SEM - VII - 2022-23
CNS Lab

B1 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 16

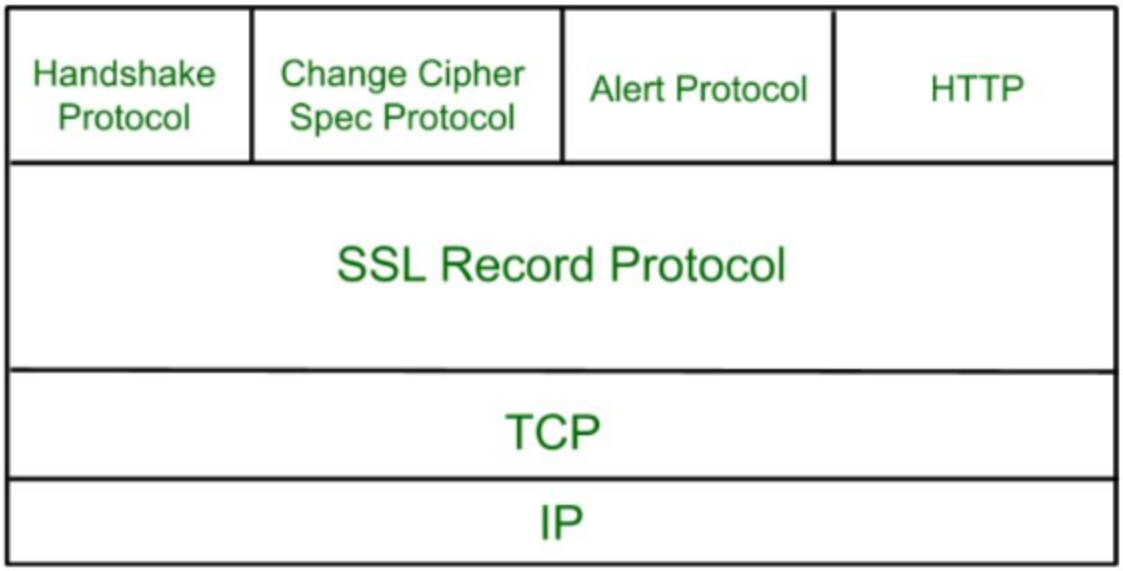
Title:- SSL/TLS Handshake Analysis using Wireshark

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

Theory:-

- SSL/TLS is used to secure TCP connections, and it is widely used as part of the secure web: HTTPS is SSL over HTTP .
- Secure Socket Layer (SSL) provides security to the data that is transferred between web browser and server.
- SSL encrypts the link between a web server and a browser which ensures that all data passed between them remains private and free from attack.
- Secure Socket Layer Protocols:
    a. SSL record protocol
    b. Handshake protocol
    c. Change-cipher spec protocol
    d. Alert protocol

SSL Protocol Stack:



Objectives of SSL

The goals of SSL are as follows –

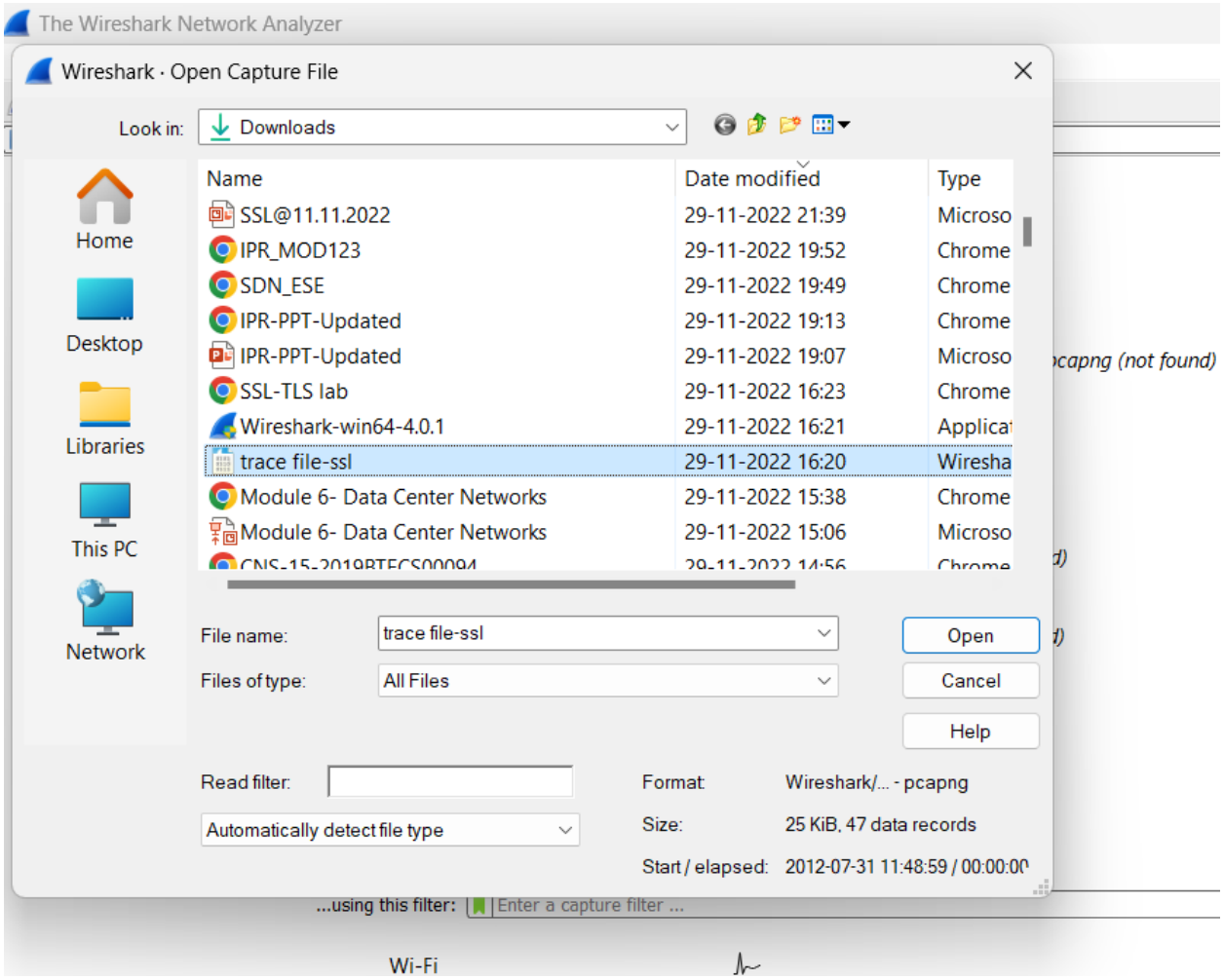
- ***Data integrity*** – Information is safe from tampering. The SSL Record Protocol, SSL Handshake Protocol, SSL Change CipherSpec Protocol, and SSL Alert Protocol maintain data privacy.
- ***Client-server authentication*** – The SSL protocol authenticates the client and server using standard cryptographic procedures.
- SSL is the forerunner of Transport Layer Security (TLS), a cryptographic technology for secure data transfer over the Internet.

- Wireshark is a free and open-source packet analyzer.
- It is used for network troubleshooting, analysis, software and communications protocol development, and education.

Use of Wireshark

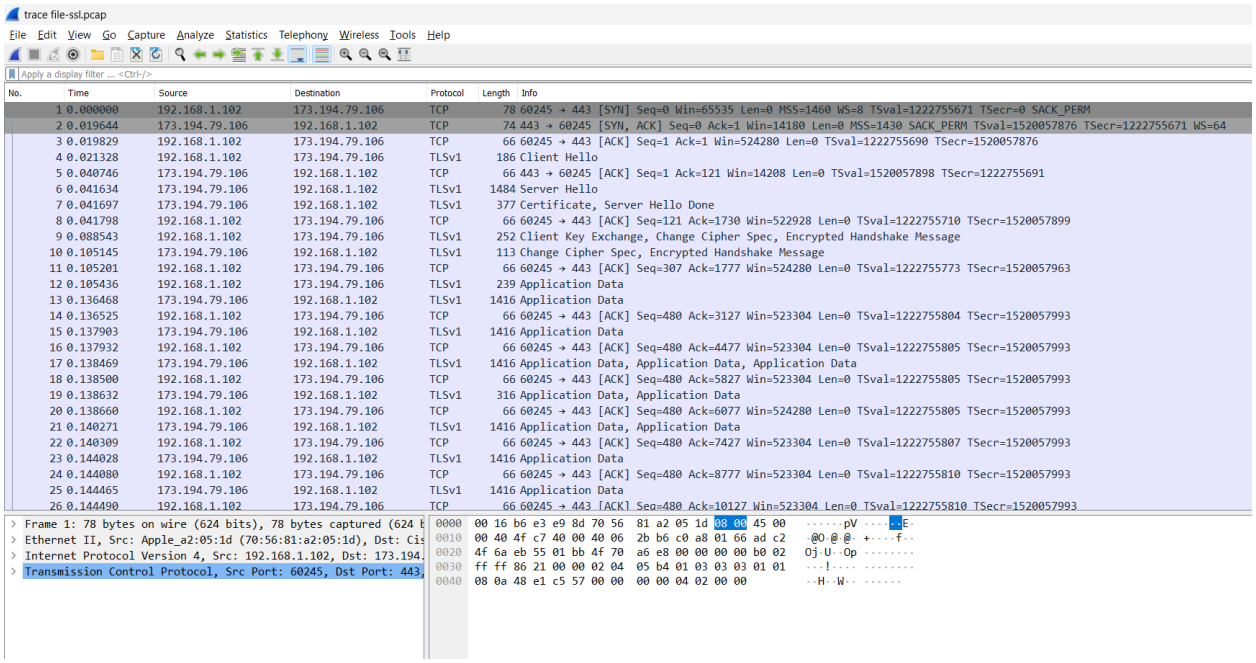
Step 1: Open a Trace you should use a supplied trace file trace-ssl.pcap.

File ⟶ Open ⟶ open from folder containing file

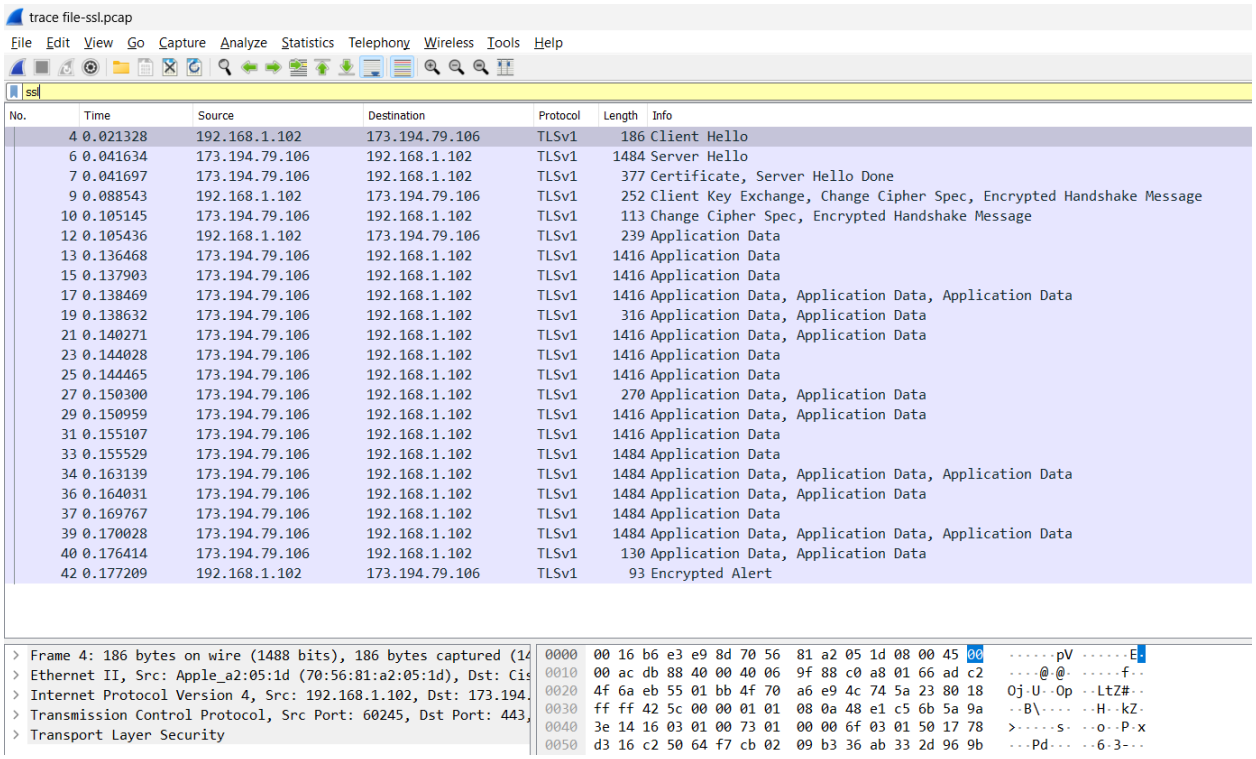


Step 2: Inspect the Trace

Now we are ready to look at the details of some SSL messages. To begin, enter and apply a display filter of ssl. This filter will help to simplify the display by showing only SSL and TLS messages. It will exclude other TCP segments that are part of the trace, such as Acks and connection open/close. Select a TLS message somewhere in the middle of your trace for which the Info field reads Application Data, and expand its Secure Sockets Layer block(by using triangular icon on left side). Application Data is a generic TLS message carrying contents for the application, such as the web page. It is a good place for us to start looking at TLS messages. Look for the following protocol blocks and fields in the message



Applying SSL Filter



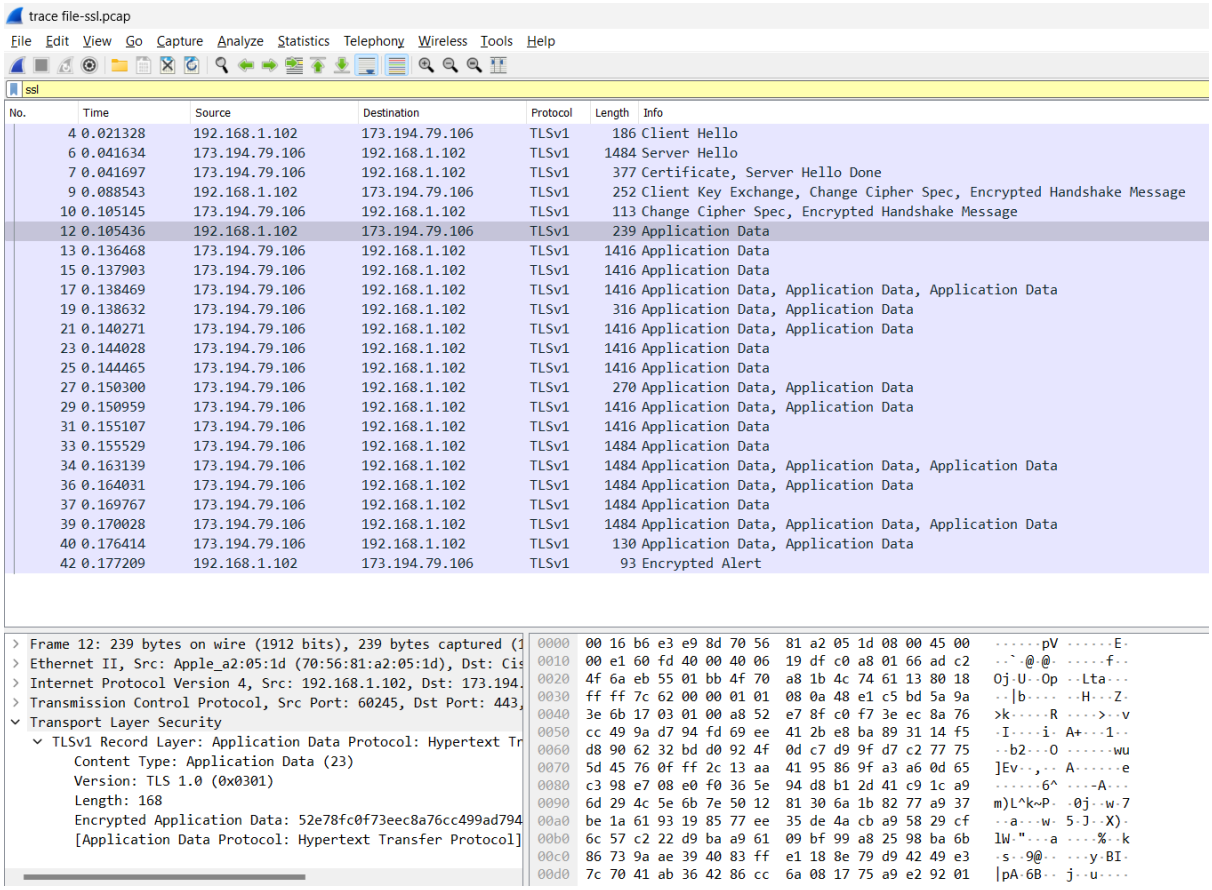
- The lower layer protocol blocks are TCP and IP because SSL runs on top of TCP/IP.]
- The SSL layer contains a TLS Record Layer. This is the foundational sublayer for TLS. All messages contain records. Expand this block to see its details.
- Each record starts with a Content Type field. This tells us what is in the contents of the record. Then comes a Version identifier.It will be a constant value for the SSL connection.
- It is followed by a Length field giving the length of the record.
  Last comes the contents of the record. Application Data records are sent after SSL has secured the connection, so the contents will show up as encrypted data.

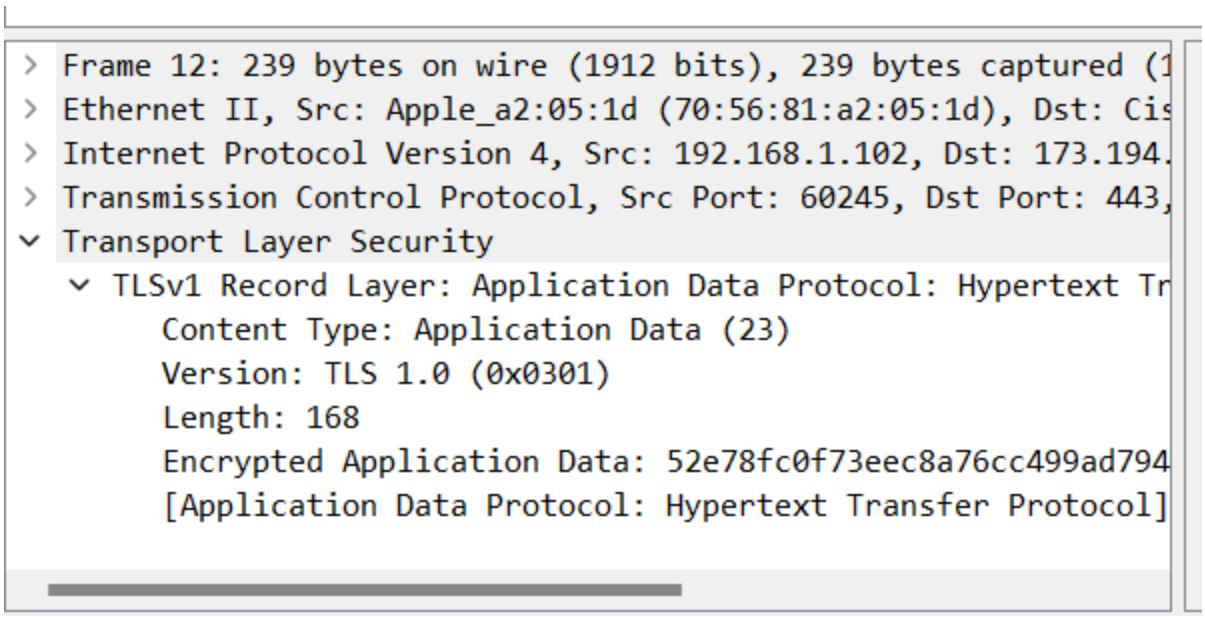
Note that, unlike other protocols we will see such as DNS, there may be multiple records in a single message. Each record will show up as its own block. Look at the Info column, and you will see messages with more than one block.

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

Ans:

The Content Type is Application Data.

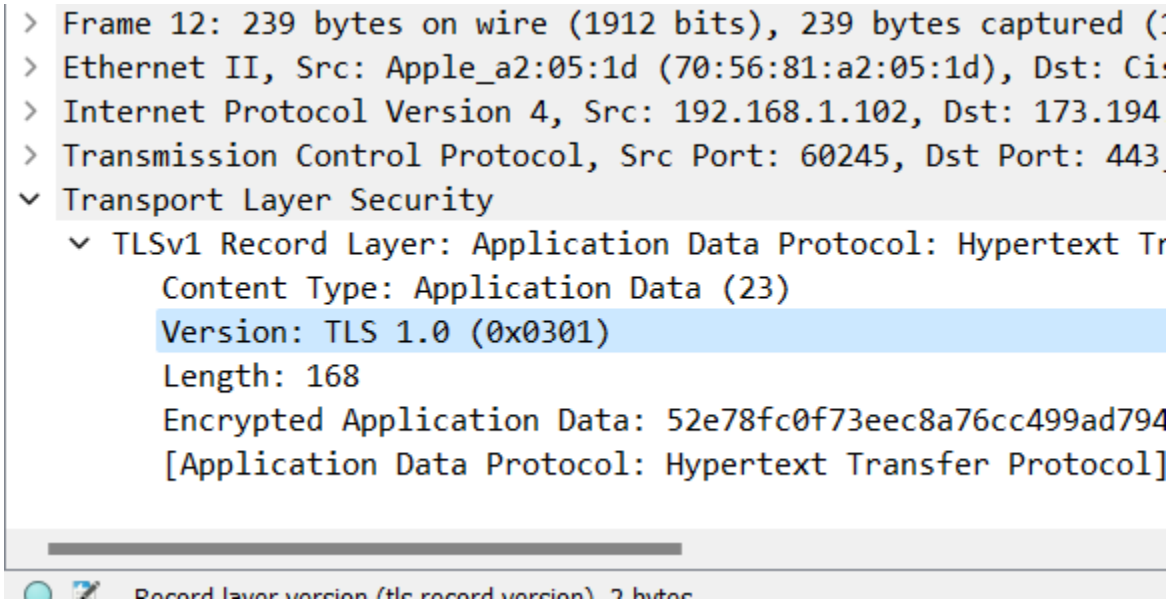



Frame 12: 239 bytes on wire (1912 bits), 239 bytes captured (1
Ethernet II, Src: Apple_a2:05:1d (70:56:81:a2:05:1d), Dst: Cis
Internet Protocol Version 4, Src: 192.168.1.102, Dst: 173.194.
Transmission Control Protocol, Src Port: 60245, Dst Port: 443,
Transport Layer Security
TLSv1 Record Layer: Application Data Protocol: Hypertext Tr
Content Type: Application Data (23)
Version: TLS 1.0 (0x0301)
Length: 168
Encrypted Application Data: 52e78fc0f73eec8a76cc499ad794
[Application Data Protocol: Hypertext Transfer Protocol]

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

Ans:

The version of TLS used is 1.0



Step 3: SSL Handshake

An important part of SSL is the initial handshake that establishes a secure connection. The handshake proceeds in several phases. There are slight differences for different versions of TLS and depending on the encryption scheme that is in use. The usual outline for a brand new connection is:

- Client (the browser) and Server(the web server) both send their Hellos
- Server sends its certificate to Client to authenticate (and optionally asks for Client Certificate)
- Client sends keying information and signals a switch to encrypted data.
- Server signals a switch to encrypted data.
- Both Client and Server send encrypted data.
- An Alert is used to tell the other party that the connection is closing. Note that there is also a mechanism to resume sessions for repeat connections between the same client and server to skip most of steps b and c.

