Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 1

Title: To implement Ceaser Cipher.

Theory:
The Caesar cipher method is based on a mono-alphabetic cipher and is also called a shift cipher or additive cipher. Julius Caesar used the shift cipher (additive cipher) technique to communicate with his officers. For this reason, the shift cipher technique is called the Caesar cipher. The Caesar cipher is a kind of replacement (substitution) cipher, where all letter of plain text is replaced by another letter.

The formula of encryption is:
En (x) = (x + n) mod 26

The formula of decryption is:
Dn (x) = (xi - n) mod 26

If any case (Dn) value becomes negative (-ve), in this case, we will add 26 in the negative value.
E denotes the encryption
D denotes the decryption
x denotes the letters value
n denotes the key value (shift value)

Example:
Encryption:
Message - The sky is pink
Key - 5
Plaintext: T → 20 En: (20 + 5) mod 26 Ciphertext: 25→ Y
Likewise,
Encrypted message - Ymj xpd nx unsp

Decryption:
Message - Ymj xpd nx unsp
Key – 5
Plaintext: Y → 25 Dn: (25 - 5) mod 26 Ciphertext: 20→ T

Likewise,
Decrypted message - The sky is pink

Code Snapshots:

```
#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";
   cout<<"Your Choice: ";
   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 Snapshots:

Encryption:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ CeaserCipherED.cpp -o CeaserCipherED } ;
 if ($?) { .\CeaserCipherED }
Enter the message:
The sky is pink
Enter key: 5
Enter your choice
1. Encryption
2. Decryption
Your Choice: 1
Encrypted message: Ymj xpd nx unsp
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Decryption:

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ CeaserCipherED.cpp -o CeaserCipherED } ;
 if ($?) { .\CeaserCipherED }
Enter the message:
Ymj xpd nx unsp
Enter key: 5
Enter your choice
1. Encryption
2. Decryption
Your Choice: 2
Decrypted message: The sky is pink
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

1. Caesar cipher algorithm can be implemented in many encryption projects to make data secure and better.
2. Implementation of Caesar Cipher Algorithm is easy in comparison with other algorithms.
3. Caesar cipher is not that strong in terms of providing security.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 2

Title: Crack the code. (Cryptanalysis)

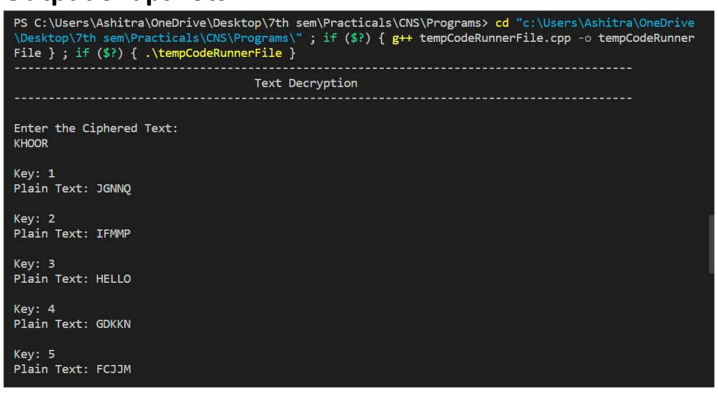
Theory:
The text gets encrypted using particular algorithm. Here, we used ceaser cipher algorithm to encrypt text. And we have tried certain permutations and combinations on the key. The many different we will get from the various keys. Choose the relevant and meaningful word. And that world which is formed will be your decrypted message. And the key used for that decrypted word will be your answered key.

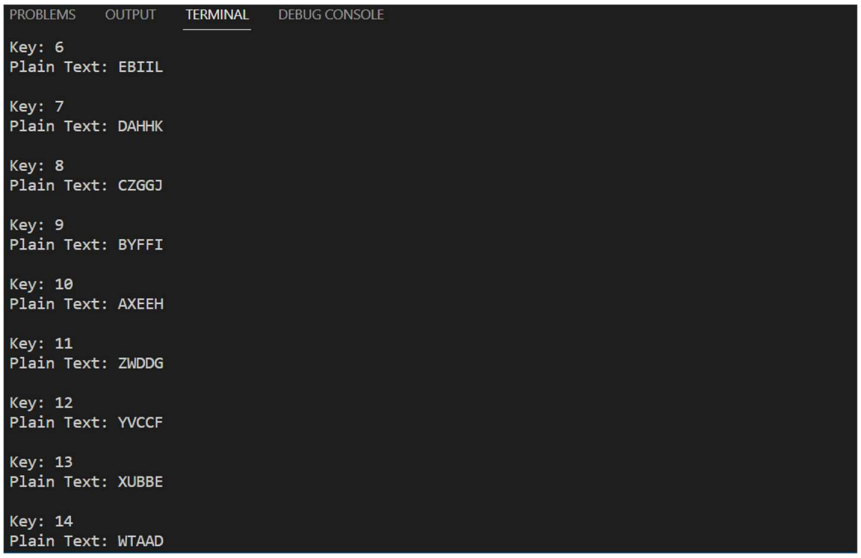
Code Snapshots:

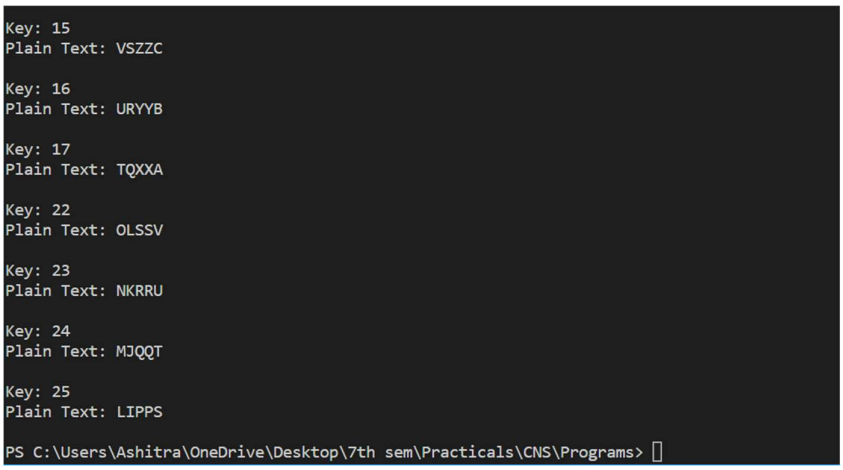
```
// Note: The following program deciphers the text using different keys.
// The user needs to identify the correct plain text from various options

#include <iostream>
#include <iomanip>

using namespace std;

int main() {

    char patternChar = '-';
    char resetChar = ' ';
    int lineWidth = 90;
    int initialWidth = 50;

    cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;
    cout << setfill(resetChar);
    cout << setw(initialWidth) << "Text Decryption" << endl;
    cout << setfill(patternChar) << setw(lineWidth) << patternChar << endl;
    cout << setfill(resetChar);


    cout << endl;
    cout << "Enter the Ciphered Text: " << endl;
```

```
    string cipherText;
    getline(cin, cipherText);

    cout << endl;


    for(size_t i = 1; i < 26; i++){
        int currentKey = i;
        string currPlainText = "";

        for(size_t j = 0; j<cipherText.size(); j++){
            char decipheredAlpha;
            if(isalpha(cipherText[j])){
                if(cipherText[j] >= 'A' && cipherText[j] <= 'Z'){
                    decipheredAlpha = (((cipherText[j]-'A')-
currentKey)+26)%26+'A';
                }else{
                    decipheredAlpha = (((cipherText[j]-'a')-
currentKey)+26)%26+'a';
                }
            }else{
                decipheredAlpha = cipherText[j];
            }

            currPlainText += decipheredAlpha;
        }

        cout << "Key: " << currentKey << endl;
        cout << "Plain Text: " << currPlainText << "\n" << endl;
    }

    return 0;
}
```

Output Snapshots:







Conclusion:

1. By trying permutations and combinations for the key value, the message can be decrypted easily.
2. The most relevant and meaningful world from the keys can be chosen as the decryption key for the message.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 3

Title: To implement Playfair Cipher Cipher.

Theory:
Playfair cipher is an encryption algorithm to encrypt or encode a message. It is the same as a traditional cipher. The only difference is that it encrypts a digraph (a pair of two letters) instead of a single letter.
It initially creates a key-table of 5*5 matrix. The matrix contains alphabets that act as the key for encryption of the plaintext. Note that any alphabet should not be repeated. Another point to note that there are 26 alphabets and we have only 25 blocks to put a letter inside it. Therefore, one letter is excess so, a letter will be omitted (usually J) from the matrix. Nevertheless, the plaintext contains J, then J is replaced by I. It means treat I and J as the same letter, accordingly.
Since Playfair cipher encrypts the message digraph by digraph. Therefore, the Playfair cipher is an example of a digraph substitution cipher.

Example:
Suppose, the plaintext is COMMUNICATION and the key that we will use to encipher the plaintext is COMPUTER.

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

Therefore, the plaintext COMMUNICATE gets encipher (encrypted) into OMRMPCSGPTER.

Code Snapshots:

```
#include<iostream>
#include<string>
#include<vector>
#include<map>
using namespace std;
int main(){
    int i,j,k,n;
    cout<<"Enter the message: ";
    string s,origin;
    getline(cin,origin);
    cout<<"Enter the key: ";
    string key;
    cin>>key;
    int choice;

    cout<<"Enter your choice(1 OR 2)."<<endl;
    cout<<"1. Encryption"<<endl;
    cout<<"2. Decryption"<<endl;
    cout<<"Your choice: ";
    cin>>choice;
    if(choice==1)
    {
    for(i=0;i<origin.size();i++){
        if(origin[i]!=' ')
            s+= origin[i];
    }
    vector<vector<char> > a(5,vector<char>(5,' '));
    n=5;
    map<char,int> mp;
    k=0;
    int pi,pj;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            while(mp[key[k]]>0&&k<key.size()){
                k++;
            }
            if(k<key.size()){
                a[i][j]=key[k];
                mp[key[k]]++;
                pi=i;
                pj=j;
            }
            if(k==key.size())
            break;
        }
        if(k==key.size())
            break;
```

```
    }
    k=0;
    for(;i<n;i++){
        for(;j<n;j++){
            while(mp[char(k+'a')]>0&&k<26){
                k++;
            }
            if(char(k+'a')=='j'){
                j--;
                k++;
                continue;
            }
            if(k<26){
                a[i][j]=char(k+'a');
                mp[char(k+'a')]++;
            }
        }
        j=0;
    }
    string ans;
    if(s.size()%2==1)
        s+="x";
    for(i=0;i<s.size()-1;i++){
        if(s[i]==s[i+1])
            s[i+1]='x';
    }
    map<char,pair<int,int> > mp2;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            mp2[a[i][j]] = make_pair(i,j);
        }
    }

    for(i=0;i<s.size()-1;i+=2){
        int y1 = mp2[s[i]].first;
        int x1 = mp2[s[i]].second;
        int y2 = mp2[s[i+1]].first;
        int x2 = mp2[s[i+1]].second;
        if(y1==y2){
            ans+=a[y1][(x1+1)%5];
            ans+=a[y1][(x2+1)%5];
        }
        else if(x1==x2){
            ans+=a[(y1+1)%5][x1];
            ans+=a[(y2+1)%5][x2];
        }
        else {
            ans+=a[y1][x2];
```

```
            ans+=a[y2][x1];
        }
    }
    cout<<ans<<'\n';
    }



    if(choice==2)
    {
        vector<vector<char> > a(5,vector<char>(5,' '));
    n=5;
    map<char,int> mp;
    k=0;
    int pi,pj;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            while(mp[key[k]]>0&&k<key.size()){
                k++;
            }
            if(k<key.size()){
                a[i][j]=key[k];
                mp[key[k]]++;
                pi=i;
                pj=j;
            }
            if(k==key.size())
            break;
        }
        if(k==key.size())
            break;
    }
    k=0;
    for(;i<n;i++){
        for(;j<n;j++){
            while(mp[char(k+'a')]>0&&k<26){
                k++;
            }
            if(char(k+'a')=='j'){
                j--;
                k++;
                continue;
            }
            if(k<26){
                a[i][j]=char(k+'a');
                mp[char(k+'a')]++;
            }
        }
        j=0;
```

```
    }
    string ans;
    map<char,pair<int,int> > mp2;
    for(i=0;i<n;i++){
        for(j=0;j<n;j++){
            mp2[a[i][j]] = make_pair(i,j);
        }
    }
    for(i=0;i<origin.size()-1;i+=2){
        int y1 = mp2[origin[i]].first;
        int x1 = mp2[origin[i]].second;
        int y2 = mp2[origin[i+1]].first;
        int x2 = mp2[origin[i+1]].second;
        if(y1==y2){
            ans+=a[y1][(x1-1)%5];
            ans+=a[y1][(x2-1)%5];
        }
        else if(x1==x2){
            ans+=a[(y1-1)%5][x1];
            ans+=a[(y2-1)%5][x2];
        }
        else {
            ans+=a[y1][x2];
            ans+=a[y2][x1];
        }
    }
    if(ans[ans.size()-1]=='x')
        ans[ans.size()-1]='\0';
    for(i=1;i<ans.size();i++){
        if(ans[i]=='x')
            ans[i]=ans[i-1];
    }

    cout<<ans<<'\n';
    }
    return 0;
}
```

Output Snapshots:

Encryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunner
File } ; if ($?) { .\tempCodeRunnerFile }
Enter the message: Communication
Enter the key: Computer
Enter your choice(1 OR 2).
1. Encryption
2. Decryption
Your choice: 1
ompwpqsibekCxp
```

Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunner
File } ; if ($?) { .\tempCodeRunnerFile }
Enter the message: ompwpqsibekCxp
Enter the key: Computer
Enter your choice(1 OR 2).
1. Encryption
2. Decryption
Your choice: 2
Communication
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

1. Playfair cipher is a symmetric encryption technique which is rich enough to encrypt all alphabets, numerals and most commonly used special symbols.
2. It uses trigraph rather than using digraph to eliminate the fact that a diagram and its reverse will encrypt in a similar fashion.
3. It is comparatively stronger algorithm.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 4

Title: To implement vigener cipher.

Theory:
The vigenere cipher is an algorithm that is used to encrypting and decrypting the text. The vigenere cipher is an algorithm of encrypting an alphabetic text that uses a series of interwoven caesar ciphers. It is based on a keyword's letters. It is an example of a polyalphabetic substitution cipher. This algorithm is easy to understand and implement. This algorithm was first described in 1553 by Giovan Battista Bellaso. It uses a Vigenere table or Vigenere square for encryption and decryption of the text. The vigenere table is also called the tabula recta.

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

Example:
Suppose the plaintext is COMMUNICATION and the keyword is COMPUTER.

Encrypted text – ECYBOGMTCHUDH

Decrypted text - COMMUNICATION

Code Snapshots:

```
#include<iostream>

#include<vector>
#include<string>
using namespace std;
int main(){
    int i,j,k,n;
    vector<vector<char> > a(26,vector<char>(26));
    k=0;
    n=26;
    for(i=0;i<n;i++){
        k=i;
        for(j=0;j<n;j++){
            a[i][j]='A'+k;
            k++;
            if(k==26)
                k=0;
        }
    }
    cout<<"Enter the message: ";
    string s;
    getline(cin,s);
    cout<<"Enter the key: ";
    string key;
    cin>>key;
    k=0;
    int mod = key.size();

    cout<<"Enter your choice."<<endl;
    cout<<"1. Encryption\n";
    cout<<"2. Decryption\n";
    cout<<"Your choice: ";
    int choice;
    cin>>choice;

    if(choice==1)
    {
    for(i=key.size();i<s.size();i++){
        key+=key[k%mod];
        k++;
    }
    string encrypt;
    for(i=0;i<s.size();i++){
        encrypt+= a[s[i]-'A'][key[i]-'A'];
    }
    cout<<"Encrypted message: "<<encrypt<<'\n';
```

```
}
    if(choice==2)
    {
        for(i=key.size();i<s.size();i++){
        key+=key[k];
        k++;
    }
    string decrypt;
    for(i=0;i<s.size();i++){
        for(j=0;j<n;j++){
            if(a[j][key[i]-'A']==s[i]){
                decrypt += 'A'+j;
                break;
            }
        }
    }
    cout<<"Decrypted message: "<<decrypt<<'\n';
    }
    return 0;
}
```

Output Snapshots:

Encryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ VigenereED.cpp -o VigenereED } ; if ($?)
 { .\VigenereED }
Enter the message: COMMUNICATION
Enter the key: COMPUTER
Enter your choice.
1. Encryption
2. Decryption
Your choice: 1
Encrypted message: ECYBOGMTCHUDH
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ VigenereED.cpp -o VigenereED } ; if ($?)
 { .\VigenereED }
Enter the message: ECYBOGMTCHUDH
Enter the key: COMPUTER
Enter your choice.
1. Encryption
2. Decryption
Your choice: 2
Decrypted message: COMMUNICATION
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

1. Vigenere Cipher is a simple polyalphabetic substitution method that goes from a simple to advanced method.

2. The Vigenere Square or Table is an important tool used in this Cipher. You can use this cipher in three different ways as per your needs. All the three methods involve different steps.
3. The autokey method is the least secure method. Even though the keyword method has its vulnerabilities, it is more secure than the autokey method. The Python Code method is relatively the most secure method.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4

Practical No – 4

Title: To implement transposition cipher.

a. Railfence cipher
b. Columnar cipher

Theory:

a. Railfence Cipher

→

The rail fence cipher (sometimes called zigzag cipher) is a transposition cipher that jumbles up the order of the letters of a message using a basic algorithm. The rail fence cipher works by writing your message on alternate lines across the page, and then reading off each line in turn.

Example:

Let's consider the plaintext "This is a secret message".

Plaintext T H I S I S A S E C R E T M E S S A G E

To encode this message we will first write over two lines (the "rails of the fence") as follows:

Rail Fence Encoding																			
	T		I		I		A		E		R		T		E		S		G	
		H		S		S		S		C		E		M		S		A		E

Note that all white spaces have been removed from the plain text.

The **ciphertext** is then read off by writing the top row first, followed by the bottom row:

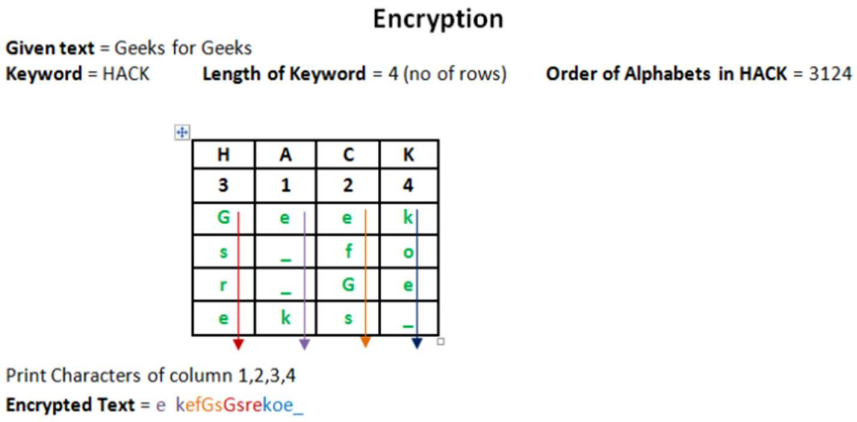
Ciphertext T I I A E R T E S G H S S S C E M S A E

b. Columnar Cipher

→

The columnar transposition cipher is a fairly simple, easy to implement cipher. It is a transposition cipher that follows a simple rule for mixing up the characters in the plaintext to form the ciphertext.

Although weak on its own, it can be combined with other ciphers, such as a substitution cipher, the combination of which can be more difficult to break than either cipher on its own. The ADFGVX cipher uses a columnar transposition to greatly improve its security.



Code Snapshots:

a. Railfence Cipher

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int t,n,m,i,j,k,sum=0;
    string s;
    cout<<"Enter the message: ";
    cin>>s;
    cout<<"Enter key: ";
    cin>>n;

     cout<<"Enter your choice."<<endl;
    cout<<"1. Encryption\n";
    cout<<"2. Decryption\n";
    cout<<"Your choice: ";
    int choice;
    cin>>choice;

    if(choice==1)
    {
    vector<vector<char>> a(n,vector<char>(s.size(),' '));
    j=0;
    int flag=0;
    for(i=0;i<s.size();i++){
        a[j][i] = s[i];
         if(j==n-1){
```

```
            flag=1;
        }
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    for(i=0;i<n;i++){
        for(j=0;j<s.size();j++){
            if(a[i][j]!=' ')
                cout<<a[i][j];
        }
    }
    cout<<'\n';
    }
    if(choice==2)
    {
        vector<vector<char>> a(n,vector<char>(s.size(),' '));
    j=0;
    int flag=0;
    for(i=0;i<s.size();i++){
        a[j][i] = '0';
         if(j==n-1){
            flag=1;
        }
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    int temp =0;
    for(i=0;i<n;i++){
        for(j=0;j<s.size();j++){
                if(a[i][j]=='0')
                    a[i][j]= s[temp++];
        }
    }
    flag=0;
    j=0;
    for(i=0;i<s.size();i++){
        cout<<a[j][i];
         if(j==n-1){
            flag=1;
        }
```

```
        else if(j==0)
            flag=0;
        if(flag==0){
            j++;
        }
        else j--;
    }
    cout<<'\n';
    }
    return 0;
}
```

b. Columnar Cipher

```
// CPP program for illustrating
// Columnar Transposition Cipher
#include<bits/stdc++.h>
using namespace std;

// Key for Columnar Transposition
string const key = "STAR";
map<int,int> keyMap;

void setPermutationOrder()
{
    // Add the permutation order into map
    for(int i=0; i < key.length(); i++)
    {
        keyMap[key[i]] = i;
    }
}

// Encryption
string encryptMessage(string msg)
{
    int row,col,j;
    string cipher = "";

    /* calculate column of the matrix*/
    col = key.length();

    /* calculate Maximum row of the matrix*/
    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')
            {
                /* Adding the padding character '_' */
                matrix[i][j] = '_';
                j++;
            }

            if( isalpha(msg[k]) || msg[k]==' ')
            {
                /* Adding only space and alphabet into matrix*/
                matrix[i][j] = msg[k];
                j++;
            }
            k++;
        }
    }

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

        // getting cipher text from matrix column wise using permuted key
        for (int i=0; i<row; i++)
        {
            if( isalpha(matrix[i][j]) || matrix[i][j]==' ' ||
matrix[i][j]=='_')
                cipher += matrix[i][j];
        }
    }

    return cipher;
}

// Decryption
string decryptMessage(string cipher)
{
    /* calculate row and column for cipher Matrix */
    int col = key.length();

    int row = cipher.length()/col;
    char cipherMat[row][col];
```

```
    /* add character into matrix column wise */
    for (int j=0,k=0; j<col; j++)
        for (int i=0; i<row; i++)
            cipherMat[i][j] = cipher[k++];

    /* update the order of key for decryption */
    int index = 0;
    for( map<int,int>::iterator ii=keyMap.begin(); ii!=keyMap.end(); ++ii)
        ii->second = index++;

    /* Arrange the matrix column wise according
    to permutation order by adding into new matrix */
    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];
        }
    }

    /* getting Message using matrix */
    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;
}

// Driver Program
int main(void)
{
    /* message */
    string msg = "THISISASECRETMESSAGE";

    setPermutationOrder();

    // Calling encryption function
    string cipher = encryptMessage(msg);
    cout << "Encrypted Message: " << cipher << endl;
```

