

Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal

New Scheme of Examination as per AICTE Flexible Curricula

VII Semester

Bachelor of Technology (B.Tech.) Computer Science and Engineering/ **(w.e.f. July, 2020)**
Computer Engineering/Computer Science & Technology]

S.No.	Subject Code	Category	Subject Name	Maximum Marks Allotted					Total Marks	Contact Hours per week			Total Credits
				Theory			Practical						
				End Sem.	Mid Sem. Exam.	Quiz/ Assignment	End Sem	Term work Lab Work & Sessional		L	T	P	
1.	CS 701	DC	Software Architectures	70	20	10	30	20	150	2	1	2	4
2.	CS 702	DE	Departmental Elective	70	20	10	-	-	100	3	1	-	4
3.	CS 703	OE	Open Elective	70	20	10	-	-	100	3	0	0	3
4.	CS 704	D Lab	Departmental Elective Lab	-	--	-	30	20	50	-	-	6	3
5.	CS 705	O/E lab	Open Elective Lab	-	-	-	30	20	50	-	-	6	3
6.	CS 706	P	Major Project-I	-	-	-	100	50	150	-	-	8	4
7.	CS 707		Evaluation of Internship -III	-	-	-	-	100	100	-	-	6	3
8.	Additional Credits [#]	*Additional credits can be earned through successful completion of credit based MOOC's Courses available on SWAYAM platform (MHRD) at respective level.											UG
			Total	210	60	30	190	210	700	8	2	28	24

Departmental Electives		Open Electives	
702(A) Computational Intelligence		703(A) Cryptography & Information Security	
702 (B) Deep & Reinforcement Learning		703(B) Data Mining and Warehousing	
702 (C) Wireless & Mobile Computing		703(C) Agile Software Development	
702 (D) Big Data		703 (D) Disaster Management	

Open Electives can be offered to students of all branches including CSE branch. However, they can be offered to students of Non-CSE branches only if they have not taken any similar courses previously and have sufficient knowledge of pre-requisite courses (if any) of respective open electives subject.

1 Hr Lecture	1 Hr Tutorial	2 Hr Practical
1 Credit	1 Credit	1 Credit

RAJIV GANDHI PROUDYOGIKI VISHWAVIDYALAYA, BHOPAL

New Scheme Based On AICTE Flexible Curricula

Computer Science and Engineering, VII-Semester

Open Elective – CS703 (A) Cryptography & Information Security

COURSE OUTCOMES:

CO1: Understanding of the basics of Cryptography and Network Security and working knowledge of Mathematics used in Cryptology.

CO2: Understanding of previous attacks on cryptosystems to prevent future attacks from securing a message over an insecure channel by various means.

CO3: Knowledge about how to maintain the Confidentiality, Integrity and Availability of a data.

CO4: Understanding of various protocols for network security to protect against the network threats.

CO5: Getting hands-on experience of various Information Security Tools.

UNIT I:

Mathematical Background for Cryptography: Abstract Algebra, Number Theory, Modular Inverse, Extended Euclid Algorithm, Fermat's Little Theorem, Euler Phi-Function, Euler's theorem.

Introduction to Cryptography: Principles of Cryptography, Classical Cryptosystem, Cryptanalysis on Substitution Cipher (Frequency Analysis), Play Fair Cipher, Block Cipher. Data Encryption Standard (DES), Triple DES, Modes of Operation, Stream Cipher.

UNIT II:

Advanced Encryption Standard (AES), Introduction to Public Key Cryptosystem, Discrete Logarithmic Problem, Diffie-Hellman Key Exchange Computational & Decisional Diffie-Hellman Problem, RSA Assumptions & Cryptosystem, RSA Signatures & Schnorr Identification Schemes, Primarily Testing, Elliptic Curve over the Reals, Elliptic curve Modulo a Prime., Chinese Remainder Theorem.

UNIT III:

Message Authentication, Digital Signature, Key Management, Key Exchange, Hash Function. Universal Hashing, Cryptographic Hash Function, MD, Secure Hash Algorithm (SHA), Digital Signature Standard (DSS), Cryptanalysis: Time-Memory Trade-off Attack, Differential Cryptanalysis. Secure channel and authentication system like Kerberos.

UNIT IV:

Information Security: Threats in Networks, Network Security Controls–Architecture, Wireless Security, Honey pots, Traffic Flow Security, Firewalls – Design and Types of Firewalls, Personal Firewalls, IDS, Email Security: Services Security for Email Attacks Through Emails, Privacy-Authentication of Source Message, Pretty Good Privacy(PGP), S-MIME. IP Security: Overview of IPSec, IP& IP version 6 Authentication, Encapsulation Security Payload ESP, Internet Key Exchange IKE, Web Security: SSL/TLS, Basic protocols of security. Encoding –Secure Electronic Transaction SET.

UNIT V: Cryptography and Information Security Tools: Spoofing tools: like Arping etc., Foot printing Tools (ex-nslookup, dig, Whois, etc.), Vulnerabilities Scanning Tools (i.e. Angry IP, HPing2, IP Scanner, Global Network Inventory Scanner, Net Tools Suite Pack.), NetBIOS Enumeration Using NetView Tool, Steganography Merge Streams, Image Hide, Stealth Files, Blindsiding: STools, Steghide, Steganos, Stegdetect, Steganalysis - Stego Watch- Stego Detection Tool, StegSpy, Trojans Detection Tools(i.e. Netstat, fPort, TCPView, CurrPorts Tool, Process Viewer), Lan Scanner Tools (i.e. look@LAN, Wireshark, Tcpdump). DoS Attack Understanding Tools- Jolt2, Bubonic.c, Land and LaTierra, Targa, Nemesis Blast, Panther2, Crazy Flinger, Some Trouble, UDP Flood, FSMax.

Recommended Text:

1. Cryptography and Network Security Principles and Practice Fourth Edition, William Stallings, Pearson Education.
2. Network Security Essentials: Applications and Standards, by William Stallings. Prentice Hall.
3. Behrouz A Ferouzan, "Cryptography and Network Security" Tata Mc Graw Hills, 2007
4. Charles P. Pfleeger, Shari Lawrence Pfleeger "Security in Computing", 4th Edition Prentice Hall of India, 2006.
5. Introduction to Modern Cryptography by Jonathan Katz and Yehuda Lindell, Chapman and Hall/CRC

COURSE OBJECTIVES

1. To provide deeper understanding into cryptography, its application to network security, threats/vulnerabilities to networks and countermeasures.
2. To explain various approaches to Encryption techniques, strengths of Traffic Confidentiality, Message Authentication Codes.
3. To familiarize symmetric and asymmetric cryptography

COURSE OUTCOMES

At the end of this course students will be able to:

1. Identify basic security attacks and services
2. Use symmetric and asymmetric key algorithms for cryptography
3. Make use of Authentication functions

Lab Plan

S.No	Name of the Experiment	Planned Date	Date of completion	Remark
1	Study and implementation of DES Algorithm			
2	Study of Playfair cipher Algorithm			
3	Study and implementation of RSA Algorithm			
4	Study and implementation of AES Algorithm			
5	Handson study of Spoofing tools (like Arping)			
6	Handson study of Vulnerabilities scanning tool (Angry IP, HPing2, IP Scanner)			
7	Handson study of Steganography Tools (STools, Steghide, Steganos.Stegdetect).			
8	Handson study of LAN Scanner Tools (look@LAN, Wireshark, Tcpcdump).			
9	Implementation of Caesar Cipher technique			

Experiment No 1

Study and implementation of DES Algorithm

The Data Encryption Standard (DES) is a symmetric-key block cipher published by the National Institute of Standards and Technology (NIST).

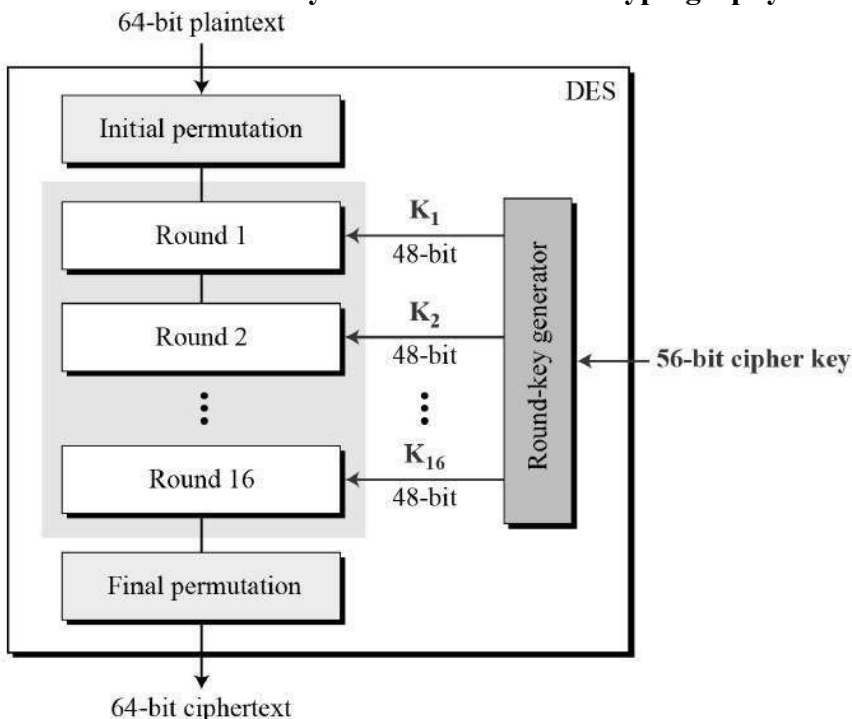
DES is an implementation of a Feistel Cipher. It uses a 16 round Feistel structure. The block size is 64-bit. Though, key length is 64-bit, DES has an effective key length of 56 bits, since 8 of the 64 bits of the key are not used by the encryption algorithm (function as check bits only). General Structure of DES is depicted in the following illustration –

Since DES is based on the Feistel Cipher, all that is required to specify DES is –

- Round function
- Key schedule
- Any additional processing – Initial and final permutation

