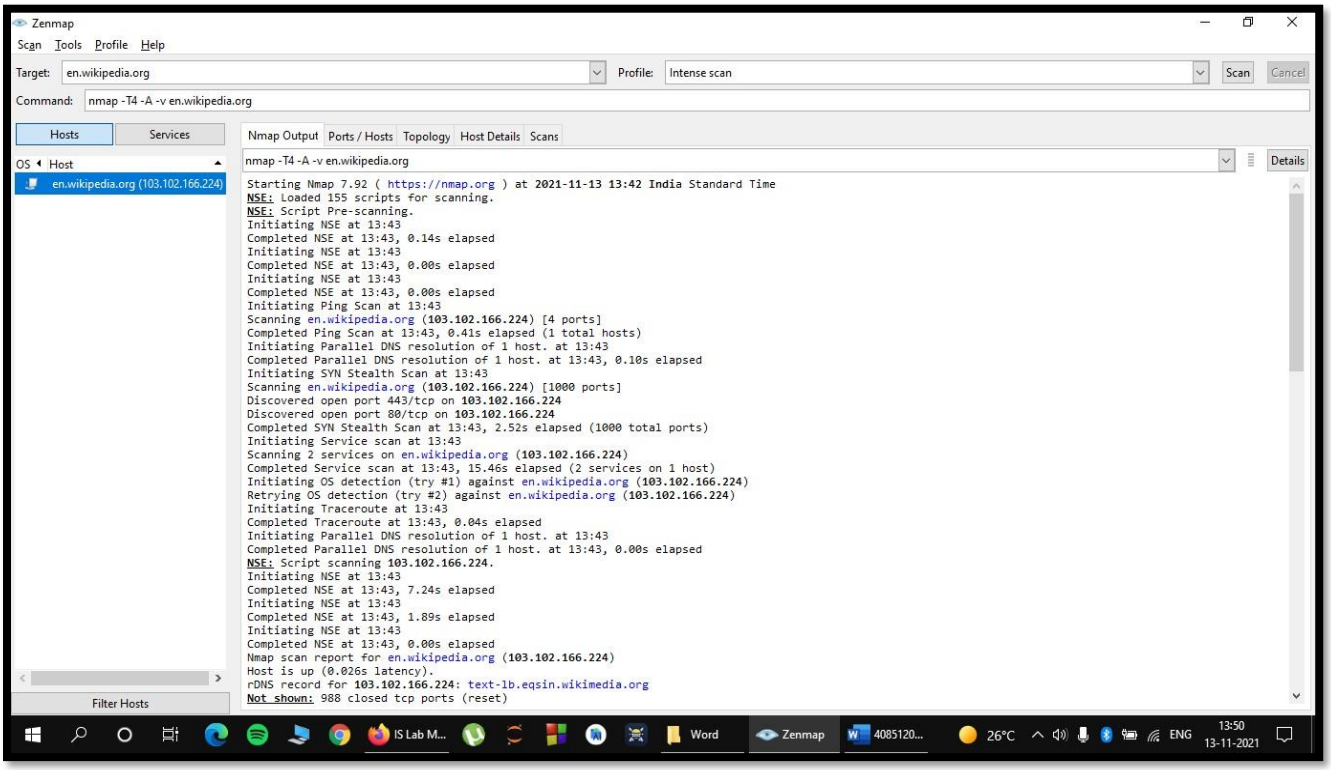


Fig2: Results of port scan against Wikipedia

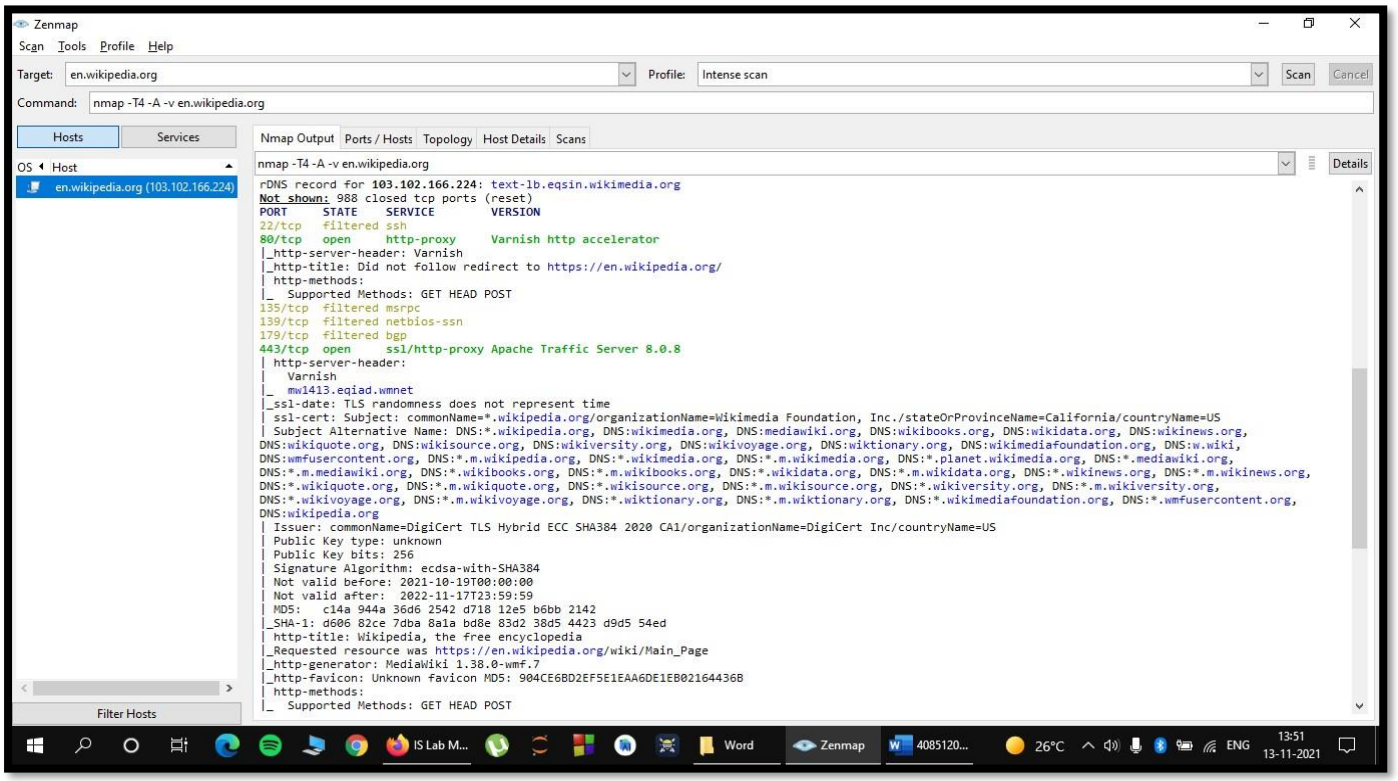


Fig3: Results of port scan against [Wikipedia]