```
    // Calling Decryption function
    cout << "Decrypted Message: " << decryptMessage(cipher) << endl;

    return 0;
}
```

Output Snapshots:

a. Railfence CIpher

Encryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ RailFenceED.cpp -o RailFenceED } ; if ($
?) { .\RailFenceED }
Enter the message: THISISASECRETMESSAGE
Enter key: 3
Enter your choice.
1. Encryption
2. Decryption
Your choice: 1
TIETSHSSSCEMSAEIAREG
```

Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ RailFenceED.cpp -o RailFenceED } ; if ($
?) { .\RailFenceED }
Enter the message: TIETSHSSSCEMSAEIAREG
Enter key: 3
Enter your choice.
1. Encryption
2. Decryption
Your choice: 2
THISISASECRETMESSAGE
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

b. Columnar Cipher

Encryption And Decryption:

```
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive
\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ ColumnarED.cpp -o ColumnarED } ; if ($?)
 { .\ColumnarED }
Encrypted Message: IAREGSSESETIETSHSCMA
Decrypted Message: THISISASECRETMESSAGE
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

a. Railfence Cipher

- 1. The Rail Fence algorithm is a simple cryptography algorithm. However, it is not secure.
- The key is how many rows is implemented. It can be guessed by making a brute-force attack.

b. Columnar Cipher

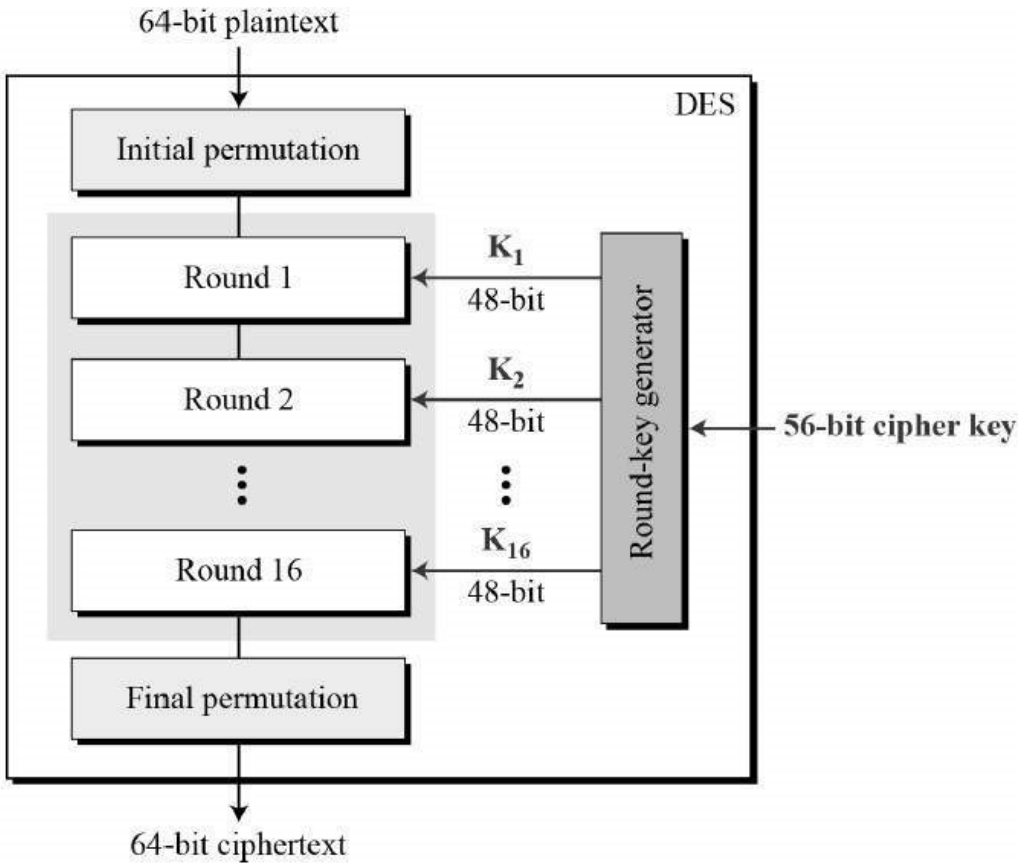
- The Columnar Transposition Cipher is a form of transposition cipher just like Rail Fence Cipher.
- Columnar Transposition involves writing the plaintext out in rows, and then reading the ciphertext off in columns one by one.

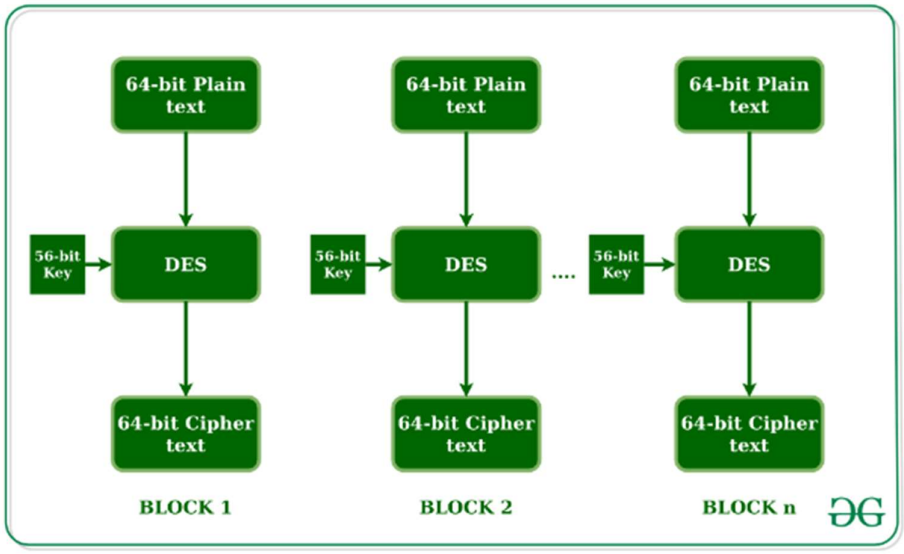
Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 6

Title: Implementation of DES Algorithm.

Theory:

Data encryption standard (DES) has been found vulnerable to very powerful attacks and therefore, the popularity of DES has been found slightly on the decline. DES is a block cipher and encrypts data in blocks of size of 64 bits each, which means 64 bits of plain text go as the input to DES, which produces 64 bits of ciphertext. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits. The basic idea is shown in the figure:





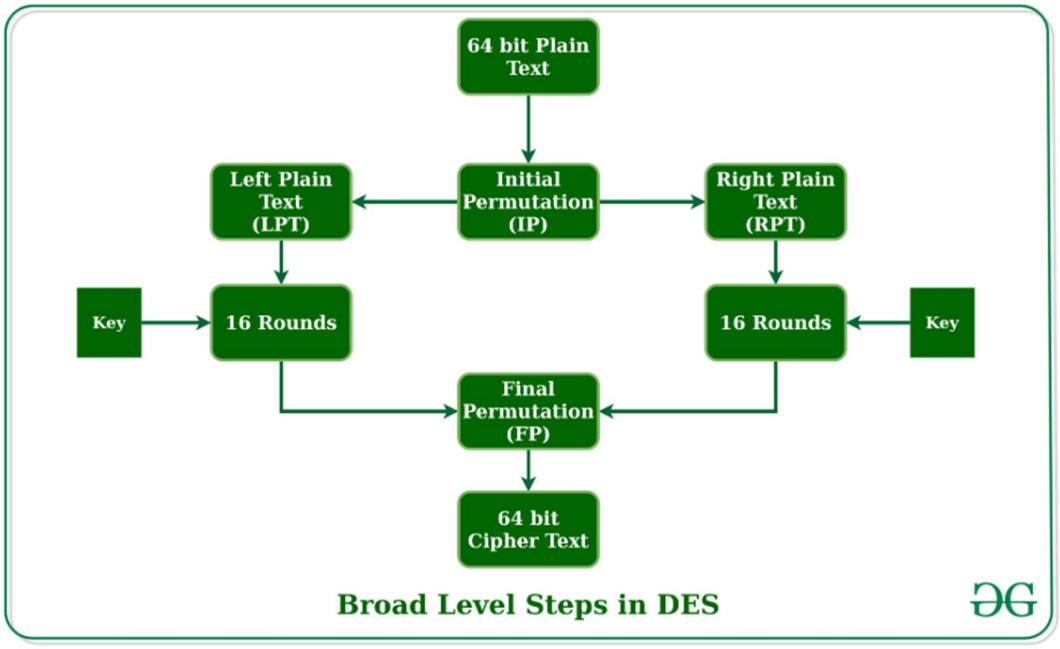
We have mentioned that DES uses a 56-bit key. Actually, the initial key consists of 64 bits. However, before the DES process even starts, every 8th bit of the key is discarded to produce a 56-bit key. That is bit positions 8, 16, 24, 32, 40, 48, 56, and 64 are discarded.

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

Figure - discording of every 8th bit of original key

Thus, the discarding of every 8th bit of the key produces a **56-bit key** from the original **64-bit key**. DES is based on the two fundamental attributes of cryptography: substitution (also called confusion) and transposition (also called diffusion). DES consists of 16 steps, each of which is called a round. Each round performs the steps of substitution and transposition. Let us now discuss the broad-level steps in DES.

- In the first step, the 64-bit plain text block is handed over to an initial Permutation (IP) function.
- The initial permutation is performed on plain text.
- Next, the initial permutation (IP) produces two halves of the permuted block; saying Left Plain Text (LPT) and Right Plain Text (RPT).
- Now each LPT and RPT go through 16 rounds of the encryption process.
- In the end, LPT and RPT are rejoined and a Final Permutation (FP) is performed on the combined block
- The result of this process produces 64-bit ciphertext.



Initial Permutation (IP):

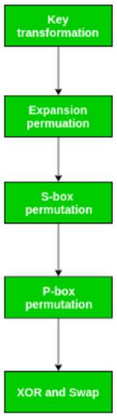
As we have noted, the initial permutation (IP) happens only once and it happens before the first round. It suggests how the transposition in IP should proceed, as shown in the figure. For example, it says that the IP replaces the first bit of the original plain text block with the 58th bit of the original plain text, the second bit with the 50th bit of the original plain text block, and so on.

This is nothing but jugglery of bit positions of the original plain text block. the same rule applies to all the other bit positions shown in the figure.

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	33	45	37	29	21	13	5	63	55	47	39	31	23	15	7

Figure - Initial permutation table

As we have noted after IP is done, the resulting 64-bit permuted text block is divided into two half blocks. Each half-block consists of 32 bits, and each of the 16 rounds, in turn, consists of the broad-level steps outlined in the figure.



Step-1: Key transformation:

We have noted initial 64-bit key is transformed into a 56-bit key by discarding every 8th bit of the initial key. Thus, for each a 56-bit key is available. From this 56-bit key, a different 48-bit Sub Key is generated during each round using a process called key transformation. For this, the 56-bit key is divided into two halves, each of 28 bits. These halves are circularly shifted left by one or two positions, depending on the round.

For example: if the round numbers 1, 2, 9, or 16 the shift is done by only one position for other rounds, the circular shift is done by two positions. The number of key bits shifted per round is shown in the figure.

Round	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
#key bits shifted	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

Figure - number of key bits shifted per round

After an appropriate shift, 48 of the 56 bits are selected. for selecting 48 of the 56 bits the table is shown in the figure given below. For instance, after the shift, bit number 14 moves to the first position, bit number 17 moves to the second position, and so on. If we observe the table carefully, we will realize that it contains only 48-bit positions. Bit number 18 is discarded (we will not find it in the table), like 7 others, to reduce a 56-bit key to a 48-bit key. Since the key transformation process involves permutation as well as a selection of a 48-bit subset of the original 56-bit key it is called Compression Permutation.

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

Figure - compression permutation

Because of this compression permutation technique, a different subset of key bits is used in each round. That makes DES not easy to crack.

Step-2: Expansion Permutation:

Recall that after the initial permutation, we had two 32-bit plain text areas called Left Plain Text(LPT) and Right Plain Text(RPT). During the expansion permutation, the RPT is expanded from 32 bits to 48 bits. Bits are permuted as well hence called expansion permutation. This happens as the 32-bit RPT is divided into 8 blocks, with each block consisting of 4 bits. Then, each 4-bit block of the previous step is then expanded to a corresponding 6-bit block, i.e., per 4-bit block, 2 more bits are added.

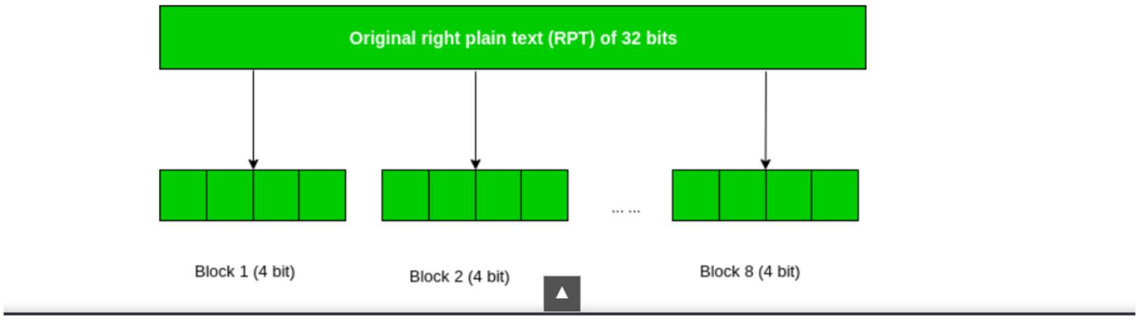


Figure - division of 32 bit RPT into 8 bit blocks

This process results in expansion as well as a permutation of the input bit while creating output. The key transformation process compresses the 56-bit key to 48 bits. Then the expansion permutation process expands the **32-bit RPT** to **48-bits**. Now the 48-bit key is XOR with 48-bit RPT and the resulting output is given to the next step, which is the **S-Box substitution**.

Code Snapshots:

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

string hexToBin(string s) {
    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";
    stringstream bin;
    for (int i = 0; i < s.size(); i++) {
        bin << mp[s[i]];
    }
    return bin.str();
}
string binToHex(string s) {
    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";
    stringstream hex;
    for (int i = 0; i < s.length(); i += 4) {
```

```
        string ch = s.substr(i, 4);
        hex << mp[ch];
    }
    return hex.str();
}

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

string shiftLeft(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) {
    stringstream ans;
    for (int i = 0; i < a.size(); i++) {
        if (a[i] == b[i]) {
            ans << "0";
        } else {
            ans << "1";
        }
    }
    return ans.str();
}

string encrypt(string plain, vector<string> rkb, vector<string> rk) {
    // Hexadecimal to binary
    plain = hexToBin(plain);

    // 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
    plain = permute(plain, initial_perm, 64);
    cout << "After initial permutation: " << binToHex(plain) << endl;

    // Splitting
    string left = plain.substr(0, 32);
    string right = plain.substr(32, 32);
    cout << "After splitting: L0=" << binToHex(left)
         << " R0=" << binToHex(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');
```

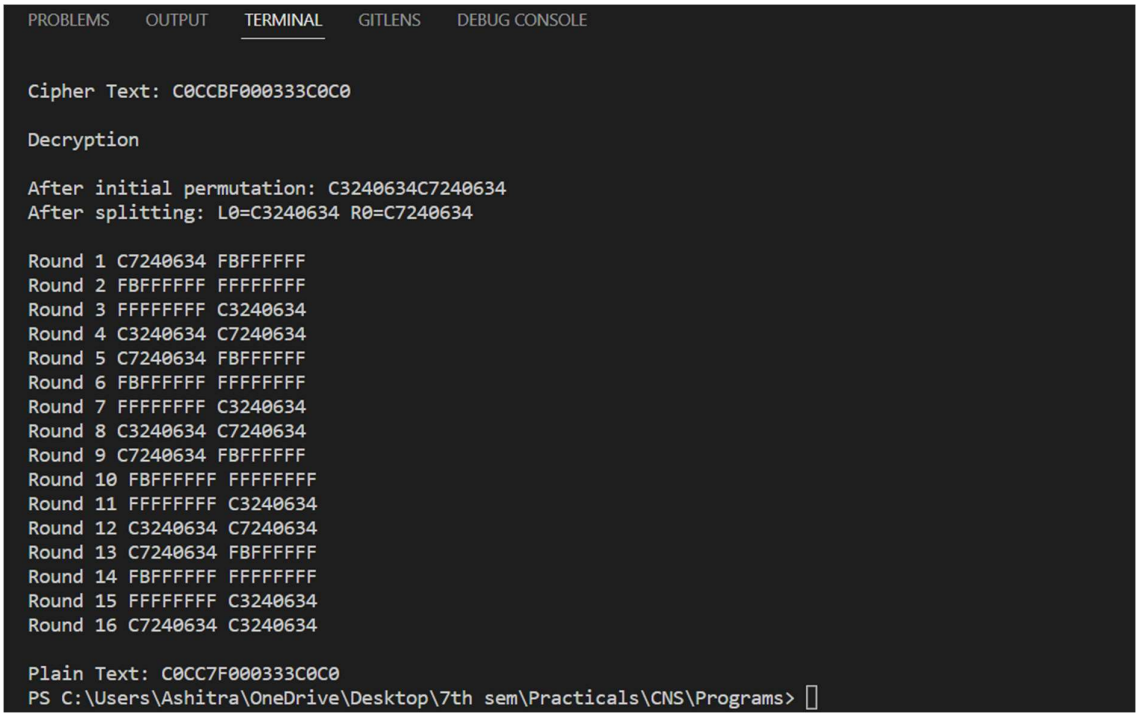
```
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Straight D-box
        op = permute(op, per, 32);

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

        left = x;

        // Swapper
        if (i != 15) {
            swap(left, right);
        }
        cout << "Round " << i + 1 << " " << binToHex(left) << " "
             << binToHex(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 = binToHex(permute(combine, final_perm, 64));
    return cipher;
}
int main() {
    string plain, key;

    // plain = "This is a test text";
    // key = "this is a test";
    // Key Generation
```

```
    cout << "Enter the plain text: ";
    getline(cin, plain);
    cout << "Enter the key: ";
    getline(cin, key);

    // Hex to binary
    key = hexToBin(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 = shiftLeft(left, shift_table[i]);
        right = shiftLeft(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(binToHex(RoundKey));
    }

    cout << "\nEncryption:\n\n";
    string cipher = encrypt(plain, 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 Snapshots:

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ DES.cpp -o DES } ; if ($?) { .\DES }
Enter the plain text: Hello World
Enter the key: test

Encryption:

After initial permutation:
After splitting: L0= R0=

Round 1  FFFFFFFF
Round 2 FFFFFFFF FBFFFFFF
Round 3 FBFFFFFF C7240634
Round 4 C7240634 C3240634
Round 5 C3240634 FFFFFFFF
Round 6 FFFFFFFF FBFFFFFF
Round 7 FBFFFFFF C7240634
Round 8 C7240634 C3240634
Round 9 C3240634 FFFFFFFF
Round 10 FFFFFFFF FBFFFFFF
Round 11 FBFFFFFF C7240634
Round 12 C7240634 C3240634
Round 13 C3240634 FFFFFFFF
Round 14 FFFFFFFF FBFFFFFF
Round 15 FBFFFFFF C7240634
Round 16 C3240634 C7240634
```



Conclusion:

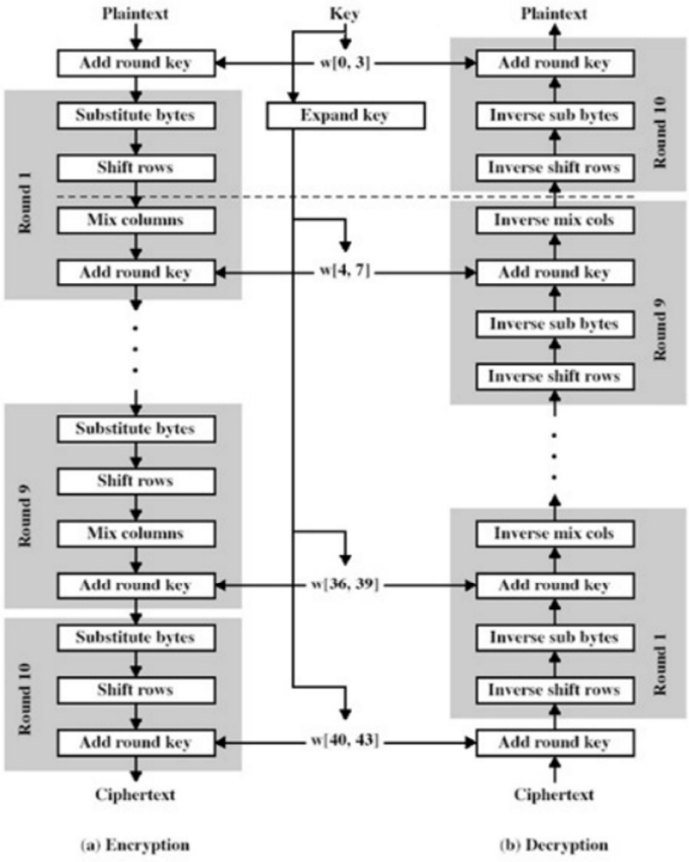
1. DES is a symmetric block cipher that can be used to encrypt 64-bits of plaintext into 64-bits of ciphertext.
2. The algorithm is the same for the process of encryption as well as decryption. The only difference is that the decryption procedure is the opposite of the encryption procedure.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 7

Title: Implementation of AES Algorithm.

Theory:

Advanced Encryption Standard (AES) is a specification for the encryption of electronic data established by the U.S National Institute of Standards and Technology (NIST) in 2001. AES is widely used today as it is a much stronger than DES and triple DES despite being harder to implement.



Points to remember

- AES is a block cipher.
- The key size can be 128/192/256 bits.
- Encrypts data in blocks of 128 bits each.

That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

Working of the cipher :

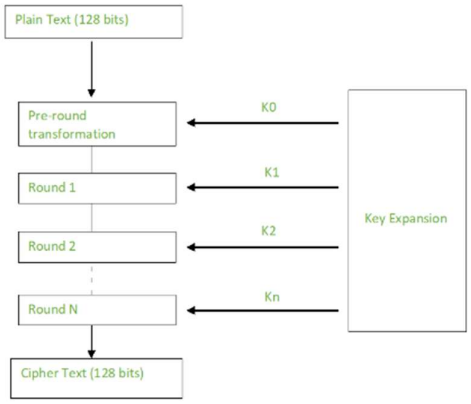
AES performs operations on bytes of data rather than in bits. Since the block size is 128 bits, the cipher processes 128 bits (or 16 bytes) of the input data at a time.

The number of rounds depends on the key length as follows :

- 128 bit key – 10 rounds
- 192 bit key – 12 rounds
- 256 bit key – 14 rounds

Creation of Round keys :

A Key Schedule algorithm is used to calculate all the round keys from the key. So the initial key is used to create many different round keys which will be used in the corresponding round of the encryption.



Encryption :

AES considers each block as a 16 byte (4 byte x 4 byte = 128) grid in a column major arrangement.

```
[ b0 | b4 | b8 | b12 |
| b1 | b5 | b9 | b13 |
| b2 | b6 | b10| b14 |
| b3 | b7 | b11| b15 ]
```

Each round comprises of 4 steps :

- SubBytes
- ShiftRows
- MixColumns
- Add Round Key

The last round doesn't have the MixColumns round.

The SubBytes does the substitution and ShiftRows and MixColumns performs the permutation in the algorithm.

SubBytes :

This step implements the substitution.

SubBytes :

This step implements the substitution.

In this step each byte is substituted by another byte. Its performed using a lookup table also called the S-box. This substitution is done in a way that a byte is never substituted by itself and also not substituted by another byte which is a compliment of the current byte. The result of this step is a 16 byte (4 x 4) matrix like before.

The next two steps implement the permutation.

ShiftRows :

This step is just as it sounds. Each row is shifted a particular number of times.