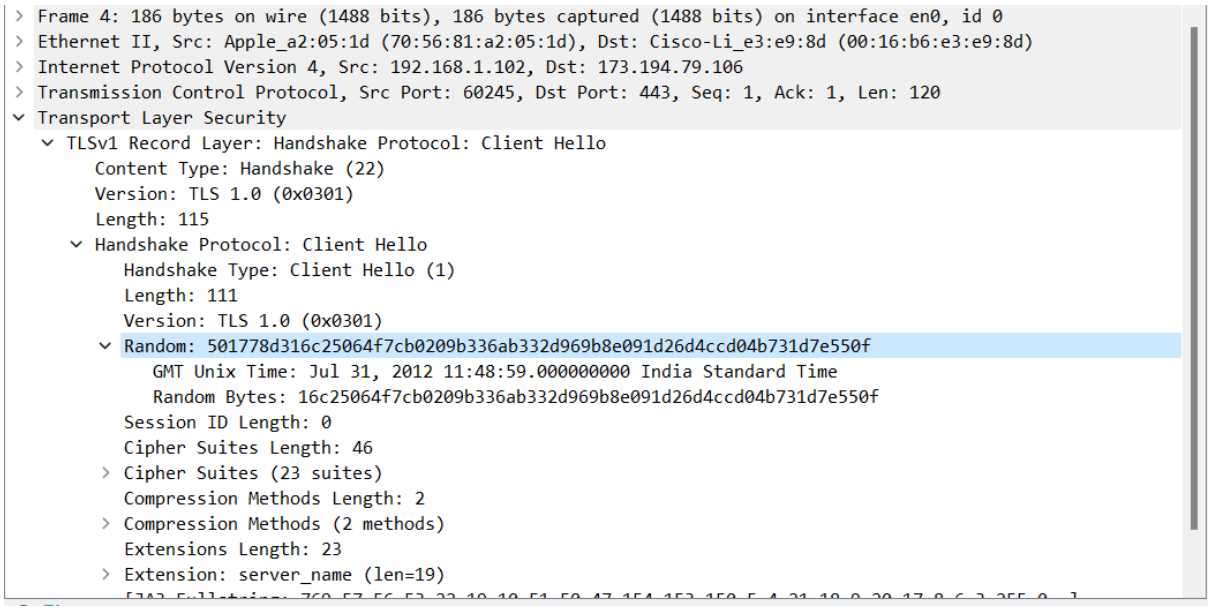
Hello Message

Find and inspect the details of the Client Hello and Server Hello messages, including expanding the Hand- shake protocol block within the TLS Record. For these initial messages, an encryption scheme is not yet established so the contents of the record are visible to us. They contain details of the secure connection setup in a Handshake protocol format.

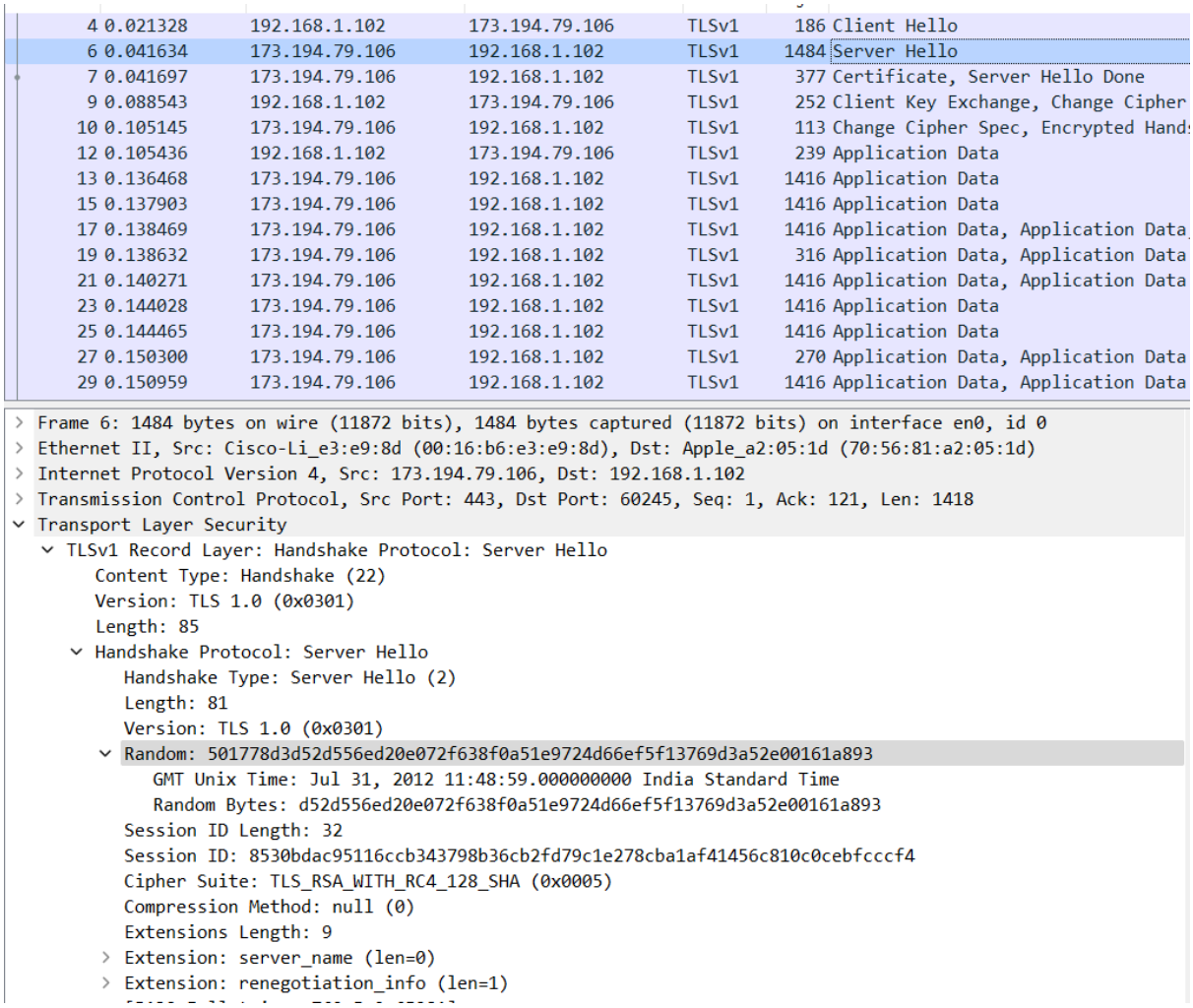
1. How long is the random data in the Hellos? Both the Client and Server include this random data (a nonce) to allow the establishment of session keys.

Ans:

Client:



Server:



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
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Hand
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
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

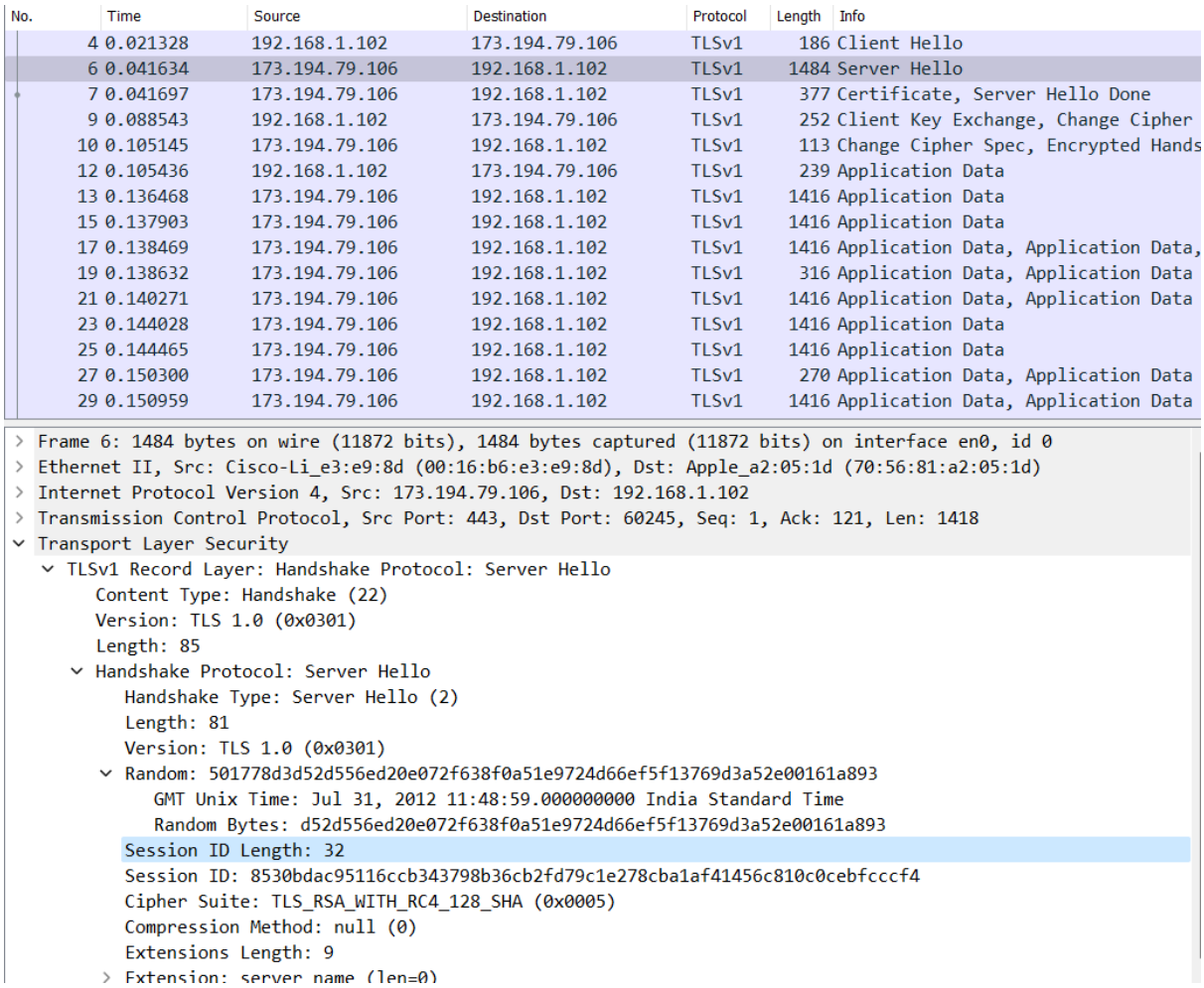
```
> Frame 6: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits) on interface en0, id 0
> Ethernet II, Src: Cisco-Li_e3:e9:8d (00:16:b6:e3:e9:8d), Dst: Apple_a2:05:1d (70:56:81:a2:05:1d)
> Internet Protocol Version 4, Src: 173.194.79.106, Dst: 192.168.1.102
> Transmission Control Protocol, Src Port: 443, Dst Port: 60245, Seq: 1, Ack: 121, Len: 1418
v Transport Layer Security
  v TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 85
    v Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 81
        Version: TLS 1.0 (0x0301)
      v Random: 501778d3d52d556ed20e072f638f0a51e9724d66ef5f13769d3a52e00161a893
          GMT Unix Time: Jul 31, 2012 11:48:59.000000000 India Standard Time
          Random Bytes: d52d556ed20e072f638f0a51e9724d66ef5f13769d3a52e00161a893
        Session ID Length: 32
        Session ID: 8530bdac95116ccb343798b36cb2fd79c1e278cba1af41456c810c0cebfcccf4
        Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
        Compression Method: null (0)
        Extensions Length: 9
      > Extension: server_name (len=0)
      > Extension: renegotiation_info (len=1)
```

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:

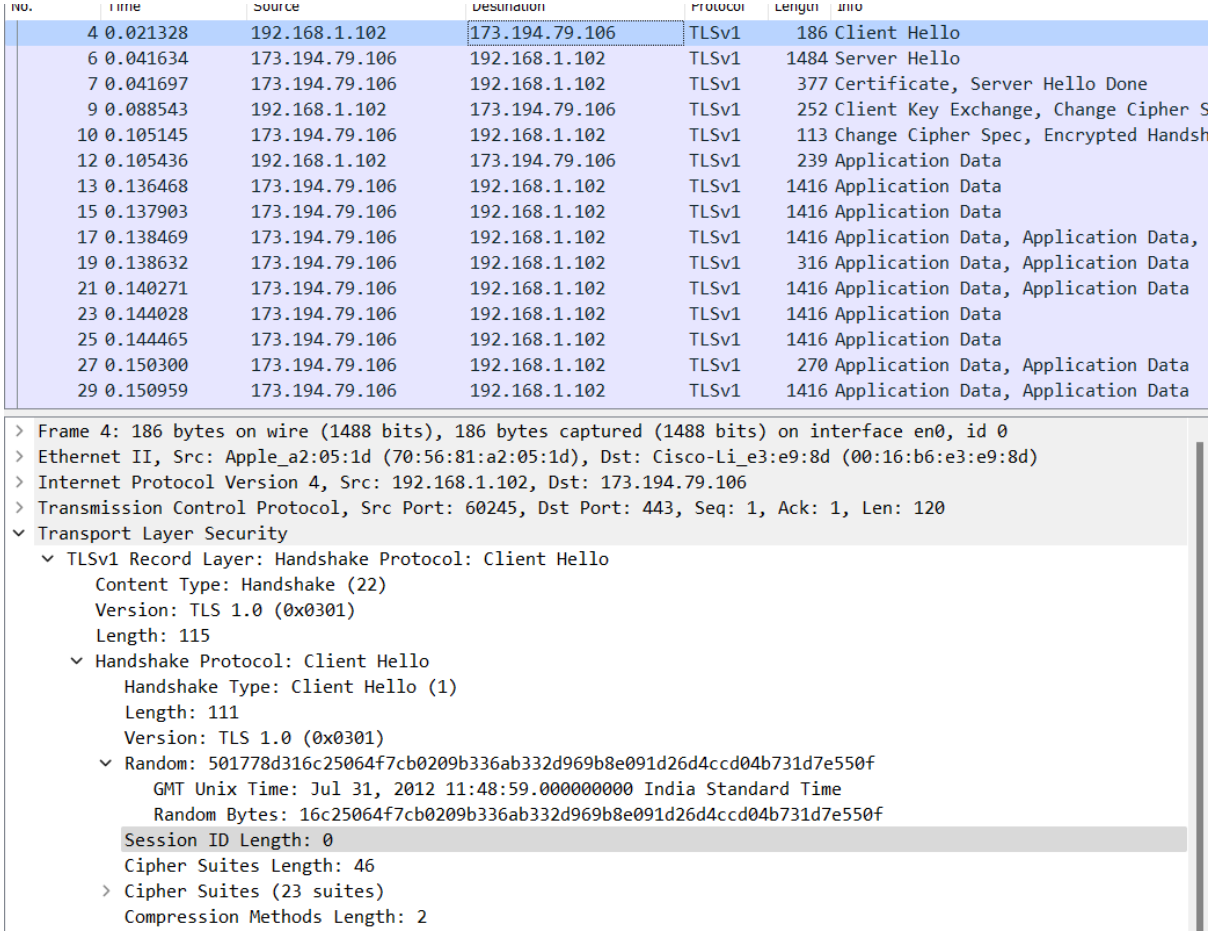
Server:

Length if Session ID is 32



Client:

Length of Session ID is 0



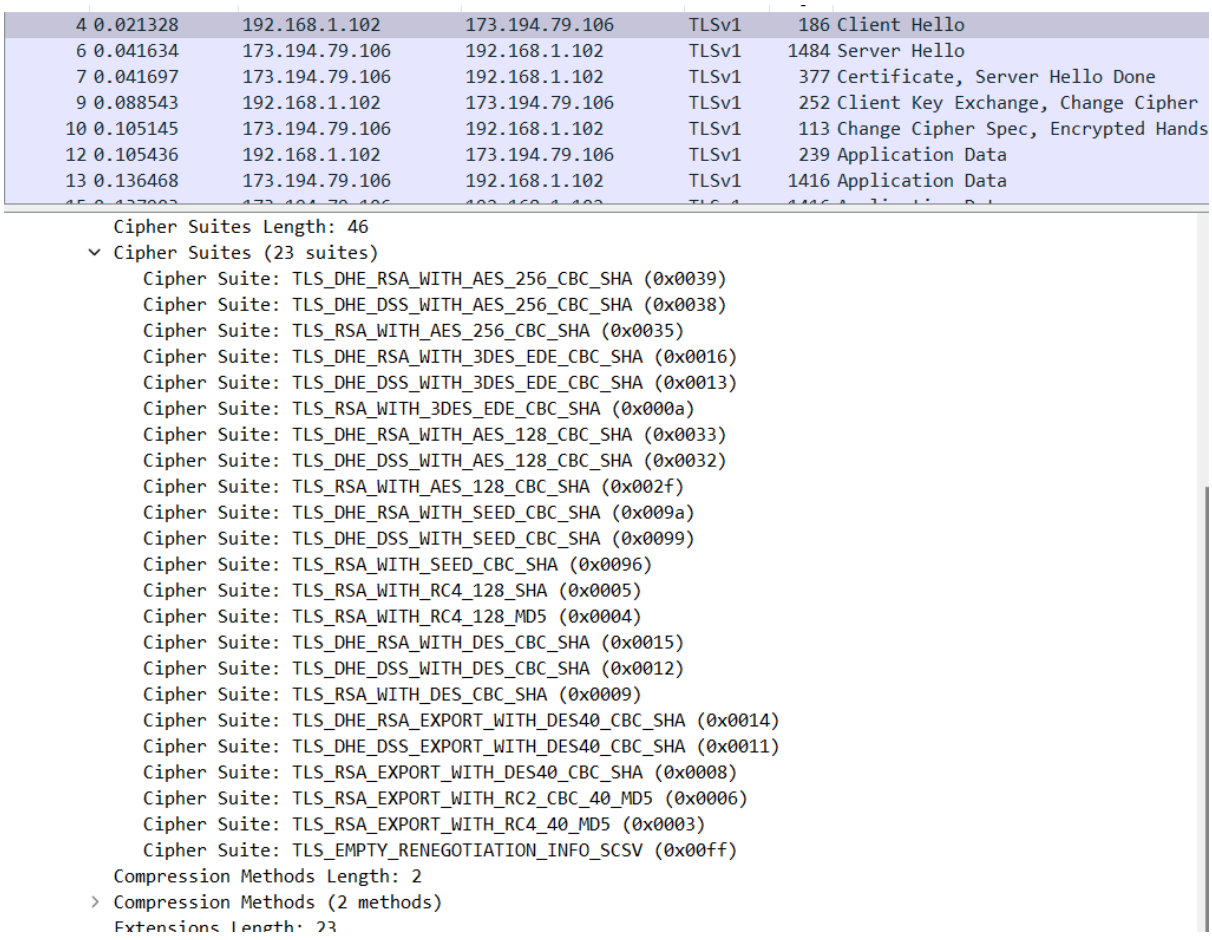
No.	Time	Source	Destination	Protocol	Length	Info
4	0.021328	192.168.1.102	173.194.79.106	TLSv1	186	Client Hello
6	0.041634	173.194.79.106	192.168.1.102	TLSv1	1484	Server Hello
7	0.041697	173.194.79.106	192.168.1.102	TLSv1	377	Certificate, Server Hello Done
9	0.088543	192.168.1.102	173.194.79.106	TLSv1	252	Client Key Exchange, Change Cipher S
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Handsh
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,
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 4: 186 bytes on wire (1488 bits), 186 bytes captured (1488 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: 1, Ack: 1, Len: 120
v Transport Layer Security
  v TLSv1 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 115
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 111
        Version: TLS 1.0 (0x0301)
      v Random: 501778d316c25064f7cb0209b336ab332d969b8e091d26d4ccd04b731d7e550f
          GMT Unix Time: Jul 31, 2012 11:48:59.000000000 India Standard Time
          Random Bytes: 16c25064f7cb0209b336ab332d969b8e091d26d4ccd04b731d7e550f
        Session ID Length: 0
        Cipher Suites Length: 46
      > Cipher Suites (23 suites)
        Compression Methods Length: 2
```

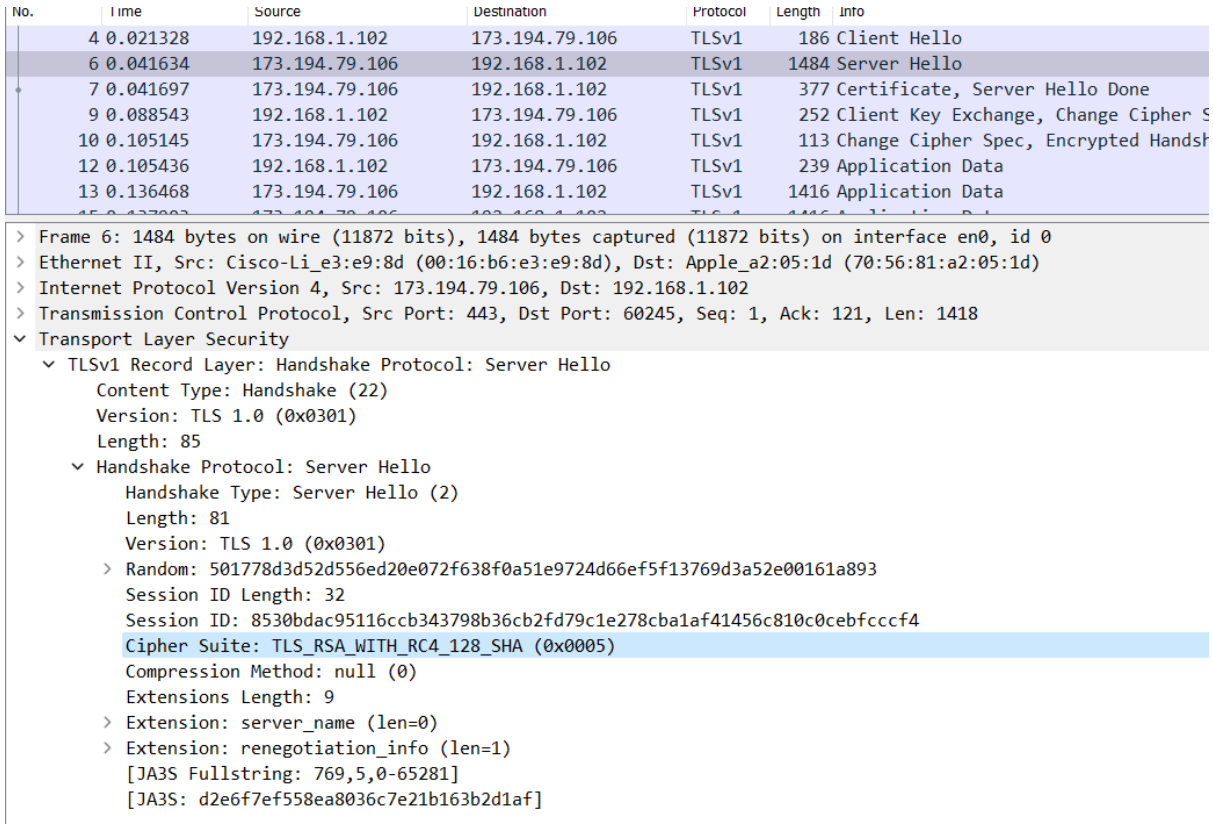
3. What Cipher suite is chosen by the Server? Give its name and value. The Client will list the different cipher methods it supports, and the Server will pick one of these methods to use.

Ans:

Client:



Server:



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 S
10	0.105145	173.194.79.106	192.168.1.102	TLSv1	113	Change Cipher Spec, Encrypted Handsh
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