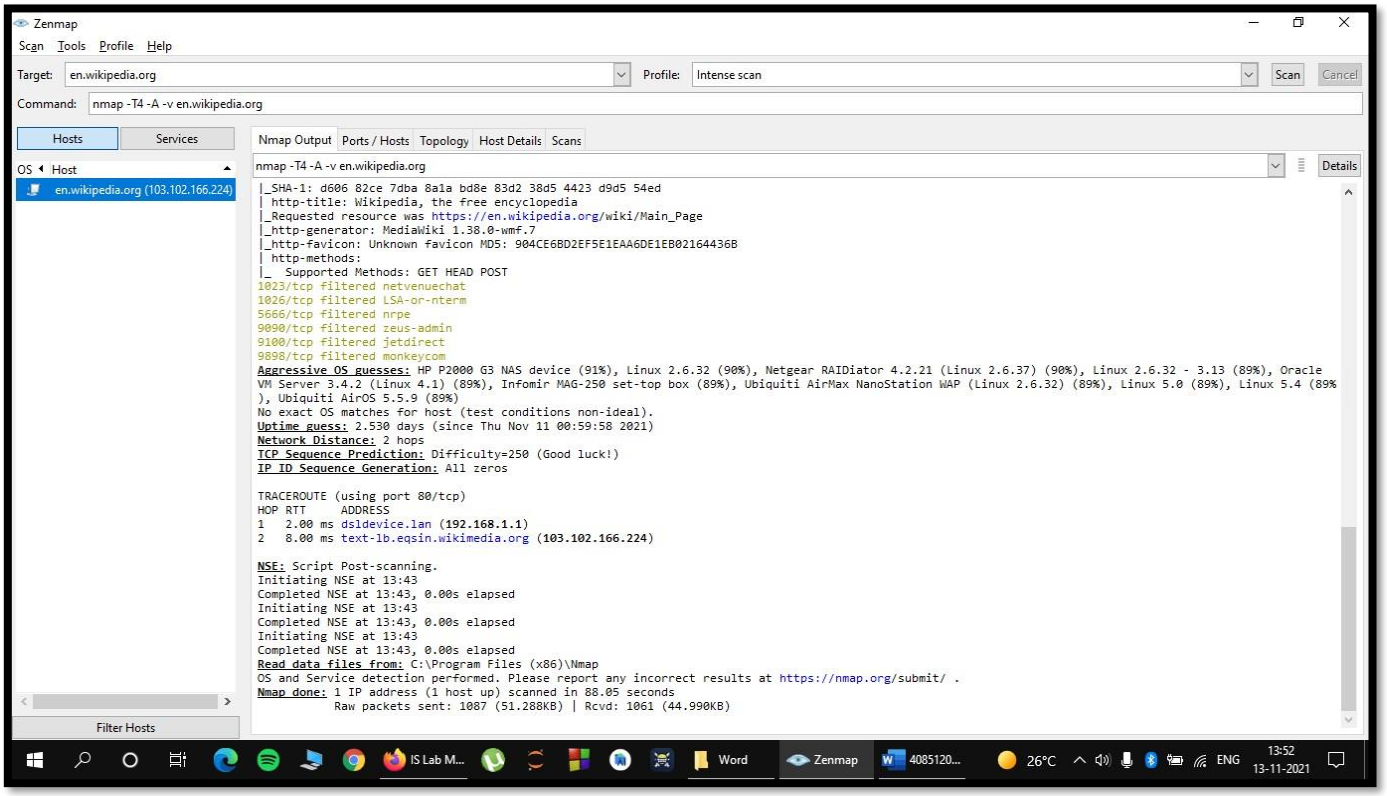


Fig4: Results of port scan against [Wikipedia]

CONCLUSION: Successfully studied and understood a simulation tool (Nmap) based on the parameters