- The first row is not shifted
- The second row is shifted once to the left.
- The third row is shifted twice to the left.
- The fourth row is shifted thrice to the left.

(A left circular shift is performed.)

```
[ b0  | b1  | b2  | b3  ]         [ b0  | b1  | b2  | b3  ]
| b4  | b5  | b6  | b7  |    ->   | b5  | b6  | b7  | b4  |
| b8  | b9  | b10 | b11 |         | b10 | b11 | b8  | b9  |
[ b12 | b13 | b14 | b15 ]         [ b15 | b12 | b13 | b14 ]
```

MixColumns :

This step is basically a matrix multiplication. Each column is multiplied with a specific matrix and thus the position of each byte in the column is changed as a result.

This step is skipped in the last round.

```
[ c0 ]         [ 2  3  1  1 ]  [ b0 ]
| c1 |  =      | 1  2  3  1 |     | b1 |
| c2 |      | 1  1  2  3 |     | b2 |
[ c3 ]      [ 3  1  1  2 ]     [ b3 ]
```

Add Round Keys :

Now the resultant output of the previous stage is XOR-ed with the corresponding round key. Here, the 16 bytes is not considered as a grid but just as 128 bits of data.

After all these rounds 128 bits of encrypted data is given back as output. This process is repeated until all the data to be encrypted undergoes this process.

Decryption :

The stages in the rounds can be easily undone as these stages have an opposite to it which when performed reverts the changes.Each 128 blocks goes through the 10,12 or 14 rounds depending on the key size.

The stages of each round in decryption is as follows :

- Add round key
- Inverse MixColumns
- ShiftRows
- Inverse SubByte

The decryption process is the encryption process done in reverse so i will explain the steps with notable differences.

Inverse MixColumns :

This step is similar to the MixColumns step in encryption, but differs in the matrix used to carry out the operation.

```
[ b0 ]         [ 14  11  13  9  ]  [ c0 ]
| b1 |  =      | 9   14  11  13 |     | c1 |
| b2 |      | 13  9   14  11 |     | c2 |
[ b3 ]         [ 11  13  9   14 ]     [ c3 ]
```

Inverse SubBytes :

Inverse S-box is used as a lookup table and using which the bytes are substituted during decryption.

Code Snapshots:

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <sstream>
#include "key_expand.h"
#include "encoding.h"
#include "decoding.h"
#include <typeinfo>
#include <unistd.h>
using namespace std;
int main()
{
    // we will read from file input.txt
    int extendedlength = 0;
    int choice;
    string myText;
label:
    cout << "Welcome to 128 bits AES encryption" << endl;
    cout << endl;
    cout << "Enter you choice " << endl;
    cout << "1- Encoding" << endl;
    cout << "2- Decoding" << endl;
    cin >> choice;

    switch (choice)
    {
    case 1:
    {
        // encryption of text data
        ifstream File;
        string filepath = "encryption.aes";
        // clearing encryption.aes before editing
        File.open(filepath.c_str(), std::ifstream::out |
std::ifstream::trunc);
        if (!File.is_open() || File.fail())
        {
            File.close();
            printf("\nError : failed to erase file content !");
        }
        File.close();
        // reading plain text from input.txt
        fstream newfile;
        newfile.open("input.txt", ios::in); // open a file to perform read
operation using file object
        if (newfile.is_open())
        { // checking whether the file is open
            cout << "Reading plain text from input.txt .........\n";
```

```
            usleep(1000);
            string tp;
            cout << "Reading KEY from key.txt ......\n";
            usleep(1000);
            cout << "Now encrypting ....\n";
            usleep(1000);
            cout << "writing encrypted data in encryption.aes ..\n";
            usleep(1000);
            cout << endl;
            while (getline(newfile, tp))
            {
                // read data from file object and put it into string.
                int messlength = tp.length();
                int extendedlength;
                if ((messlength % 16) != 0)
                {
                    extendedlength = messlength + (16 - (messlength % 16));
                }
                else
                {
                    extendedlength = messlength;
                }
                unsigned char *encryptedtext = new unsigned
char[extendedlength];
                for (int i = 0; i < extendedlength; i++)
                {
                    if (i < messlength)
                        encryptedtext[i] = tp[i];
                    else
                        encryptedtext[i] = 0;
                }
                // getting key from key.txt
                string k;
                ifstream infile;
                infile.open("key.txt");
                if (infile.is_open())
                {
                    getline(infile, k); // The first line of file should be
the key
                    infile.close();
                }

                else
                    cout << "Unable to open file";

                istringstream tempkey(k);
                unsigned char key[16];
                unsigned int x;
```

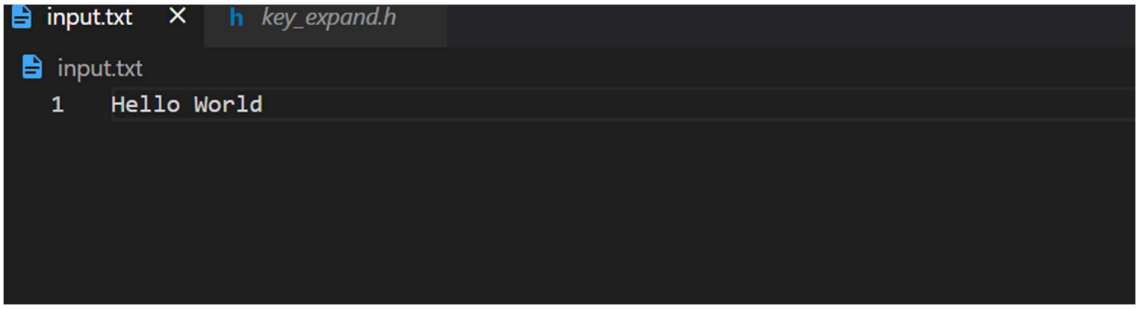
```
                for (int i = 0; i < 16; i++)
                {
                    tempkey >> hex >> x;
                    key[i] = x;
                }
                // extending key
                unsigned char extendedkeys[176];
                Key_extenxion(key, extendedkeys);

                // encrypting our plain text
                for (int i = 0; i < extendedlength; i += 16)
                {
                    unsigned char *temp = new unsigned char[16];
                    for (int j = 0; j < 16; j++)
                    {
                        temp[j] = encryptedtext[i + j];
                    }
                    encryption(temp, extendedkeys);
                    for (int j = 0; j < 16; j++)
                    {
                        encryptedtext[i + j] = temp[j];
                    }
                }
                // storing our encrypted data in encryption.aes
                ofstream fout; // Create Object of Ofstream
                ifstream fin;
                fin.open("encryption.aes");
                fout.open("encryption.aes", ios::app); // Append mode
                if (fin.is_open())
                    fout << encryptedtext << "\n"; // Writing data to file
                fin.close();
                fout.close();
            }
            cout << "128-bit AES encryption is done sucessfully\n";
            cout << "Data has been appended to file encryption.aes";
            newfile.close(); // close the file object.
        }
        break;
    }

    case 2:
    {
        cout << "Reading encrypted data from encryption.txt .........\n";
        usleep(1000);
        string tp;
        cout << "Reading KEY from key.txt ......\n";
        usleep(1000);
        cout << "Now Decrypting ....\n";
```

```
        usleep(1000);
        cout << "writing decrypted data in outputtext.txt ..\n";
        usleep(1000);
        cout << endl;
        cout << "Following is our decrypted text:- \n";
        // clearing outputtext file
        ifstream File;
        string filepath = "outputtext.txt";
        File.open(filepath.c_str(), std::ifstream::out |
std::ifstream::trunc);
        if (!File.is_open() || File.fail())
        {
            File.close();
            printf("\nError : failed to erase file content !");
        }
        File.close();

        ifstream MyReadFile;
        MyReadFile.open("encryption.aes", ios::in | ios::binary);
        if (MyReadFile.is_open())
        {
            while (getline(MyReadFile, myText))
            {
                cout.flush();
                char *x;
                x = &myText[0];
                int messlength = strlen(x);
                char *msg = new char[myText.size() + 1];

                strcpy(msg, myText.c_str());

                int n = strlen((const char *)msg);
                unsigned char *decryptedtext = new unsigned char[n];
                // decrypting our encrypted data
                for (int i = 0; i < n; i++)
                {
                    decryptedtext[i] = (unsigned char)msg[i];
                }
                // reading key from key.txt file
                string k;
                ifstream infile;
                infile.open("key.txt");
                if (infile.is_open())
                {
                    getline(infile, k); // The first line of file should be
the key
                    infile.close();
                }
```

```
            else
                cout << "Unable to open file";
            istringstream tempkey(k);
            unsigned char key[16];
            unsigned int x1;
            for (int i = 0; i < 16; i++)
            {
                tempkey >> hex >> x1;
                key[i] = x1;
            }
            // extending key
            unsigned char extendedkeys[176];
            Key_extenxion(key, extendedkeys);
            // decrypting our data
            for (int i = 0; i < messlength; i += 16)
            {
                unsigned char *temp = new unsigned char[16];
                for (int j = 0; j < 16; j++)
                    temp[j] = decryptedtext[i + j];
                decryption(temp, extendedkeys);
                for (int j = 0; j < 16; j++)
                    decryptedtext[i + j] = temp[j];
            }
            // printing our plain text
            for (int i = 0; i < messlength; i++)
            {
                cout << decryptedtext[i];
                if (decryptedtext[i] == 0 && decryptedtext[i - 1] == 0)
                    break;
            }
            // storing plain text in outputtext.txt file
            cout << endl;
            ofstream fout; // Create Object of Ofstream
            ifstream fin;
            fin.open("outputtext.txt");
            fout.open("outputtext.txt", ios::app); // Append mode
            if (fin.is_open())
                fout << decryptedtext << "\n"; // Writing data to file

            fin.close();
            fout.close(); // Closing the file
            usleep(500);
        }
    }
    else
    {
        cout << "Can not open input file\n ";
```

```
        }
        cout << "\n Data has been appended to file outputtext.txt";
        MyReadFile.close();
        break;
    }
    }
}
```

Output Snapshots:




```
                                ^~~~~~
Welcome to 128 bits AES encryption

Enter you choice
1- Encoding
2- Decoding
1
Reading plain text from input.txt .........
Reading KEY from key.txt ......
Now encrypting ....
writing encrypted data in encryption.aes ..

128-bit AES encryption is done sucessfully
Data has been appended to file encryption.aes
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Practical 7>
```

```
                                ^~~~~~
Welcome to 128 bits AES encryption

Enter you choice
1- Encoding
2- Decoding
2
Reading encrypted data from encryption.txt .........
Reading KEY from key.txt ......
Now Decrypting ....
writing decrypted data in outputtext.txt ..

Following is our decrypted text:-
Hello World
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Practical 7>
```

Conclusion:

1. Advanced encryption standard (AES) algorithm is one of the efficient algorithm and it is widely supported and adopted on hardware and software.
2. This algorithm enables to deal with different key sizes such as 128, 192, and 256 bits with 128 bits block cipher.
3. AES has the ability to provide much more security compared to other algorithms like DES, 3DES etc.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 8

Title: Implementation of Eucledian Algorithm and Extended Eucledian Algorithm.

Theory:

1. **Euclidian Algorithm:**

Euclid's algorithm, is an efficient method for computing the greatest common divisor (GCD) of two integers (numbers), the largest number that divides them both without a remainder. It is named after the ancient Greek mathematician Euclid, who first described it in his Elements (c. 300 BC). It is an example of an algorithm, a step-by-step procedure for performing a calculation according to well-defined rules, and is one of the oldest algorithms in common use. It can be used to reduce fractions to their simplest form, and is a part of many other number-theoretic and cryptographic calculations.

Input: $a, b \in R$.

Output: $g \in R$ a gcd of a and b.

$r_0 := a$
$r_1 := b$
$i := 2$
while $r_{i-1} \neq 0$ **repeat**
  $r_i := r_{i-2}$ rem r_{i-1}
  $i := i + 1$
return r_{i-2}

2. **Extended Eucledian Algorithm:**

The Extended Euclidean algorithm is arguably one of the oldest and most widely known algorithms. It is a method of computing the greatest common divisor (GCD) of two integers aa and bb. It allows computers to do a variety of simple number-theoretic tasks, and also serves as a foundation for more complicated algorithms in number theory.

Algorithm 3.4 Extended Euclidean algorithm

Input: Integers a and b

Output: Integers x, y, and d, where $d = \gcd(a, b) = ax + by$

Set $d_0 = a$ **Set** $x_0 = 1$ **Set** $y_0 = 0$
Set $d_1 = b$ **Set** $x_1 = 0$ **Set** $y_1 = 1$
While $d_1 \neq 0$ **Do**
  Set $q = \lfloor d_0/d_1 \rfloor$
  Set $d_2 = d_1$ **Set** $x_2 = x_1$ **Set** $y_2 = y_1$
  Set $d_1 = d_0 - qd_1$
  Set $x_1 = x_0 - qx_1$
  Set $y_1 = y_0 - qy_1$
  Set $d_0 = d_2$ **Set** $x_0 = x_2$ **Set** $y_0 = y_2$
End While

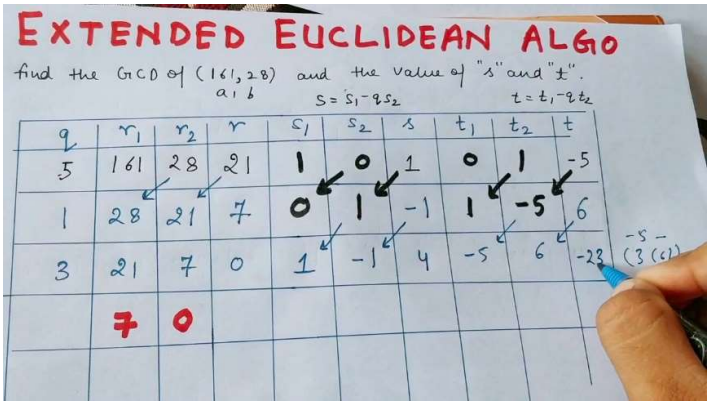
Return $[d, x, y] = [d_0, x_0, y_0]$

Example:

1. **Eucledian:**

Step	a	b	|a - b|
1	527	221	306
2	306	221	85
3	85	221	136
4	85	136	51
5	85	51	34
6	34	51	17
7	34	17	17
8	17	17	0

2. **Extended Eucledian:**



Code Snapshots:

1. **Eucledian:**

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

int findGcd(int r1, int r2)
{
    if (r2 == 0)
    {
        return r1;
    }

    int q = r1 / r2;
    int r = r1 % r2;

    cout<<"q "<<"r1 "<<"r2 "<<"r "<<endl;
    cout << q << " " << r1 << " " << r2 << " " << r << " "<< endl;

    return findGcd(r2, r);
}

int main()
{

    int num1, num2;
    cout << "Enter 2 numbers to find GCD" << endl;
    cin >> num1 >> num2;

    int gcd = findGcd(num1, num2);
    cout << "GCD is " << gcd << endl;


    return 0;
}
```

Output Snapshots:

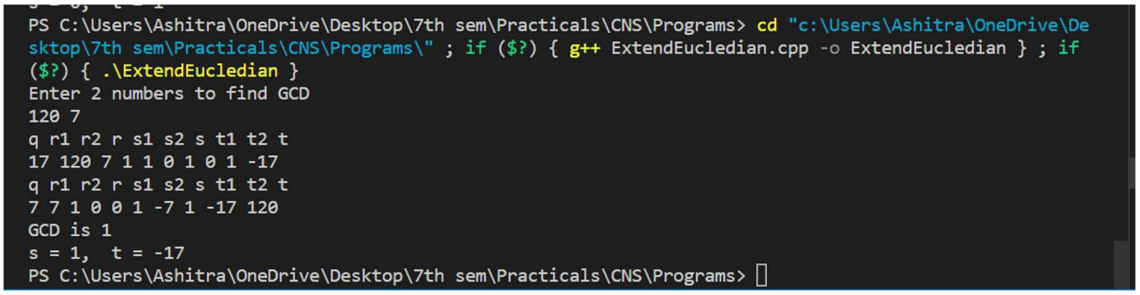
```
GCD is 4
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ Eucledian.cpp -o Eucledian } ; if ($?) { .\Eucledian }
Enter 2 numbers to find GCD
120 7
q r1 r2 r
17 120 7 1
q r1 r2 r
7 7 1 0
GCD is 1
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Code Snapshots:

2. **Extended Eucledian:**

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

int ansS, ansT;

int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)
{
    // Base Case
    if (r2 == 0)
    {
        ansS = s1;
        ansT = t1;
        return r1;
    }

    int q = r1 / r2;
    int r = r1 % r2;

    int s = s1 - q * s2;
    int t = t1 - q * t2;

    cout<<"q "<<"r1 "<<r2 "<<"r "<<"s1 "<<"s2 "<<"s "<<"t1 "<<"t2 "<<"t
"<<endl;
    cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2
<< " " << s << " " << t1 << " " << t2 << " " << t << endl;

    return findGcdExtended(r2, r, s2, s, t2, t);
}

int main()
{

    int num1, num2, s, t;
    cout << "Enter 2 numbers to find GCD" << endl;
    cin >> num1 >> num2;

    int gcd = findGcdExtended(num1, num2, 1, 0, 0, 1);
    cout << "GCD is " << gcd << endl;
    cout << "s = "<<ansS << ",  " << "t = "<< ansT << endl;

    return 0;
}
```

Output Snapshots:



Conclusion:

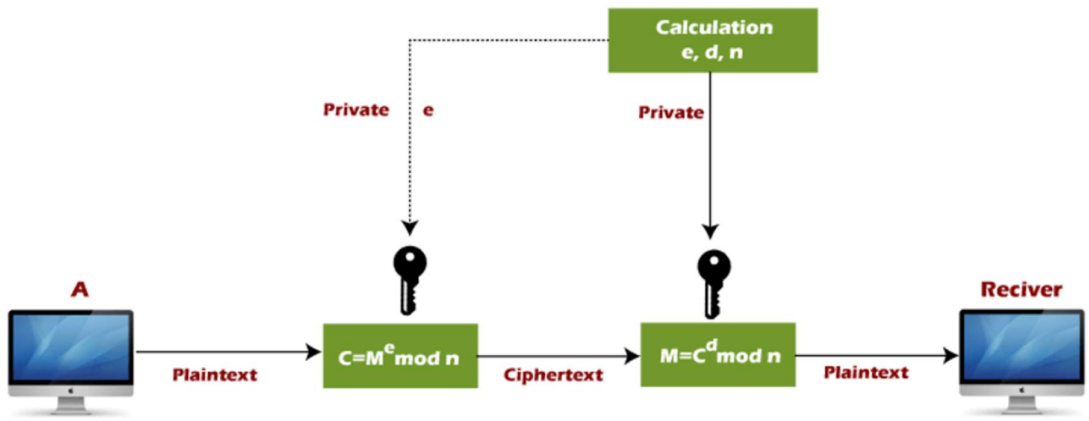
1. The Euclidean algorithm helps us in finding the greatest common divisor of two non-negative integers.
2. The Extended Euclidean algorithm builds on top of the basic Euclidean algorithm. It can solve linear diophantine equations of the form: ax + by = c, where c is divisible by the greatest common divisor of a and b.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 11

Title: Implementation of RSA Algorithm.

Theory:

RSA algorithm is a public key encryption technique and is considered as the most secure way of encryption. It was invented by Rivest, Shamir and Adleman in year 1978 and hence name RSA algorithm.



RSA

RSA algorithm uses the following procedure to generate public and private keys:

- Select two large prime numbers, p and **q**.
- Multiply these numbers to find **n = p x q,** where **n** is called the modulus for encryption and decryption.
- Choose a number **e** less than **n**, such that n is relatively prime to **(p - 1) x (q -1).** It means that **e** and **(p - 1) x (q - 1)** have no common factor except 1. Choose "e" such that 1<e < φ (n), e is prime to φ (n),
  gcd (e,d(n)) =1
- If **n = p x q,** then the public key is <e, n>. A plaintext message **m** is encrypted using public key <e, n>. To find ciphertext from the plain text following formula is used to get ciphertext C.
  $C = m^e \bmod n$
  Here**, m** must be less than **n**. A larger message (>n) is treated as a concatenation of messages, each of which is encrypted separately.
- To determine the private key, we use the following formula to calculate the d such that:
  $D_e \bmod \{(p - 1) \times (q - 1)\} = 1$
  Or
  $D_e \bmod \varphi(n) = 1$
- The private key is <d, n>. A ciphertext message **c** is decrypted using private key <d, n>. To calculate plain text **m** from the ciphertext c following formula is used to get plain text m.
  $m = c^d \bmod n$

Example:

RSA Example

1. Select primes: $p=17$ & $q=11$
2. Compute $n = pq = 17 \times 11 = 187$
3. Compute $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$
4. Select e : $\gcd(e,160)=1$; choose $e=7$
5. Determine d: $de=1 \bmod 160$ and $d < 160$
   Value is $d=23$ since $23 \times 7 = 161 = 10 \times 160 + 1$
6. Publish public key $KU=\{7,187\}$
7. Keep secret private key $KR=\{23,17,11\}$

CCLAB

연세대학교 YONSEI UNIVERSITY

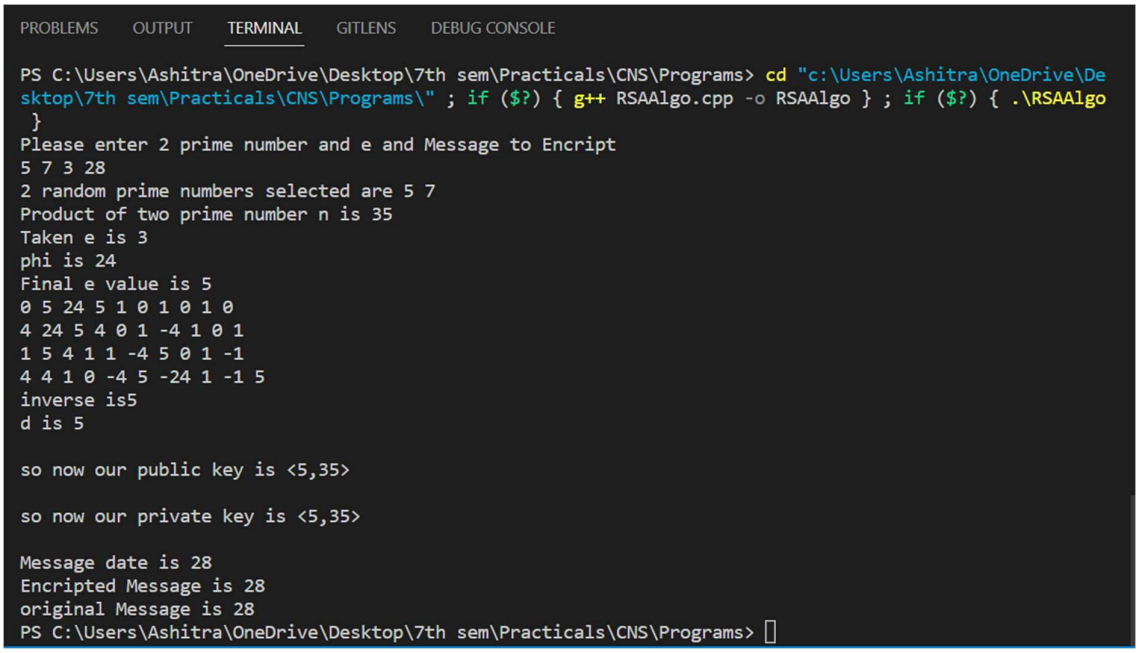
Code Snapshots:

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

// void file()
// {
// #ifndef ONLINE_JUDGE
//     freopen("input.txt", "r", stdin);
//     freopen("output.txt", "w", stdout);
// #endif
// }