```
> Frame 6: 1484 bytes on wire (11872 bits), 1484 bytes captured (11872 bits) on interface en0, id 0
> Ethernet II, Src: Cisco-Li_e3:e9:8d (00:16:b6:e3:e9:8d), Dst: Apple_a2:05:1d (70:56:81:a2:05:1d)
> Internet Protocol Version 4, Src: 173.194.79.106, Dst: 192.168.1.102
> Transmission Control Protocol, Src Port: 443, Dst Port: 60245, Seq: 1, Ack: 121, Len: 1418
v Transport Layer Security
  v TLSv1 Record Layer: Handshake Protocol: Server Hello
      Content Type: Handshake (22)
      Version: TLS 1.0 (0x0301)
      Length: 85
    v Handshake Protocol: Server Hello
        Handshake Type: Server Hello (2)
        Length: 81
        Version: TLS 1.0 (0x0301)
      > Random: 501778d3d52d556ed20e072f638f0a51e9724d66ef5f13769d3a52e00161a893
        Session ID Length: 32
        Session ID: 8530bdac95116ccb343798b36cb2fd79c1e278cba1af41456c810c0cebfcccf4
        Cipher Suite: TLS_RSA_WITH_RC4_128_SHA (0x0005)
        Compression Method: null (0)
        Extensions Length: 9
      > Extension: server_name (len=0)
      > Extension: renegotiation_info (len=1)
        [JA3S Fullstring: 769,5,0-65281]
        [JA3S: d2e6f7ef558ea8036c7e21b163b2d1af]
```

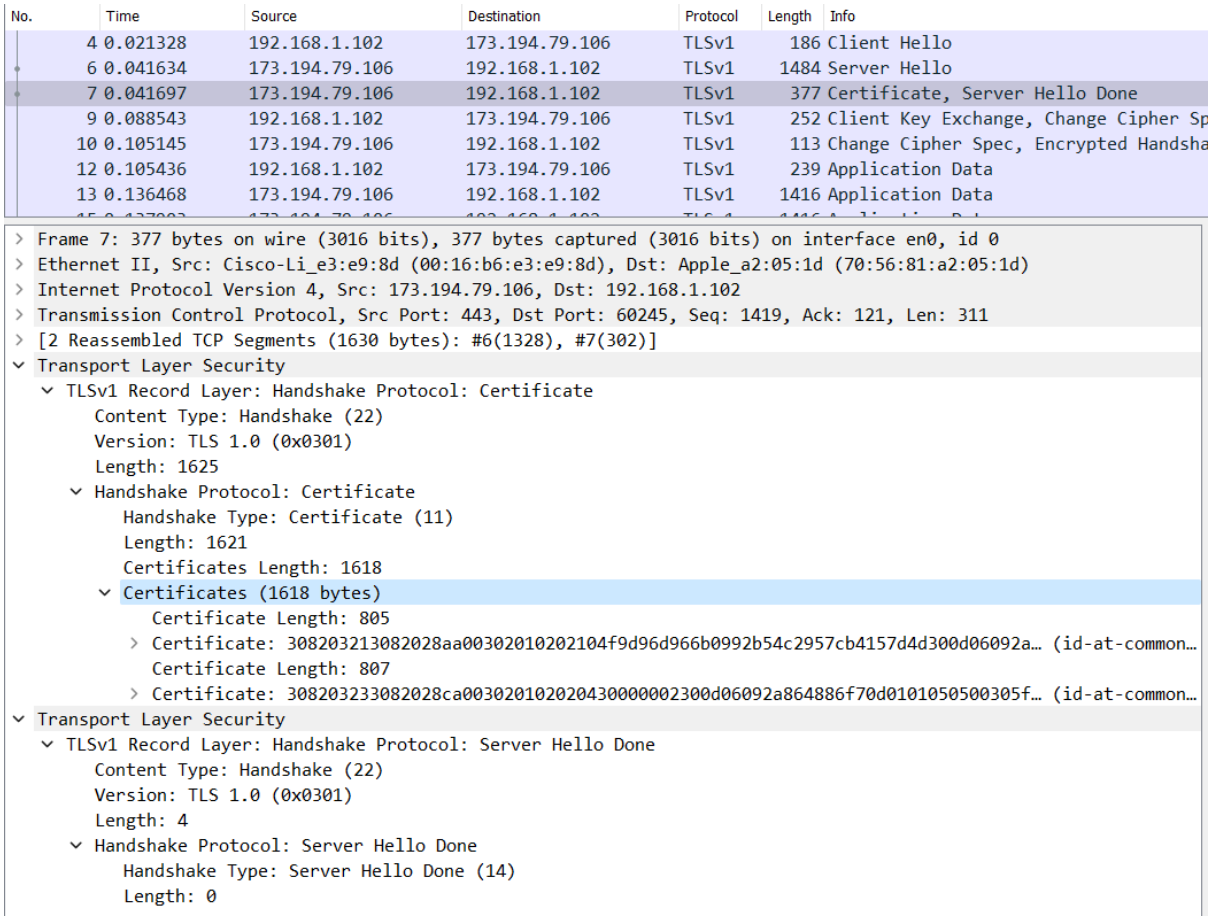
Certificate Messages:

Next, find and inspect the details of the Certificate message, including expanding the Handshake protocol block within the TLS Record. As with the Hellos, the contents of the Certificate message are visible because an encryption scheme is not yet established. It should come after the Hello 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:

The Server sends Certificate to the client



A Certificate message will contain one or more certificates, as needed for one party to verify the identity of the other party from its roots of trust certificates. You can inspect those certificates in your browser.

Client Key Exchange and Change Cipher Messages

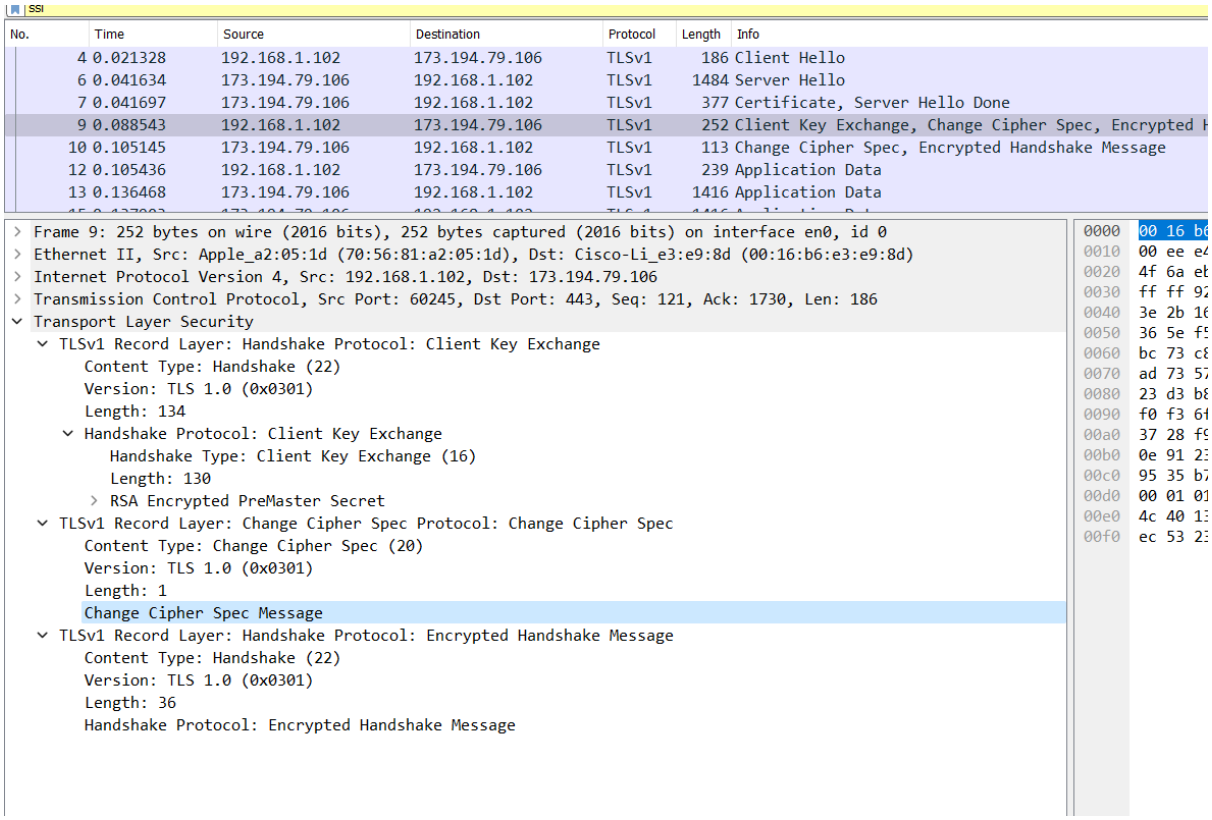
Find and inspect the details of the Client Key Exchange and Change Cipher messages, expanding their various details. The key exchange message is sent to pass keying information so that both sides will have the same secret session key. The change cipher message signal a switch to a new encryption scheme to the other party. This means that it is the last unencrypted message sent by the party.

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

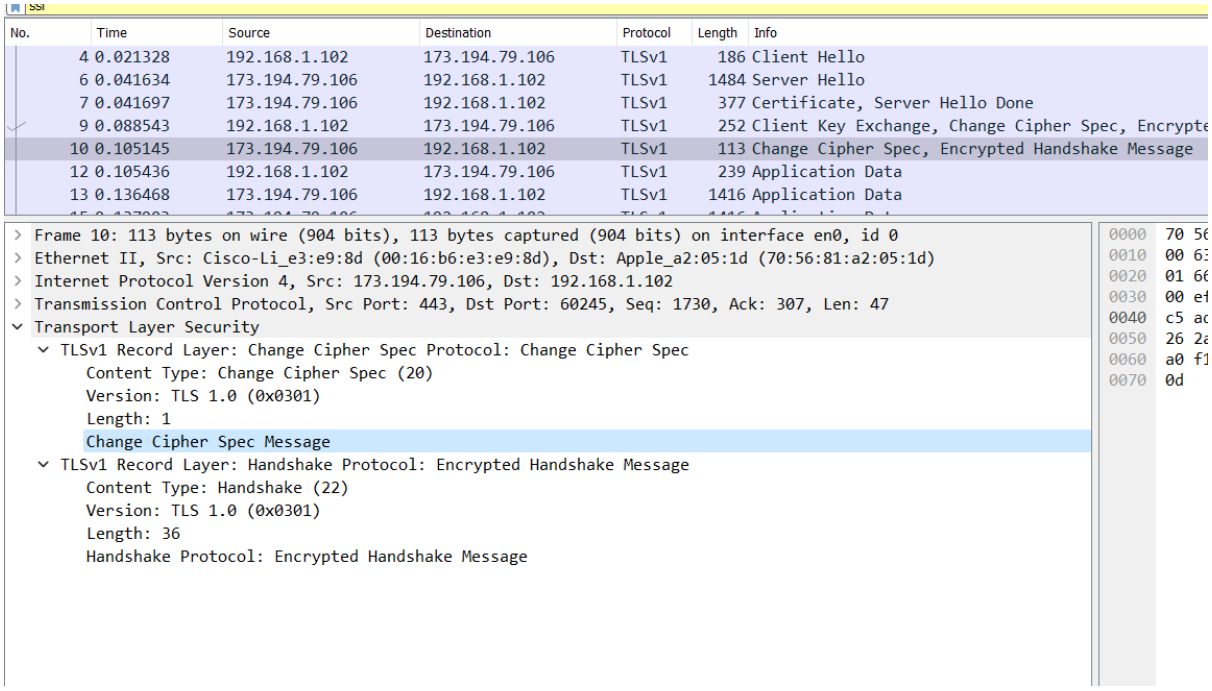
Ans:

Both the server and the client sends the Change Cipher Spec Message

Client:

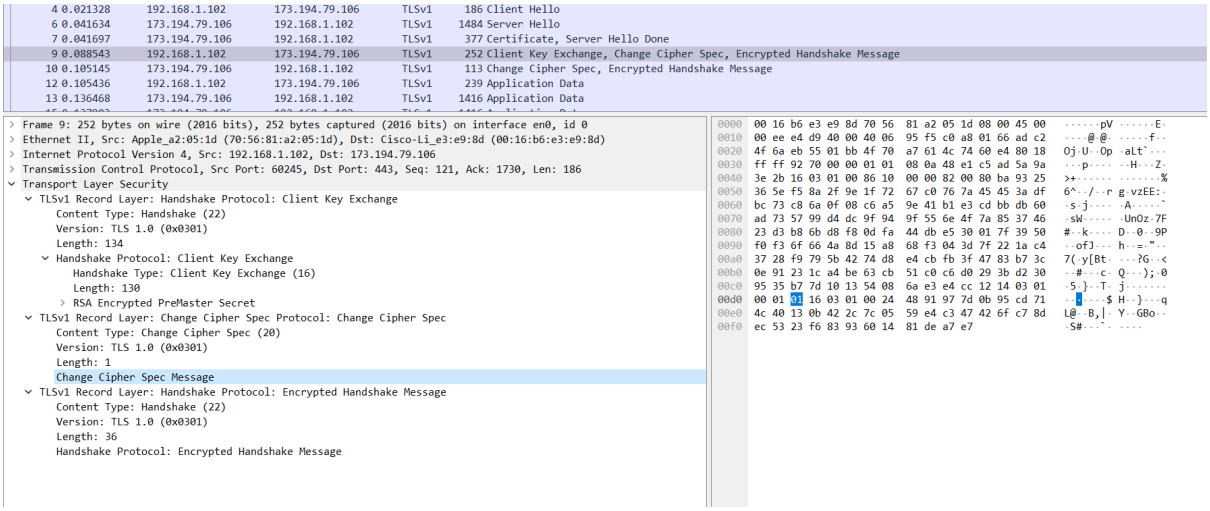


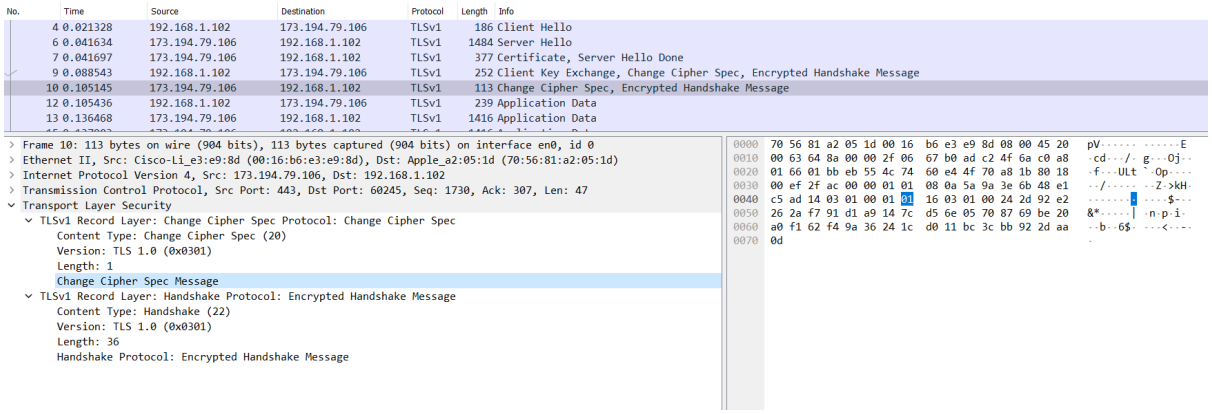
Server:



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

Ans:





Conclusion:

Performed the experiment successfully.

Wireshark is used to analyse the packets of various protocols such as TCP, UDP, SSL, TLS, etc.

SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 1

Caesar Cipher

Caesar Cipher:

The Caesar cipher is one of the earliest known and simplest ciphers. It is a type of substitution cipher in which each letter in the plaintext is 'shifted' a certain number of places down the alphabet. For example, with a shift of 1, A would be replaced by B, B would become C, and so on. The method is named after Julius Caesar, who apparently used it to communicate with his generals.

$$e(x) = (x + k) \pmod{26}$$

(Encryption Phase with shift n)

$$e(x) = (x - k) \pmod{26}$$

(Decryption Phase with shift n)

Code:

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

int main()
{
    int k;
    string s, x;
```

```
    getline(cin, s);
    for (int i = 0; i < s.length(); i++)
        if (s[i] != ' ')
            x += s[i];
    s = x;

    cin >> k;

    cout << "\nPlain text is: " << s << endl;
    cout << "key is: " << k << endl;

    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] >= 'a' and s[i] <= 'z')
            s[i] = (s[i] - 'a' + k + 26) % 26 + 'a';
        if (s[i] >= 'A' and s[i] <= 'Z')
            s[i] = (s[i] - 'A' + k + 26) % 26 + 'A';
    }

    cout << "\nCipher text is: " << s;

    for (int i = 0; i < s.length(); i++)
    {
        if (s[i] >= 'a' and s[i] <= 'z')
            s[i] = (s[i] - 'a' - k + 26) % 26 + 'a';
        if (s[i] >= 'A' and s[i] <= 'Z')
            s[i] = (s[i] - 'A' - k + 26) % 26 + 'A';

    }

    cout << "\n\nPlain text after decryption is: " << s;

    return 0;
}
```

Output:

inputf.in

```
1  Sweety Shrawan Gupta
2  3
```

outputf.in

```
1
2  Plain text is: SweetyShrawanGupta
3  key is: 3
4
5  Cipher text is: VzhhwbVkudzdqJxswd
6
7  Plain text after decription is: SweetyShrawanGupta
```

SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 2
Cryptanalysis

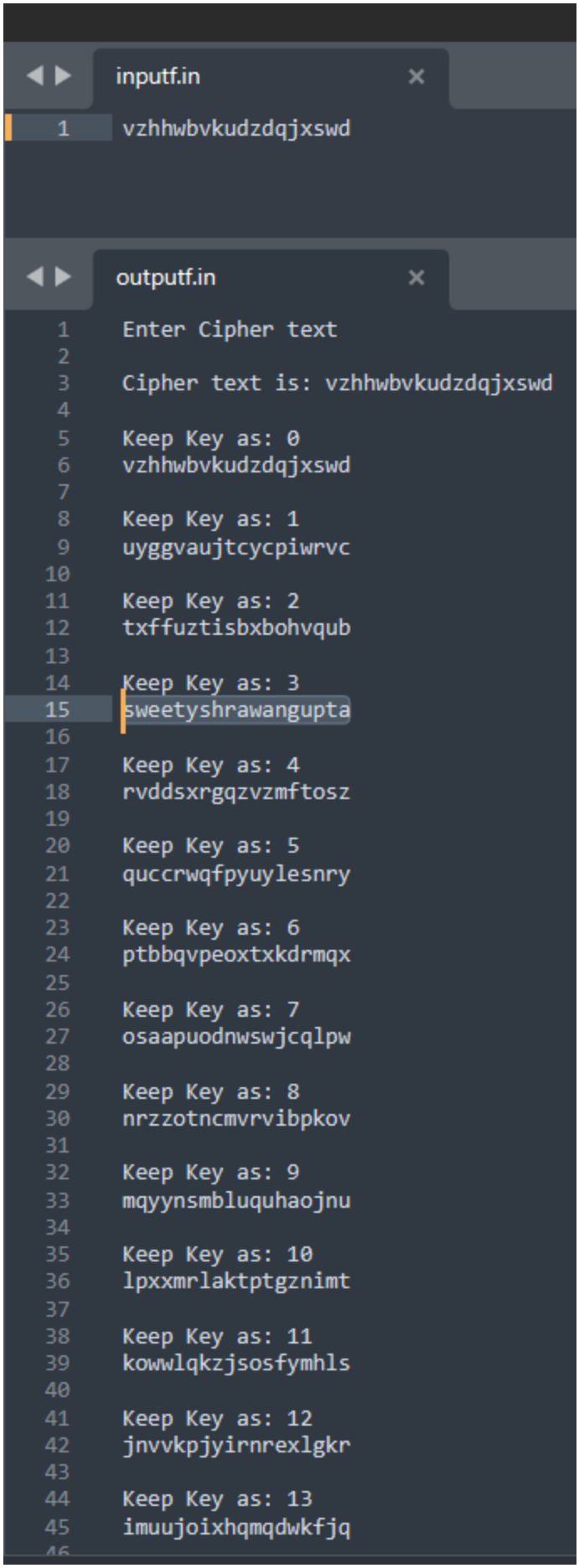
Cryptanalysis is the study of ciphertext, ciphers and cryptosystems with the aim of understanding how they work and finding and improving techniques for defeating or weakening them.