of Information Security.

Course Outcome
CO4, CO3

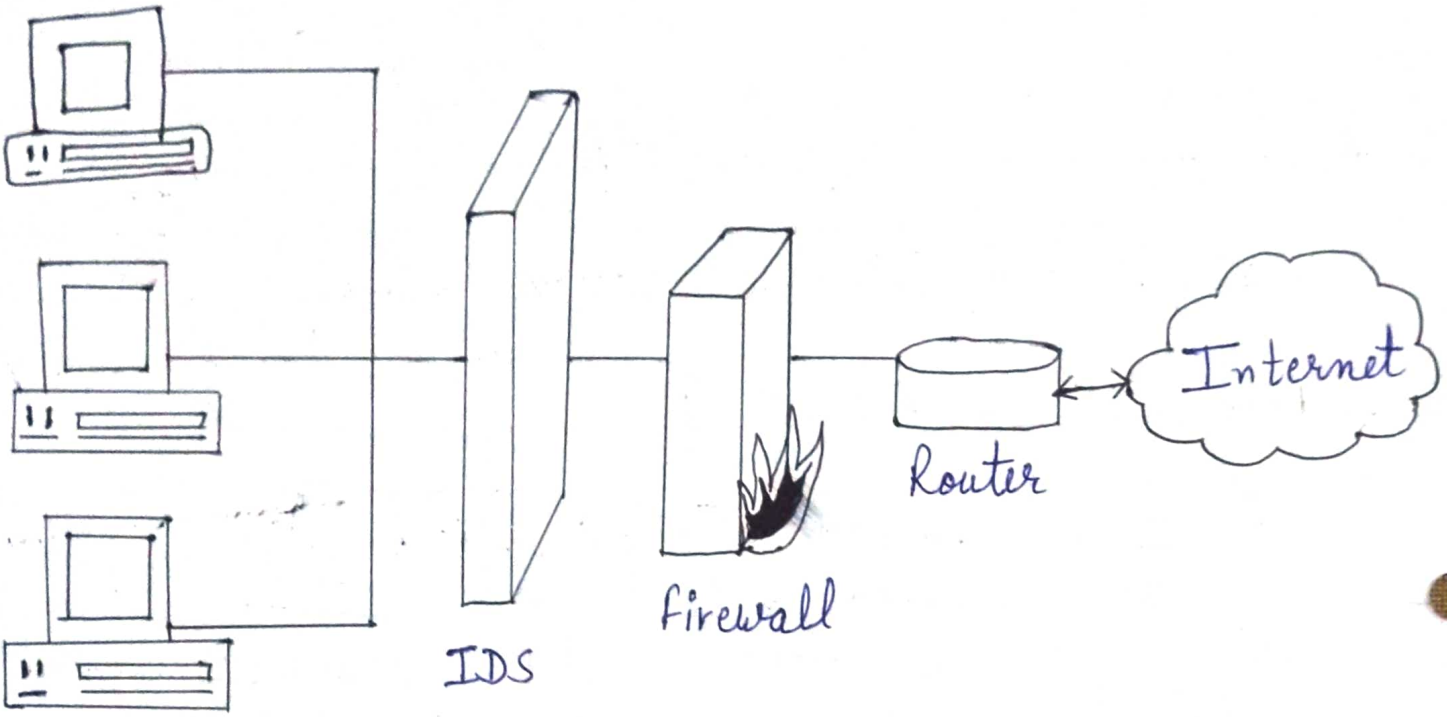
DELTA Pg No. 11
Date 11/10/21

PRACTICAL 7

Aim: To study Intrusion Detection System.