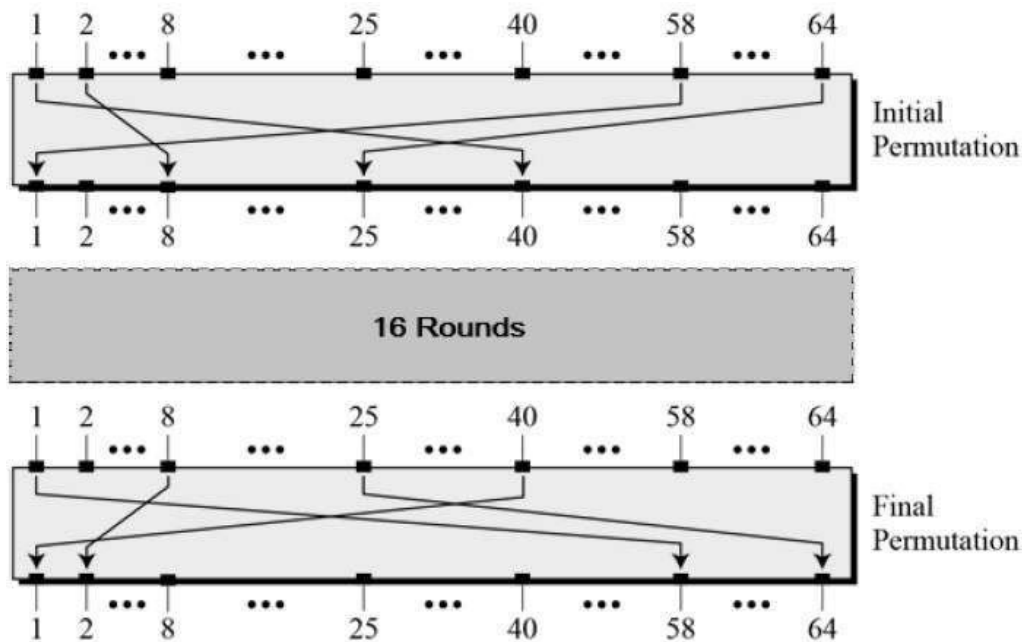
Initial and Final Permutation

The initial and final permutations are straight Permutation boxes (P-boxes) that are inverses of each other. They have no cryptography significance in DES. The



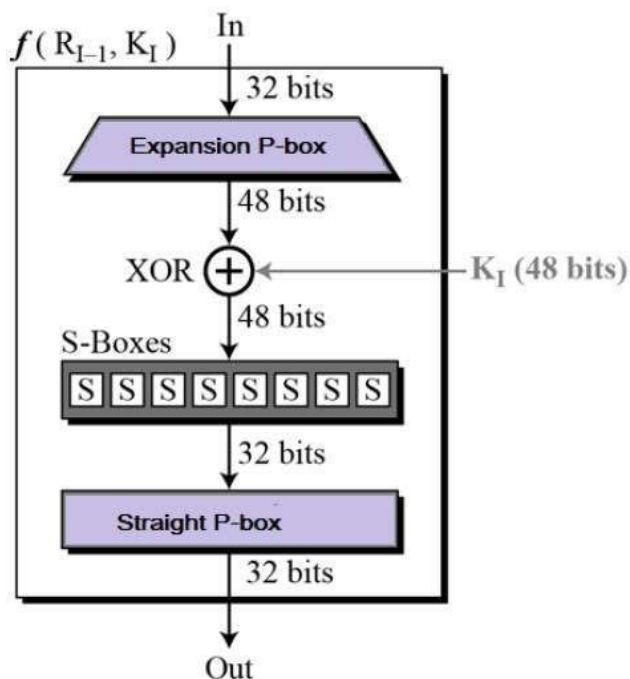
shown as follows –

initial and final permutations are

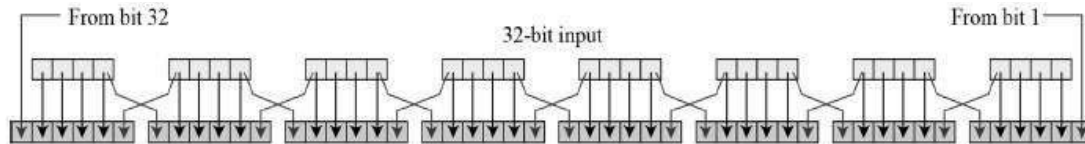


Round Function

The heart of this cipher is the DES function, f . The DES function applies a 48-bit key to the rightmost 32 bits to produce a 32-bit output.



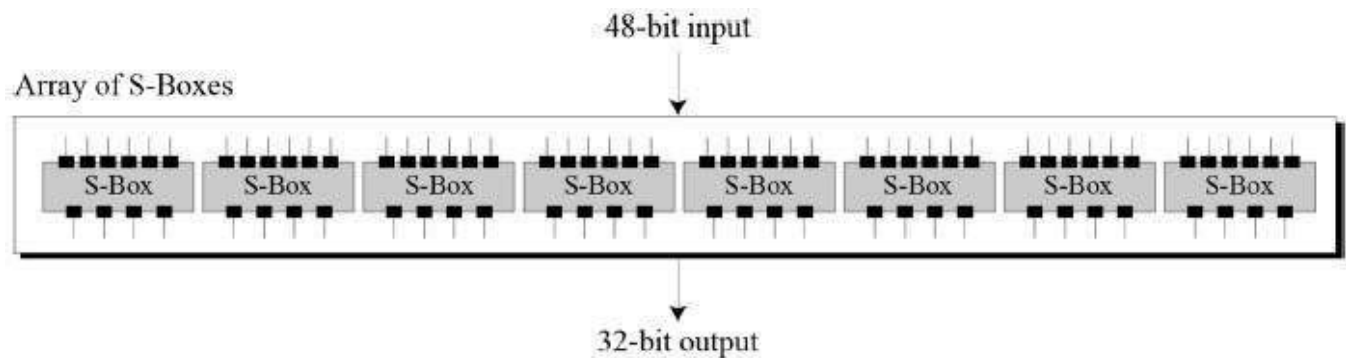
- **Expansion Permutation Box** – Since right input is 32-bit and round key is a 48-bit, we first need to expand right input to 48 bits. Permutation logic is graphically depicted in the following illustration –



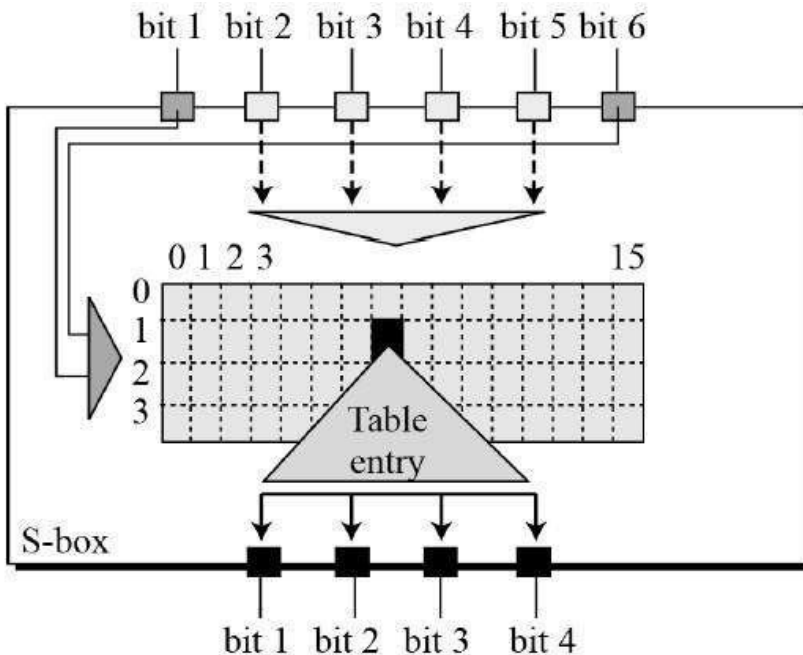
- The graphically depicted permutation logic is generally described as table in DES specification illustrated as shown –

32	01	02	03	04	05
04	05	06	07	08	09
08	09	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	31	31	32	01

- **XOR (Whitener).** – After the expansion permutation, DES does XOR operation on the expanded right section and the round key. The round key is used only in this operation.
- **Substitution Boxes.** – The S-boxes carry out the real mixing (confusion). DES uses 8 S-boxes, each with a 6-bit input and a 4-bit output. Refer the following illustration –



- The S-box rule is illustrated below –



- There are a total of eight S-box tables. The output of all eight s-boxes is then combined in to 32 bit section.
- Straight Permutation – The 32 bit output of S-boxes is then subjected to the straight permutation with rule shown in the following illustration:

16	07	20	21	29	12	28	17
01	15	23	26	05	18	31	10
02	08	24	14	32	27	03	09
19	13	30	06	22	11	04	25

Key Generation

The round-key generator creates sixteen 48-bit keys out of a 56-bit cipher key. The process of key generation is depicted in the following illustration –

The logic for Parity drop, shifting, and Compression P-box is given in the DES description.

Implementation of DES

```
# include <stdio.h>
# include <fstream>
# include <string.h>
# include <iostream>
# include <stdlib.h>
using namespace std;
```

```
int key[64]=
{
    0,0,0,1,0,0,1,1,
    0,0,1,1,0,1,0,0,
    0,1,0,1,0,1,1,1,
    0,1,1,1,1,0,0,1,
    1,0,0,1,1,0,1,1,
    1,0,1,1,1,1,0,0,
    1,1,0,1,1,1,1,1,
    1,1,1,1,0,0,0,1
};
```

```
class Des
```

```
{
public:
    int keyi[16][48],
        total[64],
        left[32],
        right[32],
        ck[28],
        dk[28],
```

expansion[48],

z[48],

xor1[48],

sub[32],

p[32],

xor2[32],

temp[64],

pc1[56],

ip[64],

inv[8][8];

char final[1000];

void IP();

void PermChoice1();

void PermChoice2();

void Expansion();

void inverse();

void xor_two();

void xor_oneE(int);

void xor_oneD(int);

void substitution();

void permutation();

void keygen();

char * Encrypt(char *);

char * Decrypt(char *);

};