Code:

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


int main()
{

    string s, org;
    cout << "Enter Cipher text" << endl;
    getline(cin, s);

    string x;
    for (int i = 0; i < s.length(); i++)
        if (s[i] != ' ')
            x += s[i];
    s = x;

    int k = 0;

    cout << "\nCipher text is: " << s << endl << endl;

    org = s;
    for (int k = 0; k < 26; k++)
    {
        cout << "Keep Key as: " << k << endl;
        s = org;
```

```
        for (int i = 0; i < s.length(); i++)
        {

            int val = s[i] - 'a';
            val = (val - k + 26) % 26;
            char ch = 'a' + val;
            s[i] = ch;
        }
        cout << s << endl << endl;
    }

    return 0;
}
```

Output:


inputf.in
1 vzhhwbvkudzdqjxswd
outputf.in
1 Enter Cipher text
2
3 Cipher text is: vzhhwbvkudzdqjxswd
4
5 Keep Key as: 0
6 vzhhwbvkudzdqjxswd
7
8 Keep Key as: 1
9 uyggvaujtcycpiwrvc
10
11 Keep Key as: 2
12 txffuztisbxbohvqub
13
14 Keep Key as: 3
15 sweetyshrawangupta
16
17 Keep Key as: 4
18 rvddsxrgqzvzmftosz
19
20 Keep Key as: 5
21 quccrwqfpyuylesnry
22
23 Keep Key as: 6
24 ptbbqvpeoxtxkdrmqx
25
26 Keep Key as: 7
27 osaapuodnwswjcqlpw
28
29 Keep Key as: 8
30 nrzzotncmvrvibpkov
31
32 Keep Key as: 9
33 mqyynsmbluquhaojnu
34
35 Keep Key as: 10
36 lpxxmrlaktptgznimt
37
38 Keep Key as: 11
39 kowwlqkzjsosfymhls
40
41 Keep Key as: 12
42 jnvvkpjyirnrexlgkr
43
44 Keep Key as: 13
45 imuujoixhqmqdwkfjq

SEM - VII - 2022-23
CNS Lab
B3 - 2019BTECS00094 - Sweety Shrawan Gupta
Assignment 3
Playfair Cipher

Playfair Cipher :

The Playfair cipher encryption technique can be used to encrypt or encode a message. It operates exactly like typical encryption. The only difference is that it encrypts a digraph, or a pair of two letters, as opposed to a single letter.

An initial 5×5 matrix key table is created. The plaintext encryption key is made out of the matrix's alphabetic characters. Be mindful that you shouldn't repeat the letters. There are 26 alphabets, however, there are only 25 spaces in which we can place a letter. The matrix will delete the extra letter because there is an excess of one letter (typically J). Despite this, J is there in the plaintext before being changed to I.

Rules for Encryption:

- **If both the letters are in the same column**: Take the letter below each one (going back to the top if at the bottom).
- **If both the letters are in the same row**: Take the letter to the right of each one (going back to the leftmost if at the rightmost position).
- **If neither of the above rules is true**: Form a rectangle with the two letters and take the letters on the horizontal opposite corner of the rectangle.

Code:

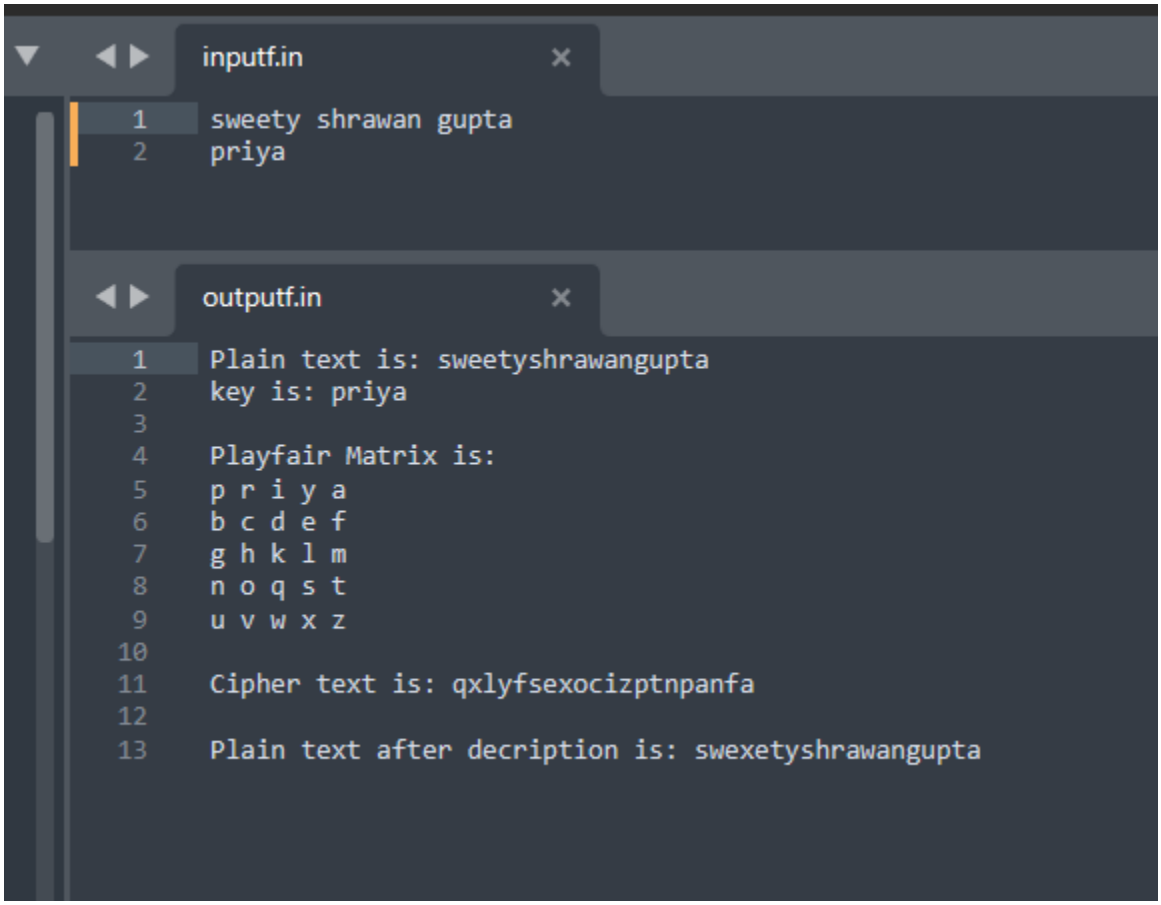
```
#include <bits/stdc++.h>
using namespace std;

int main()
{
    string pt, x, k, ct;
    getline(cin, pt);
    for (int i = 0; i < pt.length(); i++)
        if (pt[i] != ' ')
            x += pt[i];
    pt = x;

    cin >> k;
    cout << "Plain text is: " << pt << endl;
    cout << "key is: " << k << endl;

    map<char, pair<int, int>> char_index;
    vector<vector<char>> mat(5, vector<char>(5, '#'));
    int i = 0, j = 0, l = 0;

    while (i < 5 and l < k.size()) {
        if (k[l] == 'j')k[l] = 'i';
        if (char_index.find(k[l]) == char_index.end()) {
            mat[i][j] = k[l];
            char_index[k[l]] = {i, j};
            j++;
            if (j == 5) {
                i++; j = 0;
            }
        }
        l++;
    }

    for (char ch = 'a'; i < 5 and ch <= 'z'; ch++) {
        if (ch == 'j')ch++;
        if (char_index.find(ch) == char_index.end()) {
```

```
            mat[i][j] = ch;
            char_index[ch] = {i, j};
            j++;
            if (j == 5) {
                i++; j = 0;
            }
        }
    }
    cout << "\nPlayfair Matrix is: \n";
    for (int ii = 0; ii < 5; ii++) {
        for (int jj = 0; jj < 5; jj++) {
            cout << mat[ii][jj] << " ";
        }
        cout << "\n";
    }

    for (int i = 1; i < pt.size(); i++) {
        if (i % 2 and pt[i - 1] == pt[i]) {
            pt.insert(i, "x");
        }
    }
    if (pt.size() % 2)pt.push_back('z');

    cout << "\n\nPlain text after decription is: " << pt;

    for (int i = 1; i < pt.size(); i += 2) {
        int fi, fj, si, sj;
        fi = char_index[pt[i - 1]].first;
        fj = char_index[pt[i - 1]].second;
        si = char_index[pt[i]].first;
        sj = char_index[pt[i]].second;
        if (si == fi) {
            sj++;
            fj++;
            if (fj == 5)fj = 0;
            if (sj == 5)sj = 0;
            ct.push_back(mat[fi][fj]);
            ct.push_back(mat[si][sj]);
        } else if (sj == fj) {
            si++;
```

```
            fi++;
            if (fi == 5)fi = 0;
            if (si == 5)si = 0;
            ct.push_back(mat[fi][fj]);
            ct.push_back(mat[si][sj]);
        } else {
            ct.push_back(mat[fi][sj]);
            ct.push_back(mat[si][fj]);
        }
    }

    cout << "\nCipher text is: " << ct;

    pt = "";

    for (int i = 1; i < ct.size(); i += 2) {
        int fi, fj, si, sj;
        fi = char_index[ct[i - 1]].first;
        fj = char_index[ct[i - 1]].second;
        si = char_index[ct[i]].first;
        sj = char_index[ct[i]].second;
        if (si == fi) {
            sj--;
            fj--;
            if (fj == -1)fj = 4;
            if (sj == -1)sj = 4;
            pt.push_back(mat[fi][fj]);
            pt.push_back(mat[si][sj]);
        } else if (sj == fj) {
            si--;
            fi--;
            if (fi == -1)fi = 4;
            if (si == -1)si = 4;
            pt.push_back(mat[fi][fj]);
            pt.push_back(mat[si][sj]);
        } else {
            pt.push_back(mat[fi][sj]);
            pt.push_back(mat[si][fj]);
        }
    }
```

```
    if (pt.back() == 'z')pt.pop_back();
    for (int i = 2; i < pt.size(); i++) {
        if (pt[i - 2] == pt[i] and pt[i - 1] == 'x') {
            pt.erase(i - 1, 1);
        }
    }

    cout << "\n\nPlain text after decryption is: " << pt;

    return 0;
}
```

Output:



SEM - VII - 2022-23
CNS Lab
B3 - 2019BTECS00094 - Sweety Shrawan Gupta
Assignment 4
Vigenere Cipher

Vigenere Cipher

- 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.
- In vigenere cipher,the encryption and decryption are done by Vigenere algebraically formula in this method (convert the letters (A-Z) into the numbers (0-25)).

Formula of encryption is,

$$E_i = (P_i + K_i) \bmod 26$$

Formula of decryption is,

$$D_i = (E_i - K_i) \bmod 26$$

If any case (D_i) value becomes negative (-ve), in this case, we will add 26 in the negative value.

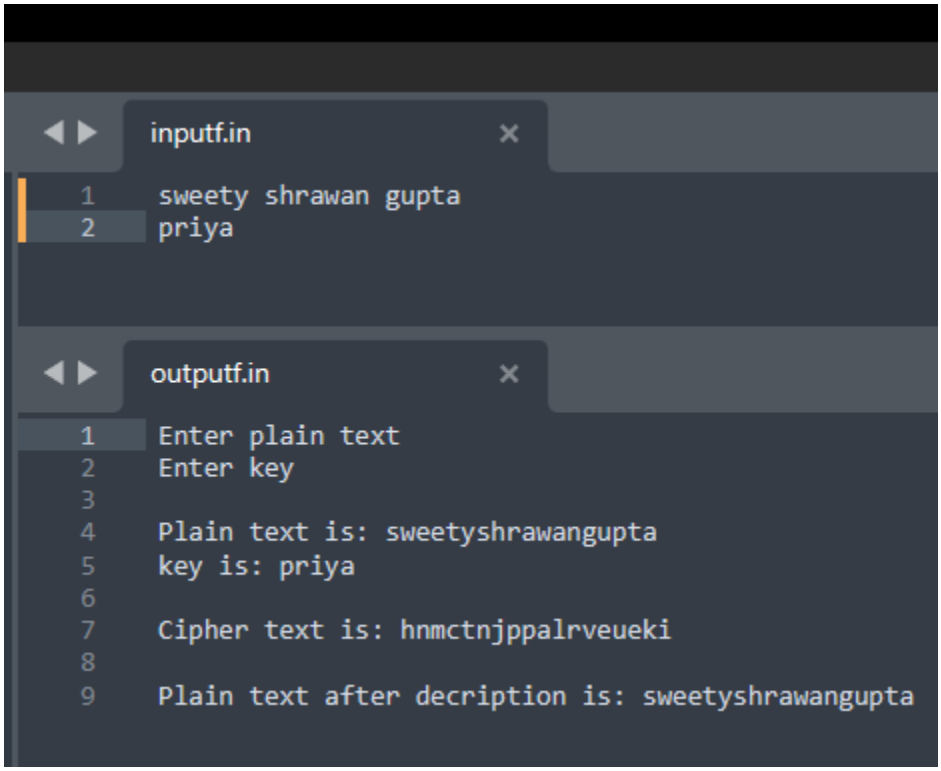
Code:

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

int main()
{
    string s, x, k;
    cout << "Enter plain text" << endl;
```

```
getline(cin, s);
for (int i = 0; i < s.length(); i++)
    if (s[i] != ' ')
        x += s[i];
s = x;

cout << "Enter key" << endl;
cin >> k;

cout << "\nPlain text is: " << s << endl;
cout << "key is: " << k << endl;

int j = 0;
for (int i = 0; i < s.length(); i++)
{
    if (s[i] >= 'a' and s[i] <= 'z')
        s[i] = (s[i] - 'a' + k[j] - 'a'  + 26) % 26 + 'a';
    if (s[i] >= 'A' and s[i] <= 'Z')
        s[i] = (s[i] - 'A' + k[j] - 'a'  + 26) % 26 + 'A';

    j++;
    if (j >= k.size())j = 0;
}

cout << "\nCipher text is: " << s;
j = 0 ;
for (int i = 0; i < s.length(); i++)
{
    if (s[i] >= 'a' and s[i] <= 'z')
        s[i] = (s[i] - 'a' - (k[j] - 'a') + 26) % 26 + 'a';
    if (s[i] >= 'A' and s[i] <= 'Z')
        s[i] = (s[i] - 'A' - (k[j] - 'a') + 26) % 26 + 'A';


    j++;
    if (j >= k.size())j = 0;
}

cout << "\n\nPlain text after decryption is: " << s;
```

```
    return 0;
}
```

Output:



SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 5

Tranposition Ciphers

a)Columnar Cipher

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

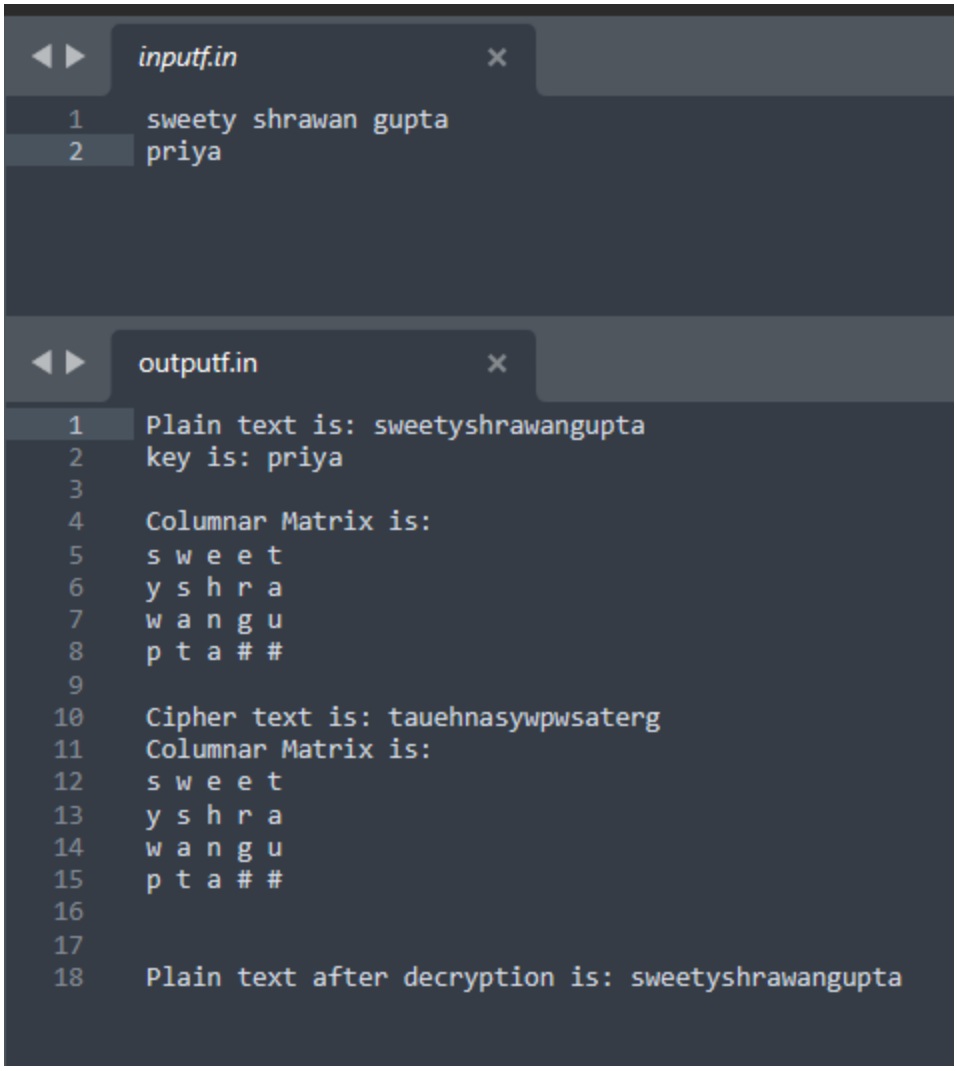
Code:

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

int main()
{
    string pt, x, k, ct;
    getline(cin, pt);
    //removing the spaces
    for (int i = 0; i < pt.length(); i++)
        if (pt[i] != ' ')
            x += pt[i];
    pt = x;

    cin >> k;
    cout << "Plain text is: " << pt << endl;
    cout << "key is: " << k << endl;
    int m = k.size(); //no. of columns
    int n = pt.size() / m + (pt.size() % m != 0); //no. of rows
    x = k;
    sort(x.begin(), x.end()); //sorting the key
    map<char, int> ma;
    for (int ii = 0; ii < m; ii++) {
```

```
        ma[k[ii]] = ii;
    }

//making a matrix and filling a plain text char row-wise
    vector<vector<char>> mat(n, vector<char>(m, '#'));
    int i = 0, j = 0, l = 0;
    while (i < n and l < pt.size()) {
        mat[i][j] = pt[l];
        j++;
        if (j == m) {
            i++; j = 0;
        }
        l++;
    }

    cout << "\nColumnar Matrix is: \n";
    for (int ii = 0; ii < n; ii++) {
        for (int jj = 0; jj < m; jj++) {
            cout << mat[ii][jj] << " ";
        }
        cout << "\n";
    }
//getting a index of letter/(column no. to read first) using a sorted key
    for (auto ch : x) {
        int jj = ma[ch];
        for (int ii = 0; ii < n and mat[ii][jj] != '#'; ii++) {
            ct += mat[ii][jj];
        }
    }

    cout << "\nCipher text is: " << ct;

    int rem = ct.size() % m; //finding no. of column field to be filled in
last row
    pt = "";
    for (int ii = 0; ii < n; ii++) {
        for (int jj = 0; jj < m; jj++) {
            mat[ii][jj] = '#';
        }
    }
```

```
    l = 0;
    for (auto ch : x) {
        int jj = ma[ch];
        int cnt = (jj < rem) ? n : n - 1; //finding no. of row to be
filled of that column no.
        for (int ii = 0; ii < n and cnt>0 and l < ct.size(); ii++) {
            mat[ii][jj] = ct[l];
            cnt--;
            l++;
        }
    }

    cout << "\nColumnar Matrix is: \n";
    for (int ii = 0; ii < n; ii++) {
        for (int jj = 0; jj < m; jj++) {
            if (mat[ii][jj] != '#')pt += mat[ii][jj];
            cout << mat[ii][jj] << " ";
        }
        cout << "\n";
    }

    cout << "\n\nPlain text after decryption is: " << pt;

    return 0;
}
```

Output:



b)Railfence Cipher

- Plaintext is written downwards and diagonally on rails and then read as row-wise
- Each letter is written in a zigzag pattern.
- Keyless transposition cipher

Code:

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

```
int main()
{
    string pt, x, ct;
    getline(cin, pt);
    for (int i = 0; i < pt.length(); i++)
        if (pt[i] != ' ')
            x += pt[i];
    pt = x;

    int n;
    cin >> n;
    int m = pt.size();
    int div = 2 * (n - 1);
    int bogus = 0;
    if (m % div > 0)bogus = div - (m % div);
    while (bogus > 0) {
        pt += 'z';
        bogus--;
    }
    cout << "Plain text is: " << pt << endl;
    cout << "depth is: " << n << endl;
    m = pt.size();
    vector<vector<char>> mat(n, vector<char>(m, '#'));
    int i = 0, j = 0, l = 0, inc = 1;

    while (l < pt.size()) {
        mat[i][j] = pt[l];
        i += inc; j++;
        if (n == 1)i = 0;
        if (i == n - 1) {
            inc = -1;
        }
        if (i == 0) {
            inc = 1;
        }
        l++;
    }
    cout << "\nColumnar Matrix is: \n";
    for (int ii = 0; ii < n; ii++) {
```

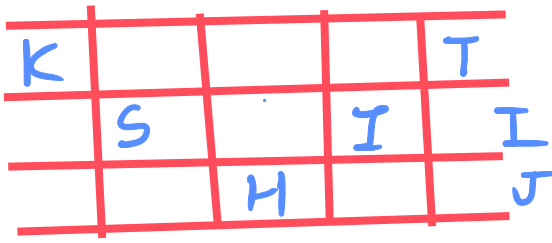
```
        for (int jj = 0; jj < m; jj++) {
            if (mat[ii][jj] != '#')ct += mat[ii][jj];
            cout << mat[ii][jj] << " ";
        }
        cout << "\n";
    }

    cout << "\nCipher text is: " << ct << "\n";


    return 0;
}
```

Output:

Railfence

Kshitij = 7

d = 3



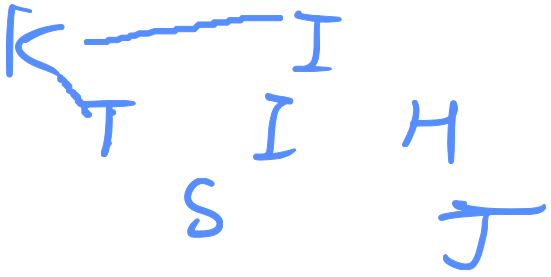
inputf.in | columnar.cpp

```
sweety shrawan gupta
3
```

outputf.in

```
Plain text is: sweetyshrawanguptazz
depth is: 3

Columnar Matrix is:
s # # # t # # # r # # # n # # # t # # #
# w # e # y # h # a # a # g # p # a # z
# # e # # # s # # # w # # # u # # # z #