// Function for extended Euclidean Algorithm
int ansS, ansT;
int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)
{
    // Base Case
    if (r2 == 0)
    {
        ansS = s1;
        ansT = t1;
        return r1;
    }

    int q = r1 / r2;
    int r = r1 % r2;

    int s = s1 - q * s2;
    int t = t1 - q * t2;

    cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2
<< " " << s << " " << t1 << " " << t2 << " " << t << endl;

    return findGcdExtended(r2, r, s2, s, t2, t);
}


int modInverse(int A, int M)
{
    int x, y;
    int g = findGcdExtended(A, M, 1, 0, 0, 1);
    if (g != 1) {
        cout << "Inverse doesn't exist";
        return 0;
    }
    else {

        // m is added to handle negative x
```

```
        int res = (ansS % M + M) % M;
        cout << "inverse is" << res << endl;
        return res;
    }
}

long long powM(long long a, long long b, long long n)
{
    if (b == 1)
        return a % n;
    long long x = powM(a, b / 2, n);
    x = (x * x) % n;
    if (b % 2)
        x = (x * a) % n;
    return x;
}

int findGCD(int num1, int num2)
{
    if (num1 == 0)
        return num2;
    return findGCD(num2 % num1, num1);
}

// Code to demonstrate RSA algorithm
int main()
{
    //file();

    // Two random prime numbers
    long long p, q, e, msg;
    //17 31 7 2

    cout << "Please enter 2 prime number and e and Message to Encript" <<
endl;
    cin >> p >> q >> e >> msg;

    cout << "2 random prime numbers selected are " << p << " " << q << endl;

    // First part of public key:
    long long n = p * q;
    cout << "Product of two prime number n is " << n << endl;

    // Finding other part of public key.
    // e stands for encrypt

    cout << "Taken e is " << e << endl;
```

```
    long long phi = (p - 1) * (q - 1);
    cout << "phi is " << phi << endl;

    while (e < phi) {
        // e must be co-prime to phi and
        // smaller than phi.
        if (findGCD(e, phi) == 1)
            break;
        else
            e++;
    }

    cout << "Final e value is " << e << endl;


    // Private key (d stands for decrypt)

    long long d = modInverse(e, phi);
    cout << "d is " << d << endl;

    cout << "\nso now our public key is " << "<" << e << "," << n << ">" <<
endl;
    cout << "\nso now our private key is " << "<" << d << "," << n << ">" <<
endl << endl;



    // Message to be encrypted

    cout << "Message date is " << msg << endl;

    // Encryption c = (msg ^ e) % n
    long long c = powM(msg, e, n);
    cout << "Encripted Message is " << c << endl;

    // Decryption m = (c ^ d) % n
    long long m = powM(c, d, n);
    cout << "original Message is " << m << endl;

    return 0;
}
```

Output Snapshots:



Conclusion:

1. It is concluded that, while establishing RSA key pairs, usage keys and general-purpose keys are integrated.

2. In usage RSA keys, two key pairs are used for encryption and signatures.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 10

Title: Implementation of RSA Factorization challenge.

Theory:
The RSA Factoring Challenge was a challenge put forward by RSA Laboratories on March 18, 1991 to encourage research into computational number theory and the practical difficulty of factoring large integers and cracking RSA keys used in cryptography. They published a list of semiprimes (numbers with exactly two prime factors) known as the RSA numbers, with a cash prize for the successful factorization of some of them. The smallest of them, a 100-decimal digit number called RSA-100 was factored by April 1, 1991. Many of the bigger numbers have still not been factored and are expected to remain unfactored for quite some time, however advances in quantum computers make this prediction uncertain due to Shor's algorithm.

The RSA Factoring Challenges ended in 2007.[5] RSA Laboratories stated: "Now that the industry has a considerably more advanced understanding of the cryptanalytic strength of common symmetric-key and public-key algorithms, these challenges are no longer active."[6] When the challenge ended in 2007, only RSA-576 and RSA-640 had been factored from the 2001 challenge numbers.

The factoring challenge was intended to track the cutting edge in integer factorization. A primary application is for choosing the key length of the RSA public-key encryption scheme. Progress in this challenge should give an insight into which key sizes are still safe and for how long. As RSA Laboratories is a provider of RSA-based products, the challenge was used by them as an incentive for the academic community to attack the core of their solutions — in order to prove its strength.

The RSA numbers were generated on a computer with no network connection of any kind. The computer's hard drive was subsequently destroyed so that no record would exist, anywhere, of the solution to the factoring challenge.

Example:

The mathematics [edit]

RSA Laboratories states that: for each RSA number *n*, there exists prime numbers *p* and *q* such that

$n = p \times q$.

The problem is to find these two primes, given only *n*.

The prizes and records [edit]

The following table gives an overview over all RSA numbers. Note that the RSA Factoring Challenge ended in 2007[5] and no further prizes will be awarded for factoring the higher numbers.

The challenge numbers in white lines are part of the original challenge and are expressed in base 10, while the challenge numbers in yellow lines are part of the 2001 expansion and are expressed in base 2

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on	Factored by
RSA100	100	330	US$1,000[6]	April 1, 1991[9]	Arjen K. Lenstra
RSA110	110	364	US$4,429[6]	April 14, 1992[9]	Arjen K. Lenstra and M.S. Manasse
RSA120	120	397	US$5,898[6]	July 9, 1993[10]	T. Denny *et al.*
RSA129 [a]	129	426	US$100	April 26, 1994[9]	Arjen K. Lenstra *et al.*
RSA130	130	430	US$14,527[6]	April 10, 1996	Arjen K. Lenstra *et al.*
RSA140	140	463	US$17,226	February 2, 1999	Herman te Riele *et al.*
RSA150	150	496		April 16, 2004	Kazumaro Aoki *et al.*
RSA155	155	512	US$9,383[6]	August 22, 1999	Herman te Riele *et al.*
RSA160	160	530		April 1, 2003	Jens Franke *et al.*, University of Bonn
RSA170 [b]	170	563		December 29, 2009	D. Bonenberger and M. Krone [c]
RSA576	174	576	US$10,000	December 3, 2003	Jens Franke *et al.*, University of Bonn
RSA180 [b]	180	596		May 8, 2010	S. A. Danilov and I. A. Popovyan, Moscow State University[11]
RSA190 [b]	190	629		November 8, 2010	A. Timofeev and I. A. Popovyan
RSA640	193	640	US$20,000	November 2, 2005	Jens Franke *et al.*, University of Bonn

Code Snapshots:

```cpp
#include <bits/stdc++.h>
#define ll long long
#define ul unsigned long long
#define pb emplace_back
#define po pop_back
#define vi vector<ll>
#define vii vector<vector<ll>>
using namespace std;
void file(){
     ios_base::sync_with_stdio(false);
     cin.tie(NULL);}
ll M = 1e9 + 7;
int rem;
string longDivision(string number, int divisor)
{
    string ans;

    int idx = 0;
    int temp = number[idx] - '0';
    while (temp < divisor && number.length()>1)
        temp = temp * 10 + (number[++idx] - '0');

    while (number.size() > idx) {
        rem = temp % divisor;
        ans += (temp / divisor) + '0';
        temp = (temp % divisor) * 10 + number[++idx] - '0';
```

```
    }

    if (ans.length() == 0)
        return "0";
    if(rem==0)
    return ans;
    else return number;
}
int main(){
    string num;
    cout<<"Prime Factors:\n";
    cout<<"Enter Number : ";
    cin>>num;
    rem=0;

    unordered_map<int,int> mp;
    int len = num.size();

    string ans = longDivision(num,2);
    while(rem == 0){
        mp[2]++;
        num =  ans;
        ans = longDivision(num,2);
    }

      for (int i = 3; i <= 1000000; i = i + 2)
    {   string ans = longDivision(num,i);
        while (ans!="0" && rem==0)
        {   mp[i]++;
            num =  ans;
            ans = longDivision(num,i);
        }
    }

    cout<<"\n";
    cout<<"Prime Factor"<<" - "<<"Power"<<"\n";
    for(auto x:mp) cout<<x.first<<" - "<<x.second<<"\n";
}
```

Output Snapshots:

```
PROBLEMS  OUTPUT  TERMINAL  GITLENS  DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\De
sktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile }
 ; if ($?) { .\tempCodeRunnerFile }
Prime Factors:
Enter Number : 295

Prime Factor - Power
59 - 1
5 - 1
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs>
```

Conclusion:

1. RSA is breakable byfactoring the “N”, the security of RSA is often basedon the integer factorization problem.
2. To resolve the prime factors of RSA, we can use integer factorization algorithm.
3. The RSA breach can be resolved easily by doing factorization on the public key.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 11

Title: Implementation of Diffie-Hellman Key Exchange.

Theory:

The Diffie-Hellman key exchange was one of the most important developments in public-key cryptography and it is still frequently implemented in a range of today's different security protocols.

It allows two parties who have not previously met to securely establish a key which they can use to secure their communications. In this article, we'll explain what it's used for, how it works on a step-by-step basis, its different variations, as well as the security considerations that need to be noted in order to implement it safely.

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

- For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime P and G (a primitive root of P) and two private values a and b.
- P and G are both publicly available numbers. Users (say Alice and Bob) pick private values a and b and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

Step by Step Explanation

Alice	Bob
Public Keys available = P, G	Public Keys available = P, G
Private Key Selected = a	Private Key Selected = b
Key generated = $x = G^a mod P$	Key generated = $y = G^b mod P$
Exchange of generated keys takes place	
Key received = y	key received = x
Generated Secret Key = $k_a = y^a mod P$	Generated Secret Key = $k_b = x^b mod P$
Algebraically, it can be shown that $k_a = k_b$	
Users now have a symmetric secret key to encrypt	

Example:

```
Step 1: Alice and Bob get public numbers P = 23, G = 9

Step 2: Alice selected a private key a = 4 and
        Bob selected a private key b = 3

Step 3: Alice and Bob compute public values
Alice:    x =(9^4 mod 23) = (6561 mod 23) = 6
        Bob:    y = (9^3 mod 23) = (729 mod 23)  = 16

Step 4: Alice and Bob exchange public numbers

Step 5: Alice receives public key y =16 and
        Bob receives public key x = 6

Step 6: Alice and Bob compute symmetric keys
        Alice:  ka = y^a mod p = 65536 mod 23 = 9
        Bob:    kb = x^b mod p = 216 mod 23 = 9

Step 7: 9 is the shared secret.
```

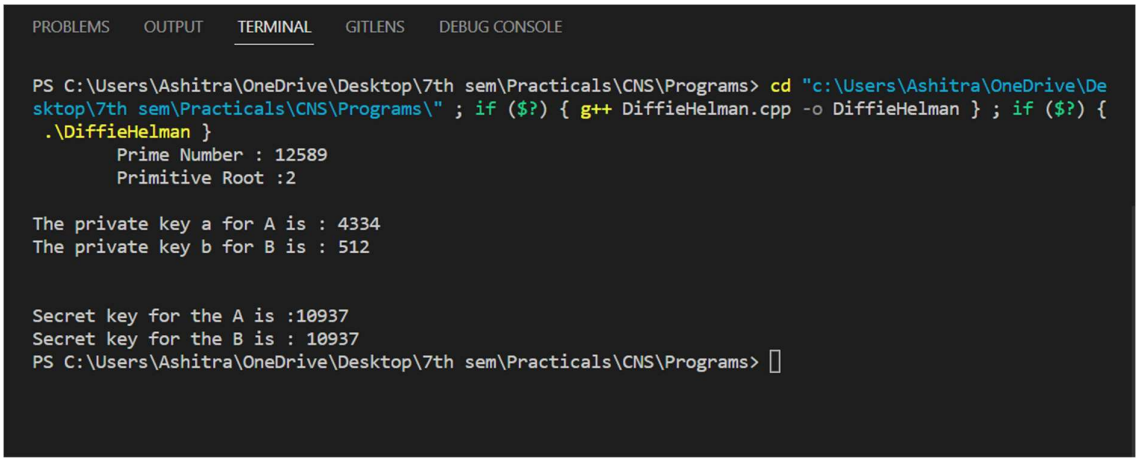
Code Snapshots:

```cpp
#include <bits/stdc++.h>
#define ll long long
#define ul unsigned long long
#define pb emplace_back
#define po pop_back
#define vi vector<ll>
#define vii vector<vector<ll>>
using namespace std;

vector<int> primeNums;
vector<bool> prime(100000001,1);

void SeiveOfEratosthenes(int n){
    for(int p=2; p*p<=n; p++){
        if(prime[p] ==  true){
            for (int i = p * p; i <= n; i += p)
            prime[i] = false;
        }
    }
```

```
    for(int i=3;i<n;i+=2){
        if(prime[i]) primeNums.push_back(i);
    }
}


ll power(ll a, ll b, ll p){
    if (b == 1)
        return a;
    else
        return (((long long int)pow(a, b)) % p);
}


void findPrimefactors(unordered_set<int> &s, int n){
    while (n%2 == 0){
        s.insert(2);
        n = n/2;
    }
    for (int i = 3; i <= sqrt(n); i = i+2){
        while (n%i == 0){
            s.insert(i);
            n = n/i;
        }
    }
    if (n > 2)
        s.insert(n);
}


int primitiveRoot(int n){
    unordered_set<int> s;
    int phi = n-1;
    findPrimefactors(s, phi);
    for (int r=2; r<=phi; r++){
        bool flag = false;
        for (auto it = s.begin(); it != s.end(); it++){
            if (power((ll)r, (ll)phi/(*it),(ll)n) == 1)
            {
                flag = true;
                break;
            }
         }
         if (flag == false)
           return r;
    }
    return -1;
}
```

```cpp
int main(){
    // prime number till 100000000
    SeiveOfEratosthenes(100000000);
    int privateNumberA, privateNumberB;
    cout<<"Enter the privateNumber of A and B respectively : ";
    cin>>privateNumberA>>privateNumberB;
    cout<<"\nFinding prime Number and a primitive root ...\n";

    srand(time(0));
    int  p = primeNums[rand() % primeNums.size()];
    int g = primitiveRoot(p);

    cout<<"\tPrime Number : "<<p<<"\n";
    cout<<"\tPrimitive Root :"<<g<<"\n";

    // calculating the private key for  a
    ll x = power(g,privateNumberA,p);
    if(x<0) x = p + x;
    cout<<"\nThe private key a for A is : "<<x<<"\n";

    // calculate private key for b
    ll y = power(g,privateNumberB,p);
    if(y<0) y = p + y;
    cout<<"The private key b for B is : "<<y<<"\n";

    ll ka = power(y, privateNumberA, p); // Secret key for A
    if(ka<0) ka = p + ka;
    ll  kb = power(x, privateNumberB, p); // Secret key for B
    if(kb<0) kb = p + kb;

    cout<<"\n\nSecret key for the A is :"<<ka;
    cout<<"\nSecret key for the B is : "<<kb<<endl;
    return 0;
}
```

Output Snapshots:



Conclusion:

1. The Diffie Hellman key Exchange has proved to be a useful key exchange system due to its advantages.
2. While it is really tough for someone snooping the network to decrypt the data and get the keys, it is still possible if the numbers generated are not entirely random.

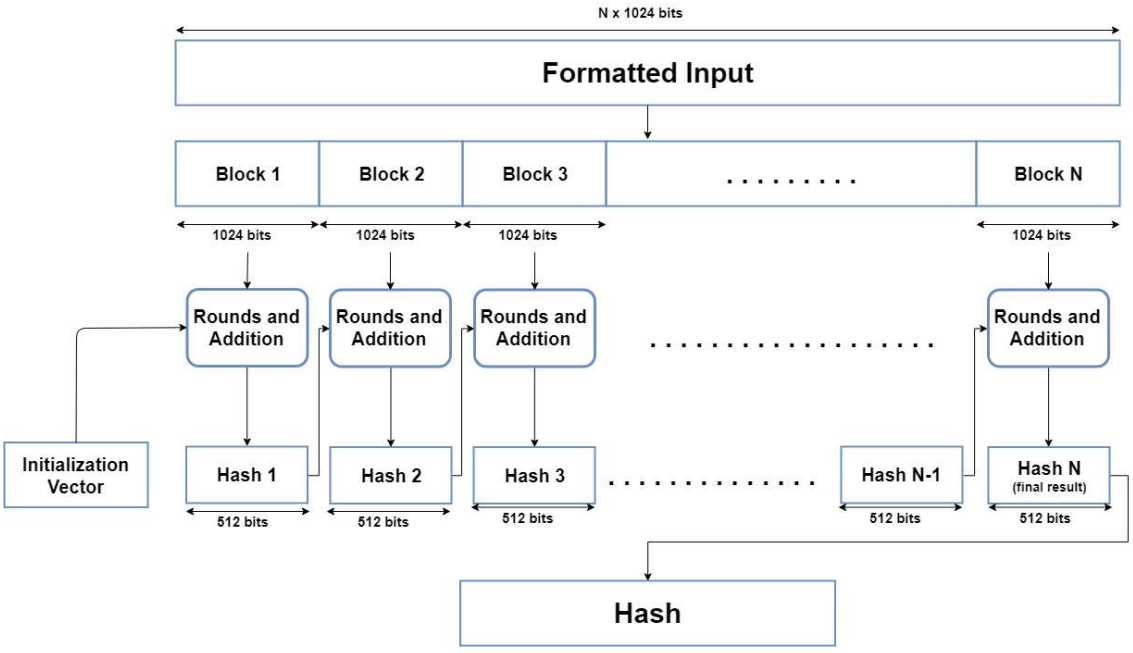
Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 12

Title: Implementation of Cryptographic Hash Function- SHA 512.

Theory:
SHA-512, or Secure Hash Algorithm 512, is a hashing algorithm used to convert text of any length into a fixed-size string. Each output produces a SHA-512 length of 512 bits (64 bytes).

This algorithm is commonly used for email addresses hashing, password hashing, and digital record verification. SHA-512 is also used in blockchain technology, with the most notable example being the BitShares network.

SHA-512 is used in real-world applications, let's look at how it compares to SHA 256 vs 512 by assessing three critical factors: security, computational efficiency, and compatibility.



Example:

Input *: hello world*
Output *:*
309ecc489c12d6eb4cc40f50c902f2b4d0ed77ee511a7c7a9bcd3ca86d4cd86f989dd35bc5ff4
99670da34255b45b0cfd830e81f605dcf7dc5542e93ae9cd76f

Input *: GeeksForGeeks*
Output *:*
acc10c4e0b38617f59e88e49215e2e894afaee5ec948c2af6f44039f03c9fe47a9210e01d5cd9
26c142bdc9179c2ad30f927a8faf69421ff60a5eaddcf8cb9c

Code Snapshots:

```
#include<bits/stdc++.h>

#define ull unsigned long long

#define SHA_512_INPUT_REPRESENTATION_LENGTH 128
#define BLOCK_SIZE 1024

#define BUFFER_COUNT 8
#define WORD_LENGTH 64
#define ROUND_COUNT 80

using namespace std;

// void file()
// {
// #ifndef ONLINE_JUDGE
//  freopen("input.txt", "r", stdin);
//  freopen("output.txt", "w", stdout);
// #endif
// }

void initialiseBuffersAndConstants(vector<ull>& buffers, vector<ull>&
constants)
{
    buffers = {
        0x6a09e667f3bcc908, 0xbb67ae8584caa73b, 0x3c6ef372fe94f82b,
0xa54ff53a5f1d36f1,
        0x510e527fade682d1, 0x9b05688c2b3e6c1f, 0x1f83d9abfb41bd6b,
0x5be0cd19137e2179
    };
```

```
    constants = {
        0x428a2f98d728ae22, 0x7137449123ef65cd, 0xb5c0fbcfec4d3b2f,
0xe9b5dba58189dbbc, 0x3956c25bf348b538,
        0x59f111f1b605d019, 0x923f82a4af194f9b, 0xab1c5ed5da6d8118,
0xd807aa98a3030242, 0x12835b0145706fbe,
        0x243185be4ee4b28c, 0x550c7dc3d5ffb4e2, 0x72be5d74f27b896f,
0x80deb1fe3b1696b1, 0x9bdc06a725c71235,
        0xc19bf174cf692694, 0xe49b69c19ef14ad2, 0xefbe4786384f25e3,
0x0fc19dc68b8cd5b5, 0x240ca1cc77ac9c65,
        0x2de92c6f592b0275, 0x4a7484aa6ea6e483, 0x5cb0a9dcbd41fbd4,
0x76f988da831153b5, 0x983e5152ee66dfab,
        0xa831c66d2db43210, 0xb00327c898fb213f, 0xbf597fc7beef0ee4,
0xc6e00bf33da88fc2, 0xd5a79147930aa725,
        0x06ca6351e003826f, 0x142929670a0e6e70, 0x27b70a8546d22ffc,
0x2e1b21385c26c926, 0x4d2c6dfc5ac42aed,
        0x53380d139d95b3df, 0x650a73548baf63de, 0x766a0abb3c77b2a8,
0x81c2c92e47edaee6, 0x92722c851482353b,
        0xa2bfe8a14cf10364, 0xa81a664bbc423001, 0xc24b8b70d0f89791,
0xc76c51a30654be30, 0xd192e819d6ef5218,
        0xd69906245565a910, 0xf40e35855771202a, 0x106aa07032bbd1b8,
0x19a4c116b8d2d0c8, 0x1e376c085141ab53,
        0x2748774cdf8eeb99, 0x34b0bcb5e19b48a8, 0x391c0cb3c5c95a63,
0x4ed8aa4ae3418acb, 0x5b9cca4f7763e373,
        0x682e6ff3d6b2b8a3, 0x748f82ee5defb2fc, 0x78a5636f43172f60,
0x84c87814a1f0ab72, 0x8cc702081a6439ec,
        0x90befffa23631e28, 0xa4506cebde82bde9, 0xbef9a3f7b2c67915,
0xc67178f2e372532b, 0xca273eceea26619c,
        0xd186b8c721c0c207, 0xeada7dd6cde0eb1e, 0xf57d4f7fee6ed178,
0x06f067aa72176fba, 0x0a637dc5a2c898a6,
        0x113f9804bef90dae, 0x1b710b35131c471b, 0x28db77f523047d84,
0x32caab7b40c72493, 0x3c9ebe0a15c9bebc,
        0x431d67c49c100d4c, 0x4cc5d4becb3e42b6, 0x597f299cfc657e2a,
0x5fcb6fab3ad6faec, 0x6c44198c4a475817
    };
}

string sha512Padding(string input)
{
    string finalPlainText = "";

    for (int i = 0 ; i < input.size() ; ++i)
    {
        finalPlainText += bitset<8>((int)input[i]).to_string();
    }

    finalPlainText += '1';

    int plainTextSize = input.size() * 8;
```

```
    int numberOfZeros = BLOCK_SIZE - ((plainTextSize +
SHA_512_INPUT_REPRESENTATION_LENGTH + 1) % BLOCK_SIZE);

    while (numberOfZeros--)
    {
        finalPlainText += '0';
    }

    finalPlainText +=
bitset<SHA_512_INPUT_REPRESENTATION_LENGTH>(plainTextSize).to_string();

    cout << "Plain text length = " << plainTextSize << endl;
    cout << "Plain text length after padding = " << finalPlainText.length() <<
endl << endl;

    return finalPlainText;
}

ull getUllFromString(string str)
{
    bitset<WORD_LENGTH> word(str);
    return word.to_ullong();
}

static inline ull rotr64(ull n, ull c)
{
    const unsigned int mask = (CHAR_BIT * sizeof(n) - 1);
    c &= mask;
    return (n >> c) | (n << ((-c)&mask ));
}

int main()
{
    //file();

    vector<ull> buffers(BUFFER_COUNT);
    vector<ull> constants(ROUND_COUNT);

    initialiseBuffersAndConstants(buffers, constants);

    cout << "Enter Text: ";
    string input;
    getline(cin, input);

    cout << "Input: " << input << endl;
    string paddedInput = sha512Padding(input);

    cout << "Padded Input:" << " " << paddedInput << endl << endl;
```

```
    for (int i = 0 ; i < paddedInput.size() ; i += BLOCK_SIZE)
    {
        string currentBlock = paddedInput.substr(i, BLOCK_SIZE);

        vector<ull> w(ROUND_COUNT);
        for (int j = 0 ; j < 16 ; ++j)
        {
            w[j] = getUllFromString(currentBlock.substr(j, WORD_LENGTH));
        }

        for (int j = 16 ; j < 80 ; ++j)
        {
            ull sigma1 = (rotr64(w[j - 15], 1)) ^ (rotr64(w[j - 15], 8)) ^
(w[j - 15] >> 7);
            ull sigma2 = (rotr64(w[j - 2], 19)) ^ (rotr64(w[j - 2], 61)) ^
(w[j - 2] >> 6);

            w[j] = w[j - 16] + sigma1 + w[j - 7] + sigma2;
        }

        ull a = buffers[0], b = buffers[1], c = buffers[2], d = buffers[3];
        ull e = buffers[4], f = buffers[5], g = buffers[6], h = buffers[7];

        for (int j = 0 ; j < ROUND_COUNT ; ++j)
        {

            ull sum0 = (rotr64(a, 28)) ^ (rotr64(a, 34)) ^ (rotr64(a, 39));
            ull sum1 = (rotr64(e, 14)) ^ (rotr64(e, 18)) ^ (rotr64(e, 41));

            ull ch = (e && f) ^ ((!e) && g);
            ull temp1 = h + sum1 + ch + constants[i] + w[i];

            ull majorityFunction = (a && b) ^ (a && c) ^ (b && c);
            ull temp2 = sum0 + majorityFunction;

            h = g;
            g = f;
            f = e;
            e = d + temp1;
            d = c;
            c = b;
            b = a;
            a = temp1 + temp2;
        }

        buffers[0] += a;
        buffers[1] += b;
```

```
        buffers[2] += c;
        buffers[3] += d;
        buffers[4] += e;
        buffers[5] += f;
        buffers[6] += g;
        buffers[7] += h;
    }
    cout << "Output of SHA-512 Algorithm: " << endl;
    for (int i = 0 ; i < BUFFER_COUNT ; ++i)
    {
        cout << setfill('0') << setw(16) << right << hex << buffers[i];
    }

    return 0;
}
```

Output Snapshots:

```
PROBLEMS    OUTPUT    TERMINAL    GITLENS    DEBUG CONSOLE

PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> cd "c:\Users\Ashitra\OneDrive\De
sktop\7th sem\Practicals\CNS\Programs\" ; if ($?) { g++ SHA512.cpp -o SHA512 } ; if ($?) { .\SHA512 }
Enter Text: Hello World
Input: Hello World
Plain text length = 88
Plain text length after padding = 1024

Padded Input: 0100100001100101011011000110110001101111001000000101011101101111011100100110110001100100
100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000001011000

Output of SHA-512 Algorithm:
65afd7bdf26905b6071db318319f35fe20fa3f66a1777faf9b754f3febb327bb71c1afad0aa218640e1c42bd491e03d235659f
1099dd7ff15b14c6579c8d37ce
PS C:\Users\Ashitra\OneDrive\Desktop\7th sem\Practicals\CNS\Programs> 
```

Conclusion:

1. The SHA-512 hashing algorithm is currently one of the best and secured hashing algorithms after hashes like MD5 and SHA-1 has been broken down.
2. Due to their complicated nature it is not well accepted and SHA-256 is a general standard, but the industry is slowing moving towards this hashing algorithm.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 13

Title: To generate a digital certificate.

Theory:
Asymmetric encryption algorithms are based on the sharing of a public key between different users. Generally, the sharing of this key is done through an electronic directory (generally in LDAP format) or a website.

However, this method of sharing has a major weakness: there is no guarantee that the key is really the same as the user to whom it is associated. Indeed, a hacker can corrupt the public key present in the directory by replacing it with his public key. Thus, the hacker will be able to decrypt all the messages that have been encrypted with the key present in the directory.

Thus a digital certificate makes it possible to associate a public key with an entity (a person, a machine, …) in order to ensure its validity. The digital certificate is the identity card of the public key, issued by an organization called a certification authority (often noted CA for Certification Authority).
The certification authority is in charge of issuing digital certificates, assigning them a validity date (equivalent to the expiration date), as well as possibly revoking certificates before this date in case of compromise of the key (or the owner).

❖ **Structure of a certificate** :
Digital certificates are small files divided into two parts:

- The part containing the information
- The part containing the signature of the certification authority

The structure of digital certificates is standardized by the ITU X.509 standard (more precisely X.509v3), which defines the information contained in the digital certificate:

- The version of X.509 to which the digital certificate corresponds;
- The serial number of the digital certificate;
- The encryption algorithm used to sign the digital certificate;

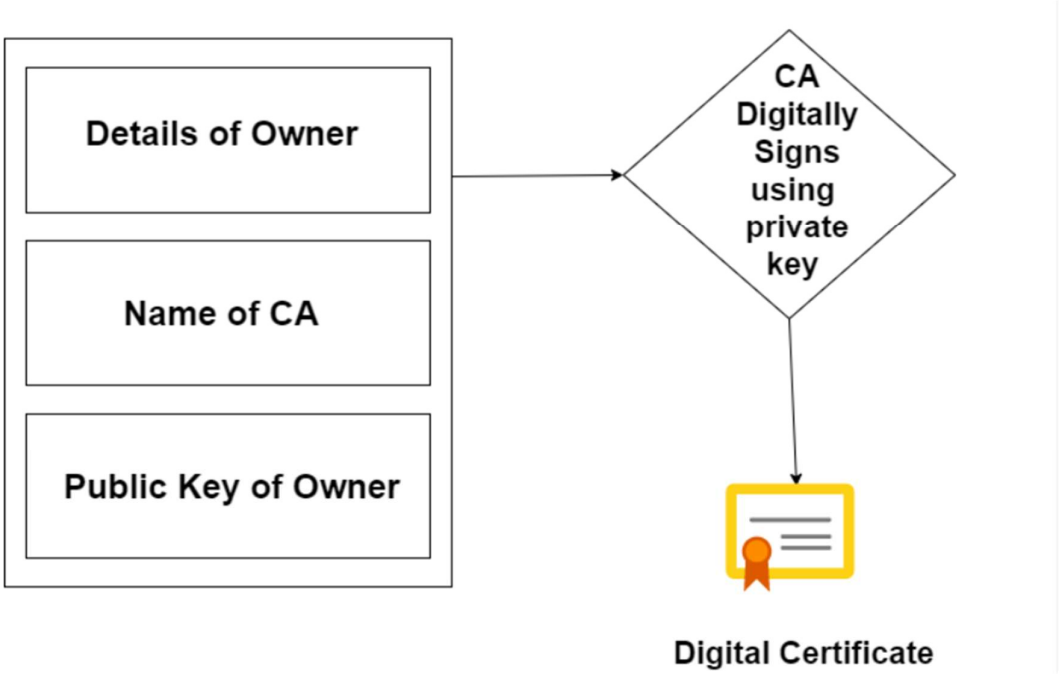
- The name (DN, for Distinguished Name) of the issuing Certification Authority(CA);
- The validity start date of the digital certificate;
- The digital certificate's expiry date;
- The purpose of the public key use;
- The public key of the digital certificate owner;
- The signature of the digital certificate issuer (thumbprint).

Types of digital certificates

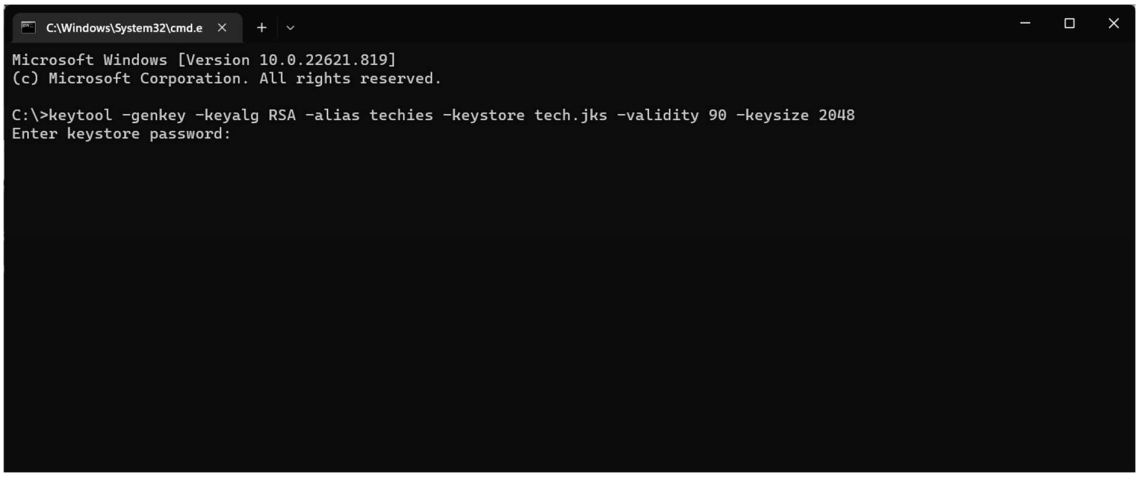
There are different types of digital certificates depending on the signature level:

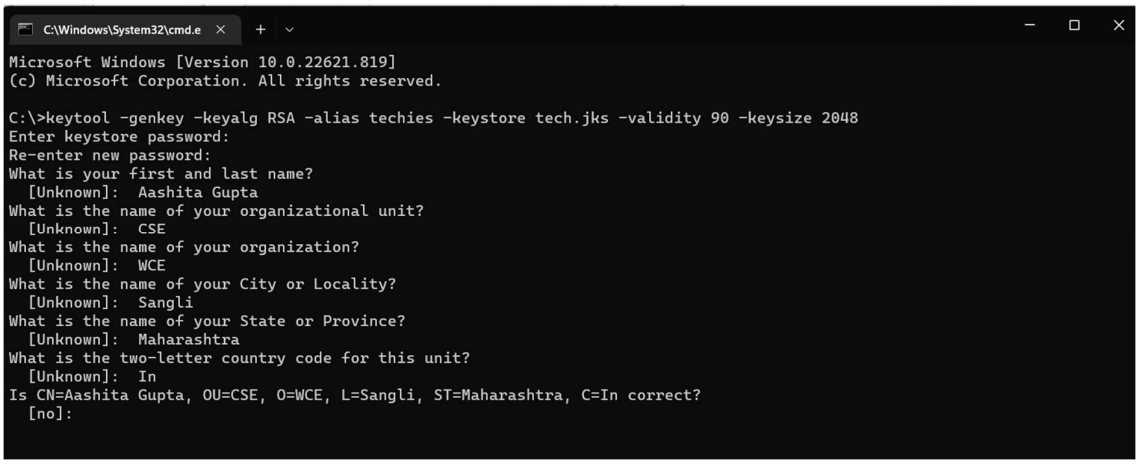
Self-signed certificates are certificates for internal use. Signed by a local server, this type of certificate guarantees the confidentiality of exchanges within an organization, for example for the needs of an intranet. It is thus possible to authenticate users using self-signed certificates.

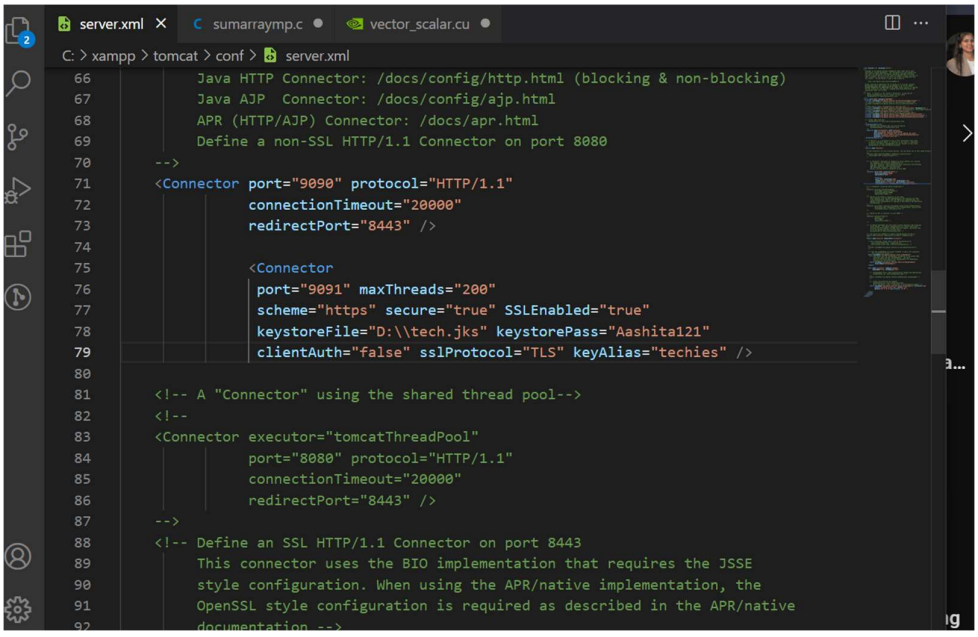
Certificates signed by a certification authority are necessary to ensure the security of exchanges with anonymous users, for example for a secure website accessible to the general public. The third-party certifier assures the user that the certificate belongs to the same organization to which it is claimed to belong.



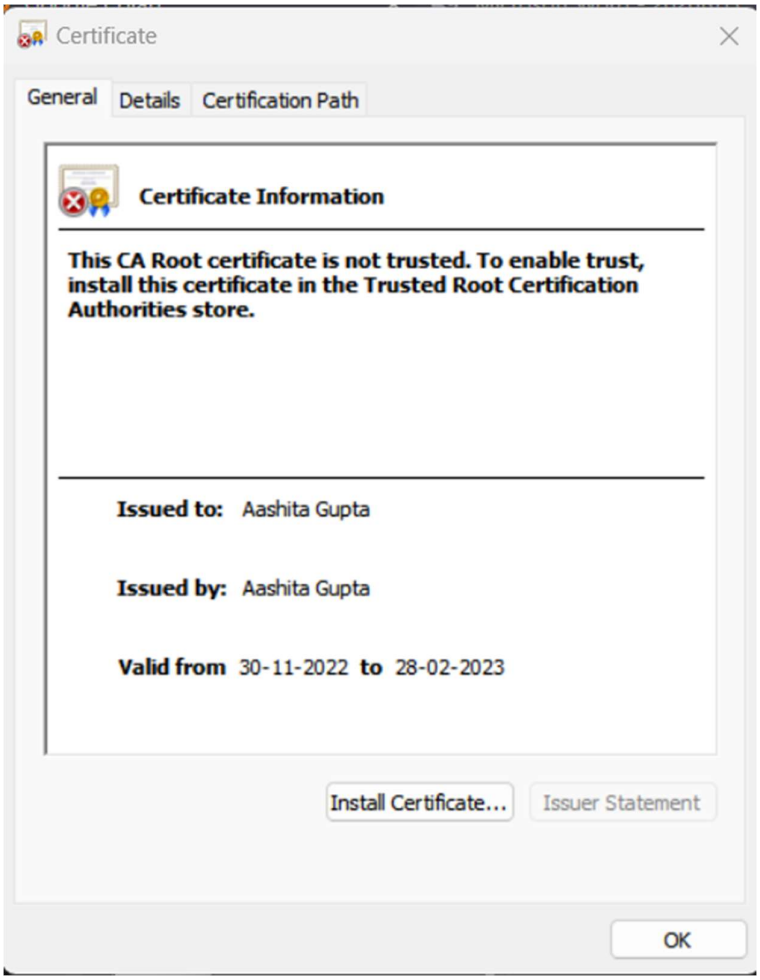
Snapshots:

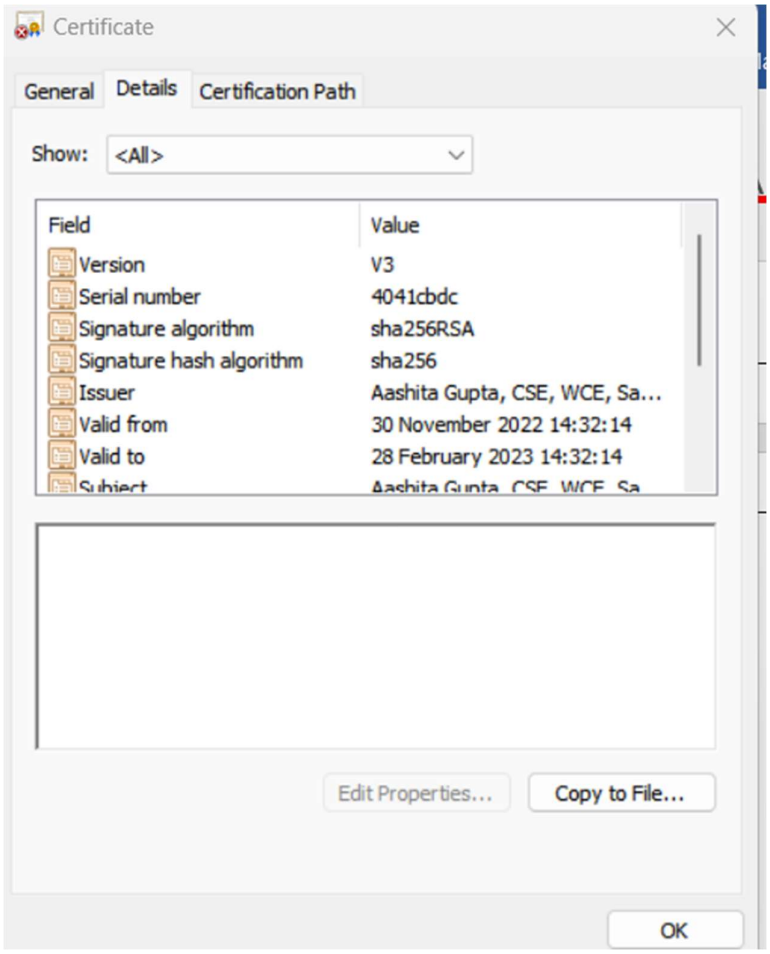

C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. All rights reserved.
C:\>keytool -genkey -keyalg RSA -alias techies -keystore tech.jks -validity 90 -keysize 2048
Enter keystore password:


C:\Windows\System32\cmd.e
Microsoft Windows [Version 10.0.22621.819]
(c) Microsoft Corporation. All rights reserved.
C:\>keytool -genkey -keyalg RSA -alias techies -keystore tech.jks -validity 90 -keysize 2048
Enter keystore password:
Re-enter new password:
What is your first and last name?
[Unknown]: Aashita Gupta
What is the name of your organizational unit?
[Unknown]: CSE
What is the name of your organization?
[Unknown]: WCE
What is the name of your City or Locality?
[Unknown]: Sangli
What is the name of your State or Province?
[Unknown]: Maharashtra
What is the two-letter country code for this unit?
[Unknown]: In
Is CN=Aashita Gupta, OU=CSE, O=WCE, L=Sangli, ST=Maharashtra, C=In correct?
[no]:


server.xml
sumarraymp.c
vector_scalar.cu
C: > xampp > tomcat > conf > server.xml
Java HTTP Connector: /docs/config/http.html (blocking & non-blocking)
Java AJP Connector: /docs/config/ajp.html
APR (HTTP/AJP) Connector: /docs/apr.html
Define a non-SSL HTTP/1.1 Connector on port 8080
<Connector port="9090" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
<Connector
port="9091" maxThreads="200"
scheme="https" secure="true" SSLEnabled="true"
keystoreFile="D:\\tech.jks" keystorePass="Aashita121"
clientAuth="false" sslProtocol="TLS" keyAlias="techies" />
<!-- A "Connector" using the shared thread pool-->
<!--
<Connector executor="tomcatThreadPool"
port="8080" protocol="HTTP/1.1"
connectionTimeout="20000"
redirectPort="8443" />
-->
<!-- Define an SSL HTTP/1.1 Connector on port 8443
This connector uses the BIO implementation that requires the JSSE
style configuration. When using the APR/native implementation, the
OpenSSL style configuration is required as described in the APR/native

```
C:\Windows\System32\cmd.e
operable program or batch file.

D:\>RSA -alias techies -keystore tech.jks -validity 90 -keysize 2048
'RSA' is not recognized as an internal or external command,
operable program or batch file.

D:\>keytool -genkey -keyalg RSA -alias techies -keystore tech.jks -validity 90 -keysize 2048
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  Aashita Gupta
What is the name of your organizational unit?
  [Unknown]:  CSE
What is the name of your organization?
  [Unknown]:  WCE
What is the name of your City or Locality?
  [Unknown]:  Sangli
What is the name of your State or Province?
  [Unknown]:  Maharashtra
What is the two-letter country code for this unit?
  [Unknown]:  In
Is CN=Aashita Gupta, OU=CSE, O=WCE, L=Sangli, ST=Maharashtra, C=In correct?
  [no]:  yes


D:\>keytool -export -alias techies -file "D:\tech_cert.cer" -keystore "D:\tech.jks"
Enter keystore password:
Certificate stored in file <D:\tech_cert.cer>

D:\>
```





Conclusion: The digital certificate is the identity card of the public key, issued by an organization called a certification authority (often noted CA for Certification Authority).

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 14

Title: Implementation of Chinese Remainder Theorem.

Theory:
Chinese Remainder Theorem states that there always exists an x that satisfies given congruences.
Let num[0], num[1], …num[k-1] be positive integers that are pairwise coprime. Then, for any given sequence of integers rem[0], rem[1], … rem[k-1], there exists an integer x solving the following system of simultaneous congruences.

$$\begin{cases} x \equiv rem[0] & (\text{mod } num[0]) \\ \quad \cdots & \\ x \equiv rem[k-1] & (\text{mod } num[k-1]) \end{cases}$$

Furthermore, all solutions x of this system are congruent modulo the product, prod = num[0] * num[1] * ... * nun[k-1]. Hence

$$x \equiv y \ (\text{mod } num[i]), \quad 0 \leq i \leq k-1 \quad \Longleftrightarrow \quad x \equiv y \ (\text{mod } prod).$$

The first part is clear that there exists an x. The second part basically states that all solutions (including the minimum one) produce the same remainder when divided by-product of num[0], num[1], .. num[k-1]. In the above example, the product is 3*4*5 = 60. And 11 is one solution, other solutions are 71, 131, .. etc. All these solutions produce the same remainder when divided by 60, i.e., they are of form 11 + m*60 where m >= 0.
A Naive Approach to find x is to start with 1 and one by one increment it and check if dividing it with given elements in num[] produces corresponding remainders in rem[]. Once we find such an x, we return it.

Example:

```
Input:  num[] = {5, 7}, rem[] = {1, 3}
Output: 31
Explanation:
31 is the smallest number such that:
  (1) When we divide it by 5, we get remainder 1.
  (2) When we divide it by 7, we get remainder 3.