void Des::IP() //Initial Permutation

{

```
int k=58,i;
for(i=0; i<32; i++)
{
    ip[i]=total[k-1];
    if(k-8>0) k=k-8;
    else    k=k+58;
}
k=57;
for( i=32; i<64; i++)
{
    ip[i]=total[k-1];
    if(k-8>0) k=k-8;
    else    k=k+58;
}
}

void Des::PermChoice1() //Permutation Choice-1
{
    int k=57,i;
    for(i=0; i<28; i++)
    {
        pc1[i]=key[k-1];
        if(k-8>0) k=k-8;
        else    k=k+57;
    }
    k=63;
    for( i=28; i<52; i++)
    {
        pc1[i]=key[k-1];
```

```
    if(k-8>0)  k=k-8;
    else      k=k+55;
}
k=28;
for(i=52; i<56; i++)
{
    pc1[i]=key[k-1];
    k=k-8;
}

}

void Des::Expansion() //Expansion Function applied on `right' half
{
    int exp[8][6],i,j,k;
    for( i=0; i<8; i++)
    {
        for( j=0; j<6; j++)
        {
            if((j!=0)||(j!=5))
            {
                k=4*i+j;
                exp[i][j]=right[k-1];
            }
            if(j==0)
            {
                k=4*i;
                exp[i][j]=right[k-1];
            }
        }
    }
}
```



```
    if(j==5)
    {
        k=4*i+j;
        exp[i][j]=right[k-1];
    }
}

exp[0][0]=right[31];
exp[7][5]=right[0];

k=0;
for(i=0; i<8; i++)
    for(j=0; j<6; j++)
        expansion[k++]=exp[i][j];
}

void Des::PermChoice2()
{
    int per[56],i,k;
    for(i=0; i<28; i++) per[i]=ck[i];
    for(k=0,i=28; i<56; i++) per[i]=dk[k++];

    z[0]=per[13];
    z[1]=per[16];
    z[2]=per[10];
    z[3]=per[23];
    z[4]=per[0];
    z[5]=per[4];
    z[6]=per[2];
```

```
z[7]=per[27];
z[8]=per[14];
z[9]=per[5];
z[10]=per[20];
z[11]=per[9];
z[12]=per[22];
z[13]=per[18];
z[14]=per[11];
z[15]=per[3];
z[16]=per[25];
z[17]=per[7];
z[18]=per[15];
z[19]=per[6];
z[20]=per[26];
z[21]=per[19];
z[22]=per[12];
z[23]=per[1];
z[24]=per[40];
z[25]=per[51];
z[26]=per[30];
z[27]=per[36];
z[28]=per[46];
z[29]=per[54];
z[30]=per[29];
z[31]=per[39];
z[32]=per[50];
z[33]=per[46];
z[34]=per[32];
```

```
z[35]=per[47];
z[36]=per[43];
z[37]=per[48];
z[38]=per[38];
z[39]=per[55];
z[40]=per[33];
z[41]=per[52];
z[42]=per[45];
z[43]=per[41];
z[44]=per[49];
z[45]=per[35];
z[46]=per[28];
z[47]=per[31];
}

void Des::xor_oneE(int round) //for Encrypt
{
    int i;
    for(i=0; i<48; i++)
        xor1[i]=expansion[i]^keyi[round-1][i];
}

void Des::xor_oneD(int round) //for Decrypt
{
    int i;
    for(i=0; i<48; i++)
        xor1[i]=expansion[i]^keyi[16-round][i];
}

void Des::substitution()
```

```
{  
  int s1[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  
  };  

```

```
  int s2[4][16]=  
  {  
    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  
  };  

```

```
  int s3[4][16]=  
  {  
    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  
  };  

```

```
  int s4[4][16]=  
  {  
    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  
};
```

```
int s5[4][16]=  
{  
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  
};
```

```
int s6[4][16]=  
{  
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  
};
```

```
int s7[4][16]=  
{  
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  
};
```

```
int s8[4][16]=  
{  
    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  
};  
int a[8][6],k=0,i,j,p,q,count=0,g=0,v;
```

```
for(i=0; i<8; i++)  
{  
    for(j=0; j<6; j++)  
    {  
        a[i][j]=xor1[k++];  
    }  
}
```

```
for( i=0; i<8; i++)  
{  
    p=1;  
    q=0;  
    k=(a[i][0]*2)+(a[i][5]*1);  
    j=4;  
    while(j>0)  
    {  
        q=q+(a[i][j]*p);  
        p=p*2;
```



```
j--;  
}  
count=i+1;  
switch(count)  
{  
case 1:  
    v=s1[k][q];  
    break;  
case 2:  
    v=s2[k][q];  
    break;  
case 3:  
    v=s3[k][q];  
    break;  
case 4:  
    v=s4[k][q];  
    break;  
case 5:  
    v=s5[k][q];  
    break;  
case 6:  
    v=s6[k][q];  
    break;  
case 7:  
    v=s7[k][q];  
    break;  
case 8:  
    v=s8[k][q];
```

```
        break;
    }

    int d,i=3,a[4];
    while(v>0)
    {
        d=v%2;
        a[i--]=d;
        v=v/2;
    }
    while(i>=0)
    {
        a[i--]=0;
    }

    for(i=0; i<4; i++)
        sub[g++]=a[i];
}

void Des::permutation()
{
    p[0]=sub[15];
    p[1]=sub[6];
    p[2]=sub[19];
    p[3]=sub[20];
    p[4]=sub[28];
    p[5]=sub[11];
```

```
p[6]=sub[27];  
p[7]=sub[16];  
p[8]=sub[0];  
p[9]=sub[14];  
p[10]=sub[22];  
p[11]=sub[25];  
p[12]=sub[4];  
p[13]=sub[17];  
p[14]=sub[30];  
p[15]=sub[9];  
p[16]=sub[1];  
p[17]=sub[7];  
p[18]=sub[23];  
p[19]=sub[13];  
p[20]=sub[31];  
p[21]=sub[26];  
p[22]=sub[2];  
p[23]=sub[8];  
p[24]=sub[18];  
p[25]=sub[12];  
p[26]=sub[29];  
p[27]=sub[5];  
p[28]=sub[21];  
p[29]=sub[10];  
p[30]=sub[3];  
p[31]=sub[24];  
}
```

```
void Des::xor_two()
{
    int i;
    for(i=0; i<32; i++)
    {
        xor2[i]=left[i]^p[i];
    }
}

void Des::inverse()
{
    int p=40,q=8,k1,k2,i,j;
    for(i=0; i<8; i++)
    {
        k1=p;
        k2=q;
        for(j=0; j<8; j++)
        {
            if(j%2==0)
            {
                inv[i][j]=temp[k1-1];
                k1=k1+8;
            }
            else if(j%2!=0)
            {
                inv[i][j]=temp[k2-1];
                k2=k2+8;
            }
        }
    }
}
```

```
    }
    p=p-1;
    q=q-1;
}
}

char * Des::Encrypt(char *Text1)
{
    int i,a1,j,nB,m,iB,k,K,B[8],n,t,d,round;
    char *Text=new char[1000];
    strcpy(Text,Text1);
    i=strlen(Text);
    int mc=0;
    a1=i%8;
    if(a1!=0) for(j=0; j<8-a1; j++,i++) Text[i]=' ';
    Text[i]='\0';
    keygen();
    for(iB=0,nB=0,m=0; m<((strlen(Text)/8); m++) //Repeat for TextLenth/8 times.
    {
        for(iB=0,i=0; i<8; i++,nB++)
        {
            n=(int)Text[nB];
            for(K=7; n>=1; K--)
            {
                B[K]=n%2; //Converting 8-Bytes to 64-bit Binary Format
                n/=2;
            }
            for(; K>=0; K--) B[K]=0;
```

for(K=0; K<8; K++,iB++) total[iB]=B[K]; //Now `total' contains the 64-Bit binary format of 8-Bytes

}

IP(); //Performing initial permutation on `total[64]'

for(i=0; i<64; i++) total[i]=ip[i]; //Store values of ip[64] into total[64]

for(i=0; i<32; i++) left[i]=total[i]; // +--> left[32]

// total[64]--|

for(; i<64; i++) right[i-32]=total[i]; // +--> right[32]

for(round=1; round<=16; round++)

{

Expansion(); //Performing expansion on `right[32]' to get `expansion[48]'

xor_oneE(round); //Performing XOR operation on expansion[48],z[48] to get xor1[48]

substitution(); //Perform substitution on xor1[48] to get sub[32]

permutation(); //Performing Permutation on sub[32] to get p[32]

xor_two(); //Performing XOR operation on left[32],p[32] to get xor2[32]

for(i=0; i<32; i++) left[i]=right[i]; //Dumping right[32] into left[32]

for(i=0; i<32; i++) right[i]=xor2[i]; //Dumping xor2[32] into right[32]

}

for(i=0; i<32; i++) temp[i]=right[i]; // Dumping -->[swap32bit]

for(; i<64; i++) temp[i]=left[i-32]; // left[32],right[32] into temp[64]

inverse(); //Inversing the bits of temp[64] to get inv[8][8]

/* Obtaining the Cypher-Text into final[1000]*/

k=128;

d=0;

for(i=0; i<8; i++)

{


```
    for(j=0; j<8; j++)
    {
        d=d+inv[i][j]*k;
        k=k/2;
    }
    final[mc++]=(char)d;
    k=128;
    d=0;
}
} //for loop ends here

final[mc]='\0';
return(final);
}

char * Des::Decrypt(char *Text1)
{
    int i,a1,j,nB,m,iB,k,K,B[8],n,t,d,round;
    char *Text=new char[1000];
    unsigned char ch;
    strcpy(Text,Text1);
    i=strlen(Text);
    keygen();
    int mc=0;
    for(iB=0,nB=0,m=0; m<((strlen(Text)/8); m++) //Repeat for TextLenth/8 times.
    {
        for(iB=0,i=0; i<8; i++,nB++)
        {
            ch=Text[nB];
            n=(int)ch;//(int)Text[nB];
```

```
for(K=7; n>=1; K--)
{
    B[K]=n%2; //Converting 8-Bytes to 64-bit Binary Format
    n/=2;
}
for(; K>=0; K--) B[K]=0;

for(K=0; K<8; K++,iB++) total[iB]=B[K]; //Now `total' contains the 64-Bit binary format
of 8-Bytes
}

IP(); //Performing initial permutation on `total[64]'
for(i=0; i<64; i++) total[i]=ip[i]; //Store values of ip[64] into total[64]

for(i=0; i<32; i++) left[i]=total[i]; //    +--> left[32]
// total[64]--|
for(; i<64; i++) right[i-32]=total[i]; //    +--> right[32]
for(round=1; round<=16; round++)
{
    Expansion(); //Performing expansion on `right[32]' to get `expansion[48]'
    xor_oneD(round);
    substitution(); //Perform substitution on xor1[48] to get sub[32]
    permutation(); //Performing Permutation on sub[32] to get p[32]
    xor_two(); //Performing XOR operation on left[32],p[32] to get xor2[32]
    for(i=0; i<32; i++) left[i]=right[i]; //Dumping right[32] into left[32]
    for(i=0; i<32; i++) right[i]=xor2[i]; //Dumping xor2[32] into right[32]
} //rounds end here

for(i=0; i<32; i++) temp[i]=right[i]; // Dumping  -->[ swap32bit ]
for(; i<64; i++) temp[i]=left[i-32]; //    left[32],right[32] into temp[64]
```

```
inverse(); //Inversing the bits of temp[64] to get inv[8][8]

/* Obtaining the Cypher-Text into final[1000]*/

k=128;

d=0;

for(i=0; i<8; i++)
{
    for(j=0; j<8; j++)
    {
        d=d+inv[i][j]*k;

        k=k/2;
    }

    final[mc++]=(char)d;

    k=128;

    d=0;
}

} //for loop ends here

final[mc]='\0';

char *final1=new char[1000];

for(i=0,j=strlen(Text); i<strlen(Text); i++,j++)

    final1[i]=final[j];

final1[i]='\0';

return(final);

}

int main()

{

    Des d1,d2;

    char *str=new char[1000];

    char *str1=new char[1000];
```

```
//strcpy(str,"PHOENIX it & ece solutions.");  
cout<<"Enter a string : ";  
cin >> str;  
str1=d1.Encrypt(str);  
cout<<"\n/p Text: "<<str<<endl;  
cout<<"\nCypher : "<<str1<<endl;  
// ofstream fout("out2_fil.txt"); fout<<str1; fout.close();  
cout<<"\n/o/p Text: "<<d2.Decrypt(str1)<<endl;  
}
```

```
void Des::keygen()  
{  
    PermChoice1();  
  
    int i,j,k=0;  
    for(i=0; i<28; i++)  
    {  
        ck[i]=pc1[i];  
    }  
    for(i=28; i<56; i++)  
    {  
        dk[k]=pc1[i];  
        k++;  
    }  
    int noshift=0,round;  
    for(round=1; round<=16; round++)  
    {  
        if(round==1||round==2||round==9||round==16)
```

```
    noshift=1;
else
    noshift=2;
while(noshift>0)
{
    int t;
    t=ck[0];
    for(i=0; i<28; i++)
        ck[i]=ck[i+1];
    ck[27]=t;
    t=dk[0];
    for(i=0; i<28; i++)
        dk[i]=dk[i+1];
    dk[27]=t;
    noshift--;
}
PermChoice2();
for(i=0; i<48; i++)
    keyi[round-1][i]=z[i];
}
}
```

Experiment 2

Study and Implementation of Playfair cipher Algorithm

first understand concept of playfair cipher The Playfair cipher was the first practical digraph substitution cipher. The scheme was invented in 1854 by Charles Wheatstone but was named after Lord Playfair who promoted the use of the cipher. In playfair cipher unlike traditional cipher we encrypt a pair of alphabets(digraphs) instead of a single alphabet. Playfair is reasonably fast to use and requires no special equipment.