Cipher text is: strntweyhaagpazeswuz
```

KT S I I HJ



depth = no. of row

length of pt = no. of column

SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 6

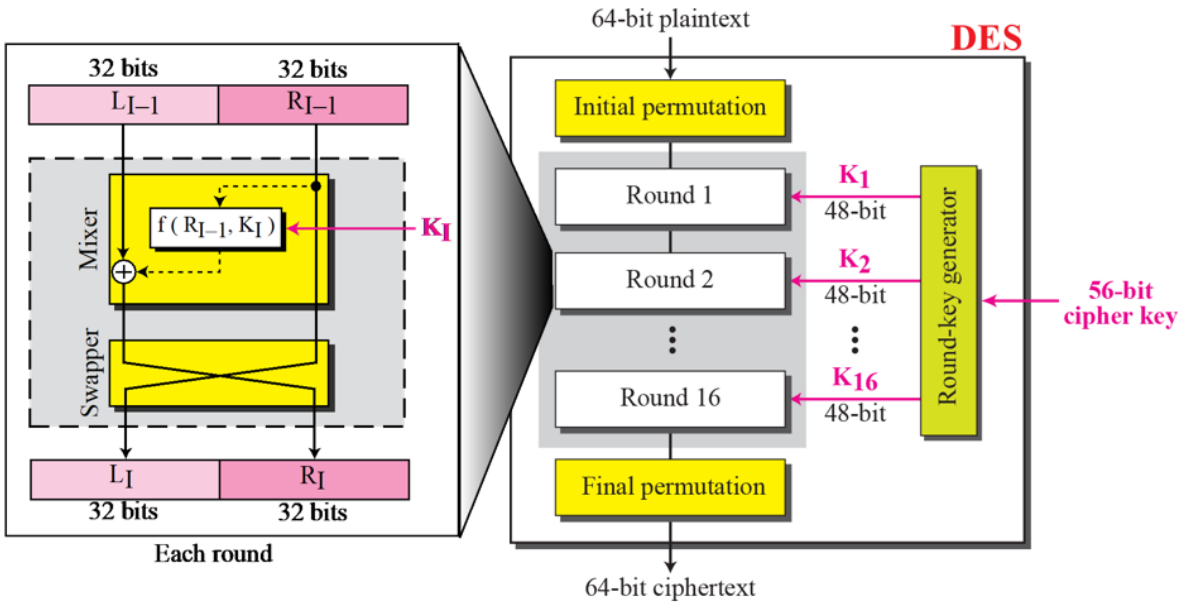
DES

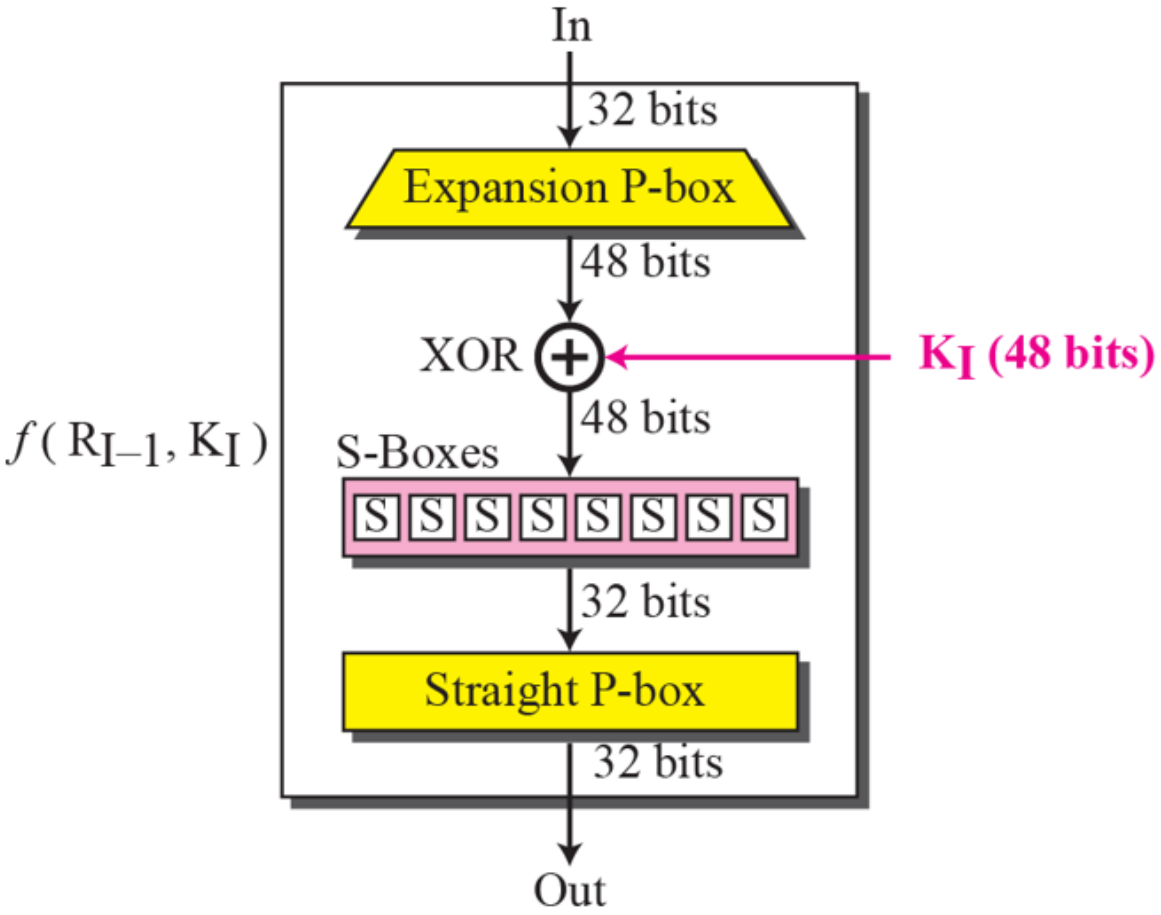
DES - Data Encryption Standard

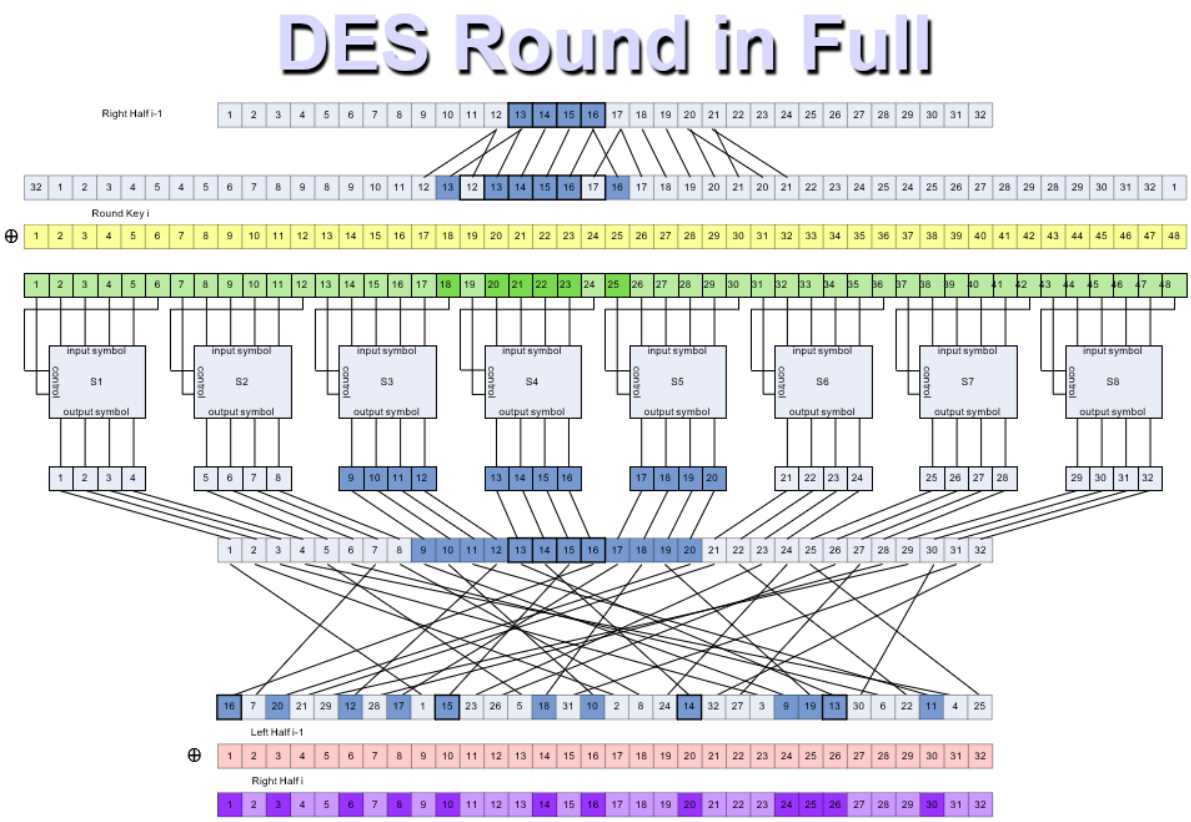
Theory:

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**.

- 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.


64-bit plaintext
DES
32 bits
32 bits
L_{I-1}
R_{I-1}
Mixer
$f(R_{I-1}, K_I)$
K_I
Swapper
L_I
R_I
32 bits
32 bits
Each round
Initial permutation
Round 1
Round 2
Round 16
Final permutation
K_1
48-bit
K_2
48-bit
K_{16}
48-bit
Round-key generator
56-bit cipher key
64-bit ciphertext


In
32 bits
Expansion P-box
48 bits
XOR
K_I (48 bits)
48 bits
$f(R_{I-1}, K_I)$
S-Boxes
S S S S S S S S
32 bits
Straight P-box
32 bits
Out



Code:

```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
string hex2bin(string s)
{
// hexadecimal to binary conversion
    map<char, string> mp;
    mp['0'] = "0000";
    mp['1'] = "0001";
    mp['2'] = "0010";
    mp['3'] = "0011";
    mp['4'] = "0100";
    mp['5'] = "0101";
    mp['6'] = "0110";
    mp['7'] = "0111";
    mp['8'] = "1000";
    mp['9'] = "1001";
```

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

```
    }
    return hex;
}

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

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

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

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

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

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

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

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

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

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

// Straight Permutation Table
```

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

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

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

        // S-boxes
        string op = "";
        for (int i = 0; i < 8; i++) {
            int row = 2 * int(x[i * 6] - '0') + int(x[i * 6 + 5] - '0');
            int col = 8 * int(x[i * 6 + 1] - '0') + 4 * int(x[i * 6 + 2] -
'0') + 2 * int(x[i * 6 + 3] - '0') + int(x[i * 6 + 4] - '0');
            int val = s[i][row][col];
            op += char(val / 8 + '0');
            val = val % 8;
            op += char(val / 4 + '0');
            val = val % 4;
            op += char(val / 2 + '0');
            val = val % 2;
            op += char(val + '0');
        }
        // Straight D-box
        op = permute(op, per, 32);

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

        left = x;
```

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

// Combination
    string combine = left + right;

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

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

    pt = "123456ABCD1325F7";
    key = "1CBB0918273CCDD1";
// Key Generation
```

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

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

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

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

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

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

    vector<string> rkb; // rkb for RoundKeys in binary
```

```
    vector<string> rk; // rk for RoundKeys in hexadecimal
    for (int i = 0; i < 16; i++) {
        // Shifting
        left = shift_left(left, shift_table[i]);
        right = shift_left(right, shift_table[i]);

        // Combining
        string combine = left + right;

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

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

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

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

Output:

outputf.in

```
Plain text:123456ABCD1325F7

Encryption:

After initial permutation: 94A7D6F898CA18AD
After splitting: L0=94A7D6F8 R0=98CA18AD

Round 1 98CA18AD E98FF09F 1944C074D6A8
Round 2 E98FF09F 4CDE422D 4528581AFC5C
Round 3 4CDE422D CB006489 06ECA069D5B4
Round 4 CB006489 D7D55F5E DA2D02896CAB
Round 5 D7D55F5E 12E9DECD 68A609EE5A15
Round 6 12E9DECD 9520EBB2 41940E9343FE
Round 7 9520EBB2 C4354D7F 6088D2959B81
Round 8 C4354D7F D82AC081 34E822D22675
Round 9 D82AC081 EEB57875 849B44F1D0C7
Round 10 EEB57875 170A32C8 02724706A6AF
Round 11 170A32C8 5EC71E48 295560BE3DC5
Round 12 5EC71E48 335A2EC9 C041E92AC3F3
Round 13 335A2EC9 17E7FA69 91C31157ED03
Round 14 17E7FA69 FDE5C7B9 051B83EE0558
Round 15 FDE5C7B9 5434F99B 3330C5C9F34E
Round 16 1E362632 5434F99B 081C1D4D8F69

Cipher Text: 0A57F44AFB3D880A
```

```
Decryption

After initial permutation: 1E3626325434F99B
After splitting: L0=1E362632 R0=5434F99B

Round 1 5434F99B FDE5C7B9 081C1D4D8F69
Round 2 FDE5C7B9 17E7FA69 3330C5C9F34E
Round 3 17E7FA69 335A2EC9 051B83EE0558
Round 4 335A2EC9 5EC71E48 91C31157ED03
Round 5 5EC71E48 170A32C8 C041E92AC3F3
Round 6 170A32C8 EEB57875 295560BE3DC5
Round 7 EEB57875 D82AC081 02724706A6AF
Round 8 D82AC081 C4354D7F 849B44F1D0C7
Round 9 C4354D7F 9520EBB2 34E822D22675
Round 10 9520EBB2 12E9DECD 6088D2959B81
Round 11 12E9DECD D7D55F5E 41940E9343FE
Round 12 D7D55F5E CB006489 68A609EE5A15
Round 13 CB006489 4CDE422D DA2D02896CAB
Round 14 4CDE422D E98FF09F 06ECA069D5B4
Round 15 E98FF09F 98CA18AD 4528581AFC5C
Round 16 94A7D6F8 98CA18AD 1944C074D6A8

Plain Text: 123456ABCD1325F7
```

SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 7

AES - Advanced Encryption Standard

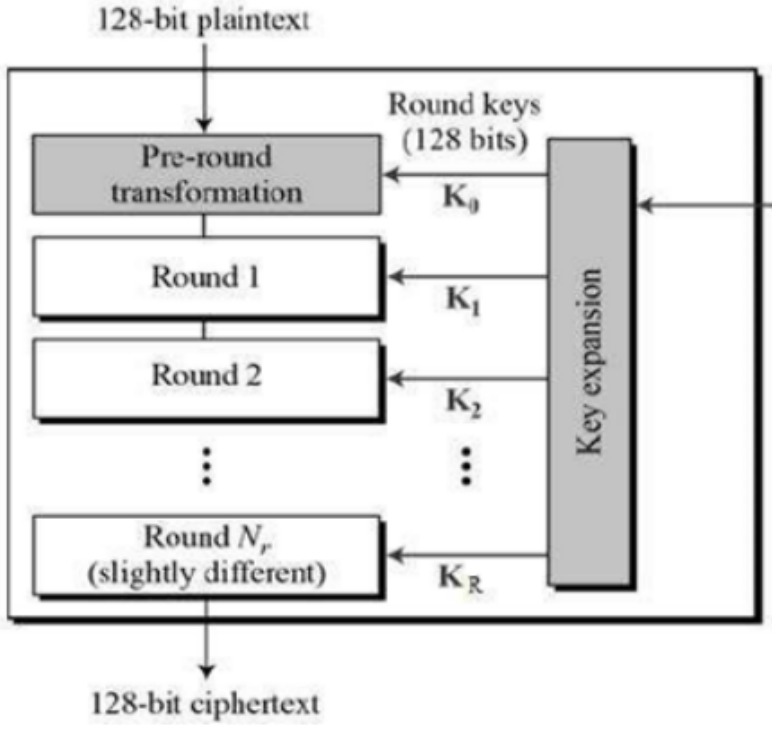
Objective: To study and implement encryption and decryption using AES

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.

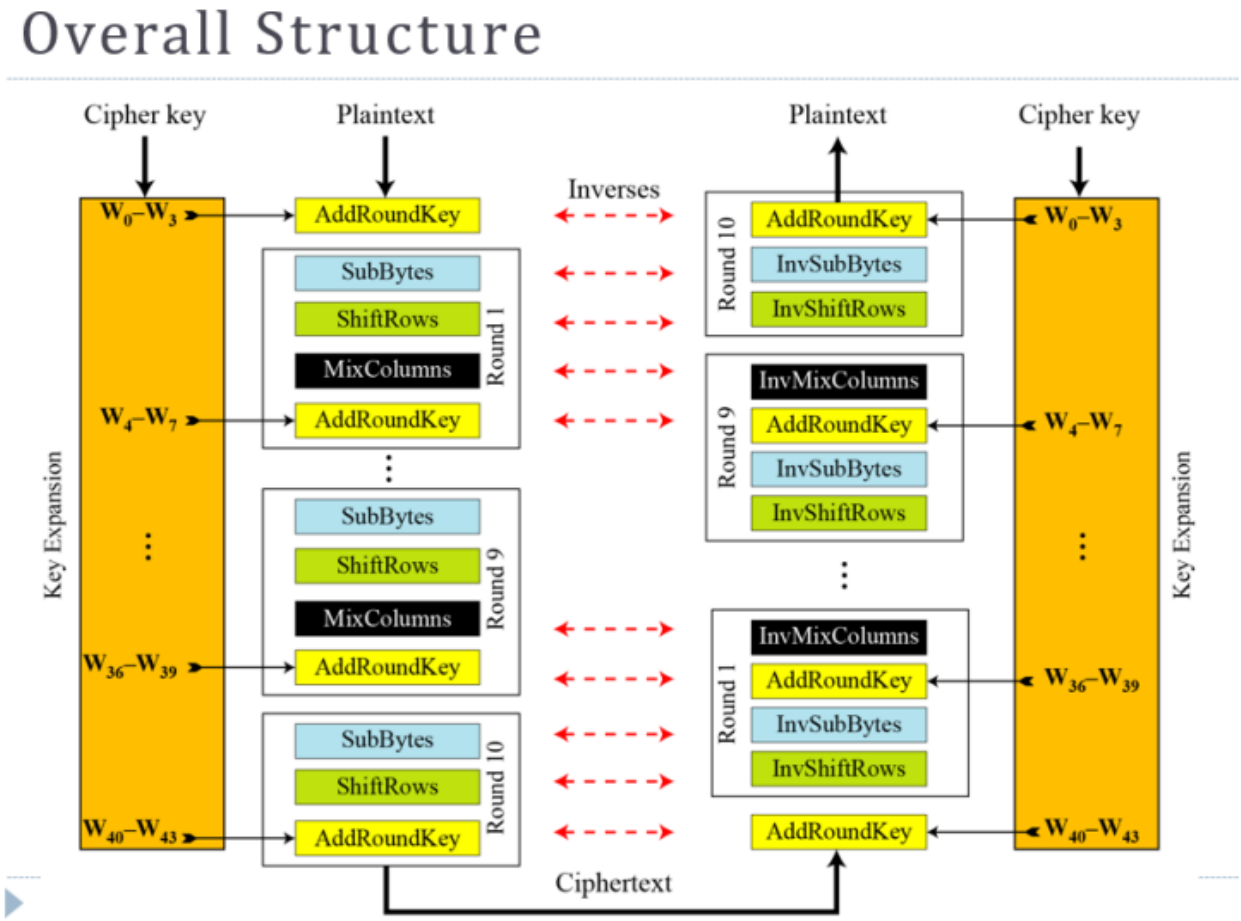
AES is a block cipher. The key size can be 128/192/256 bits. It encrypts data in blocks of 128 bits each. That means it takes 128 bits as input and outputs 128 bits of encrypted cipher text as output. AES relies on substitution-permutation network principle which means it is performed using a series of linked operations which involves replacing and shuffling of the input data.

The schematic of AES structure



R	Key size
10	128
12	192
14	256

Relationship between number of rounds(R) and cipher key size



Code:

```
/*
 * Advanced Encryption Standard
 * @author Dani Huertas
 * @email huertas.dani@gmail.com
 *
 * Based on the document FIPS PUB 197
 */
#include <stdio.h>
```

```c
#include "aes.h"

int main() {

    uint8_t i;

    /*
     * Appendix A - Key Expansion Examples
     */

    /* 128 bits */
    /* uint8_t key[] = {
         0x2b, 0x7e, 0x15, 0x16,
         0x28, 0xae, 0xd2, 0xa6,
         0xab, 0xf7, 0x15, 0x88,
         0x09, 0xcf, 0x4f, 0x3c}; */

    /* 192 bits */
    /* uint8_t key[] = {
         0x8e, 0x73, 0xb0, 0xf7,
         0xda, 0x0e, 0x64, 0x52,
         0xc8, 0x10, 0xf3, 0x2b,
         0x80, 0x90, 0x79, 0xe5,
         0x62, 0xf8, 0xea, 0xd2,
         0x52, 0x2c, 0x6b, 0x7b}; */

    /* 256 bits */
    /* uint8_t key[] = {
         0x60, 0x3d, 0xeb, 0x10,
         0x15, 0xca, 0x71, 0xbe,
         0x2b, 0x73, 0xae, 0xf0,
         0x85, 0x7d, 0x77, 0x81,
         0x1f, 0x35, 0x2c, 0x07,
         0x3b, 0x61, 0x08, 0xd7,
         0x2d, 0x98, 0x10, 0xa3,
         0x09, 0x14, 0xdf, 0xf4};
    */

    /* uint8_t in[] = {
         0x32, 0x43, 0xf6, 0xa8,
```

```
        0x88, 0x5a, 0x30, 0x8d,
        0x31, 0x31, 0x98, 0xa2,
        0xe0, 0x37, 0x07, 0x34}; // 128
    */

    /*
     * Appendix C - Example Vectors
     */

    /* 128 bit key */
    /* uint8_t key[] = {
        0x00, 0x01, 0x02, 0x03,
        0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b,
        0x0c, 0x0d, 0x0e, 0x0f}; */

    /* 192 bit key */
    /* uint8_t key[] = {
        0x00, 0x01, 0x02, 0x03,
        0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b,
        0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13,
        0x14, 0x15, 0x16, 0x17}; */

    /* 256 bit key */
    uint8_t key[] = {
        0x00, 0x01, 0x02, 0x03,
        0x04, 0x05, 0x06, 0x07,
        0x08, 0x09, 0x0a, 0x0b,
        0x0c, 0x0d, 0x0e, 0x0f,
        0x10, 0x11, 0x12, 0x13,
        0x14, 0x15, 0x16, 0x17,
        0x18, 0x19, 0x1a, 0x1b,
        0x1c, 0x1d, 0x1e, 0x1f};

    uint8_t in[] = {
        0x00, 0x11, 0x22, 0x33,
        0x44, 0x55, 0x66, 0x77,
        0x88, 0x99, 0xaa, 0xbb,
```

```
        0xcc, 0xdd, 0xee, 0xff};

    uint8_t out[16]; // 128

    uint8_t *w; // expanded key

    w = aes_init(sizeof(key));

    aes_key_expansion(key, w);

    printf("Plaintext message:\n");
    for (i = 0; i < 4; i++) {
        printf("%02x %02x %02x %02x ", in[4*i+0], in[4*i+1], in[4*i+2],
in[4*i+3]);
    }

    printf("\n");

    aes_cipher(in /* in */, out /* out */, w /* expanded key */);

    printf("Ciphered message:\n");
    for (i = 0; i < 4; i++) {
        printf("%02x %02x %02x %02x ", out[4*i+0], out[4*i+1], out[4*i+2],
out[4*i+3]);
    }

    printf("\n");

    aes_inv_cipher(out, in, w);

    printf("Original message (after inv cipher):\n");
    for (i = 0; i < 4; i++) {
        printf("%02x %02x %02x %02x ", in[4*i+0], in[4*i+1], in[4*i+2],
in[4*i+3]);
    }

    printf("\n");

    free(w);
```

```
    return 0;
}
```

Output:

```
d:\CNS Lab\AES>aes.exe
Plaintext message:
00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
Ciphered message:
8e a2 b7 ca 51 67 45 bf ea fc 49 90 4b 49 60 89
Original message (after inv cipher):
00 11 22 33 44 55 66 77 88 99 aa bb cc dd ee ff
```

SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 8

Euclidean,Extended Euclidean,Multiplicative Inverse

Theory:-

Euclidean

The **Euclidean Algorithm** is a technique for quickly finding the **GCD** of two integers.

The Algorithm

The Euclidean Algorithm for finding GCD(A,B) is as follows:

- If A = 0 then GCD(A,B)=B, since the GCD(0,B)=B, and we can stop.
- If B = 0 then GCD(A,B)=A, since the GCD(A,0)=A, and we can stop.
- Write A in quotient remainder form (A = B·Q + R)
- Find GCD(B,R) using the Euclidean Algorithm since GCD(A,B) = GCD(B,R)

Extended Euclidean

In arithmetic and computer programming, the extended Euclidean algorithm is an extension to the Euclidean algorithm, and computes, in addition to the greatest common divisor (gcd) of integers a and b, also the coefficients of Bézout's identity, which are integers x and y such that

Multiplicative Inverse

A modular multiplicative inverse of an integer a is an integer x such that $a \cdot x$ is congruent to 1 modular some modulus m. To write it in a formal way: we want to find an integer x so that

$$a \cdot x \equiv 1 \mod m.$$

We will also denote x simply with a^{-1}.

We should note that the modular inverse does not always exist. For example, let $m = 4$, $a = 2$. By checking all possible values modulo m is should become clear that we cannot find a^{-1} satisfying the above equation. It can be proven that the modular inverse exists if and only if a and m are relatively prime (i.e. $\gcd(a, m) = 1$).

We use Extended Euclidean Algorithm to find the solution.

Euclidean

Code:

Cpp:

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

int main()
{

    int q, r1, r2, r;
    cin >> r1 >> r2;
    cout << r1 << " " << r2 << "\n";
    while (r2 > 0)
    {
        q = r1 / r2;
        r = r1 - r2 * q;
        r1 = r2;
        r2 = r;
    }
    cout << r1 << "\n";

    return 0;
}
```

```
d:\CNS Lab>cd "d:\CNS Lab\" && g++ euclidean.cpp -o euclidean && "d:\CNS Lab\"euclidean
4 8
4 8
4
```

Python:

```python
def euclidean(a: int, b: int):
  r1 = a
  r2 = b

  print(f'{"q"} {"r1":>20} {"r2":>20} {"r":>20}')
  print("")

  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2
    print(f'{q} {"%20.0d" % r1} {"%20.0d" % r2} {"%20.0d" % r}')

    r1 = r2
    r2 = r

  print(f'{"-"} {"%20.0d" % r1} {"%20.0d" % r2} {"-":>20}')

  print(f'GCD({a}, {b}) = {r1}')




def main():

  a = int(input('Enter a: '))
  b = int(input('Enter b: '))
  euclidean(a,b)




main()
```