Theory: An intrusion detection system (IDS) is a device or software application that monitors network and/or system activities for malicious activities or policy violations and produces reports to a Management Station. Some systems may attempt to stop an intrusion attempt but this is neither required nor expected of a monitoring system. Intrusion detection & prevention system (IDPS) are primarily focused on identifying possible incidents, logging information about them and reporting attempts. In addition, organizations use IDPS for other purposes, such as identifying problems with security policies, documenting existing threats, & deterring individuals from violating security policies. IDPSes have become a necessary addition to the security infrastructure of nearly every organization.

Snort: Snort is a light weight network intrusion detection system, capable of performing real-time traffic analysis & packet logging on IP networks. It can perform protocol analysis, content searching/matching & can be used to detect a variety of attacks & probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS finger printing attempts & much more.



Intrusion Detection System in Security

CISCO

SNORT™

Snort Logo

Course outcome
CO4, CO3

DELTA Pg No. 12
Date 18/10/2024

PRACTICAL 8

Aim: To implement Vernam Cipher Algorithm.

Software used: VS Code

Theory: It is a method of encryption alphabet text. It is one of the transposition technique for converting a plain text to cipher text.

In modern terminology, a Vernam cipher is a symmetrical stream cipher in which the plaintext is combined with a random or pseudorandom stream of data (the "keystream") of the same length, to generate the ciphertext, using the Boolean "exclusive or" (XOR) function.

Key generation →
We take a key to encrypt the plain text whose length should be equal to the length of plain ~~text~~ text.

Encryption algorithm (for decryption use reverse of this)

1) Assign a number to each character of plain text and key according to alphabetical order.
2) Add both numbers corresponding to each other.
3) Subtract # number from 26 if number added is greater than 26.
4) Get cipher text by getting corresponding character from the number.

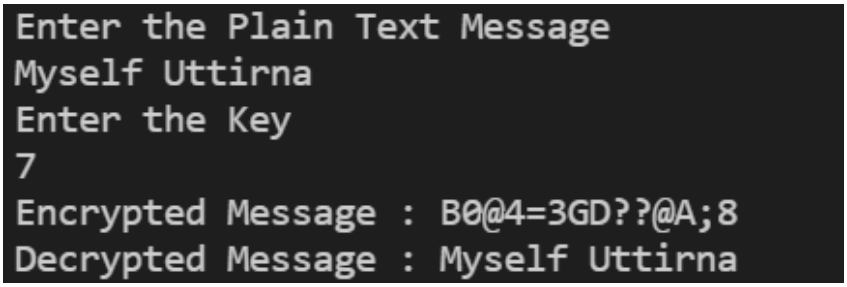
⇒ Ciphertext 1 ⊕ Ciphertext 2 = Plaintext 1 ⊕ Plaintext 2

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;

class vernam{
public:
string s,k;
char enc[1000],dec[1000];
vernam(string a, string b){
s = a;
k = b;
}
void encrypt(){
int i,j=0;
for(i=0;i<s.size();i++){
enc[i] = s[i]^k[j];
j++;
if(j>=k.size()){
j =0;
}
}
}
void decrypt(){
int i,j=0;
for(i=0;i<s.size();i++)
{
dec[i] = enc[i]^k[j];
j++;
if(j>=k.size())
{
j =0;
}
}
}
void printenc(){
int i;
char c;
for(i=0;i<s.size();i++)
{
c = enc[i]%26 + 48;
cout<<c;
}
cout<<endl;
}
void printdec(){
int i;
for(i=0;i<s.size();i++)
```

```
{
cout<<dec[i];
}
}
};

int main(){
string s,k;
cout<<”Enter the Plain Text Message”<<endl;
getline(cin,s);
cout<<”Enter the Key”<<endl;
getline(cin,k);
vernam v(s,k);
v.encrypt();
cout<<”Encrypted Message : “;
v.printenc();
//cout<<endl;
v.decrypt();
cout<<”Decrypted Message : “;
v.printdec();
return 0;
}
```

OUTPUT:



RESULT: Successfully studied and implemented Vernam Cipher.

EXPECTED OUTPUT ATTAINED: YES