The Playfair Cipher Encryption Algorithm:

The Algorithm consists of 2 steps:

Step 1 : Generate the key Square(5×5):

Step 2: Algorithm to encrypt the plain text:

The plaintext is split into pairs of two letters (digraphs).

If there is an odd number of letters, a Z is added to the last letter.

Step 1:

The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext.

Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets).

If the plaintext contains J, then it is replaced by I.

The initial alphabets in the key square are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

Key : Monarchy

Plaintext : instruments

The key is "monarchy"

Thus the initial entries are

'm', 'o', 'n', 'a', 'r', 'c', 'h', 'y'

followed by remaining characters of

a-z(except 'j')

in that order.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Algorithm to encrypt the plain text: The plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z is added to the last letter.

PlainText: "instruments"

After Split: 'in' 'st' 'ru' 'me' 'nt' 'sz'

Rules for Encryption:

If both the letters are in the same column:

Take the letter below each one
(going back to the top if at the bottom).

Diagraph: "me"

Encrypted Text: cl

Encryption:

m -> c

e -> l

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

If both the letters are in the same row: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Diagraph: "nt"

Encrypted Text: rq

Encryption:

n -> r

t -> q

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Plain Text: "instrumentsz"

Encrypted Text: gatlmzclrqtx

Encryption:

i -> g

n -> a

s -> t

t -> l

r -> m

u -> z

m -> c

e -> l

n -> r

t -> q

s -> t

z -> x

in:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

st:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

ru:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

me:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

nt:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

sz:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Decryption process play fair cipher

Decrypting the Playfair cipher is as simple as doing the same process in reverse. The receiver has the same key and can create the same key table, and then decrypt any messages made using that key.

key : monarchy

ciphertext : gatlmzclrqtx

The Playfair Cipher Decryption Algorithm:

The Algorithm consists of 2 steps:

Generate the key Square(5×5) at the receiver's end:

The key square is a 5×5 grid of alphabets that acts as the key for encrypting the plaintext.

Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table (as the table can hold only 25 alphabets).

If the plaintext contains J, then it is replaced by I.

The initial alphabets in the key square

are the unique alphabets of the key in the order in which they appear followed by the remaining letters of the alphabet in order.

The key is "monarchy"

Thus the initial entires are

'm', 'o', 'n', 'a', 'r', 'c', 'h', 'y'

followed by remaining characters of a-z(except 'j') in that order.

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Algorithm to decrypt the ciphertext: The ciphertext is split into pairs of two letters (digraphs)

CipherText: "gatlmzclrqtx"

After Split: 'ga' 'tl' 'mz' 'cl' 'rq' 'tx'

Rules for Decryption:

If both the letters are in the same column: Take the letter above each one (going back to the bottom if at the top).

Diagraph: "cl"

Decrypted Text: me

Decryption:

c -> m

l -> e

Diagraph: "tl"

Decrypted Text: st

Decryption:

t -> s

l -> t

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

If neither of the above rules is true: Form a rectangle with the two letters and take the letters on the

horizontal opposite corner of the rectangle.

Diagraph: "rq"

Decrypted Text: nt

Decryption:

r -> n

q -> t

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

Plain Text: "gatlmzclrqtx"

Decrypted Text: instrumentsz

Decryption:

(red)-> (green)

ga -> in

tl -> st

mz -> ru

cl -> me

rq -> nt

tx -> sz

in:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

st:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

ru:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

me:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

nt:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

sz:

M	O	N	A	R
C	H	Y	B	D
E	F	G	I	K
L	P	Q	S	T
U	V	W	X	Z

coding for play fair cipher

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#define SIZE 30
```

// Function to convert the string to lowercase

```
void toLowerCase(char plain[], int ps)
{
    int i;
    for (i = 0; i < ps; i++) {
        if (plain[i] > 64 && plain[i] < 91)
            plain[i] += 32;
    }
}
```

// Function to remove all spaces in a string

```
int removeSpaces(char* plain, int ps)
{
    int i, count = 0;
    for (i = 0; i < ps; i++)
        if (plain[i] != ' ')
            plain[count++] = plain[i];
    plain[count] = '\0';
    return count;
}
```

// Function to generate the 5x5 key square

```
void generateKeyTable(char key[], int ks, char keyT[5][5])
{
    int i, j, k, flag = 0, *dicty;

    // a 26 character hashmap
    // to store count of the alphabet
    dicty = (int*)calloc(26, sizeof(int));
    for (i = 0; i < ks; i++) {
        if (key[i] != 'j')
            dicty[key[i] - 97] = 2;
    }

    dicty['j' - 97] = 1;

    i = 0;
    j = 0;

    for (k = 0; k < ks; k++) {
        if (dicty[key[k] - 97] == 2) {
            dicty[key[k] - 97] -= 1;
            keyT[i][j] = key[k];
        }
    }
}
```

```
j++;
if (j == 5) {
    i++;
    j = 0;
}
}
}

for (k = 0; k < 26; k++) {
    if (dicty[k] == 0) {
        keyT[i][j] = (char)(k + 97);
        j++;
        if (j == 5) {
            i++;
            j = 0;
        }
    }
}
}

// Function to search for the characters of a digraph
// in the key square and return their position
void search(char keyT[5][5], char a, char b, int arr[])
{
    int i, j;

    if (a == 'j')
        a = 'i';
    else if (b == 'j')
        b = 'i';

    for (i = 0; i < 5; i++) {
        for (j = 0; j < 5; j++) {
            if (keyT[i][j] == a) {
                arr[0] = i;
                arr[1] = j;
            }
            else if (keyT[i][j] == b) {
                arr[2] = i;
                arr[3] = j;
            }
        }
    }
}
```

```
// Function to find the modulus with 5
int mod5(int a)
{
    return (a % 5);
}

// Function to make the plain text length to be even
int prepare(char str[], int ptrs)
{
    if (ptrs % 2 != 0) {
        str[ptrs++] = 'z';
        str[ptrs] = '\0';
    }
    return ptrs;
}

// Function for performing the encryption
void encrypt(char str[], char keyT[5][5], int ps)
{
    int i, a[4];

    for (i = 0; i < ps; i += 2) {

        search(keyT, str[i], str[i + 1], a);

        if (a[0] == a[2]) {
            str[i] = keyT[a[0]][mod5(a[1] + 1)];
            str[i + 1] = keyT[a[0]][mod5(a[3] + 1)];
        }
        else if (a[1] == a[3]) {
            str[i] = keyT[mod5(a[0] + 1)][a[1]];
            str[i + 1] = keyT[mod5(a[2] + 1)][a[1]];
        }
        else {
            str[i] = keyT[a[0]][a[3]];
            str[i + 1] = keyT[a[2]][a[1]];
        }
    }
}

// Function to encrypt using Playfair Cipher
void encryptByPlayfairCipher(char str[], char key[])
{
    char ps, ks, keyT[5][5];
```



```
// Key
ks = strlen(key);
ks = removeSpaces(key, ks);
toLowerCase(key, ks);

// Plaintext
ps = strlen(str);
toLowerCase(str, ps);
ps = removeSpaces(str, ps);

ps = prepare(str, ps);

generateKeyTable(key, ks, keyT);

encrypt(str, keyT, ps);
}

// Driver code
int main()
{
    char str[SIZE], key[SIZE];

    // Key to be encrypted
    strcpy(key, "Monarchy");
    printf("Key text: %s\n", key);

    // Plaintext to be encrypted
    strcpy(str, "instruments");
    printf("Plain text: %s\n", str);

    // encrypt using Playfair Cipher
    encryptByPlayfairCipher(str, key);

    printf("Cipher text: %s\n", str);

    return 0;
}
```

Experiment 3

Implement RSA Encryption Algorithm

RSA is an asymmetric cryptography algorithm which works on two keys-public key and private key.

Algorithms

Begin

1. Choose two prime numbers p and q .
2. Compute $n = p * q$.
3. Calculate $\phi = (p-1) * (q-1)$.
4. Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$; i.e., e and $\phi(n)$ are coprime.
5. Calculate d as $d \equiv e^{-1} \pmod{\phi(n)}$; here, d is the modular multiplicative inverse of e modulo $\phi(n)$.
6. For encryption, $c = m^e \bmod n$, where m = original message.
7. For decryption, $m = c^d \bmod n$.

End

Example

```
#include<iostream>

#include<math.h>

using namespace std;

// find gcd

int gcd(int a, int b) {
    int t;
    while(1) {
        t= a%b;
        if(t==0)
            return b;
        a = b;
        b= t;
    }
}
```

```
}  
  
int main() {  
    //2 random prime numbers  
    double p = 13;  
    double q = 11;  
    double n=p*q;//calculate n  
    double track;  
    double phi= (p-1)*(q-1);//calculate phi  
    //public key  
    //e stands for encrypt  
    double e=7;  
    //for checking that  $1 < e < \phi(n)$  and  $\gcd(e, \phi(n)) = 1$ ; i.e., e and  $\phi(n)$  are coprime.  
    while(e<phi) {  
        track = gcd(e,phi);  
        if(track==1)  
            break;  
        else  
            e++;  
    }  
    //private key  
    //d stands for decrypt  
    //choosing d such that it satisfies  $d*e = 1 \bmod \phi$   
    double d1=1/e;  
    double d=fmod(d1,phi);  
    double message = 9;  
    double c = pow(message,e); //encrypt the message  
    double m = pow(c,d);  
    c=fmod(c,n);
```