Output:

```
In [1]: runfile('D:/CNS Lab/euclidean.py', wdir='D:/CNS Lab')

Enter a: 56738901534

Enter b: 45778901246
q                   r1                   r2                    r

1          56738901534          45778901246          10960000288
4          45778901246          10960000288           1938900094
5          10960000288           1938900094           1265499818
1           1938900094           1265499818            673400276
1           1265499818            673400276            592099542
1            673400276            592099542             81300734
7            592099542             81300734             22994404
3             81300734             22994404             12317522
1             22994404             12317522             10676882
1             12317522             10676882              1640640
6             10676882              1640640               833042
1              1640640               833042               807598
1               833042               807598                25444
31               807598                25444                18834
1                 25444                18834                 6610
2                 18834                 6610                 5614
1                  6610                 5614                  996
5                  5614                  996                  634
1                   996                  634                  362
1                   634                  362                  272
1                   362                  272                   90
3                   272                   90                    2
45                   90                    2                    0
-                     2                    0                    -
GCD(56738901534, 45778901246) = 2
```

```
In [4]: runfile('D:/CNS Lab/euclidean.py', wdir='D:/CNS Lab')

Enter a: 07512779912565014928652150296195898826835317022090

Enter b: 97569354727979857921728118865439213176360669242153
q                  r1                  r2                  r

0 75127799125650149286521502961958988268353170220907 97569354727979857921728118865439213176360669242153 75127799125650149286521502961958988268353170220907
12 97569354727979857921728118865439213176360669242153 75127799125650149286521502961958988268353170220907 74159957771996787779023153110884272543368649770737
1 75127799125650149286521502961958988268353170220907 74159957771996787779023153110884272543368649770737 96784135365336150749834985107471572498452045017
76 74159957771996787779023153110884272543368649770737 96784135365336150749834985107471572498452045017 60401489434131320914856442920587744454509555781
1 96784135365336150749834985107471572498452045017 60401489434131320914856442920587744454509555781 36382645931204829834978542186883828043942489236
1 60401489434131320914856442920587744454509555781 36382645931204829834978542186883828043942489236 24018843502926491079877900733703916410567066545
1 36382645931204829834978542186883828043942489236 24018843502926491079877900733703916410567066545 12363802428278338755100641453179911633375422691
1 24018843502926491079877900733703916410567066545 12363802428278338755100641453179911633375422691 11655041074648152324777259280524004777191643854
1 12363802428278338755100641453179911633375422691 11655041074648152324777259280524004777191643854 70876135363018643032338217265590685618377883 7
16 11655041074648152324777259280524004777191643854 70876135363018643032338217265590685618377883 7 31485941656516943960314451802949507825118246 2
2 70876135363018643032338217265590685618377883 7 31485941656516943960314451802949507825118246 2 79042520499847551117093136596916699681413913
3 31485941656516943960314451802949507825118246 2 79042520499847551117093136596916699681413913 7 77318550656267862518651082387449792069407 23
1 79042520499847551117093136596916699681413913 77731855065626786251865108238744979206940723 13106654342207648652280283581717204744731 90
59 77731855065626786251865108238744979206940723 13106654342207648652280283581717204744731 90 40259444660165920341143510661347121302251 3
3 13106654342207648652280283581717204744731 90 40259444660165920341143510661347121302251 3 10288209441578725499372303833130683540565 1
3 40259444660165920341143510661347121302251 3 10288209441578725499372303833130683540565 1 93948163354297438430265991619550706805560 8
1 10288209441578725499372303833130683540565 1 93948163354297438430265991619550706805560 8 93393106148981656345704671175612860009 1
10 93948163354297438430265991619550706805560 8 93393106148981656345704671175612860009 1 46088527393992727956955245019894208040650 4
1 8933931061489816563457046711756128600091 4608852739399272795695524501989420804650 1 43250783220905437677615222097667077954 41
1 4608852739399272795695524501989420804650 1 43250783220905437677615222097667077954 41 28377441730872902793400229222271300920 9
15 43250783220905437677615222097667077954 41 28377441730872902793400229222271300920 9 68462062459608348751487826426012657306 9
4 28377441730872902793400229222271300920 9 68462062459608348751487826426012657306 9 99261674702956329280509865186623799 8
6 68462062459608348751487826426012657306 9 99261674702956329280509865186623799 8 89050576378345511831819071314038377396 1
1 99261674702956329280509865186623799 8 89050576378345511831819071314038377396 1 10211098324610817448690792046240025 89
8 89050576378345511831819071314038377396 1 10211098324610817448690792046240025 89 73617897814589722422927367704635668 4
1 10211098324610817448690792046240025 89 73617897814589722422927367704635668 4 28493085431518452063980552757764590 5
2 73617897814589722422927367704635668 4 28493085431518452063980552757764590 5 16631726951552818294966262189106487 4
1 28493085431518452063980552757764590 5 16631726951552818294966262189106487 4 11861358479965633769014290568658103 1
```

```
1 1291253980046550917312769649257153 125489770712128157678638924024174 4 3635627292526934052638040901540 9
34 125489770712128157678638924024174 4 3635627292526934052638040901540 9 1878442766212399888945533371783 8
1 3635627292526934052638040901540 9 1878442766212399888945533371783 8 1757184526314534163692507529757 1
1 1878442766212399888945533371783 8 1757184526314534163692507529757 1 1212582398978657252530258420267
14 1757184526314534163692507529757 1 1212582398978657252530258420267 5956916774441401015014574138 3
2 1212582398978657252530258420267 5956916774441401015014574138 3 2119904409037704952734359260 1
28 5956916774441401015014574138 3 2119904409037704952734359260 1 2118442913582714735836821005 1
10 2119904409037704952734359260 1 2118442913582714735836821005 1 1461495454990216897538255 1
144 2118442913582714735836821005 1 1461495454990216897538255 1 1388945839680240338173366 1
1 1461495454990216897538255 1 1388945839680240338173366 1 7254961530997655936488 90
19 1388945839680240338173366 1 7254961530997655936488 90 1050314879068571024047 51
6 7254961530997655936488 90 1050314879068571024047 51 9530722565862297922038 4
1 1050314879068571024047 51 9530722565862297922038 4 9724262248234123184367
9 9530722565862297922038 4 9724262248234123184367 7788865424515870561081
1 9724262248234123184367 7788865424515870561081 1935396823718252623286
4 7788865424515870561081 1935396823718252623286 4727812964286006793 7
40 1935396823718252623286 4727812964286006793 7 4427163800384990581 6
1 4727812964286006793 7 4427163800384990581 6 3006491639010162131
14 4427163800384990581 6 3006491639010162131 2 1807550577076359 7
1 6 3006491639010162131 2 1807550577076359 7 8257365813025261 59
2 2 1807550577076359 7 8257365813025261 59 2928189510258365 4
1 7 8257365813025261 59 2928189510258365 4 2964546861999425 5
1 59 2928189510258365 4 2964546861999425 5 2328272089026411 49
1 4 2964546861999425 5 2328272089026411 49 6362747729730135 6
3 5 2328272089026411 49 6362747729730135 6 4194477701073708 1
1 49 6362747729730135 6 4194477701073708 1 2168270028656427 5
1 6 4194477701073708 1 2168270028656427 5 2026207672417280 6
1 1 2168270028656427 5 2026207672417280 6 142062356239146 9
14 5 2026207672417280 6 142062356239146 9 37334685069224 0
3 6 142062356239146 9 37334685069224 0 3005830103147 49
1 9 37334685069224 0 3005830103147 49 7276384037749 1
4 0 3005830103147 49 7276384037749 1 952764880478 5
7 49 7276384037749 1 952764880478 5 607029874399 6
1 1 952764880478 5 607029874399 6 345735006078 9
1 5 607029874399 6 345735006078 9 261294868320 7
1 6 345735006078 9 261294868320 7 84440137758 2
3 9 261294868320 7 84440137758 2 9744550461
```

```
1       373346850692240       300583010314749       72763840377491
4       300583010314749       72763840377491        9527648804785
7       72763840377491        9527648804785         6070298743996
1       9527648804785         6070298743996         3457350060789
1       6070298743996         3457350060789         2612948683207
1       3457350060789         2612948683207         844401377582
3       2612948683207         844401377582          79744550461
10      844401377582          79744550461           46955872972
1       79744550461           46955872972           32788677489
1       46955872972           32788677489           14167195483
2       32788677489           14167195483           4454286523
3       14167195483           4454286523            804335914
5       4454286523            804335914             432606953
1       804335914             432606953             371728961
1       432606953             371728961             60877992
6       371728961             60877992              6461009
9       60877992              6461009               2728911
2       6461009               2728911               1003187
2       2728911               1003187               722537
1       1003187               722537                280650
2       722537                280650                161237
1       280650                161237                119413
1       161237                119413                41824
2       119413                41824                 35765
1       41824                 35765                 6059
5       35765                 6059                  5470
1       6059                  5470                  589
9       5470                  589                   169
3       589                   169                   82
2       169                   82                    5
16      82                    5                     2
2       5                     2                     1
2       2                     1                     0
-       1                     0                     -
GCD(751277991256501492865215029619589882683531702209O, 97569354727979857921728118865439213176360669242153) = 1
```

Extended Euclidean

Code:

Cpp:

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

int main()
{

    int q, r1, r2, r, s1 = 1, s2 = 0, s, t1 = 0, t2 = 1, t;
    cin >> r1 >> r2;
    cout << r1 << " " << r2 << "\n";
    while (r2 > 0)
    {
        q = r1 / r2;
        r = r1 - r2 * q;
        r1 = r2;
```

```
        r2 = r;
        s = s1 - s2 * q;
        s1 = s2;
        s2 = s;
        t = t1 - t2 * q;
        t1 = t2;
        t2 = t;
    }
    cout << r1 << " " << s1 << " " << t1 << "\n";

    return 0;
}
```

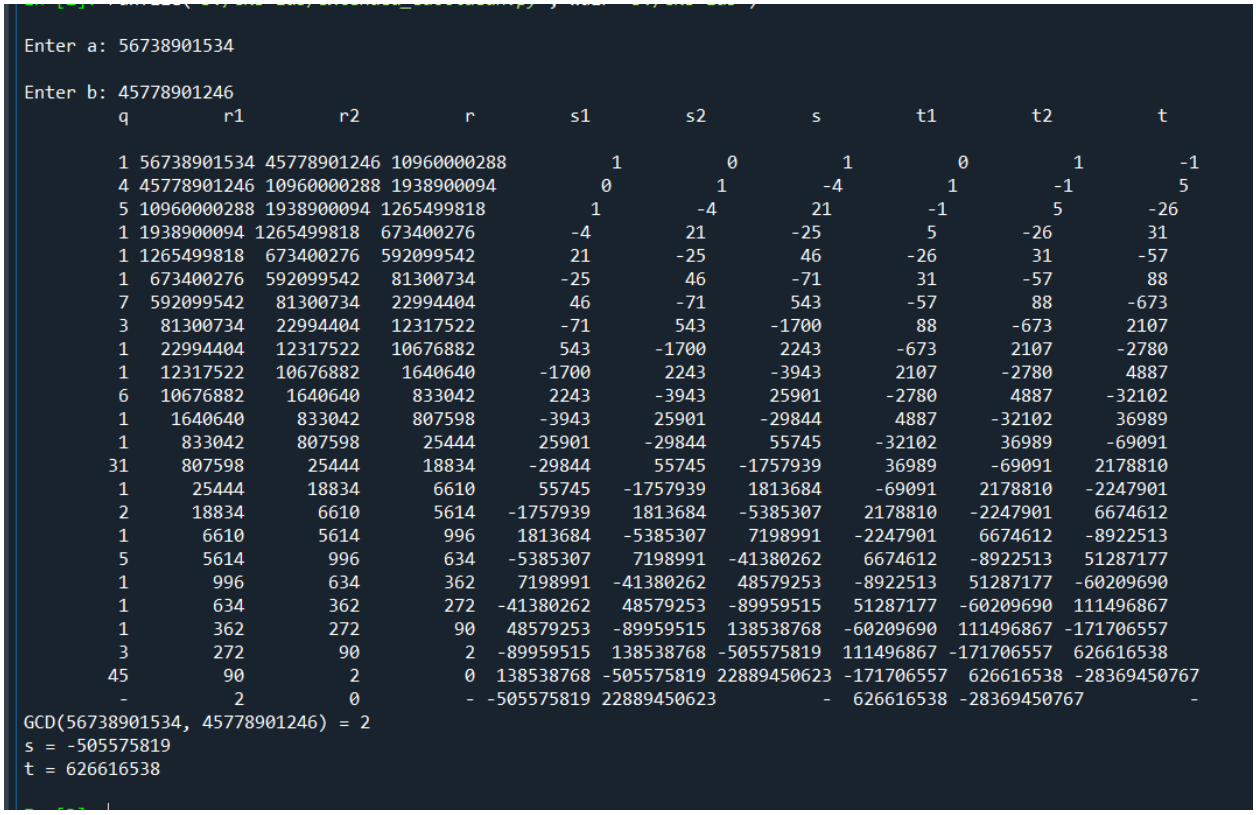
Output:

```
d:\CNS Lab>cd "d:\CNS Lab\" && g++ ext_euclidean.cpp -o ext_euclidean ean && "d:\CNS Lab\"ext_
ean && "d:\CNS Lab\"ext_euclidean
56 32
56 32
8 -1 2
```

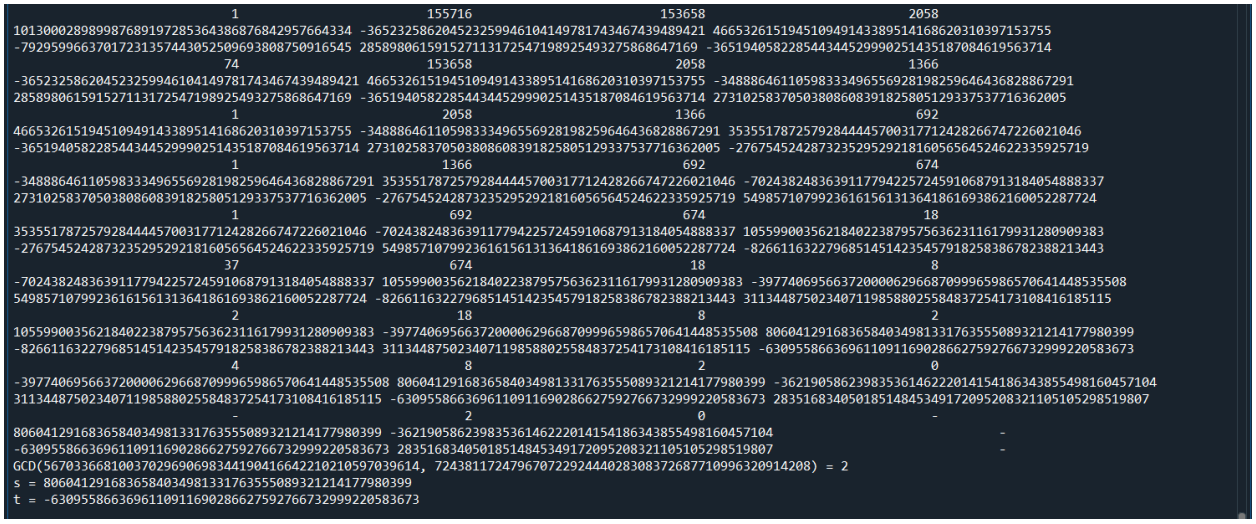
Python:

```
def extEuclidean(a: int, b: int):
  r1 = a
  r2 = b
  s1 = 1
  s2 = 0
  t1 = 0
  t2 = 1

  print(f'{"q":>10} {"r1":>10} {"r2":>10} {"r":>10} {"s1":>10} {"s2":>10}
{"s":>10} {"t1":>10} {"t2":>10} {"t":>10}')
  print(" ")
  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2
    s = s1 - q * s2
    t = t1 - q * t2
```

```
    print(f'{"%10.0d" % q} {"%10.0d" % r1} {"%10.0d" % r2} {"%10.0d" % r}
{"%10.0d" % s1} {"%10.0d" % s2} {"%10.0d" % s} {"%10.0d" % t1} {"%10.0d" %
t2} {"%10.0d" % t}')

    r1 = r2
    r2 = r
    s1 = s2
    s2 = s
    t1 = t2
    t2 = t

  print(f'{"-":>10} {"%10.0d" % r1} {"%10.0d" % r2} {"-":>10} {"%10.0d" %
s1} {"%10.0d" % s2} {"-":>10} {"%10.0d" % t1} {"%10.0d" % t2} {"-":>10}')

  print(f'GCD({a}, {b}) = {r1}\ns = {s1}\nt = {t1}')



def main():

  a = int(input('Enter a: '))
  b = int(input('Enter b: '))
  extEuclidean(a,b)




main()
```



```
Enter a: 56738901534

Enter b: 45778901246
        q         r1         r2          r         s1         s2          s         t1         t2          t

        1 56738901534 45778901246 10960000288          1          0          1          0          1         -1
        4 45778901246 10960000288 1938900094          0          1         -4          1         -1          5
        5 10960000288 1938900094 1265499818          1         -4         21         -1          5        -26
        1 1938900094 1265499818  673400276         -4         21        -25          5        -26         31
        1 1265499818  673400276  592099542         21        -25         46        -26         31        -57
        1  673400276  592099542   81300734        -25         46        -71         31        -57         88
        7  592099542   81300734   22994404         46        -71        543        -57         88       -673
        3   81300734   22994404   12317522        -71        543      -1700         88       -673       2107
        1   22994404   12317522   10676882        543      -1700       2243       -673       2107      -2780
        1   12317522   10676882    1640640      -1700       2243      -3943       2107      -2780       4887
        6   10676882    1640640     833042       2243      -3943      25901      -2780       4887     -32102
        1    1640640     833042     807598      -3943      25901     -29844       4887     -32102      36989
        1     833042     807598      25444      25901     -29844      55745     -32102      36989     -69091
       31     807598      25444      18834     -29844      55745   -1757939      36989     -69091    2178810
        1      25444      18834       6610      55745   -1757939    1813684     -69091    2178810   -2247901
        2      18834       6610       5614   -1757939    1813684   -5385307    2178810   -2247901    6674612
        1       6610       5614        996    1813684   -5385307    7198991   -2247901    6674612   -8922513
        5       5614        996        634   -5385307    7198991  -41380262    6674612   -8922513   51287177
        1        996        634        362    7198991  -41380262   48579253   -8922513   51287177  -60209690
        1        634        362        272  -41380262   48579253  -89959515   51287177  -60209690  111496867
        1        362        272         90   48579253  -89959515  138538768  -60209690  111496867 -171706557
        3        272         90          2  -89959515  138538768 -505575819  111496867 -171706557  626616538
       45         90          2          0  138538768 -505575819 22889450623 -171706557  626616538 -28369450767
        -          2          0          - -505575819 22889450623          -  626616538 -28369450767          -
GCD(56738901534, 45778901246) = 2
s = -505575819
t = 626616538
```



```
                    1                    155716                    153658                    2058
1013000289899876891972853643868768429576643344 -365232586204523259946104149781743467439489421 4665326151945109491433895141686203103971537555
-792959966370172313574430525096938087509165455 2858980615915271131725471989254932758686471697 -365194058228544344529990251435187084619563714
                   74                    153658                    2058                    1366
-365232586204523259946104149781743467439489421 4665326151945109491433895141686203103971537555 -34888646110598333496556928198259646436828867291
2858980615915271131725471989254932758686471697 -365194058228544344529990251435187084619563714 27310258370503808608391825805129337537716362005
                   1                    2058                    1366                    692
4665326151945109491433895141686203103971537555 -34888646110598333496556928198259646436828867291 35355178725792844445700317712428266747226021046
-365194058228544344529990251435187084619563714 27310258370503808608391825805129337537716362005 -276754524287323529529218160565645246223359257 19
                   1                    1366                    692                    674
-34888646110598333496556928198259646436828867291 35355178725792844445700317712428266747226021046 -702438248363911779422572459106879131840548883337
27310258370503808608391825805129337537716362005 -276754524287323529529218160565645246223359257 19 549857107992361615613136418616938621600522877 24
                   1                    692                    674                    18
35355178725792844445700317712428266747226021046 -702438248363911779422572459106879131840548883337 10559900356218402238795756362311617993128090938 3
-276754524287323529529218160565645246223359257 19 549857107992361615613136418616938621600522877 24 -82661163227968514514235457918258386782388213443
                   37                    674                    18                    8
-702438248363911779422572459106879131840548883337 10559900356218402238795756362311617993128090938 3 -397740695663720000629668709996598657064144853550 8
549857107992361615613136418616938621600522877 24 -82661163227968514514235457918258386782388213443 31134487502340711985880255848372541731084161851 15
                   2                    18                    8                    2
10559900356218402238795756362311617993128090938 3 -397740695663720000629668709996598657064144853550 8 806041291683658403498133176355508932121417798 39
-82661163227968514514235457918258386782388213443 31134487502340711985880255848372541731084161851 15 -6309558663696110911690286627592766732999220583673
                   4                    8                    2                    0
-397740695663720000629668709996598657064144853550 8 806041291683658403498133176355508932121417798 39 -362190586239835361462220141541863438554981604571 04
31134487502340711985880255848372541731084161851 15 -6309558663696110911690286627592766732999220583673 28351683405018514845349172095208321105105298519807
                   -                    2                    0                    -
806041291683658403498133176355508932121417798 39 -362190586239835361462220141541863438554981604571 04                    -
-6309558663696110911690286627592766732999220583673 28351683405018514845349172095208321105105298519807                    -
GCD(567033668100370296906983441904166422102105970396 14, 724381172479670722924440283083726877109963209 14208) = 2
s = 806041291683658403498133176355508932121417798 399
t = -6309558663696110911690286627592766732999220583673
```

Multiplicative Inverse

Code:

Cpp:

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

int main()
{

    int q, r1, r2, r, t1 = 0, t2 = 1, t;
    cin >> r1 >> r2;
    cout << r1 << " " << r2 << "\n";
    while (r2 > 0)
    {
        q = r1 / r2;
        r = r1 - r2 * q;
        r1 = r2;
        r2 = r;
        t = t1 - t2 * q;
        t1 = t2;
        t2 = t;
    }
    cout << r1 << " " << t1 << "\n";

    return 0;
}
```

Output:

```
d:\CNS Lab>cd "d:\CNS Lab\" && g++ multiplicative_inverse.cpp -
ultiplicative_inverse && "d:\CNS Lab\"multiplicative_inverse
4 8
4 8
4 0
```

Python:

```python
def multInv(a: int, b: int):
  r1 = a
  r2 = b
  t1 = 0
  t2 = 1
```

```
  print(f'{"q":>20} {"r1":>20} {"r2":>20} {"r":>20} {"t1":>20} {"t2":>20}
{"t":>20}')
  print(" ")

  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2
    t = t1 - q * t2

    print(f'{"%20.0d" % q} {"%20.0d" % r1} {"%20.0d" % r2} {"%20.0d" % r}
{"%20.0d" % t1} {"%20.0d" % t2} {"%20.0d" % t}')

    r1 = r2
    r2 = r
    t1 = t2
    t2 = t

  print(f'{"-":>20} {"%20.0d" % r1} {"%20.0d" % r2} {"-":>20} {"%20.0d" %
t1} {"%20.0d" % t2} {"-":>20}')

  print(f'GCD({a}, {b}) = {r1}')

  if r1 == 1:
    if t1 > 0:
      print(f'Multiplicative inverse of {b} in Z{a} = {t1}')
    else:
      print(f'Multiplicative inverse of {b} in Z{a} = {t1} or {a + t1}')
  else:
    print('Multiplicative inverse does not exist')




def main():

  a = int(input('Enter a: '))
  b = int(input('Enter b: '))
  multInv(a,b)
```