Input:  num[] = {3, 4, 5}, rem[] = {2, 3, 1}
Output: 11
Explanation:
11 is the smallest number such that:
  (1) When we divide it by 3, we get remainder 2.
  (2) When we divide it by 4, we get remainder 3.
  (3) When we divide it by 5, we get remainder 1.
```

Code Snapshots:

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


// Function for extended Euclidean Algorithm
int ansS, ansT;
int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)
{
    // Base Case
    if (r2 == 0)
    {
        ansS = s1;
        ansT = t1;
        return r1;
    }

    int q = r1 / r2;
    int r = r1 % r2;

    int s = s1 - q * s2;
    int t = t1 - q * t2;

    cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2
<< " " << s << " " << t1 << " " << t2 << " " << t << endl;

    return findGcdExtended(r2, r, s2, s, t2, t);
}
```

```cpp
int modInverse(int A, int M)
{
    int x, y;
    int g = findGcdExtended(A, M, 1, 0, 0, 1);
    if (g != 1) {
        cout << "Inverse doesn't exist";
        return 0;
    }
    else {

        // m is added to handle negative x


        int res = (ansS % M + M) % M;
        cout << "inverse is " << res << endl;
        return res;
    }
}



int findX(vector<int> num, vector<int> rem, int k)
{
    // Compute product of all numbers
    int prod = 1;
    for (int i = 0; i < k; i++)
        prod *= num[i];

    // Initialize result
    int result = 0;

    // Apply above formula
    for (int i = 0; i < k; i++) {
        int pp = prod / num[i];
        result += rem[i] * modInverse(pp, num[i]) * pp;
    }

    return result % prod;
}



int main()
{

    // 3
    // 3 4 5
```

```
    // 2 3 1
cout<<"Enter value of k: ";
    int k;
    cin >> k;

    vector<int> num(k), rem(k);
    cout<<"Value of numbers: ";
    for (int i = 0; i < k; i++)
        cin >> num[i];
    cout<<"Value of Remainders: ";
    for (int i = 0; i < k; i++)
        cin >> rem[i];

    int x = findX(num, rem, k);
    cout << "\nx is " << x;


    return 0;
}
```

Output Snapshots:

Conclusion:

1. The Chinese Remainder Theorem determines a number n that when divided by some given divisors give some remainders.
2. In conclusion, with the help of the Chinese Remainder Theorem, the only knowledge to guide with two remaining secret messages to get what we desire is the solution of the form: $142 + (143 \times i)$.
3. It can further be concluded that the Chinese Remainder Theorem has a lot of uses in our everyday life activities.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 15

Title: Snort Intrusion Detection System (IDS)
In this lab we will explore the Snort IDS. This is a signature based intrusion detection system used to detect network attacks. Snort can also be used as a simple packet logger, however we won't be doing that in this lab. Snort has multiple modes of operation, for the lab we will use snort as a packet sniffer, not inline.

Theory:

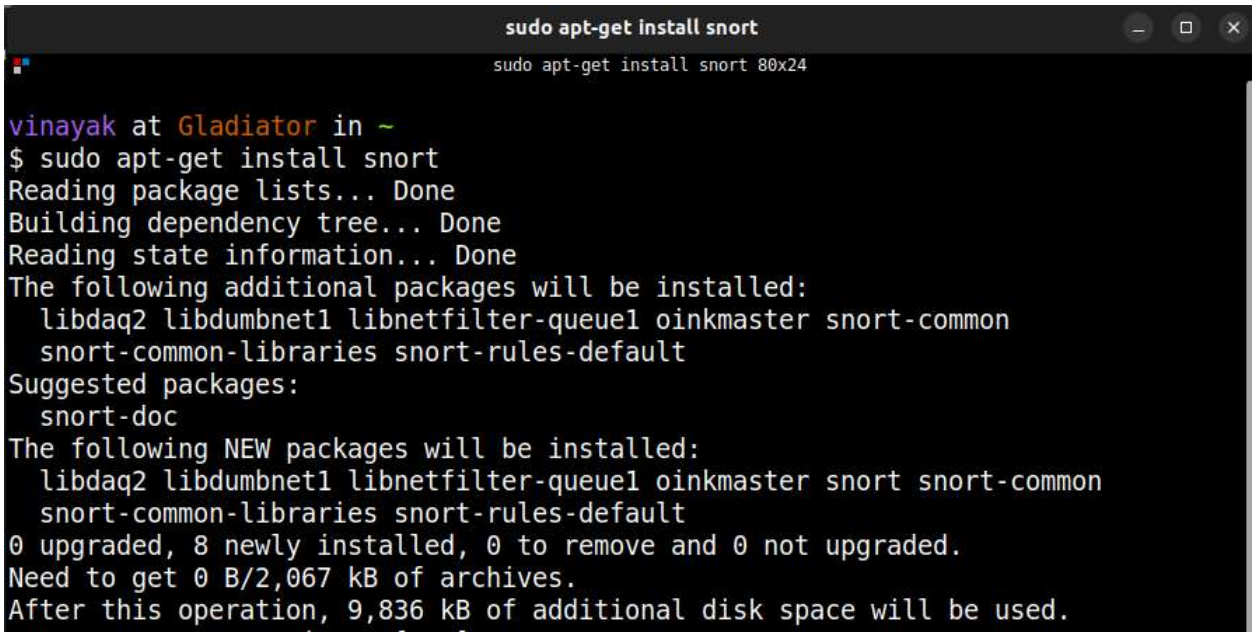
SNORT

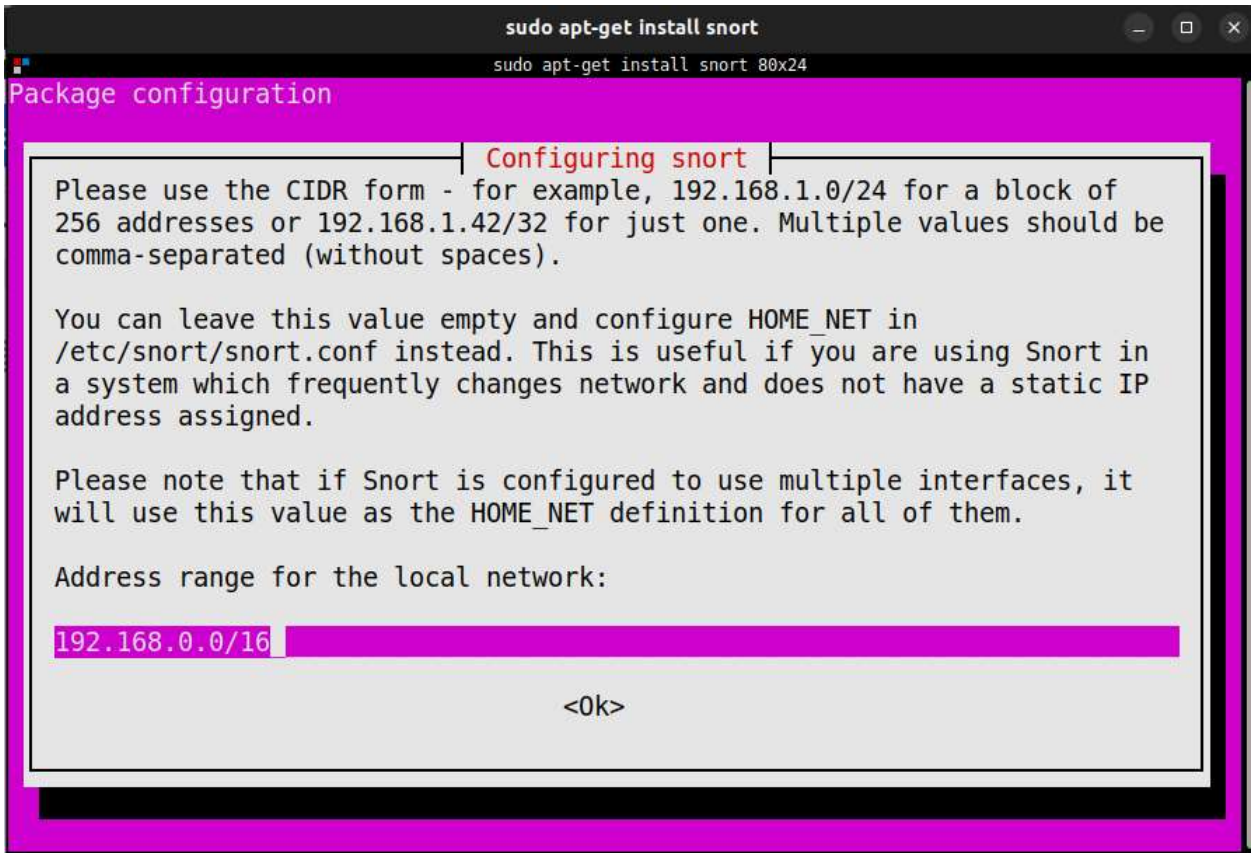
Snort is the world's foremost Open Source Intrusion Prevention System (IPS). Snort IPS uses a series of rules that help define malicious network activity and uses those rules to find packets that match against them and generates alerts for users.

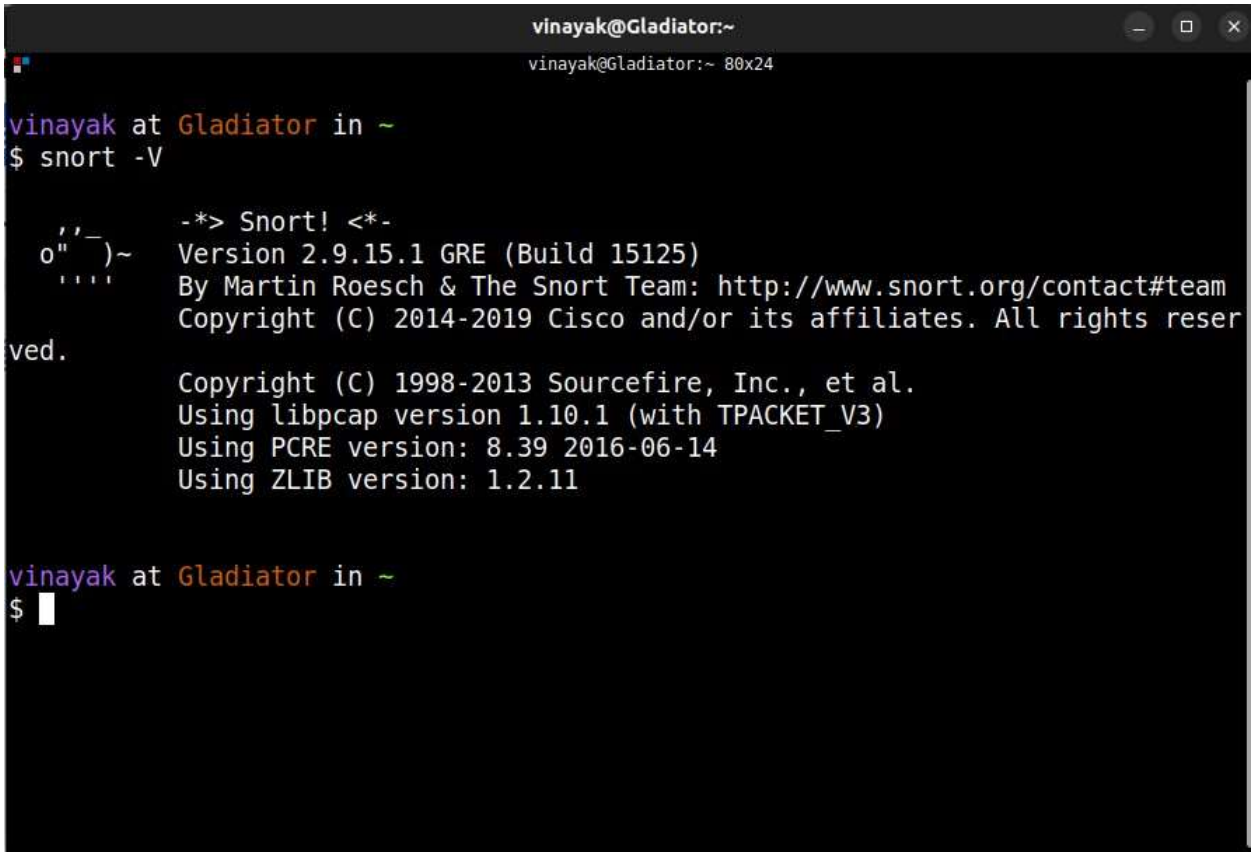
Snort can be deployed inline to stop these packets, as well. Snort has three primary uses: As a packet sniffer like tcpdump, as a packet logger – which is useful for network traffic debugging or can be used as a full-blown network intrusion prevention system. Snort can be downloaded and configured for personal and business use alike.

Snapshots:

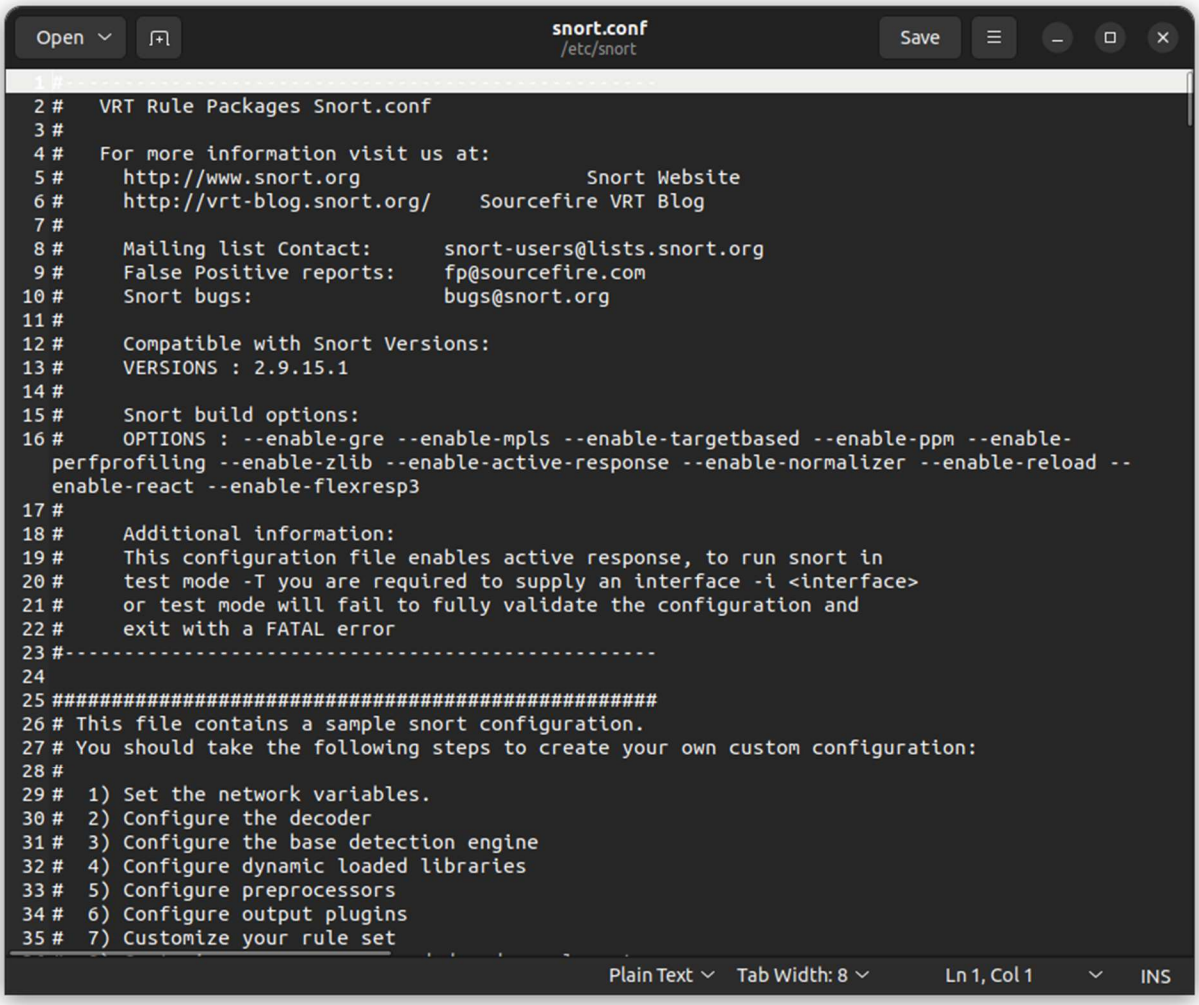
Snort Installation:



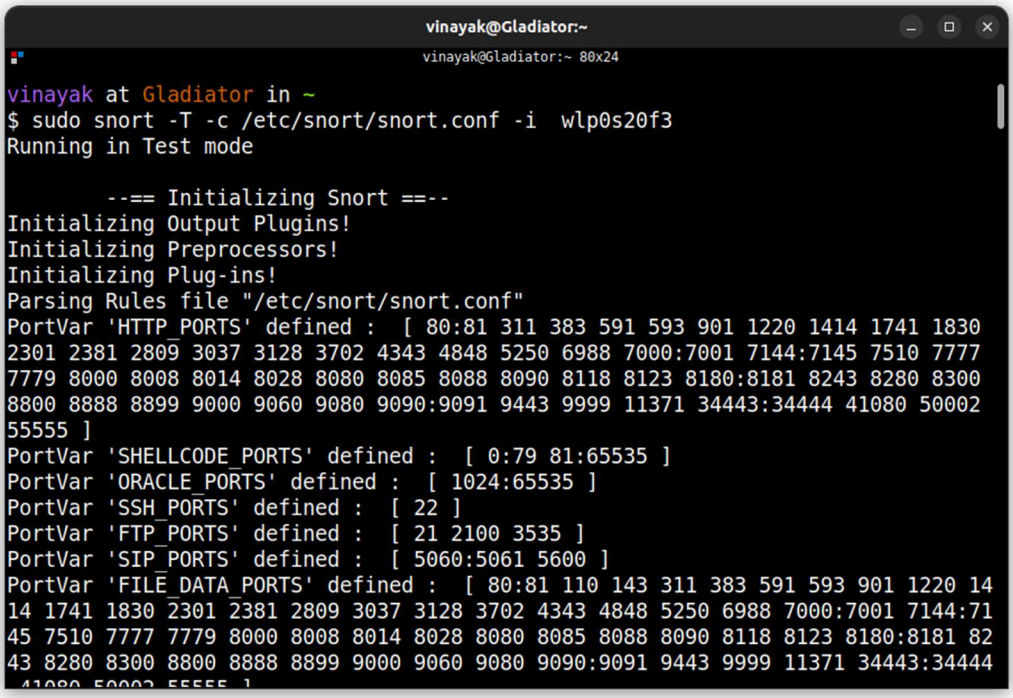

sudo apt-get install snort
sudo apt-get install snort 80x24
Package configuration
Configuring snort
Please use the CIDR form - for example, 192.168.1.0/24 for a block of
256 addresses or 192.168.1.42/32 for just one. Multiple values should be
comma-separated (without spaces).
You can leave this value empty and configure HOME_NET in
/etc/snort/snort.conf instead. This is useful if you are using Snort in
a system which frequently changes network and does not have a static IP
address assigned.
Please note that if Snort is configured to use multiple interfaces, it
will use this value as the HOME_NET definition for all of them.
Address range for the local network:
192.168.0.0/16
<Ok>

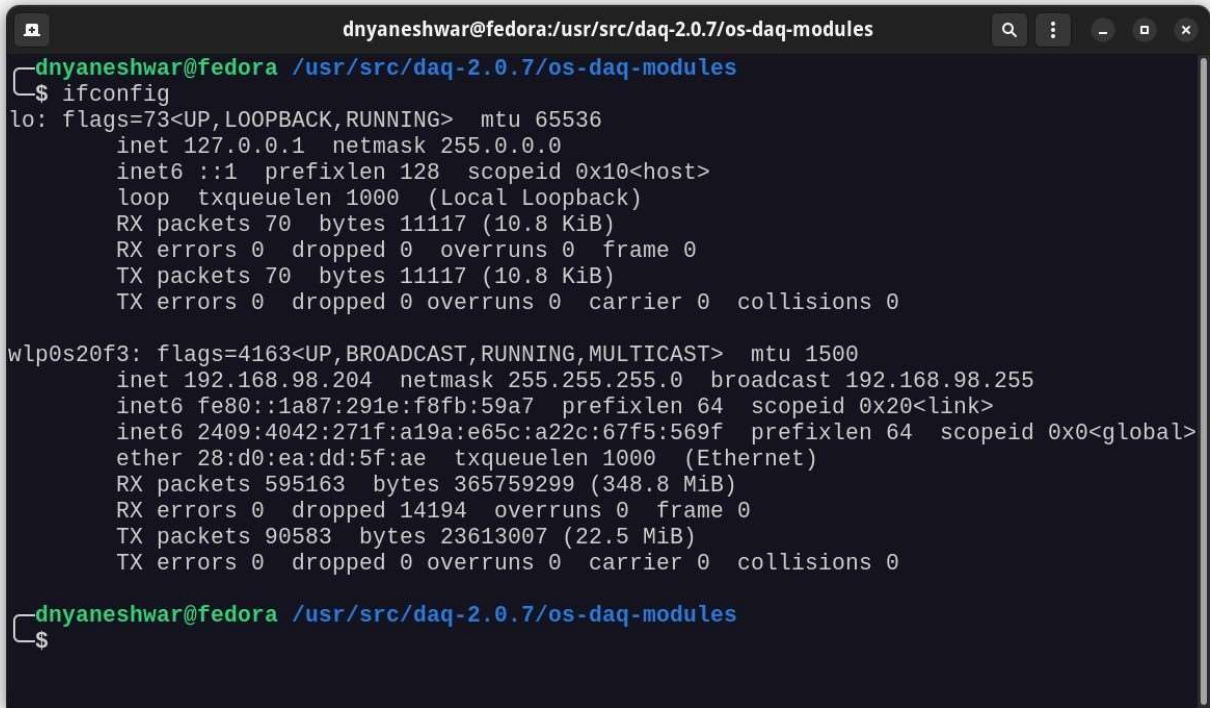


SNORT Configuration



Checking configurations for the wireless interface

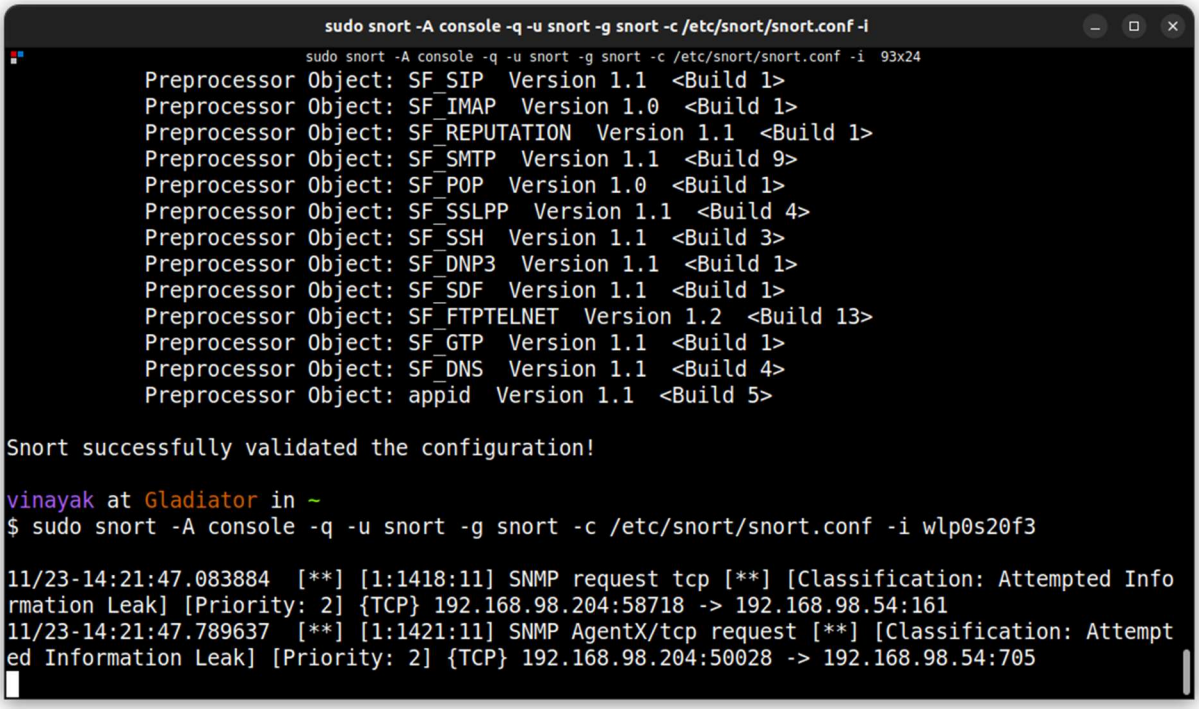




Attacking device Information:

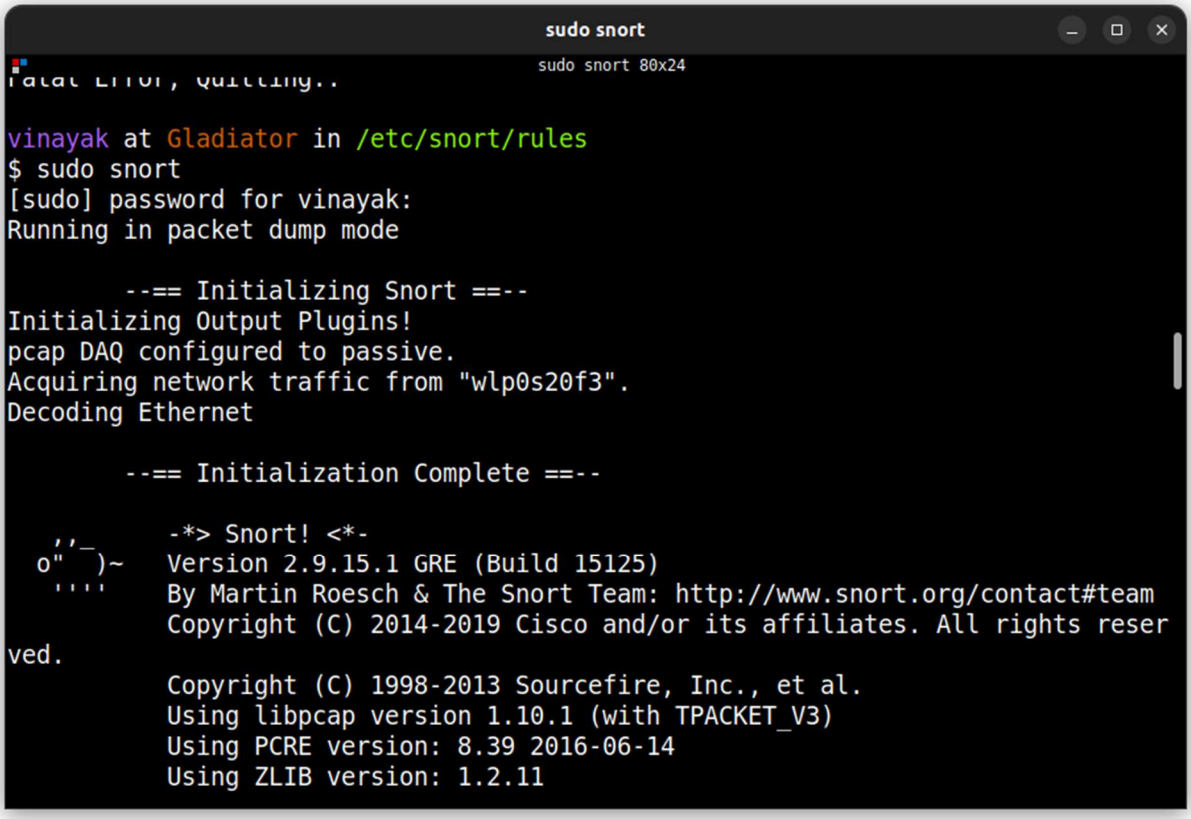
Running NMAP from another device

Starting SNORT in detection mode:

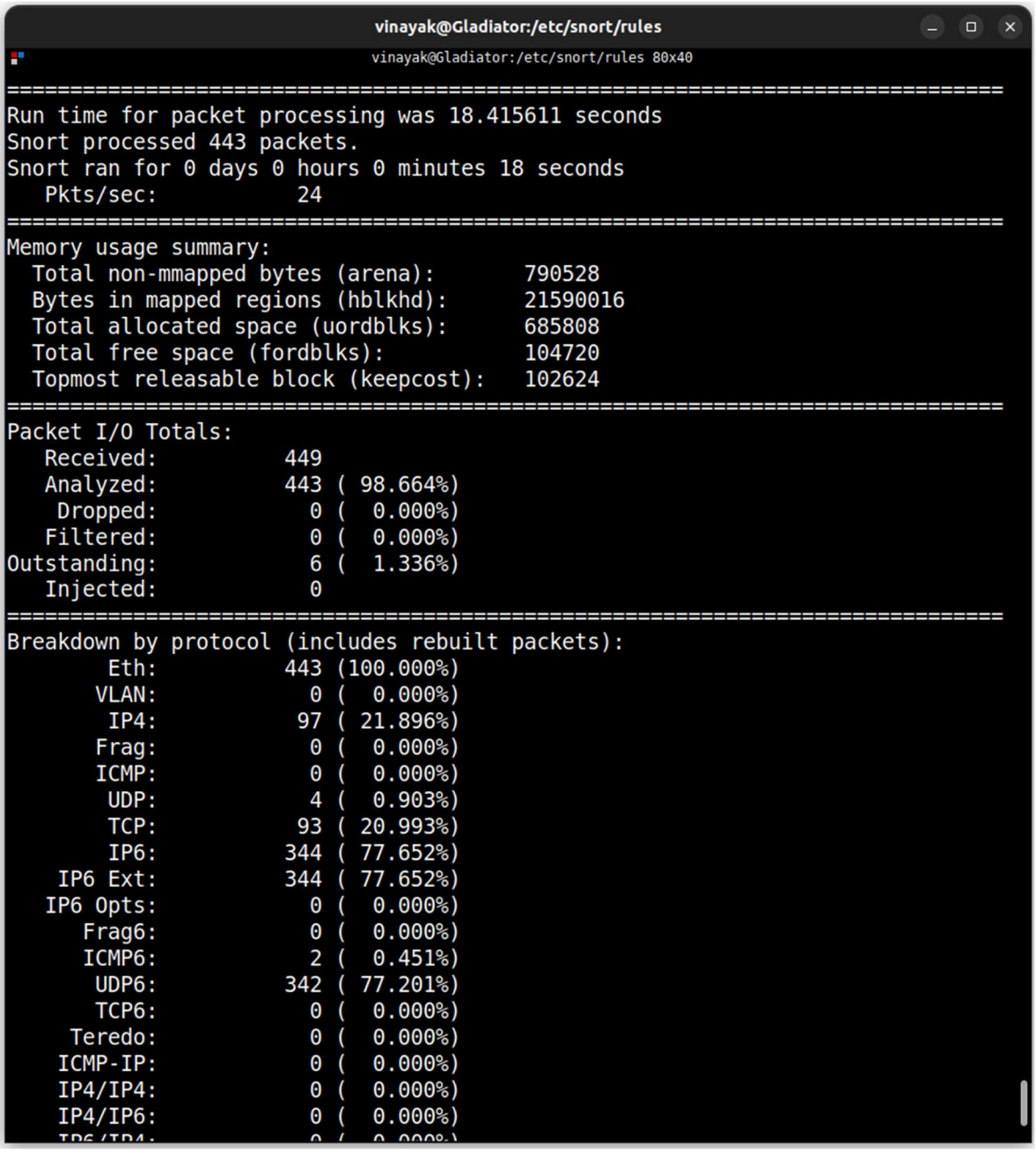


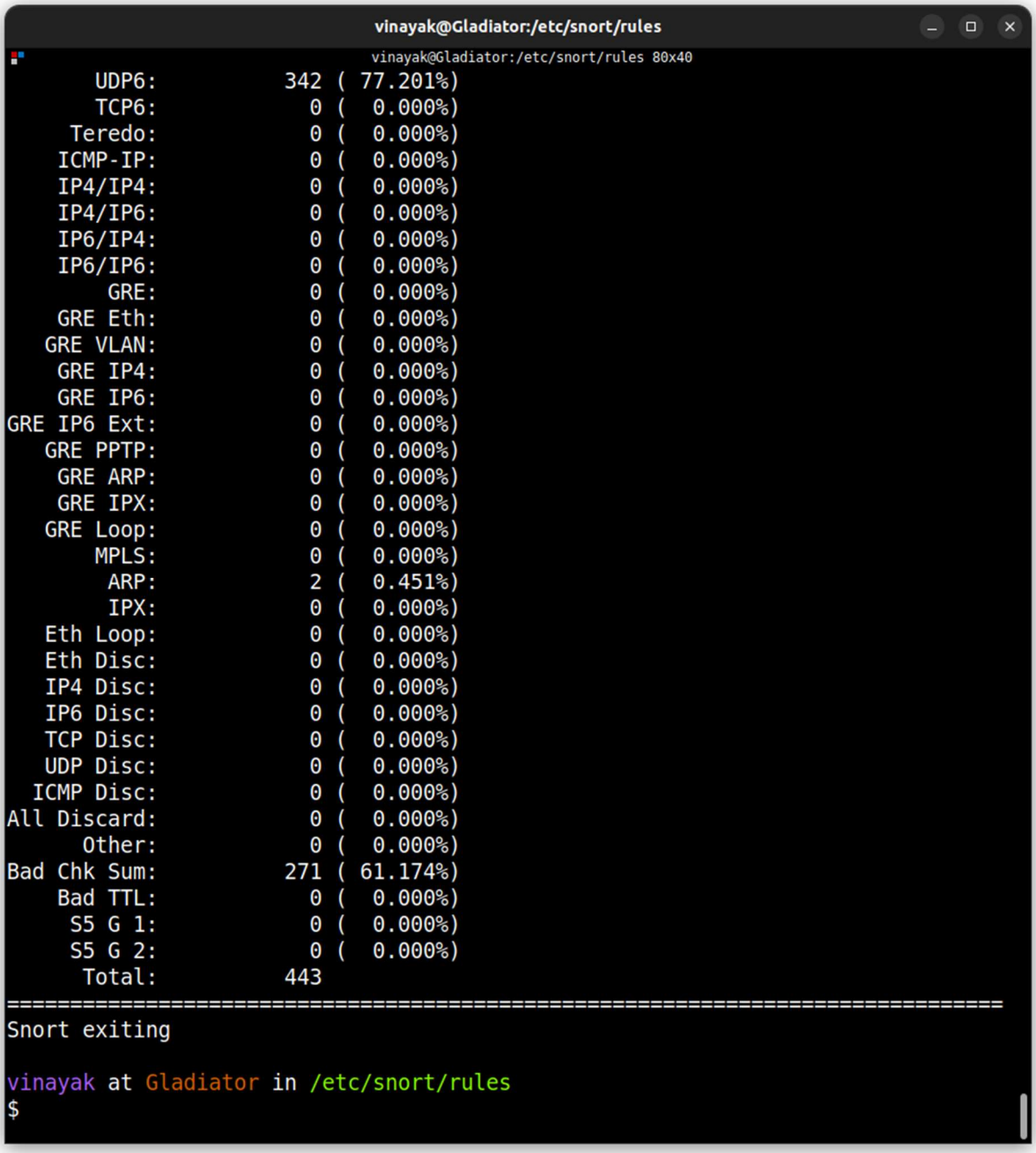
The above screenshot shows that SNORT has successfully detected the NMAP attack and an alert has been generated

Running snort in sniffing mode:



Sniffing result:


vinayak@Gladiator:/etc/snort/rules
vinayak@Gladiator:/etc/snort/rules 80x40
Run time for packet processing was 18.415611 seconds
Snort processed 443 packets.
Snort ran for 0 days 0 hours 0 minutes 18 seconds
Pkts/sec: 24
Memory usage summary:
Total non-mmapped bytes (arena): 790528
Bytes in mapped regions (hblkhd): 21590016
Total allocated space (uordblks): 685808
Total free space (fordblks): 104720
Topmost releasable block (keepcost): 102624
Packet I/O Totals:
Received: 449
Analyzed: 443 (98.664%)
Dropped: 0 (0.000%)
Filtered: 0 (0.000%)
Outstanding: 6 (1.336%)
Injected: 0
Breakdown by protocol (includes rebuilt packets):
Eth: 443 (100.000%)
VLAN: 0 (0.000%)
IP4: 97 (21.896%)
Frag: 0 (0.000%)
ICMP: 0 (0.000%)
UDP: 4 (0.903%)
TCP: 93 (20.993%)
IP6: 344 (77.652%)
IP6 Ext: 344 (77.652%)
IP6 Opts: 0 (0.000%)
Frag6: 0 (0.000%)
ICMP6: 2 (0.451%)
UDP6: 342 (77.201%)
TCP6: 0 (0.000%)
Teredo: 0 (0.000%)
ICMP-IP: 0 (0.000%)
IP4/IP4: 0 (0.000%)
IP4/IP6: 0 (0.000%)


vinayak@Gladiator:/etc/snort/rules
vinayak@Gladiator:/etc/snort/rules 80x40
UDP6: 342 (77.201%)
TCP6: 0 (0.000%)
Teredo: 0 (0.000%)
ICMP-IP: 0 (0.000%)
IP4/IP4: 0 (0.000%)
IP4/IP6: 0 (0.000%)
IP6/IP4: 0 (0.000%)
IP6/IP6: 0 (0.000%)
GRE: 0 (0.000%)
GRE Eth: 0 (0.000%)
GRE VLAN: 0 (0.000%)
GRE IP4: 0 (0.000%)
GRE IP6: 0 (0.000%)
GRE IP6 Ext: 0 (0.000%)
GRE PPTP: 0 (0.000%)
GRE ARP: 0 (0.000%)
GRE IPX: 0 (0.000%)
GRE Loop: 0 (0.000%)
MPLS: 0 (0.000%)
ARP: 2 (0.451%)
IPX: 0 (0.000%)
Eth Loop: 0 (0.000%)
Eth Disc: 0 (0.000%)
IP4 Disc: 0 (0.000%)
IP6 Disc: 0 (0.000%)
TCP Disc: 0 (0.000%)
UDP Disc: 0 (0.000%)
ICMP Disc: 0 (0.000%)
All Discard: 0 (0.000%)
Other: 0 (0.000%)
Bad Chk Sum: 271 (61.174%)
Bad TTL: 0 (0.000%)
S5 G 1: 0 (0.000%)
S5 G 2: 0 (0.000%)
Total: 443
Snort exiting
vinayak at Gladiator in /etc/snort/rules
$

Conclusion:

Snort is referred to as a packet sniffer that monitors network traffic, scrutinizing each packet closely to detect a dangerous payload or suspicious anomalies.

Final Year B. Tech, Sem VII 2022-23
PRN – 2020BTECS00211
Name – Aashita Narendra Gupta
Cryptography And Network Security Lab
Batch: B4
Practical No – 16

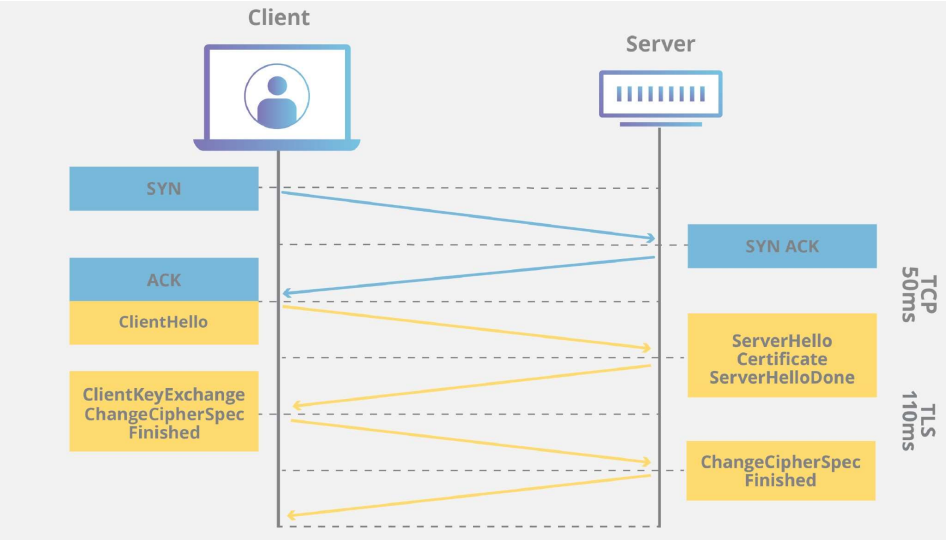
Title: To observe SSL/TLS (Secure Sockets Layer/ Transport Layer Security)in action.

Theory:

SSL

SSL (Secure Sockets Layer) and its successor, TLS (Transport Layer Security), are protocols for establishing authenticated and encrypted links between networked computers. Although the SSL protocol was deprecated with the release of TLS 1.0 in 1999, it is still common to refer to these related technologies as “SSL” or “SSL/TLS.” The most current version is TLS 1.3, defined in RFC 8446

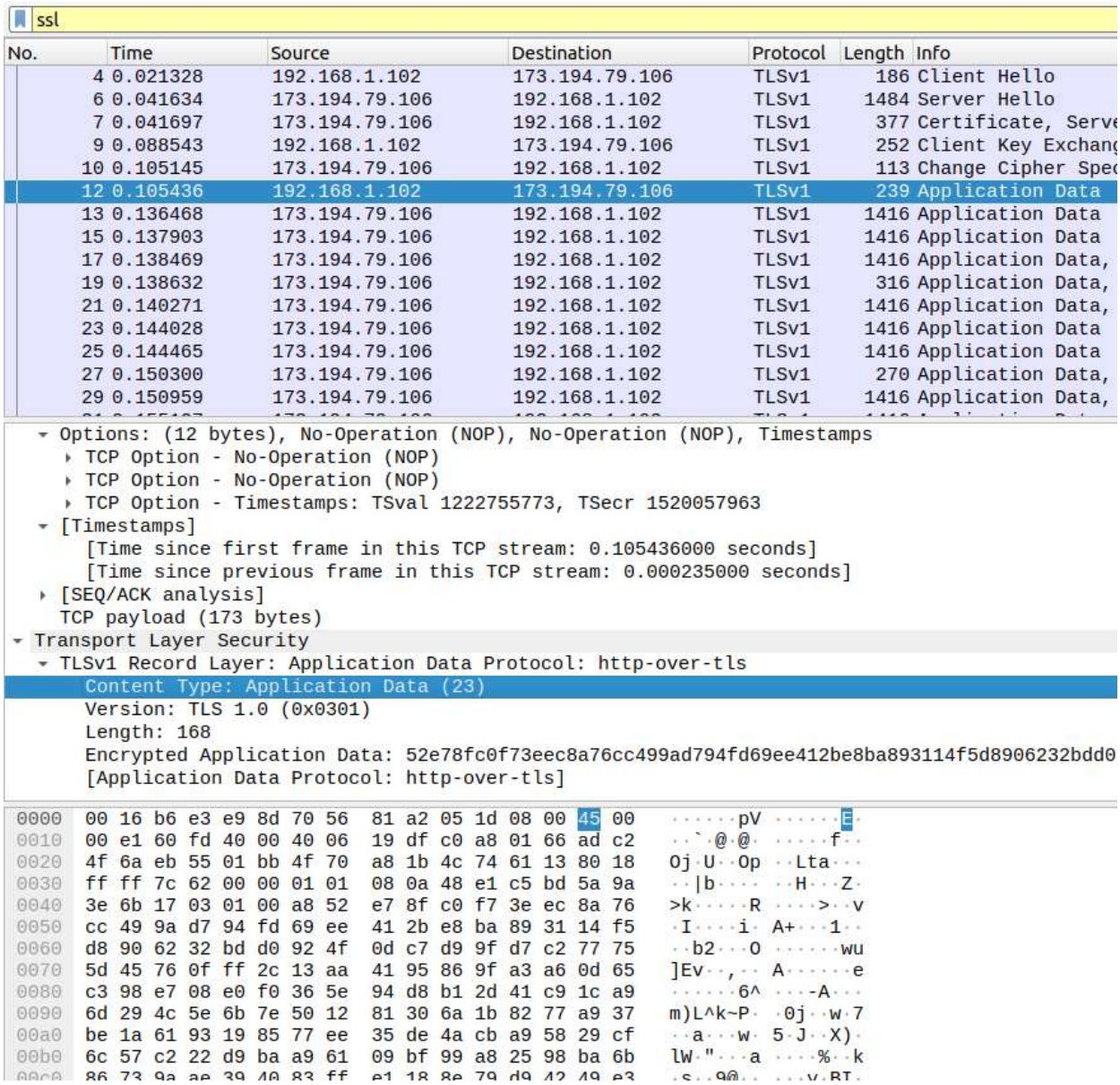
TLS (Transport Layer Security), released in 1999, is the successor to the SSL (Secure Sockets Layer) protocol for authentication and encryption. TLS 1.3 is defined in in RFC 8446 (August 2018).