```
m=fmod(m,n);  
cout<<"Original Message = "<<message;  
cout<<"\n"<<"p = "<<p;  
cout<<"\n"<<"q = "<<q;  
cout<<"\n"<<"n = pq = "<<n;  
cout<<"\n"<<"phi = "<<phi;  
cout<<"\n"<<"e = "<<e;  
cout<<"\n"<<"d = "<<d;  
cout<<"\n"<<"Encrypted message = "<<c;  
cout<<"\n"<<"Decrypted message = "<<m;  
return 0;  
}
```

Output

p = 13

q = 11

n = pq = 143

phi = 120

e = 7

d = 0.142857

Original Message = 9

Encrypted message = 48

Decrypted message = 9

Experiment No 4

Study and implementation of AES Algorithm

The most popular and widely adopted symmetric encryption algorithm likely to be encountered nowadays is the Advanced Encryption Standard (AES). It is found at least six times faster than triple DES. A replacement for DES was needed as its key size was too small. With increasing computing power, it was considered vulnerable against exhaustive key search attack. Triple DES was designed to overcome this drawback but it was found slow.

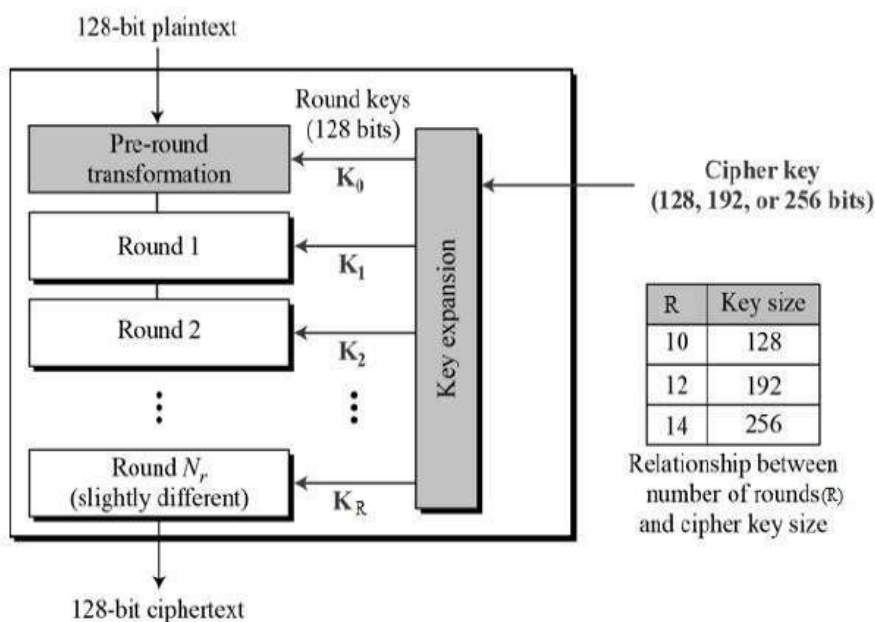
The features of AES are as follows –

- Symmetric key symmetric block cipher
- 128-bit data, 128/192/256-bit keys
- Stronger and faster than Triple-DES
- Provide full specification and design details
- Software implementable in C and Java

Operation of AES

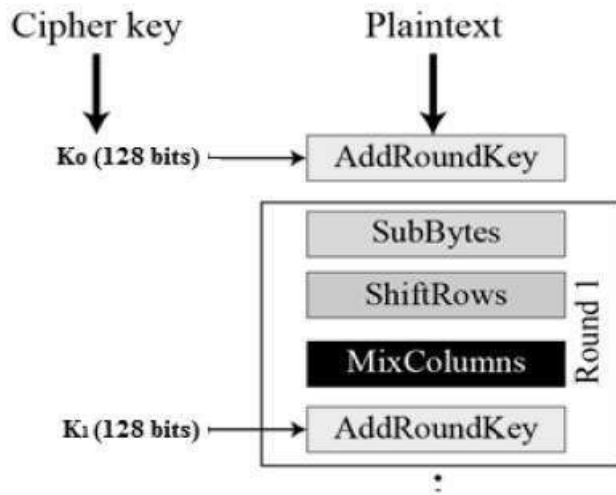
AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix – Unlike DES, the number of rounds in AES is variable and depends on the length of the key. AES uses 10 rounds for 128-bit keys, 12 rounds for 192-bit keys and 14 rounds for 256-bit keys. Each of these rounds uses a different 128-bit round key, which is calculated from the original AES key. The schematic of AES structure is given in the following illustration –



Encryption Process

Here, we restrict to description of a typical round of AES encryption. Each round comprise of four sub-processes. The first round process is depicted below –



Byte Substitution (SubBytes)

The 16 input bytes are substituted by looking up a fixed table (S-box) given in design. The result is in a matrix of four rows and four columns.

Shiftrows

Each of the four rows of the matrix is shifted to the left. Any entries that ‘fall off’ are re-inserted on the right side of row. Shift is carried out as follows –

- First row is not shifted.
- Second row is shifted one (byte) position to the left.
- Third row is shifted two positions to the left.
- Fourth row is shifted three positions to the left.
- The result is a new matrix consisting of the same 16 bytes but shifted with respect to each other.

Mix Columns

Each column of four bytes is now transformed using a special mathematical function. This function takes as input the four bytes of one column and outputs four completely new bytes, which replace the original column. The result is another new matrix consisting of 16 new bytes. It should be noted that this step is not performed in the last round.

Addroundkey

The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key. If this is the last round then the output is the ciphertext. Otherwise, the resulting 128 bits are interpreted as 16 bytes and we begin another similar round.

Decryption Process

The process of decryption of an AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the four processes conducted in the reverse order –

- Add round key

- Mix columns
- Shift rows
- Byte substitution

Since sub-processes in each round are in reverse manner, unlike for a Feistel Cipher, the encryption and decryption algorithms need to be separately implemented, although they are very closely related.

Experiment 5

Handson study of Spoofing tools (like Arping)

ARP spoofing is a type of attack in which a malicious actor sends falsified ARP (**Address Resolution Protocol**) messages over a local area network.

This results in the linking of an attacker's MAC address with the IP address of a legitimate computer or server on the network. Once the attacker's MAC address is connected to an authentic IP address, the attacker will begin receiving any data that is intended for that IP address.

ARP spoofing can enable malicious parties to intercept, modify or even stop data in-transit.

ARP spoofing attacks can only occur on local area networks that utilize the Address Resolution Protocol.

The effects of ARP spoofing attacks can have serious implications for enterprises.

In their most basic application, ARP spoofing attacks are used to steal sensitive information.

ARP spoofing

In computer networking, ARP spoofing, ARP cache poisoning, or ARP poison routing, is a technique by which an attacker sends (spoofed) Address Resolution Protocol (ARP) messages onto a local area network. Generally, the aim is to associate the attacker's MAC address with the IP address of another host, such as the default gateway, causing any traffic meant for that IP address to be sent to the attacker instead.

ARP spoofing may allow an attacker to intercept data frames on a network, modify the traffic, or stop all traffic. Often the attack is used as an opening for other attacks, such as denial of service, man in the middle, or session hijacking attacks.

ARP vulnerabilities

The Address Resolution Protocol (ARP) is a widely used communications protocol for resolving Internet layer addresses into link layer addresses.

When an Internet Protocol (IP) datagram is sent from one host to another in a local area network, the destination IP address must be resolved to a MAC address for transmission via the data link layer.

When another host's IP address is known, and its MAC address is needed, a broadcast packet is sent out on the local network. This packet is known as an ARP request.

The destination machine with the IP in the ARP request then responds with an ARP reply that contains the MAC address for that IP.

Beyond this, ARP spoofing attacks are often used to facilitate other attacks such as:

Denial-of-service attacks: DoS attacks often leverage ARP spoofing to link multiple IP addresses with a single target's MAC address. As a result, traffic that is intended for many

different IP addresses will be redirected to the target's MAC address, overloading the target with traffic.

Session hijacking: Session hijacking attacks can use ARP spoofing to steal session IDs, granting attackers access to private systems and data.

Man-in-the-middle attacks: MITM attacks can rely on ARP spoofing to intercept and modify traffic between victims.

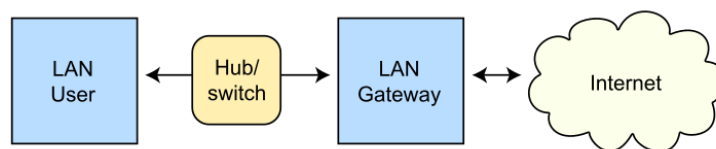
How ARP Spoofing is performed

ARP spoofing attacks typically follow a similar progression.

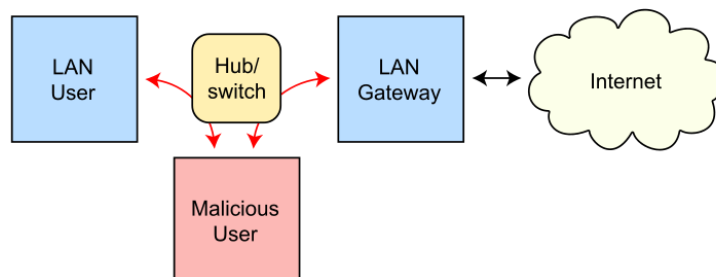
The steps to an ARP spoofing attack usually include:

- The attacker opens an ARP spoofing tool and sets the tool's IP address to match the IP subnet of a target. Examples of popular ARP spoofing software include **Arpspoof**, **Cain & Abel**, **Arpoison** and **Ettercap**.
- The attacker uses the ARP spoofing tool to scan for the IP and MAC addresses of hosts in the target's subnet.
- The attacker chooses its target and begins sending ARP packets across the LAN that contain the attacker's MAC address and the target's IP address.
- As other hosts on the LAN cache the spoofed ARP packets, data that those hosts send to the victim will go to the attacker instead. From here, the attacker can steal data or launch a more sophisticated follow-up attack.

Routing under normal operation



Routing subject to ARP cache poisoning



ARP spoofing detection and prevention software

Software that detects ARP spoofing generally relies on some form of certification or cross-checking of ARP responses. Uncertified ARP responses are then blocked.

These techniques may be integrated with the DHCP server so that both dynamic and static IP addresses are certified. This capability may be implemented in individual hosts or may be

integrated into Ethernet switches or other network equipment. The existence of multiple IP addresses associated with a single MAC address may indicate an ARP spoof attack, although there are legitimate uses of such a configuration. In a more passive approach a device listens for ARP replies on a network, and sends a notification via email when an ARP entry changes.

AntiARP also provides Windows-based spoofing prevention at the kernel level.

ArpStar is a Linux module for kernel 2.6 and Linksys routers that drops invalid packets that violate mapping, and contains an option to repoison/heal.

Some virtualized environments such as KVM also provide security mechanisms to prevent MAC spoofing between guests running on the same host.

Additionally some **ethernet adapters provide MAC and VLAN** anti-spoofing features.

OpenBSD watches passively for hosts impersonating the local host and notifies in case of any attempt to overwrite a permanent entry

Experiment 6

Handson study of Vulnerabilities scanning tool (Angry IP, HPing2, IP Scanner)

The vulnerability scanner uses a database to compare details about the target attack surface. The database references known flaws, coding bugs, packet construction anomalies, default configurations, and potential paths to sensitive data that can be exploited by attackers

Vulnerability scanner definition

Vulnerability scanners are automated tools that allow organizations to check if their networks, systems and applications have security weaknesses that could expose them to attacks.

Scanning has three types:

Port scanning - used to list open ports and services.

Network scanning - used to list IP addresses.

Vulnerability scanning - used to discover the presence of known vulnerabilities.

Scanning of computer networks (searching for addresses with known properties) is a practice that is often used by both network administrators and crackers.

Although it is widely accepted that activity of the latter is often illegal, most of the time they depend on exactly the same tools that can be used for perfectly legitimate network administration – just like a kitchen knife that can be used maliciously.

Thanks to the recent activity of mass-media on the subject (that popularized the wrong term for a cracker – a ‘hacker’), nowadays every educated person more or less understands the reasons and goals that stand behind malicious cracking: curiosity, stealing of information, making damage, showing self-importance to the world, etc.

But why do administrators need to scan their own networks? There are plenty of answers: to check status of computers and various network devices (are they up or down), find spare addresses in statically-addressed networks, monitor the usage of server-type or P2P applications, make inventory of available hardware and software, check for recently discovered holes in order to patch them, and much more things that are even difficult to foresee.

Angry IP Scanner is a widely-used open-source and multi-platform network scanner.

As a rule, almost all such programs are open-source, because they are developed with the collaboration of many people without having any commercial goals.

Secure networks are possible only with the help of open-source systems and tools, possibly reviewed by thousands of independent experts and hackers alike.

Certainly, there are other network scanners in existence (especially single-host port scanners), however, most of them are not cross-platform, are too simple and do not offer the same level of extensibility and user-friendliness as Angry IP Scanner.