Output:

1st input:

```
main()
```

```
In [3]: runfile('D:/CNS Lab/MI.py', wdir='D:/CNS Lab')

Enter a: 1234568022

Enter b: 3322446688
                  q                r1                r2                 r                t1                t2                 t
                  0        1234568022        3322446688        1234568022                 0                 1                 0
                  2        3322446688        1234568022         853310644                 1                 0                 1
                  1        1234568022         853310644         381257378                 0                 1                -1
                  2         853310644         381257378          90795888                 1                -1                 3
                  4         381257378          90795888          18073826                -1                 3                -13
                  5          90795888          18073826            426758                 3               -13                68
                 42          18073826            426758            149990               -13                68             -2869
                  2            426758            149990            126778                68             -2869              5806
                  1            149990            126778             23212             -2869              5806             -8675
                  5            126778             23212             10718              5806             -8675             49181
                  2             23212             10718              1776             -8675             49181            -107037
                  6             10718              1776                62             49181            -107037            691403
                 28              1776                62                40            -107037            691403          -19466321
                  1                62                40                22            691403          -19466321          20157724
                  1                40                22                18          -19466321          20157724          -39624045
                  1                22                18                 4          20157724          -39624045          59781769
                  4                18                 4                 2          -39624045          59781769         -278751121
                  2                 4                 2                 0          59781769         -278751121         617284011
                  -                 2                 0                 -         -278751121         617284011                 -
GCD(1234568022, 3322446688) = 2
Multiplicative inverse does not exist
```

```
                  3            620806            155716            153658 4801007168047541910021804139641184961589753
4 -792959966370172313574430525096938087509165 2858980615915271131725471989254932758686471 69
                  1            155716            153658              2058 -792959966370172313574430525096938087509165
45 2858980615915271131725471989254932758686471 69 -365194058228544344529990251435187084619563714
                 74            153658              2058              1366 2858980615915271131725471989254932758686471 69
-365194058228544344529990251435187084619563714 2731025837050380860839182580512933753771636200
5
                  1              2058              1366               692 -365194058228544344529990251435187084619563714
2731025837050380860839182580512933753771636200
5 -276754524287323529529218160565645246223359257
19
                  1              1366               692               674 2731025837050380860839182580512933753771636200
5 -276754524287323529529218160565645246223359257
19 549857107992361615613136418616938621600522877
24
                  1               692               674                18 -276754524287323529529218160565645246223359257
19 549857107992361615613136418616938621600522877
24 -826611632279685145142354579182583867823882134
43
                 37               674                18                 8 549857107992361615613136418616938621600522877
24 -826611632279685145142354579182583867823882134
43 311344875023407119858802558483725417310841618511
5
                  2                18                 8                 2 -826611632279685145142354579182583867823882134
43 311344875023407119858802558483725417310841618511
5 -630955866369611091169028662759276673299922058367
3
                  4                 8                 2                 0 311344875023407119858802558483725417310841618511
5 -630955866369611091169028662759276673299922058367
3 283516834050185148453491720952083211051052985198
07
                  -                 2                 0                 - -630955866369611091169028662759276673299922058367
3 283516834050185148453491720952083211051052985198
07                                                  -
GCD(567033668100370296906983441904166422102105970396
14, 724381172479670722924440283083726877109963209142
08) = 2
Multiplicative inverse does not exist
```

2nd input:

```
In [1]: runfile('D:/CNS Lab/MI.py', wdir='D:/CNS Lab')

Enter a: 07512779912565014928652150296195898826835317022090

Enter b: 97569354727979857921728118865439213176360669242153
                   q                   r1                   r2                    r                   t1                   t2                    t
                   0 75127799125650149286521502961958988268353170220900 97569354727979857921728118865439213176360669242153 75127799125650149286521502961958988268353170220900
1                  0
                  12 97569354727979857921728118865439213176360669242153 75127799125650149286521502961958988268353170220900 74159957771996787779023153110884272543368649770730
0                  1
                   1 75127799125650149286521502961958988268353170220900 74159957771996787779023153110884272543368649770730 96784135365336150749834985107471572498452045017
```

```
-12581359651207264965131327317302879238440718670
                   9                 5470                  589                  169 10709273062048234171906863389324683928832664000 -12581359651207264965131327317302879238440718670
12394150992291361885808880924505059707479913203
                   3                  589                  169                   82 -12581359651207264965131327317302879238440718670 12394150992291361885808880924505059707479913203
-38440588941994812153939775505245467046283811476
                   2                  169                   82                    5 12394150992291361885808880924505059707479913203 -38440588941994812153939775505245467046283811476
89275328876280986193688431934995993800047536155
                  16                   82                    5                    2 -38440588941994812153939775505245467046283811476 89275328876280986193688431934995993800047536155
-14668458509624905912529546864651813678470443899560
                   2                    5                    2                    1 89275328876280986193688431934995993800047536155 -14668458509624905912529546864651813678470443899560
30229670308012621686995978048653587294941363160670
                   2                    2                    1                    0 -14668458509624905912529546864651813678470443899560 30229670308012621686995978048653587294941363160670
-75127799125650149286521502961958988268353170220900
                   -                    1                    0                    - 30229670308012621686995978048653587294941363160670 -75127799125650149286521502961958988268353170220900
GCD(75127799125650149286521502961958988268353170220900, 97569354727979857921728118865439213176360669242153) = 1
Multiplicative inverse of 97569354727979857921728118865439213176360669242153 in Z75127799125650149286521502961958988268353170220900 = 30229670308012621686995978048653587294941363160670
```

SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS 00094 - Sweety Shrawan Gupta

Assignment 9

Chinese Remainder Theorem

Theory:-

Chinese Remainder Theorem

Given pairwise coprime positive integers $n_1, n_2, \ldots, n_k$ and arbitrary integers $a_1, a_2, \ldots, a_k$, the system of simultaneous congruences

$$\begin{aligned} x &\equiv a_1 \pmod{n_1} \\ x &\equiv a_2 \pmod{n_2} \\ &\vdots \\ x &\equiv a_k \pmod{n_k} \end{aligned}$$

has a solution, and the solution is unique modulo $N = n_1 n_2 \cdots n_k$.

The CRT is a theorem which gives a unique solution to simultaneous linear congruences with coprime moduli. In its basic form, the CRT will determine a number p that, when divided by some given divisors, leaves given remainders.

Code:

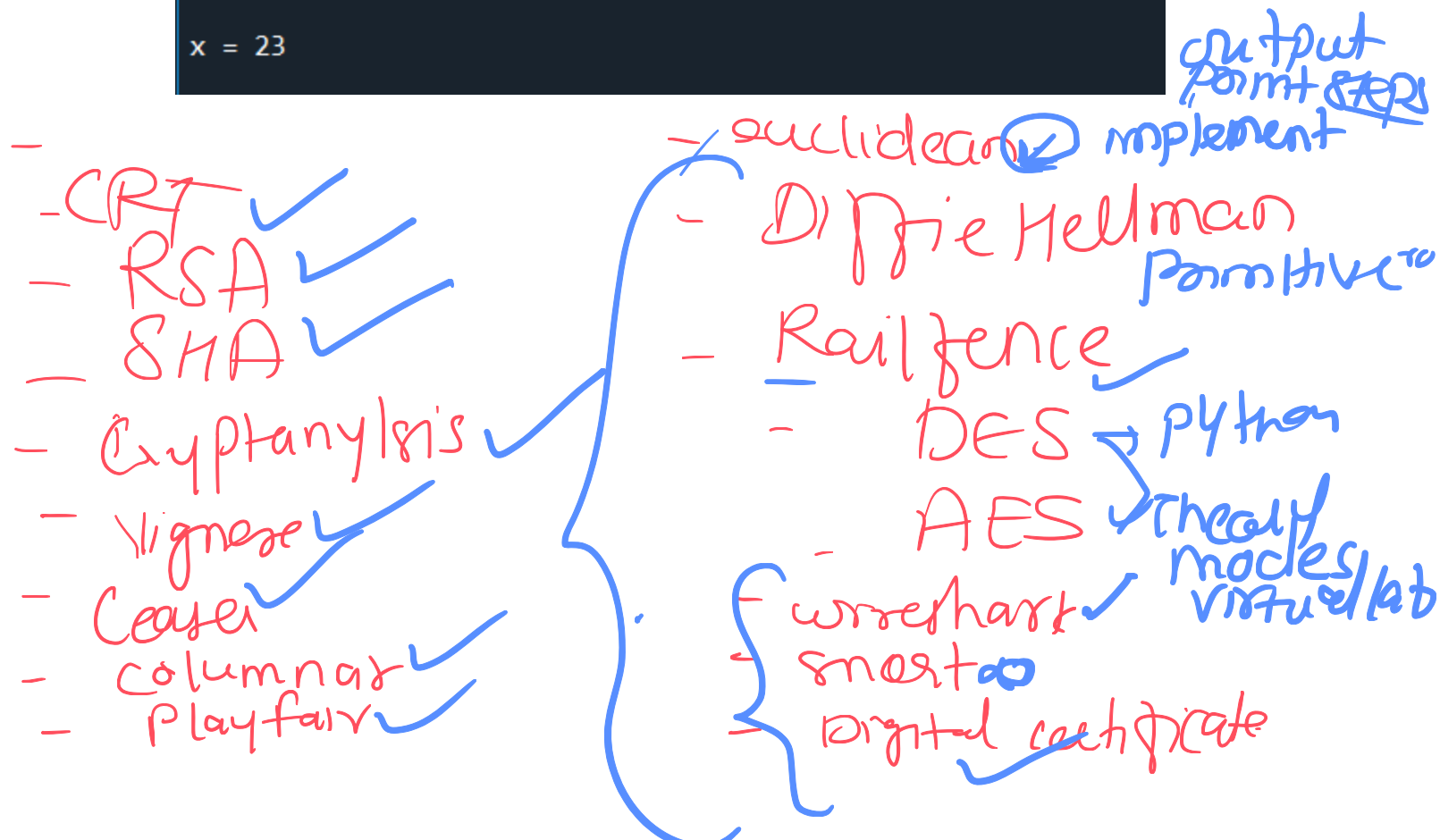
```
def calcMultInv(a: int, b: int):
  r1 = a
  r2 = b
  t1 = 0
  t2 = 1

  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2
    t = t1 - q * t2
```

```
    r1 = r2
    r2 = r
    t1 = t2
    t2 = t

  if r1 == 1:
    if t1 > 0:
      return t1
    else:
      return a + t1
  else:
    return None

def main():
  n = int(input('Enter no. of equations: '))
  a = list(map(int, input('Enter a0 a1 a2 ... an: ').split()))
  m = list(map(int, input('Enter m0 m1 m2 ... mn: ').split()))

  pM = 1
  for i in range(n):
    pM *= m[i]

  M = []
  for i in range(n):
    M.append(pM // m[i])

  MInv = []
  for i in range(n):
    MInv.append(calcMultInv(m[i], M[i]))

  x = 0
  for i in range(n):
    x = (x + a[i] * M[i] * MInv[i]) % pM

  print('\na =', a)
  print('m =', m)
  print('pM =', pM)
  print('M =', M)
  print('MInv =', MInv)
```

```
  print('\nx =', x)

main()
```

Output:

```
In [21]: runfile('D:/CNS Lab/CRT.py', wdir='D:/CNS Lab')

Enter no. of equations: 3

Enter a0 a1 a2 ... an: 2 3 2

Enter m0 m1 m2 ... mn: 3 5 7

a = [2, 3, 2]
m = [3, 5, 7]
pM = 105
M = [35, 21, 15]
MInv = [2, 1, 1]

x = 23
```



SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 11

Diffie-Hellman Key Exchange Algorithm

Theory:-

Diffie–Hellman key exchange is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as conceived by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.

Code:

```
# Enter the approved prime number and the primitive root g.
Prime_no = int(input("Enter Prime No. q: "))
g = int(input("Enter Primitive root (a<q) : "))
# Enter  private key chosen by A and B
PkXa = int(input("Enter Private key of A (xa<q) : "))
PkXb = int(input("Enter Private key of B (xb<q) : "))
# Calculate public key of A and B
ya = g**PkXa % Prime_no
yb = g**PkXb % Prime_no
# Calculate shared session key K
ka = yb**PkXa % Prime_no
kb = ya**PkXb % Prime_no
print("A's Public Key Ya =",ya)
print("B's Public Key Yb =",yb)
print("Shared session key k =",ka)
```

Output:

```
In [1]: runfile('D:/CNS Lab/Diffie_hellman.py', wdir='D:/CNS Lab')

Enter Prime No. q: 23

Enter Primitive root (a<q) : 9

Enter Private key of A (xa<q) : 4

Enter Private key of B (xb<q) : 3
A's Public Key Ya = 6
B's Public Key Yb = 16
Shared session key k = 9

In [2]:
```

Diffie Hellman key exchange - live interaction of 2 programs (socket programming)

Code:

Server:

```
import java.net.*;
import java.io.*;

public class Server {
    public static void main(String[] args) throws IOException {
        try {
            int port = 8088;

            // Server Key
```

```
            int b = 3;

            // Client p, g, and key
            double clientP, clientG, clientA, B, Bdash;
            String Bstr;

            // Established the Connection
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Waiting for client on port " +
serverSocket.getLocalPort() + "...");
            Socket server = serverSocket.accept();
            System.out.println("Just connected to " +
server.getRemoteSocketAddress());

            // Server's Private Key
            System.out.println("From Server : Private Key = " + b);

            // Accepts the data from client
            DataInputStream in = new
DataInputStream(server.getInputStream());

            clientP = Integer.parseInt(in.readUTF()); // to accept p
            System.out.println("From Client : P = " + clientP);

            clientG = Integer.parseInt(in.readUTF()); // to accept g
            System.out.println("From Client : G = " + clientG);

            clientA = Double.parseDouble(in.readUTF()); // to accept A
            System.out.println("From Client : Public Key = " + clientA);

            B = ((Math.pow(clientG, b)) % clientP); // calculation of B
            Bstr = Double.toString(B);

            // Sends data to client
            // Value of B
            OutputStream outToclient = server.getOutputStream();
            DataOutputStream out = new DataOutputStream(outToclient);

            out.writeUTF(Bstr); // Sending B
```

```
            Bdash = ((Math.pow(clientA, b)) % clientP); // calculation of
Bdash

            System.out.println("Secret Key to perform Symmetric Encryption
= "
                    + Bdash);
            server.close();
        }

        catch (SocketTimeoutException s) {
            System.out.println("Socket timed out!");
        } catch (IOException e) {
        }
    }
}
```

```
D:\CNS Lab>java Server
Waiting for client on port 8088...
```

```
Just connected to /127.0.0.1:62912
From Server : Private Key = 3
From Client : P = 23.0
From Client : G = 9.0
From Client : Public Key = 6.0
Secret Key to perform Symmetric Encryption = 9.0
```

Client:

```
import java.net.*;
import java.io.*;

public class Client {
    public static void main(String[] args) {
        try {
```

```
            String pstr, gstr, Astr;
            String serverName = "localhost";
            int port = 8088;

            // Declare p, g, and Key of client
            int p = 23;
            int g = 9;
            int a = 4;
            double Adash, serverB;

            // Established the connection
            System.out.println("Connecting to " + serverName
                    + " on port " + port);
            Socket client = new Socket(serverName, port);
            System.out.println("Just connected to "
                    + client.getRemoteSocketAddress());

            // Sends the data to client
            OutputStream outToServer = client.getOutputStream();
            DataOutputStream out = new DataOutputStream(outToServer);

            pstr = Integer.toString(p);
            out.writeUTF(pstr); // Sending p

            gstr = Integer.toString(g);
            out.writeUTF(gstr); // Sending g

            double A = ((Math.pow(g, a)) % p); // calculation of A
            Astr = Double.toString(A);
            out.writeUTF(Astr); // Sending A

            // Client's Private Key
            System.out.println("From Client : Private Key = " + a);

            // Accepts the data
            DataInputStream in = new
DataInputStream(client.getInputStream());

            serverB = Double.parseDouble(in.readUTF());
            System.out.println("From Server : Public Key = " + serverB);
```

```
            Adash = ((Math.pow(serverB, a)) % p); // calculation of Adash

            System.out.println("Secret Key to perform Symmetric Encryption
= "
                    + Adash);
            client.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

```
D:\CNS Lab>java Client
Connecting to localhost on port 8088
Just connected to localhost/127.0.0.1:8088
From Client : Private Key = 4
From Server : Public Key = 16.0
Secret Key to perform Symmetric Encryption = 9.0
```

SEM - VII - 2022-23
CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 12

Prime Factorization

Theory:

Prime factorisation is a method to find the prime factors of a given number, say a composite number. These factors are nothing but the prime numbers. A prime number is a number which has only two factors, i.e. 1 and the number itself. For example, 2 is a prime number which has two factors, 2 × 1

Code:

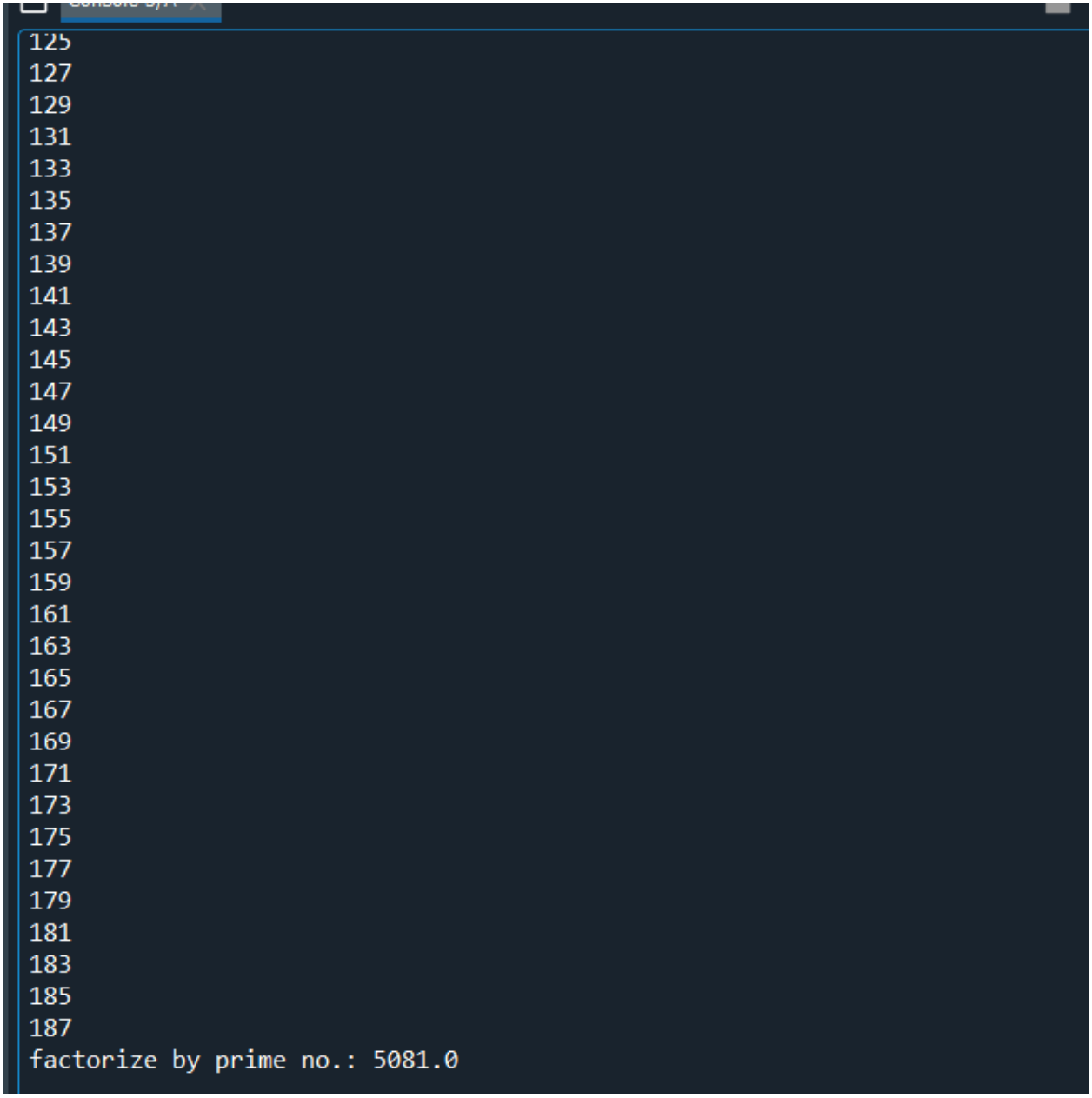
```
import math

def prime_factors(num):
    flag=0;
    while num % 2 == 0:
        flag=1
        num = num / 2

    if flag==1:
         print('factorize by prime no.:',2)

    for i in range(3, int(math.sqrt(num)) + 1, 2):

        # while i divides n , print i ad divide n
        print(i)
        while num % i == 0:
            print('factorize by prime no.:',i)
            num = num / i
    if num > 2:
```

```
        print('factorize by prime no.:',num)
# calling function

num = 35567
prime_factors(num)
```

Output:

```
In [6]: runfile('D:/CNS Lab/prime_factorization.py', wdir='D:/CNS Lab')
3
5
7
factorize by prime no.: 7
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49
51
53
55
57
59
61
63
65
67
```