Questions:

1. What is the Content-Type for a record containing Application Data?

Content-Type is: Application Data (23)

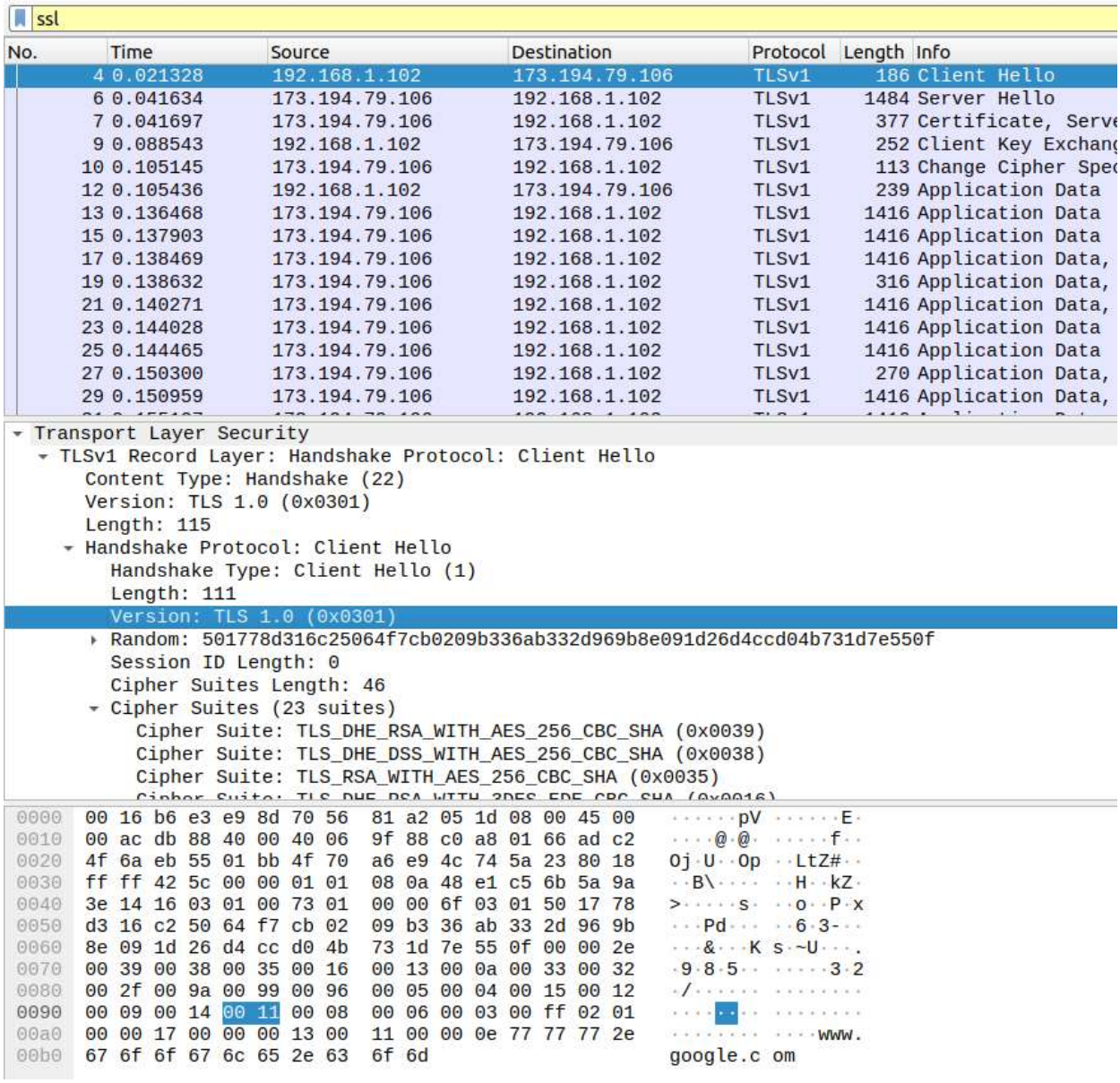


2. What version constant is used in your trace, and which version of TLS does it represent?

Ans:

TLS version : 1.0

TLS version constant: 0x0301

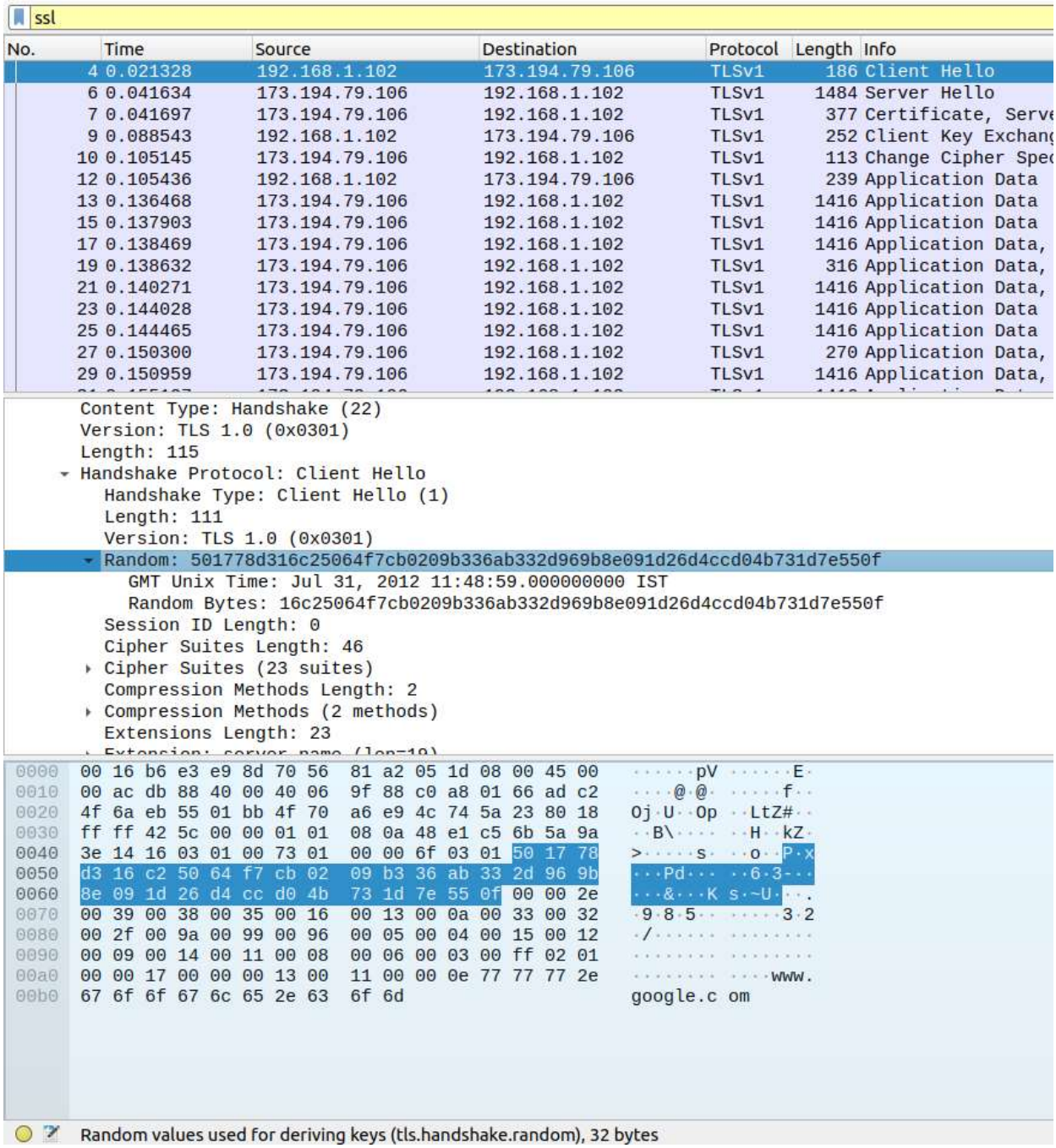


4.1 Hello Message

1. How long in bytes is the random data in the Hellos? Both the Client andServer include this random data (a nonce) to allow the establishment of session keys.

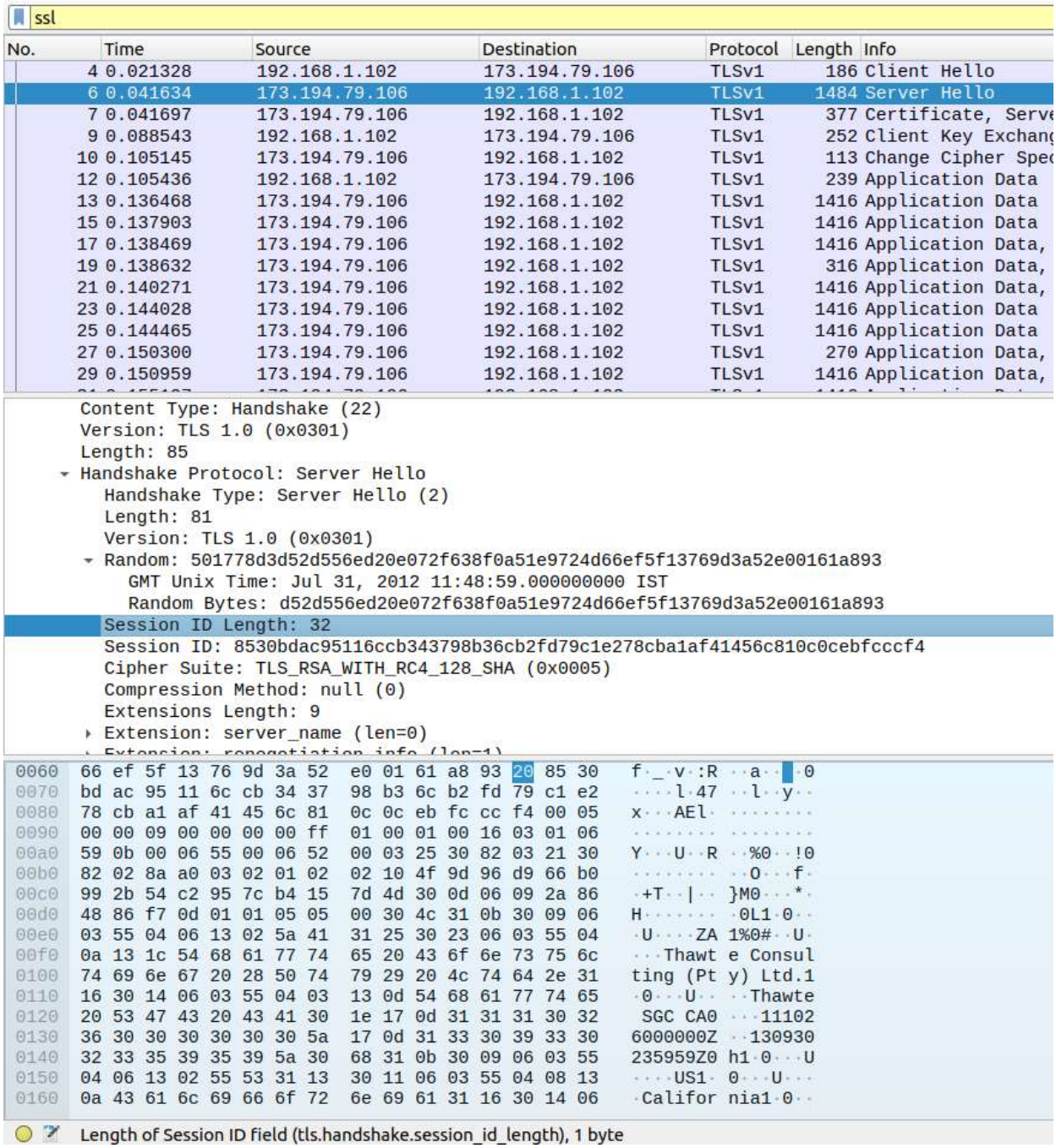
Ans:

Length of Random data filed: 32 bytes



2. How long in bytes is the session identifier sent by the server?This identifier allows later resumption of the session with an abbreviated handshake when both the client and server indicate the same value. In our case, the client likely sent no session ID as there was nothing to resume.

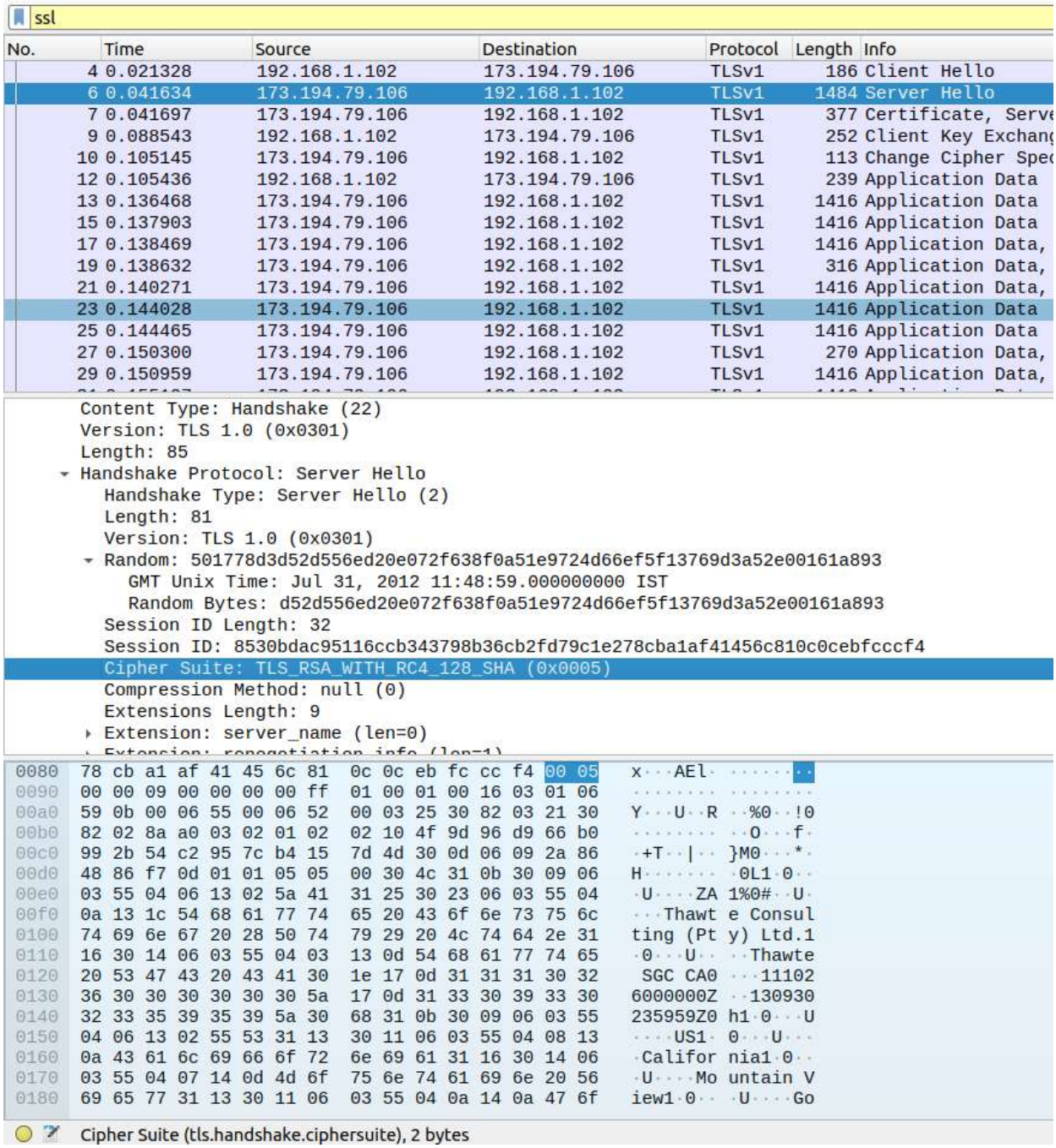
Ans: Session ID length : 32



3. What Cipher suite is chosen by the Server? Give its name and value. The Client will list the different cipher methods it supports, and the Server will pick one of these methods to use.

Ans:

Cipher Suit name: TLS_RSA_WITH_RC4_128_SHA (0x0005)

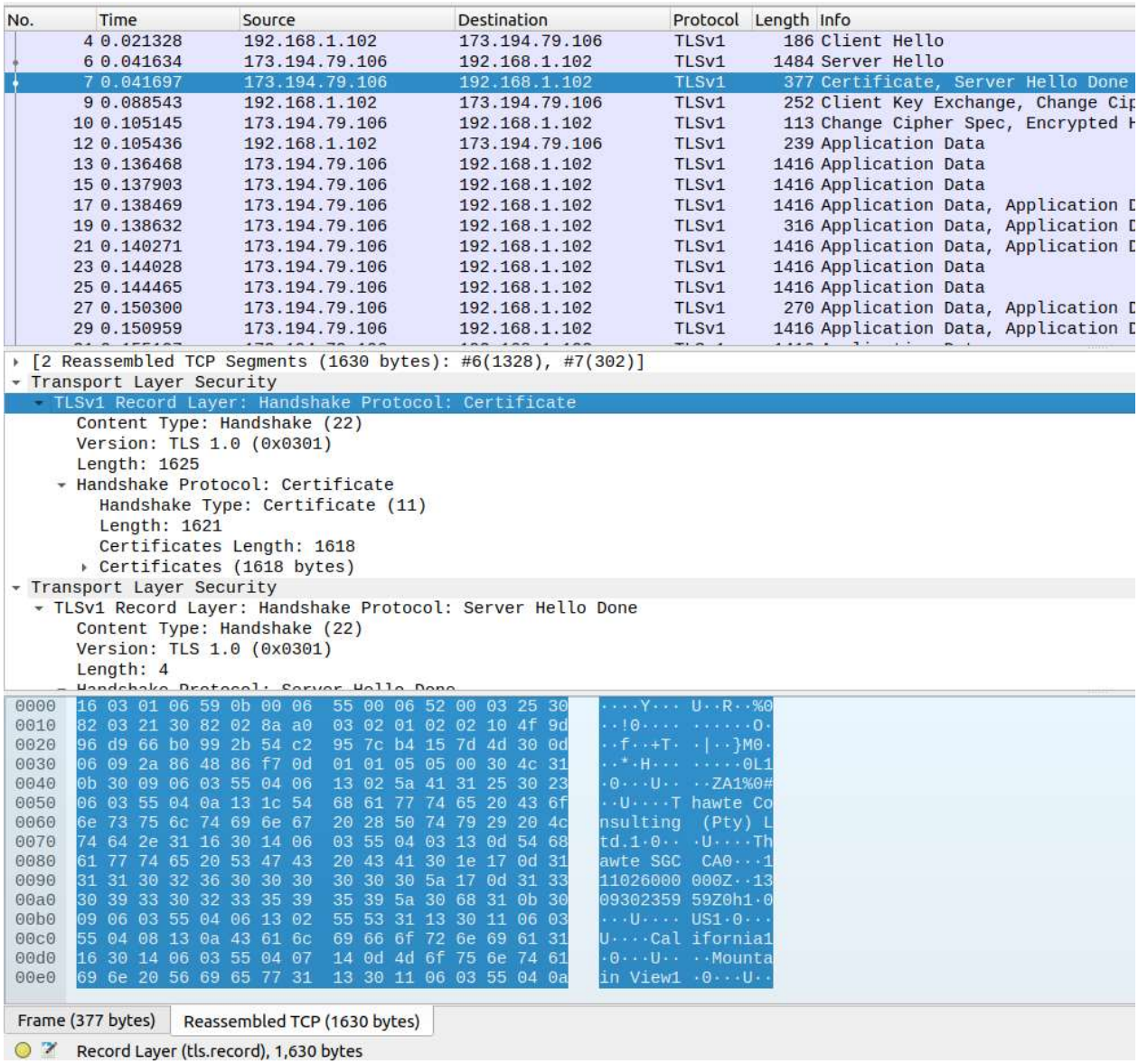


4.2 Certificate Messages

1. Who sends the Certificate, the client, the server, or both?A certificate is sent by one party to let the other party authenticate that it is who it claims to be. Based on this usage, you should be able to guess who sends the certificate and check the messages in your trace.

Ans:

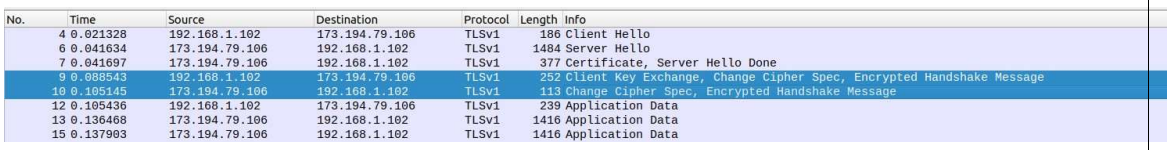
Server sends the certificate



4.3 Client Key Exchange and Change Cipher Messages

1. Who sends the Change Cipher Spec message, the client, the server, or both?

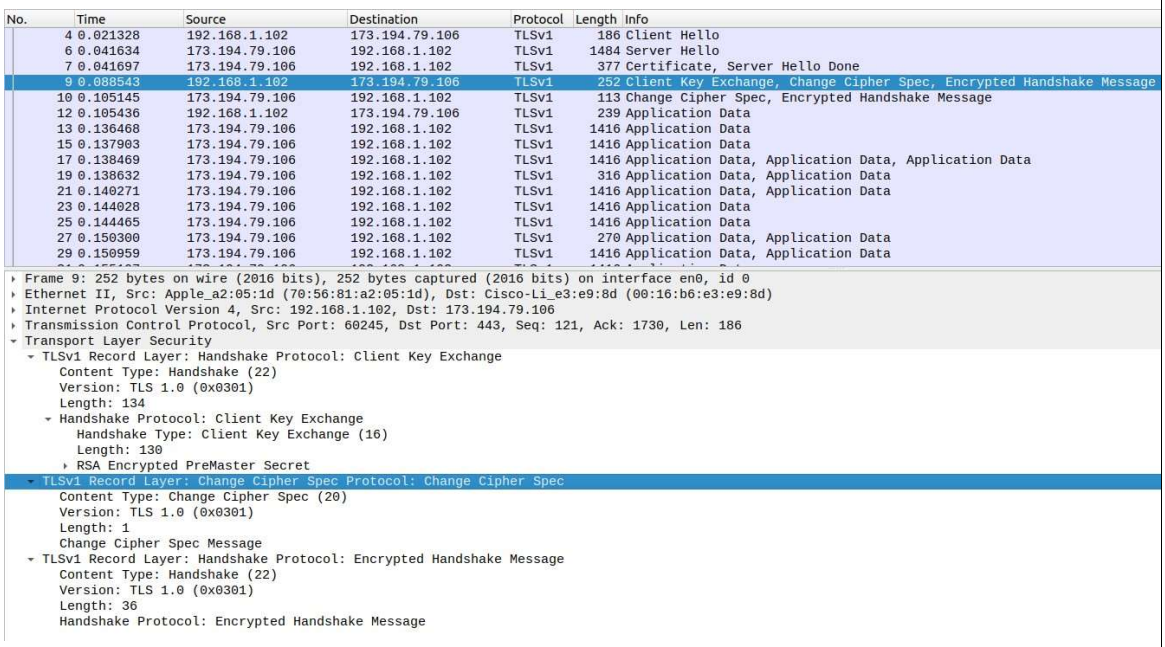
Ans: Both server and client sends the Change Cipher Spec message



No.	Time	Source	Destination	Protocol	Length	Info
4	0.021328	192.168.1.102	173.194.79.106	TLSv1	186	Client Hello
6	0.041634	173.194.79.106	192.168.1.102	TLSv1	1484	Server Hello
7	0.041697	173.194.79.106	192.168.1.102	TLSv1	377	Certificate, Server Hello Done
9	0.088543	192.168.1.102	173.194.79.106	TLSv1	252	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
12	0.105436	192.168.1.102	173.194.79.106	TLSv1	239	Application Data
13	0.136468	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data
15	0.137903	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data

2. What are the contents carried inside the Change Cipher Spec message? Lookpast the Content Type and other headers to see the message itself

Ans: Change Cipher Spec message contains:
Content Type, Version, Length and Change Cipher Spec Message



No.	Time	Source	Destination	Protocol	Length	Info
4	0.021328	192.168.1.102	173.194.79.106	TLSv1	186	Client Hello
6	0.041634	173.194.79.106	192.168.1.102	TLSv1	1484	Server Hello
7	0.041697	173.194.79.106	192.168.1.102	TLSv1	377	Certificate, Server Hello Done
9	0.088543	192.168.1.102	173.194.79.106	TLSv1	252	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Handshake Message
12	0.105436	192.168.1.102	173.194.79.106	TLSv1	239	Application Data
13	0.136468	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data
15	0.137903	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data
17	0.138469	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data, Application Data, Application Data
19	0.138632	173.194.79.106	192.168.1.102	TLSv1	316	Application Data, Application Data
21	0.140271	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data, Application Data
23	0.144028	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data
25	0.144465	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data
27	0.150300	173.194.79.106	192.168.1.102	TLSv1	270	Application Data, Application Data
29	0.150959	173.194.79.106	192.168.1.102	TLSv1	1416	Application Data, Application Data

```
▸ Frame 9: 252 bytes on wire (2016 bits), 252 bytes captured (2016 bits) on interface en0, id 0
▸ Ethernet II, Src: Apple_a2:05:1d (70:56:81:a2:05:1d), Dst: Cisco-Li_e3:e9:8d (00:16:b6:e3:e9:8d)
▸ Internet Protocol Version 4, Src: 192.168.1.102, Dst: 173.194.79.106
▸ Transmission Control Protocol, Src Port: 60245, Dst Port: 443, Seq: 121, Ack: 1730, Len: 186
▾ Transport Layer Security
  ▾ TLSv1 Record Layer: Handshake Protocol: Client Key Exchange
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 134
    ▾ Handshake Protocol: Client Key Exchange
        Handshake Type: Client Key Exchange (16)
        Length: 130
      ▸ RSA Encrypted PreMaster Secret
  ▾ TLSv1 Record Layer: Change Cipher Spec Protocol: Change Cipher Spec
      Content Type: Change Cipher Spec (20)
      Version: TLS 1.0 (0x0301)
      Length: 1
      Change Cipher Spec Message
  ▾ TLSv1 Record Layer: Handshake Protocol: Encrypted Handshake Message
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 36
      Handshake Protocol: Encrypted Handshake Message
```

Conclusion:

1. SSL/TLS certificates are vital to any website, but especially important to the performance of online stores or eCommerce sites.