The program's target audience are network administrators, consultants, developers, who all use the tool every day and therefore have advanced requirements for usability, configurability, and extensibility.

However, Angry IP Scanner aims to be very friendly to novice users as well.

Theory of network scanning

Networking

Computer networks, especially large ones, are very heterogeneous – they are composed of many interconnected devices into subnetworks using different topologies, which are in their own turn interconnected into larger networks, etc.

The point here is that thanks to bridges between networks, all of them can use different physical (and data link) mediums for communication, with PPP over dial-up, IEEE 802.3 (Ethernet), and 802.11 (Wi-Fi) being the most popular.

The Internet Protocol

IP, in “IP address” and “IP scanner”, means nothing more complex than Internet Protocol. Nowadays, thanks to the Internet, TCP/IP is the most widely spread network protocol that over the years has replaced many other LAN and WAN protocols – it is now used in the majority of networks not even directly connected to the Internet.

An IP address is the unique identifier of a network interface in the network.

Most of the world still uses the older IPv4 version of the protocol, that limits the address space to 32 bits, making the maximum number of directly addressable nodes to be less than 4 billion, which will soon not be enough for current Earth's population of over 6 billion and the increasing usage of computers and mobile devices.

In order to fix the problem, IPv6 was introduced at the end of the previous decade, and among other features provides a much broader address space of 128 bits.

However, different tricks were employed in order to prolong IPv4 usage, such as CIDR and NAT – it is too expensive to make such a big switch.

Transport layer

While only IP protocol is fine for sending packets between hosts, there is a need to differentiate multiple senders and receivers on each host (sockets). This possibilities are provided by transport protocols UDP (User Datagram Protocol), TCP (Transmission Control Protocol), and their companion, ICMP (Internet Control Message Protocol). All these protocols are independent of whether IPv4 or IPv6 is used underneath.

Both UDP and TCP define ‘ports’ – endpoints on each host that are differentiated using their numbers (16 bits, 0 to 65535).

The notion of ‘port’ is analogous to the external ports every computer has (that are used for communicating with printers, mice, keyboards and any sort of external devices) – both provide endpoints of communication with the outside world, so physical and ‘virtual’ ports have the same name.

UDP is the simplest addition to IP possible, providing unreliable point-to-point packet transmission, while TCP encapsulates ‘streams’ of data, providing internal handshaking and acknowledgment sending mechanisms in order to provide reliable data transmission.

While majority of services rely on TCP, there are some that do not require the overhead of handshakes, automatic retransmissions, etc – they use UDP, e.g. DNS (Domain Name Service), real-time audio and video streaming, multiplayer games, etc. ICMP is for various control messages interchanged by hosts and other network devices, used for TCP, UDP, and general IP packet transmission.

Scanning

The word scan is derived from the Latin word scandere, which means to climb and later came to mean “to scan a verse of poetry,” because one could beat the rhythm by lifting and putting down one’s foot.

The Middle English verb scannen, derived from scandere, came into Middle English in this sense (first recorded in a text composed before 1398).

In the 16th century this highly specialized sense having to do with the close analysis of verse developed other senses, such as “to criticize, examine minutely, interpret, perceive.” From these senses having to do with examination and perception, it was an easy step to the sense “to look at searchingly” (first recorded in 1798). In modern language, it usually may mean: “to examine closely”, “to look over quickly and systematically”, “to analyze”. In electronics, it usually means: “to move a finely focused beam [of light, electrons, radar] in a systematic pattern”. All these definitions can also be applied to the meaning used in computer technology that we are going to discuss below.

What does a network scanner able to do? There are usually two types of network scanners: port scanners and IP scanners. Port scanners usually scan TCP and sometimes UDP ports of a single host by sequentially probing each of them. This is similar to walking around a shopping mall and writing down the list of all the shops you see there along with their status (open or closed).

Another type are IP scanners that scan many hosts and then gather additional information about those of them that are available (alive).

According to the shopping mall analogy, that would be walking around the city looking for all shopping malls and then discovering all kinds of shops that exist in each of the malls.

As Angry IP Scanner is an IP scanner, designed for scanning of multiple hosts, this will be the type of network scanner reviewed in the following text

As a rule, user provides a list of IP addresses to the scanner with the goal of sequentially probing all of them and gathering interesting information about each address as well as overall statistics.

The gathered information may include the following:

whether the host is up (alive, responding) or down (dead, not responding)

average roundtrip time (of IP packets to the destination address and back) – the same value as shown by the ping program

TTL (time to live) field value from the IP packet header, which can be used to find out the rough distance to the destination address (in number of routers the packet has traveled)

host and domain name (by using a DNS reverse lookup)

versions of particular services running on the host (e.g., “Apache 2.0.32 (Linux 2.6.9)” in case of a web server) open (responding) and filtered TCP and UDP port numbers

... and much more

The list of addresses for scanning is most often provided as a range, with specified starting and ending addresses, or as a network, with specified network address and corresponding netmask.

Other options are also possible, e.g. loading from a file or generation of random addresses according to some particular rules.

Angry IP Scanner has several different modules for generation of IP addresses called feeders.

Additional feeders can be added with the help of plugins

Safety and Security

The question of safety is always asked about security tools, like network scanners. So, how safe it is to use such programs?

Fortunately, the short response is that it is both legal and safe, however with some exceptions.

Even though nowadays legal laws do not catch up with the fast development of the IT world, network scanning has existed for almost as long as the networks themselves, meaning that there was probably enough time to update the laws.

Nevertheless, scanning itself remains perfectly legal, because in most cases it neither harms the scanned systems in any way nor provides any direct possibilities of breaking into them.

Network scanning is even used by some popular network applications for automatic

Modularity and extensibility

Angry IP Scanner is an open-source program.

At first sight it may seem that open-source software can be monolithic – users will be able to extend its functionality anyway, by editing the source code. However, in most cases this is not true.

Very often, in order to make a significant change in the code, developer must spend quite a lot of time reading and debugging the code to understand how it works and where to make the exact modification.

And it is widely known that reading of code is often more difficult than writing, especially if the original author has not put any effort to make the code extensible.

Feeders

Cryptography and Information Security 705 | SIRTE Bhopal

User selects one feeder prior to scanning and configures appropriately in order to provide the desired sequence of IP addresses to the Scanner. Built-in feeders include:

IP Range – iterates IP addresses beginning and ending with the two provided addresses, e.g. from 192.168.0.1 to 192.168.0.255

Random – generates the requested number of random IP addresses according to the provided bit mask (in order to define some portions of each generated address), e.g. 100 addresses starting with 192. and ending with .125.

IP List File – extracts IP addresses from any text file provided by the user.

The file may be in any format – the feeder looks for all tokens similar to IP addresses in it, so output of any exporter can be used later as an input for a new scan.

Advanced – provides the ability to specify more complex ruler for generation in textual form (for advanced users), e.g. 192.168-170.150.1-255 or 192.168.0.0/24.

Fetchers

User selects several fetchers prior to scanning.

The list of selected fetchers defines the type and amount of information collected about each scanned IP address.

Built-in fetchers include (implemented and planned):

IP address – the simplest fetcher that just shows the current scanned IP address

Ping – displays the roundtrip time of packets to the host and back using the selected Pinger (see below)

TTL – displays the TTL value from the IP header of returned packets from the host (available in case of certain Pingers only) Hostname – displays the hostname obtained using the DNS reverse lookup using the IP address

MAC address – MAC (hardware) address of the host's physical network interface (if available), obtained using an ARP request.

This only makes sense on a local network.

NetBIOS username/computer/workgroup – three fetchers specific to Windows hosts, use NetBIOS requests to obtain the information.

Ports – obtains the list of open TCP ports on the scanned host.

Note that only port numbers specified by user are scanned.

Filtered ports – obtains the list of filtered TCP ports on the scanning host. Filtered ports are those filtered by firewalls or routers, so there must be some reason why specifically these ports are being hidden.

Version detection – tries to detect the services and their versions behind open ports

Pingers

Pingers are used internally by Ping and TTL fetchers, but they are special because subsequent fetchers depend on pinging results for decision whether to continue scanning the address and adaptation of timeouts. The following internal pingers are implemented (each appropriate for different situations):

ICMP echo – the standard pinging method used by the ping program. The drawback of this method is that many firewalls specifically block these packets and sending of them requires the usage of raw sockets and therefore, administrator privileges on most platforms.

Windows ICMP.DLL – because of the recent removal of raw socket support from Windows (see Scanning on Different Platforms, page 40), there was a need for alternative Windows-specific implementation for ICMP echo pinging.

UDP – sends UDP packets to a port that is likely to be closed. Hosts usually respond with an negative * ICMP packet if this is the case, telling the scanner that host is actually alive (responding)

TCP – makes a connection attempt to port 80 on the host, because it is likely not to be filtered. Both positive and negative responses from the host mean that it is alive.

ARP – not implemented yet, but may provide advantages over all other methods in local networks by using a physical layer ARP requests, bypassing firewalls.

Exporters

Exporters are used after the scanning has been completed in order to export the results outside of Angry IP Scanner, most often to a file in some format. Built-in exporters include:

TXT – human readable plain text file

CSV – comma-separated values (in the order of selected fetchers)

XML – well-formed XML for machine processing. Can be later post-processed by a custom XSL template.

IP:Port list – outputs IP:port line for each open port of each alive host. These files can be read by some popular programs.

Openers

Openers – are used for “opening” any scanned host in the result list. As a rule, openers execute external programs in order to connect or send something to particular hosts, e.g. open a Web browser or send a shutdown message.

Angry IP scanner is a very fast IP address and port scanner.

It can scan IP addresses in any range as well as any of their ports. It is cross-platform and lightweight.

Not requiring any installations, it can be freely copied and used anywhere.

Angry IP scanner simply pings each IP address to check if it's alive, then optionally it is resolving its hostname, determines the MAC address, scans ports, etc.

The amount of gathered data about each host can be extended with plugins.

It also has additional features, like NetBIOS information (computer name, workgroup name, and currently logged in Windows user), favorite IP address ranges, web server detection, customizable openers, etc.

Scanning results can be saved to CSV, TXT, XML or IP-Port list files.

With help of plugins, Angry IP Scanner can gather any information about scanned IPs.

Anybody who can write Java code is able to write plugins and extend functionality of Angry IP Scanner.

Is the program infected with a virus?

Important: There are no trojans or viruses in Angry IP Scanner's ipscan.exe.

Meaning of TTL in Angry IP

TTL means Time To Live.

This is the value present in every IP (Internet Protocol) packet's header (TCP, UDP, ICMP - all have it).

TTL field's size is one byte, so its value is 0-255.

Originally, the value of TTL meant for how long the packet is supposed to be traveling in the network (in seconds) in order to prevent 'lost' or unroutable packets traveling in the network forever.

How it works

Every router that forwards the packet is supposed to decrease its TTL value before sending it further either by 1 or the number of seconds it took the packet to reach it from the previous node, so as the networks are generally fast, TTL actually means the maximum number of nodes the packet is allowed to travel.

Different platforms set initial TTL value to different values, however the most common ones are 64, 128, and 255.

Windows machines usually set it to 128, Linux ones to 64.

TTL column in scanning results

Angry IP Scanner shows the TTL value of received ping packets. From its value you can have the idea of 'how far' the scanned host is from you, in the number of routers/nodes.

For example, if TTL column shows 119, then it means that most probably:

Initial value was 128

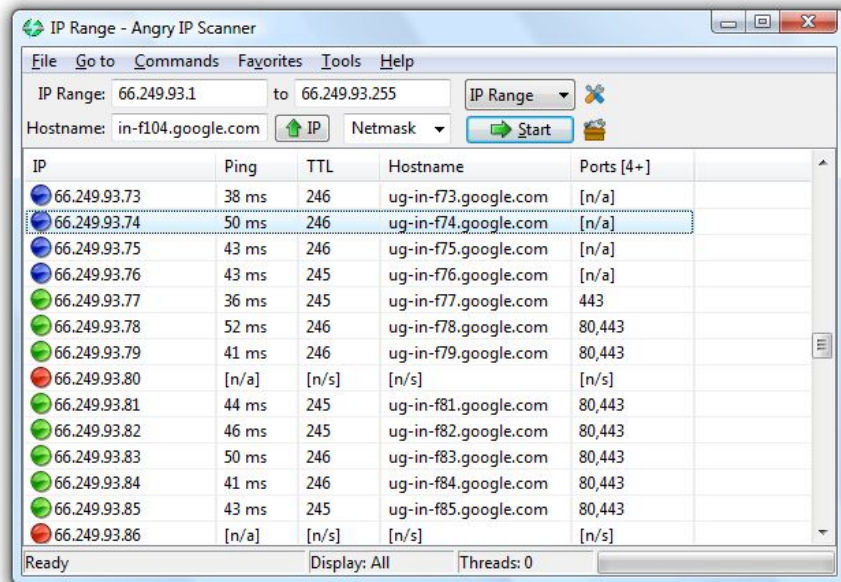
Scanned host is a Windows box

The host is 9 routers away from you

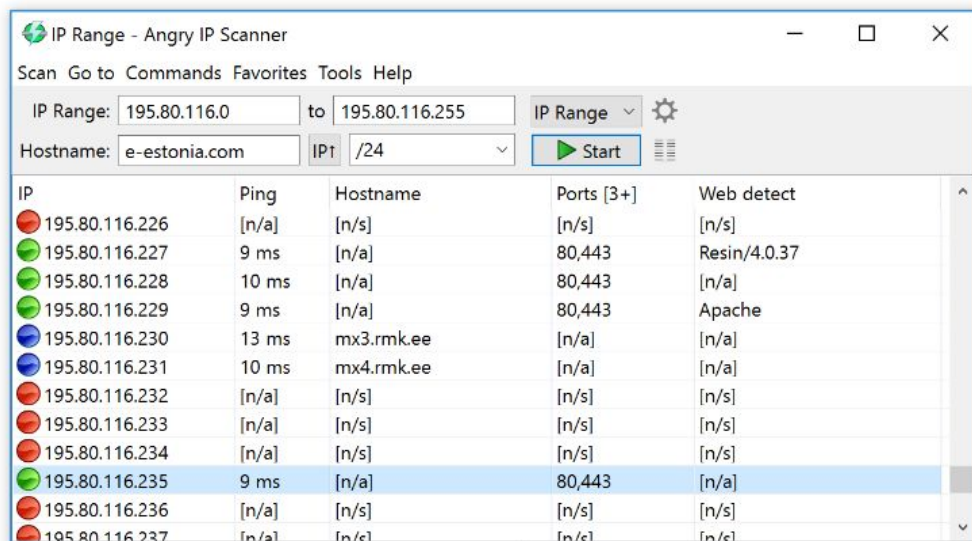
<https://angryip.org/>

Screen shot of Angry IP scanner

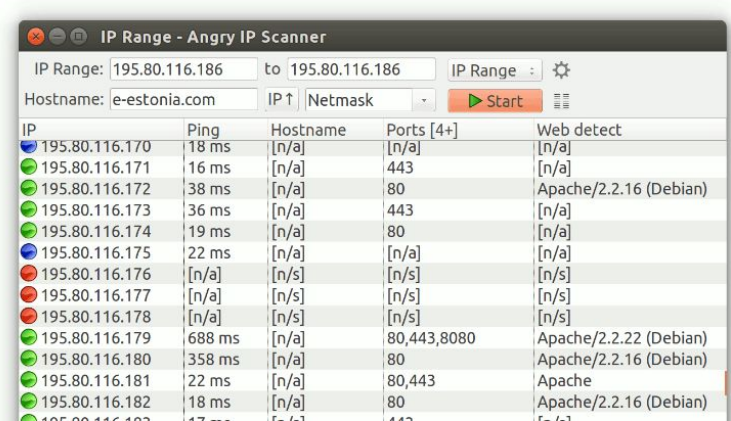
Windows 7/Vista



Windows 10 HiDPI

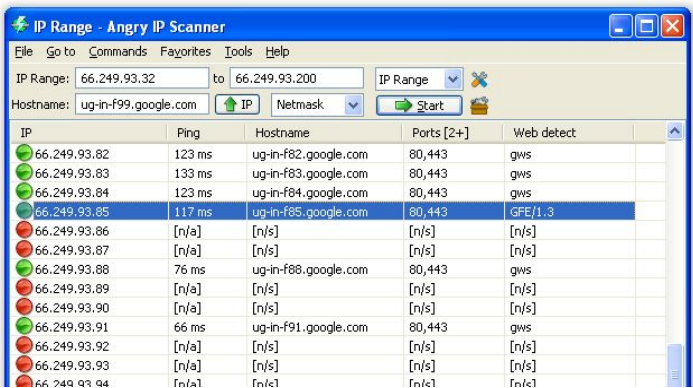


Ubuntu HiDPI



IP	Ping	Hostname	Ports [4+]	Web detect
195.80.116.170	18 ms	[n/a]	[n/a]	[n/a]
195.80.116.171	16 ms	[n/a]	443	[n/a]
195.80.116.172	38 ms	[n/a]	80	Apache/2.2.16 (Debian)
195.80.116.173	36 ms	[n/a]	443	[n/a]
195.80.116.174	19 ms	[n/a]	80	[n/a]
195.80.116.175	22 ms	[n/a]	[n/a]	[n/a]
195.80.116.176	[n/a]	[n/s]	[n/s]	[n/s]
195.80.116.177	[n/a]	[n/s]	[n/s]	[n/s]
195.80.116.178	[n/a]	[n/s]	[n/s]	[n/s]
195.80.116.179	688 ms	[n/a]	80,443,8080	Apache/2.2.22 (Debian)
195.80.116.180	358 ms	[n/a]	80	Apache/2.2.16 (Debian)
195.80.116.181	22 ms	[n/a]	80,443	Apache
195.80.116.182	18 ms	[n/a]	80	Apache/2.2.16 (Debian)

Windows XP (no longer supported)



IP	Ping	Hostname	Ports [2+]	Web detect
66.249.93.82	123 ms	ug-in-f82.google.com	80,443	gws
66.249.93.83	133 ms	ug-in-f83.google.com	80,443	gws
66.249.93.84	123 ms	ug-in-f84.google.com	80,443	gws
66.249.93.85	117 ms	ug-in-f85.google.com	80,443	GFE/1.3
66.249.93.86	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.87	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.88	76 ms	ug-in-f88.google.com	80,443	gws
66.249.93.89	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.90	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.91	66 ms	ug-in-f91.google.com	80,443	gws
66.249.93.92	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.93	[n/a]	[n/s]	[n/s]	[n/s]
66.249.93.94	[n/a]	[n/s]	[n/s]	[n/s]

Experiment No 7

Handson study of steganography tools (Stools, Steghide, Stegno's, Stegdata)

Image Steganography : Image Steganography is another free program for hiding your information in image files. You can hide text messages or files inside an image file. Just select the source file in which you want to hide the secret message and then select the file to hide or write the text message to hide. Select the output image location and then click on the start button to start encoding the file. The encoded image will have the secret message inside the image. You can use the decode option of the same tool to decode the hidden file or message.

Steghide: Steghide is open-source steganography software that lets you hide your secret file in an image or audio file. You will not notice any change in the image or audio file. However, your secret file will be inside the original image or audio file. This is command-line software. Therefore, you need to learn the commands to use the tool. Commands will be used to embed files in the image or audio file. In addition, to extract your file from the image or audio file, you need to use another command.

Crypture: Crypture is another command-line tool that performs steganography. You can use this tool to hide your sensitive data inside a BMP image file. But there is one requirement: the BMP file should be eight times larger than the data file you want to hide inside the BMP file. If you have a small amount of data to hide, you can use this tool. This tool is very small and is only 6KB in size. It does not need any kind of installation.

SteganographX Plus: SteganographX Plus is another small tool that lets you hide your confidential data inside a BMP image. It also does not need any installation and is of only 496 KB in size. This is simple and offers an easy-to use-interface. You can use this tool to hide your sensitive data inside a BMP file and recover your data from that file.

rSteg : rSteg is a Java-based tool that lets you hide textual data inside an image. It has two buttons: one to encrypt and second to decrypt the text. Just select the image file, enter the PIN and then enter the text which you want to hide in the image. It will generate a target image file with the hidden text inside. If you want to read that text again, use this tool and select the decrypt option.

SSuite Pícel : SSuite Pícel is a free portable application to hide text inside an image file. However, it has a different approach. It uses the image file as a key to protect your hidden text inside an image. Don't be confused. Actually, this tool can hide text inside an image file. Nevertheless, to hide and reveal text inside an image, you need to enter another image as a key. To hide text inside the image, select the image in which you want to hide the text and select another image for the key. Now you can hide your text inside the first image. To reveal the text, you need to enter the key image.

Our Secret: Our Secret is another tool that is used to hide sensitive information in a file.

The interface of the tool is divided into two parts: one part is to hide the data in a file and the other part is to reveal. You need to select the carrier file in which you want to hide your data. Then select the data

or file you want to hide. Enter the password to encrypt your message and then hide the data in the file. Use the same tool again to reveal the data.

Camouflage: Camouflage is another steganography tool that lets you hide any type of file inside of a file. There is no kind of restriction in the software for hiding the file. Use of the tool is simple and easy: you can just right-click on any file and select the Camouflage option. To extract your sensitive data from the file, right-click and select Uncamouflage. You can also set a password to encrypt the hidden data inside the file. The project is no longer in development, but you can use the old file for your work. It still performs well and you can use it to hide your confidential data inside an image.

SteganPEG: SteganPEG lets you hide any kind of file in a JPG image file. You can attach any file and give a password to hide inside a JPG file. Only SteganPEG can extract your file from that output JPG image. The output file will act like an ordinary image file and one cannot tell that the image was modified just by looking at it. The interface of the software is simple with fewer options. One can easily use this tool to hide data in a JPG image file

Hide'N'Send :

Hide'N'Send is a small utility which offers steganography. It lets you hide any kind of file behind a JPG image file. It supports hashing and encryption too. This means that you can hide your data by encrypting. This adds an extra layer of security. The interface of the tool is simple and offers two tabs — one to hide data and other to extract data. You can select the options accordingly. Just run the tool, select the image file, then select the file which you want to hide, select the encryption type and then hide the data in the image. Use the same tool again to extract the hidden information in the image. These are a few tools for hiding data inside a file. There are many other free tools on various software-hosting websites. However, you will get desired results from these tools. This technique was developed for secure communication. However, criminals and terrorist organizations also started using this for their communication. Therefore, it is very important to develop a system to detect steganography in a file. There is no easy way to detect a hidden file inside a file. You can't just open the file to see if there is something suspicious; there needs to be a proper investigation. We'll be exploring steganography detection in another article soon.

Experiment No 08

Handson study of LAN Scanner Tools (look@LAN, Wireshark, Tcpdump).

Tcpdump is a most powerful and widely used command-line packets sniffer or package analyzer tool which is used to capture or filter TCP/IP packets that are received or transferred over a network on a specific interface. It is available under most of the Linux/Unix based operating systems. tcpdump also gives us an option to save captured packets in a file for future analysis. It saves the file in a pcap format that can be viewed by tcpdump command or an open source GUI based tool called Wireshark (Network Protocol Analyzer) that reads tcpdump pcap format files.

How to Install tcpdump in Linux

Many of Linux distributions already shipped with tcpdump tool, if in case you don't have it on systems, you can install it using the following Yum command.

```
# yum install tcpdump
```

Once a tcpdump tool is installed on systems, you can continue to browse following commands with their examples.

Capture Packets from Specific Interface

The command screen will scroll up until you interrupt and when we execute tcpdump command it will capture from all the interfaces, however with -i switch only capture from desire interface.

```
# tcpdump -i eth0
```

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode

listening on eth0, link-type EN10MB (Ethernet), capture size 65535 bytes

```
11:33:31.976358 IP 172.16.25.126.ssh > 172.16.25.125.apwi-rxspooler: Flags [P.], seq 3500440357:3500440553, ack 3652628334, win 18760, length 196
```

```
11:33:31.976603 IP 172.16.25.125.apwi-rxspooler > 172.16.25.126.ssh: Flags [.), ack 196, win 64487, length 0
```

```
11:33:31.977243 ARP, Request who-has tecmint.com tell 172.16.25.126, length 28
```

```
11:33:31.977359 ARP, Reply tecmint.com is-at 00:14:5e:67:26:1d (oui Unknown), length 46
```

```
11:33:31.977367 IP 172.16.25.126.54807 > tecmint.com: 4240+ PTR? 125.25.16.172.in-addr.arpa. (44)
```

```
11:33:31.977599 IP tecmint.com > 172.16.25.126.54807: 4240 NXDomain 0/1/0 (121)
```

```
11:33:31.977742 IP 172.16.25.126.44519 > tecmint.com: 40988+ PTR? 126.25.16.172.in-addr.arpa. (44)
```

```
11:33:32.028747 IP 172.16.20.33.netbios-ns > 172.16.31.255.netbios-ns: NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST
```

Cryptography and Information Security 705 | SIRTE Bhopal

11:33:32.112045 IP 172.16.21.153.netbios-ns > 172.16.31.255.netbios-ns: NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST

11:33:32.115606 IP 172.16.21.144.netbios-ns > 172.16.31.255.netbios-ns: NBT UDP PACKET(137): QUERY; REQUEST; BROADCAST

11:33:32.156576 ARP, Request who-has 172.16.16.37 tell old-oraclehp1.midcorp.mid-day.com, length 46

11:33:32.348738 IP tecmint.com > 172.16.25.126.44519: 40988 NXDomain 0/1/0 (121)

Experiment No 9

Implementation of Caesar Cipher technique

It is a mono-alphabetic cipher wherein each letter of the plaintext is substituted by another letter to form the ciphertext. It is a simplest form of substitution cipher scheme.

This cryptosystem is generally referred to as the Shift Cipher. The concept is to replace each alphabet by another alphabet which is 'shifted' by some fixed number between 0 and 25.

For this type of scheme, both sender and receiver agree on a 'secret shift number' for shifting the alphabet. This number which is between 0 and 25 becomes the key of encryption.

The name 'Caesar Cipher' is occasionally used to describe the Shift Cipher when the 'shift of three' is used.

Process

- In order to encrypt a plaintext letter, the sender positions the sliding ruler underneath the first set of plaintext letters and slides it to LEFT by the number of positions of the secret shift.
- The plaintext letter is then encrypted to the ciphertext letter on the sliding ruler underneath. The result of this process is depicted in the following illustration for an agreed shift of three positions. In this case, the plaintext 'tutorial' is encrypted to the ciphertext 'wxwrueldo'. Here is the ciphertext alphabet for a Shift of 3 –

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

- On receiving the ciphertext, the receiver who also knows the secret shift, positions his sliding ruler underneath the ciphertext alphabet and slides it to RIGHT by the agreed shift number, 3 in this case.
- He then replaces the ciphertext letter by the plaintext letter on the sliding ruler underneath. Hence the ciphertext 'wxwrueldo' is decrypted to 'tutorial'. To decrypt a message encoded with a Shift of 3, generate the plaintext alphabet using a shift of '-3' as shown below –

Ciphertext Alphabet	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Plaintext Alphabet	x	y	z	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w

Here is the implementation of the above process in C++.

Steps and pseudocodes

Take the message and key as input –

For encryption

- **Input:** tutorial.
- **Output:** wxwrueldo

For decryption

- **Input:** wxwrueldo
- **Output:** tutorial

For encryption

Begin


```
For i = 0 to msg[i] != '\0'
    ch = msg[i]
    //encrypt for lowercase letter
    If (ch >= 'a' and ch <= 'z')
        ch = ch + key
        if (ch > 'z')
            ch = ch - 'z' + 'a' - 1
        done
    msg[i] = ch
    //encrypt for uppercase letter
    else if (ch >= 'A' and ch <= 'Z')
        ch = ch + key
        if (ch > 'Z')
            ch = ch - 'Z' + 'A' - 1
        done
    msg[i] = ch
done
done
Print Encrypted message
```

End

For decryption

Begin

```
For i = 0 to msg[i] != '\0'
    ch = msg[i]
    //decrypt for lowercase letter
    if(ch >= 'a' and ch <= 'z')
        ch = ch - key
        if (ch < 'a')
            ch = ch + 'z' - 'a' + 1
```

```
done
msg[i] = ch
//decrypt for uppercase letter
else if (ch >= 'A' and ch <= 'Z')
    ch = ch + key
    if (ch < 'A')
        ch = ch + 'Z' - 'A' + 1
done
msg[i] = ch
done
done
Print decrypted message
```

End

Example

```
#include<iostream>
#include<string.h>
using namespace std;
int main() {
    cout<<"Enter the message:\n";
    char msg[100];
    cin.getline(msg,100); //take the message as input
    int i, j, length,choice,key;
    cout << "Enter key: ";
    cin >> key; //take the key as input
    length = strlen(msg);
    cout<<"Enter your choice \n1. Encryption \n2. Decryption \n";
    cin>>choice;
    if (choice==1) //for encryption{
        char ch;
```

```
for(int i = 0; msg[i] != '\0'; ++i) {
    ch = msg[i];
    //encrypt for lowercase letter
    If(ch >= 'a' && ch <= 'z'){
        ch = ch + key;
        if(ch > 'z') {
            ch = ch - 'z' + 'a' - 1;
        }
        msg[i] = ch;
    }
    //encrypt for uppercase letter
    else if(ch >= 'A' && ch <= 'Z'){
        ch = ch + key;
        if(ch > 'Z'){
            ch = ch - 'Z' + 'A' - 1;
        }
        msg[i] = ch;
    }
}

printf("Encrypted message: %s", msg);
}

else if (choice == 2) { //for decryption
    char ch;
    for(int i = 0; msg[i] != '\0'; ++i) {
        ch = msg[i];
        //decrypt for lowercase letter
        if(ch >= 'a' && ch <= 'z') {
            ch = ch - key;
```

```
        if(ch < 'a'){
            ch = ch + 'z' - 'a' + 1;
        }
        msg[i] = ch;
    }

    //decrypt for uppercase letter
    else if(ch >= 'A' && ch <= 'Z') {
        ch = ch - key;
        if(ch < 'A') {
            ch = ch + 'Z' - 'A' + 1;
        }
        msg[i] = ch;
    }
}

cout << "Decrypted message: " << msg;
}
}
```

Output

For encryption:

Enter the message:

tutorial

Enter key: 3

Enter your choice

1. Encryption

2. Decryption

1

Encrypted message: wxwuldo

For decryption:

Enter the message:

wxwruldo

Enter key: 3

Enter your choice

1. Encryption

2. Decryption

2

Decrypted message: tutorial