```
63
65
67
69
71
73
75
77
79
81
83
85
87
89
91
93
95
97
99
101
103
105
107
109
111
113
115
117
119
121
123
125
127
129
131
133
135
```


125
127
129
131
133
135
137
139
141
143
145
147
149
151
153
155
157
159
161
163
165
167
169
171
173
175
177
179
181
183
185
187
factorize by prime no.: 5081.0

SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 13

SHA-512

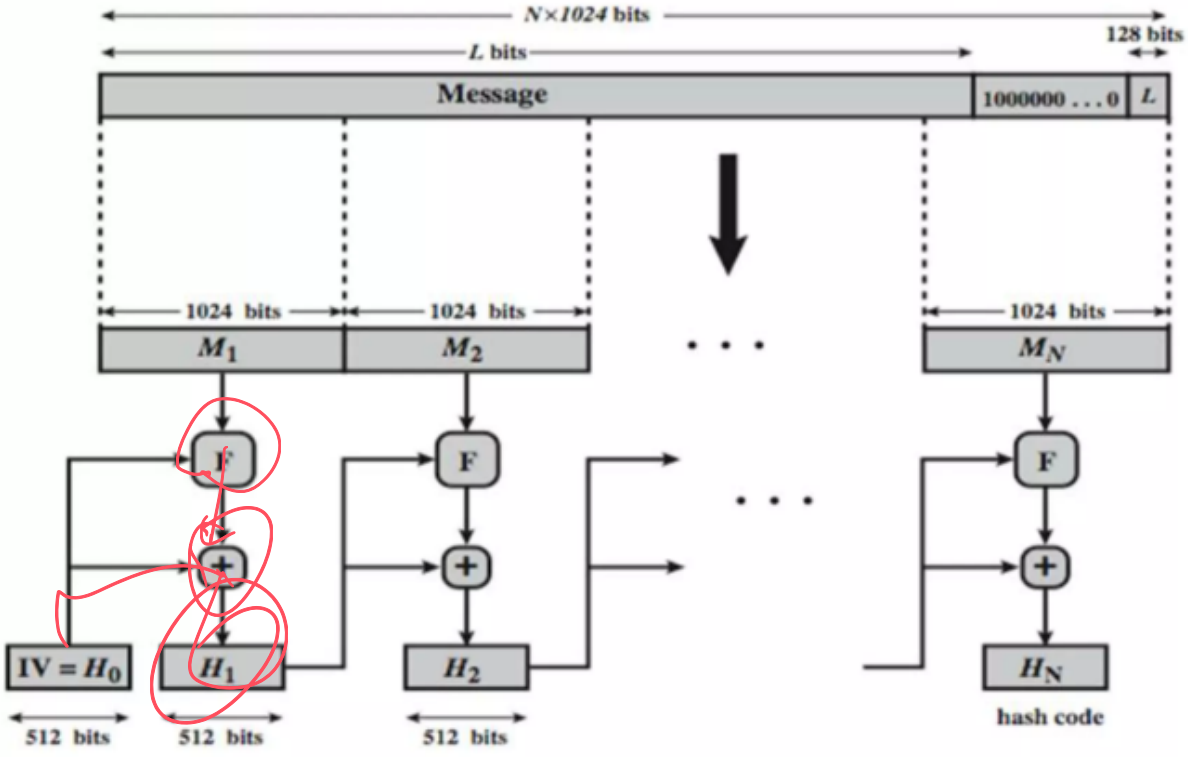
Theory:-

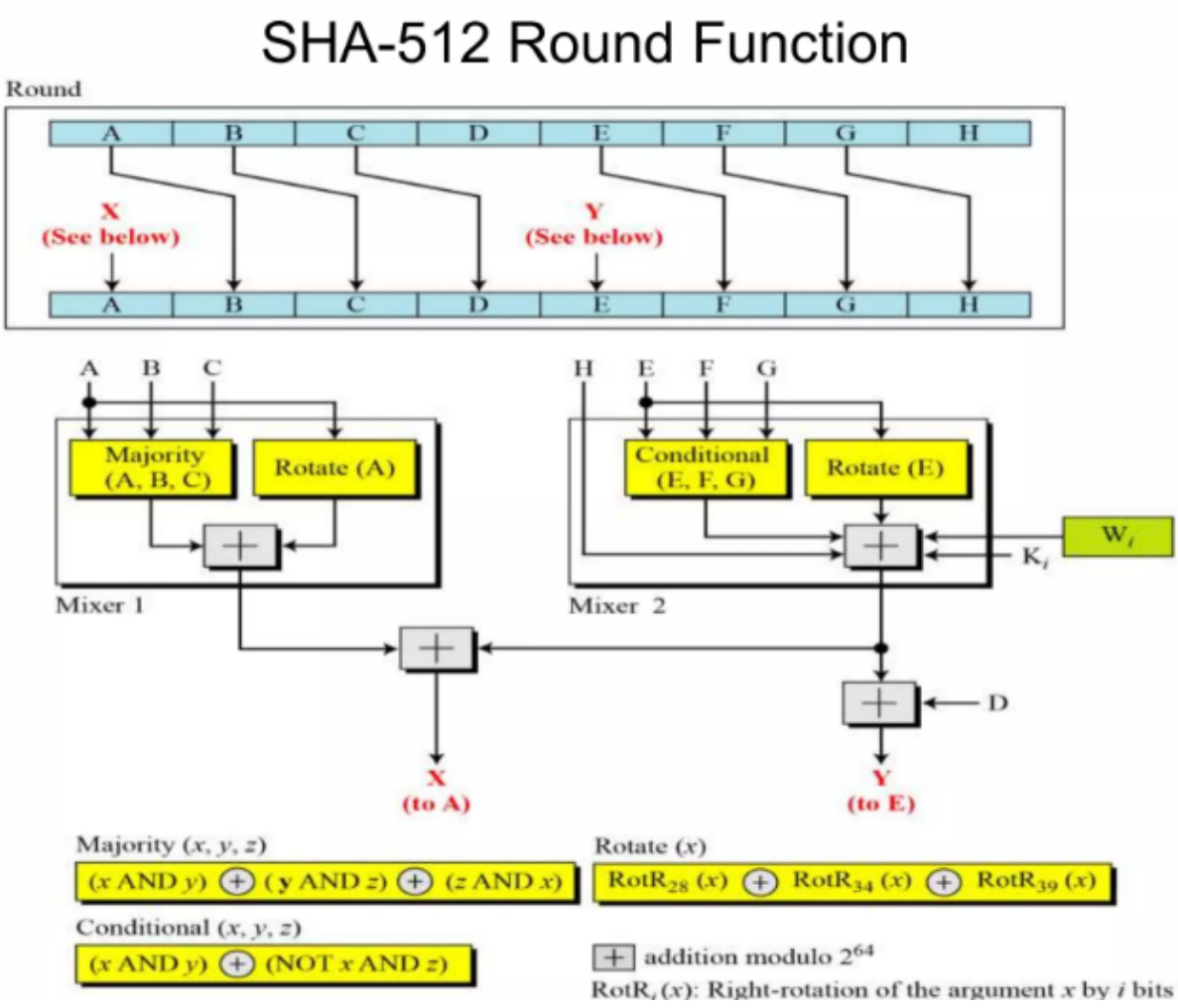
SHA-512 (Secure Hash Algorithm) is a set of cryptographic hash functions designed by the United States National Security Agency (NSA) and first published in 2001. They are built using the Merkle–Damgård construction, from a one-way compression function itself built using the Davies–Meyer structure from a specialised block cipher.

SHA-512 is a hashing algorithm that performs a hashing function on some data given to it.

SHA-512

Processing of SHA-512





Code:

```
#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 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()
{

    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:

```
D:\CNS Lab>cd "d:\CNS Lab\" && g++ SHA_512.cpp -o SHA_512 && "d:\CNS Lab\"SHA_512
Enter Text: trtyrruyttuotutyou
Input: trtyrruyttuotutyou
Plain text length = 144
Plain text length after padding = 1024

Padded Input: 0111010001110010011101000111100101110010011100100111010101111001011101000111010001110101011101000111100101101111011101011000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000010010000

Output of SHA-512 Algorithm:
fc18413cd1e0ec44ae694cdb4e9e8fb3d3023953f99f36ffca2aec34e8fe5df1be2b4e3a8f7be07d2d87fd723f223581820332fff2cced3f741c56f29e7b7918
```

SEM - VII - 2022-23
CNS Lab

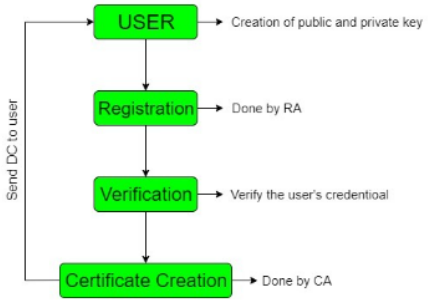
B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 14

Digital Certificate Generation

Steps for Digital Certificate Creation:

- **Step-1:** Key generation is done by either user or registration authority. The public key which is generated is sent to the registration authority and private key is kept secret by user.
- **Step-2:** In the next step the registration authority registers the user.
- **Step-3:** Next step is verification which is done by registration authority in which the user's credentials are being verified by registration authority. It also checks that the user who send the public key have corresponding private key or not.
- **Step-4:** In this step the details and sent to certificate authority by registration authority who creates the digital certificate and give it to users and also keeps a copy to itself.



1. Generation of digital certificate using java key tool and key store utilities.

Creating a certificate:

```
C:\Users\SWEETY>keytool -genkey -alias priya -keyalg RSA -keystore "D:\local.keystore"
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  s g
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]:  mh
What is the two-letter country code for this unit?
  [Unknown]:  IN
Is CN=s g, OU=cse, O=wce, L=sangli, ST=mh, C=IN correct?
  [no]:  yes

Generating 2,048 bit RSA key pair and self-signed certificate (SHA256withRSA) with a validity of 90 days
        for: CN=s g, OU=cse, O=wce, L=sangli, ST=mh, C=IN
```

Displaying a certificate:

```
D:\>keytool -v -list -keystore local.keystore
Enter keystore password:
Keystore type: PKCS12
Keystore provider: SUN

Your keystore contains 1 entry

Alias name: priya
Creation date: Nov 14, 2022
Entry type: PrivateKeyEntry
Certificate chain length: 1
Certificate[1]:
Owner: CN=s g, OU=cse, O=wce, L=sangli, ST=mh, C=IN
Issuer: CN=s g, OU=cse, O=wce, L=sangli, ST=mh, C=IN
Serial number: 88fea0e2b145c2bf
Valid from: Mon Nov 14 14:56:47 IST 2022 until: Sun Feb 12 14:56:47 IST 2023Certificate fingerprints:
         SHA1: 91:BD:9F:AE:C8:19:7E:D4:7D:39:09:CB:74:F8:D9:75:F4:2C:AE:0E
         SHA256: AF:5C:45:AB:E8:67:FF:78:AA:8B:C0:11:20:06:CD:A2:D9:89:F4:90:73:65:C4:EA:B2:F7:43:37:3E:83:AC:5D
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 2048-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: FD 3B DF FB 61 16 56 23   69 DF EB 19 1A 68 FD 6B  .;..a.V#i....h.k
0010: 7F 3A C7 8E                                        .:..
]
]



*******************************************
*******************************************
```

SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 15

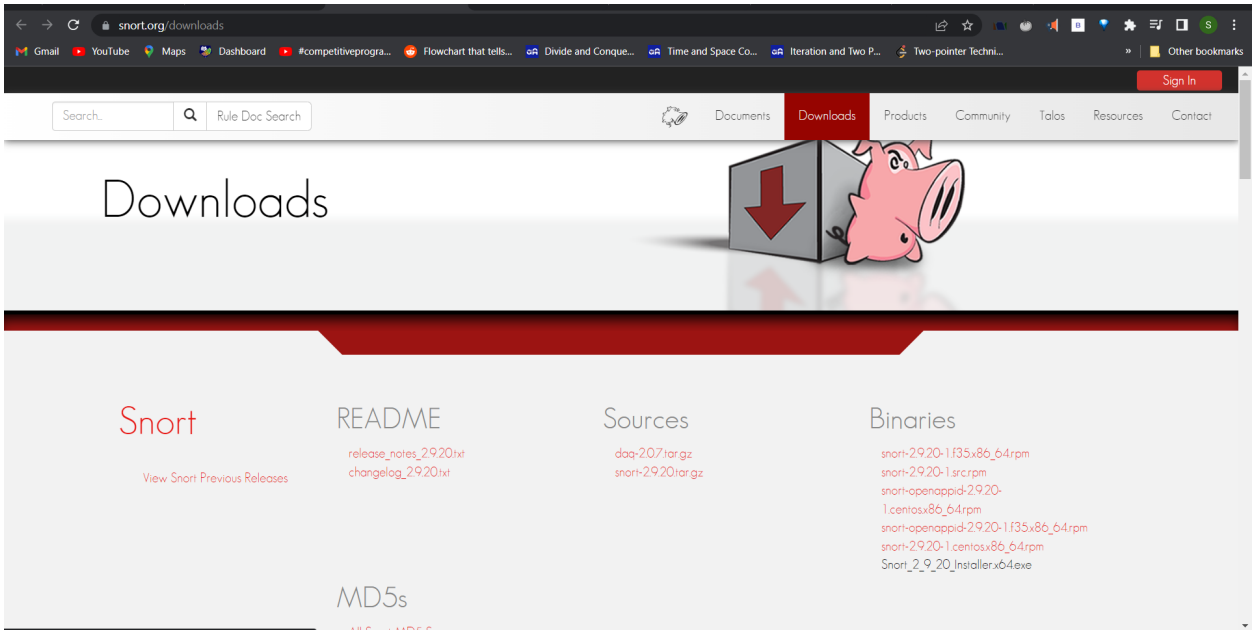
Snort Installation And Configuration

Theory:

SNORT is a network based intrusion detection system which is written in C programming language. It is free open-source software. It can also be used as a packet sniffer to monitor the system in real time. The network admin can use it to watch all the incoming packets and find the ones which are dangerous to the system. It is based on library packet capture tool. The rules are fairly easy to create and implement and it can be deployed in any kind of operating system and any kind of network environment.

Snort Installation:

https://www.snort.org/downloads



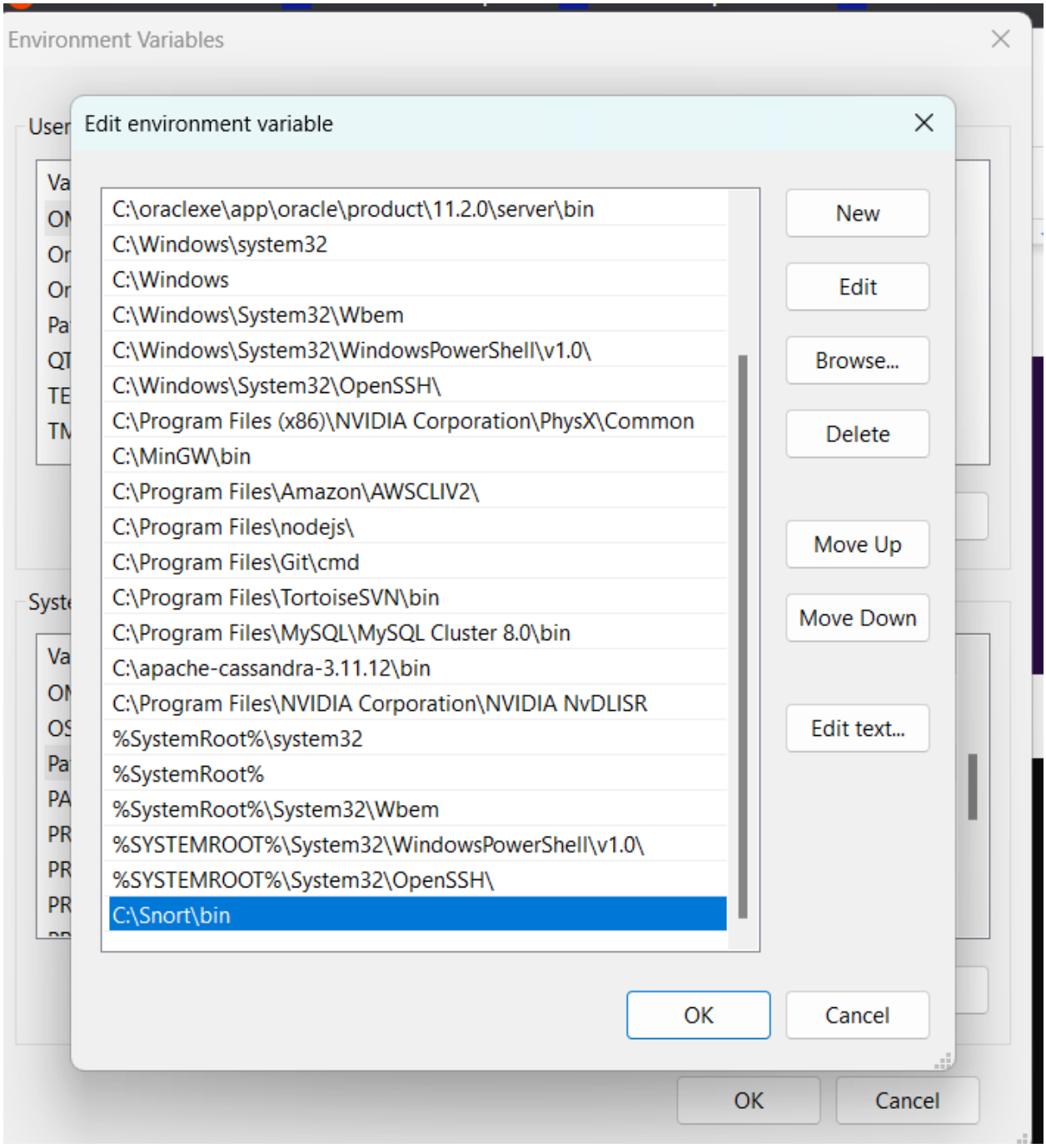
https://npcap.com/#download

Downloading and Installing Npcap Free Edition

The free version of Npcap may be used (but not externally redistributed) on up to 5 systems (free license details). It may also be used on unlimited systems where it is only used with Nmap, Wireshark, and/or Microsoft Defender for Identity. Simply run the executable installer. The full source code for each release is available, and developers can build their apps against the SDK. The improvements for each release are documented in the Npcap Changelog.

- Npcap 1.71 installer for Windows 7/2008R2, 8/2012, 8.1/2012R2, 10/2016, 2019, 11 (x86, x64, and ARM64).
- Npcap SDK 1.13 (ZIP).
- Npcap 1.71 debug symbols (ZIP).
- Npcap 1.71 source code (ZIP).

The latest development source is in our Github source repository. Windows XP and earlier are not supported; you can use WinPcap for these versions.


Environment Variables
Edit environment variable
C:\oraclexe\app\oracle\product\11.2.0\server\bin
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
C:\MinGW\bin
C:\Program Files\Amazon\AWSCLIV2\
C:\Program Files\nodejs\
C:\Program Files\Git\cmd
C:\Program Files\TortoiseSVN\bin
C:\Program Files\MySQL\MySQL Cluster 8.0\bin
C:\apache-cassandra-3.11.12\bin
C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Snort\bin
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel
OK
Cancel

```
C:\Users\SWEETY>snort -v
Running in packet dump mode

        --== Initializing Snort ==--
Initializing Output Plugins!
pcap DAQ configured to passive.
The DAQ version does not support reload.
Acquiring network traffic from "\Device\NPF_{69CDAB07-1898-44A7-AD4B-21013C6EA13E}".
Decoding Ethernet

        --== Initialization Complete ==--

           -*> Snort! <*-
  ,,_      Version 2.9.20-WIN64 GRE (Build 82)
 o"  )~    By Martin Roesch & The Snort Team: http://www.snort.org/contact#team
  ''''     Copyright (C) 2014-2022 Cisco and/or its affiliates. All rights reserved.
           Copyright (C) 1998-2013 Sourcefire, Inc., et al.
           Using PCRE version: 8.10 2010-06-25
           Using ZLIB version: 1.2.11

Commencing packet processing (pid=7492)
```

Making changes in snort.conf file

```
43
44   # Setup the network addresses you are protecting
45   ipvar HOME_NET 10.0.0.10/24
46
47   # Set up the external network addresses. Leave as "any" in most situations
48   ipvar EXTERNAL_NET !$HOME_NET
49
50   # List of DNS servers on your network
51   ipvar DNS_SERVERS $HOME_NET
52
```

```
 where snort is
 # not relative to snort.conf like the above variables
 # This is completely inconsistent with how other vars w
 # Set the absolute path appropriately
 var WHITE_LIST_PATH C:\Snort\rules
 var BLACK_LIST_PATH C:\Snort\rules
```

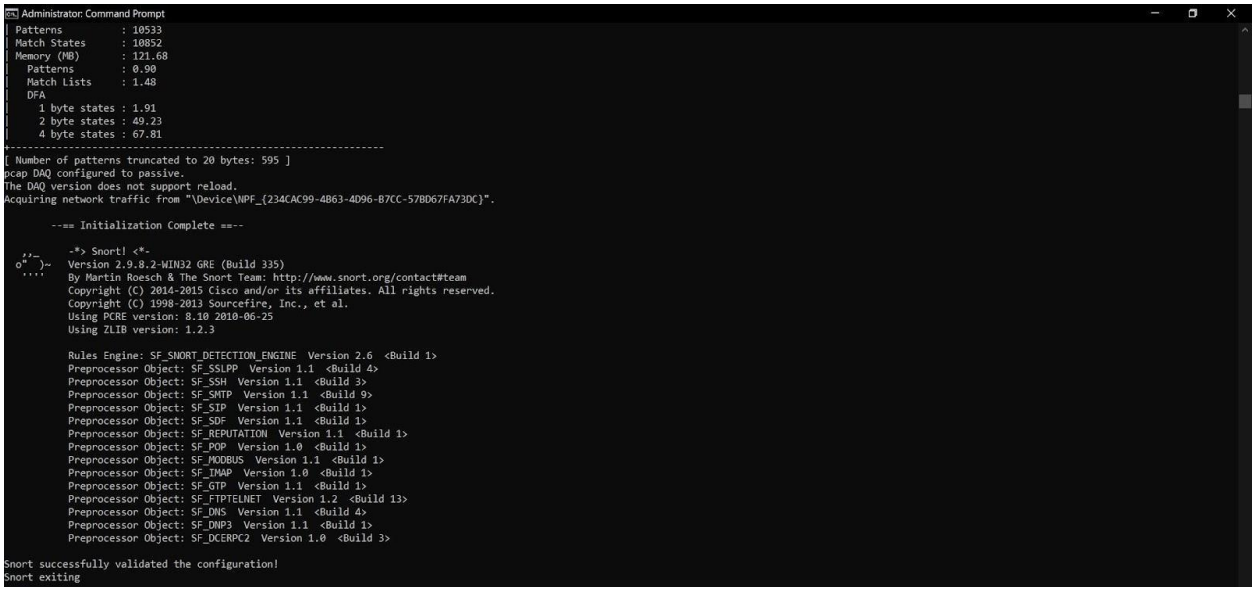
```
# path to dynamic preprocessor libraries
dynamicpreprocessor C:\Snort\lib\snort_dynamicpreprocessor

# path to base preprocessor engine
dynamicengine C:\Snort\lib\snort_dynamicengine\sf_engine.dll

# path to dynamic rules libraries
#dynamicdetection directory /usr/local/lib/snort_dynamicrules
```

```
##################################################

# site specific rules
include $RULE_PATH\local.rules

# include $RULE_PATH/app-detect.rules
# include $RULE_PATH/attack-responses.ru
```

Add rules in local.rules file

```
# to the VRT Certified Rules License Agreement (v2.0).
#
#------------
# LOCAL RULES
#------------
# Protocol types (TCP,UDP,ICMP,IP)
alert icmp any any (msg:"Testing ICMP alert";sid:1000001;)
alert udp any any (msg:"Testing UDP alert";sid:1000002;)
alert tcp any any (msg:"Testing TCP alert";sid:1000003;)
```

Initializing configuration:



SEM - VII - 2022-23

CNS Lab

B3 - 2019BTECS00094 - Sweety Shrawan Gupta

Assignment 10

RSA

Theory:

RSA algorithm is an asymmetric cryptography algorithm. Asymmetric actually means that it works on two different keys i.e. Public Key and Private Key. As the name describes that the Public Key is given to everyone and the Private key is kept private.

Code:

```
# Iterative Function to calculate (x^n)%p in O(logy)
def power(x, y, p):
  res = 1 # Initialize result

  x = x % p # Update x if it is more than or equal to p

  while (y > 0):
    # If y is odd, multiply x with result
    if (y & 1):
      res = (res * x) % p

    # y must be even now
    y = y >> 1 # y = y/2
    x = (x * x) % p
```

```
  return res

def encrypt(plaintext, e, n):
  return power(plaintext, e, n)

def decrypt(ciphertext, d, n):
  return power(ciphertext, d, n)

def main():

  print('1. Encrypt')
  print('2. Decrypt')

  choice = int(input('Enter your choice: '))

  if choice == 1:
    plaintext = int(input('Enter plaintext: '))

    print('Enter public key:')
    e = int(input('e = '))
    n = int(input('n = '))

    ciphertext = encrypt(plaintext, e, n)
    print('Ciphertext :',ciphertext)
  else:
    ciphertext = int(input('Enter ciphertext: '))

    print('Enter private key:')
    d = int(input('d = '))
    n = int(input('n = '))

    plaintext = decrypt(ciphertext, d, n)
    print("Plaintext: ",plaintext)

main()
```

Output:

```
In [4]: runfile('D:/CNS Lab/rsa.py', wdir='D:/CNS Lab')
1. Encrypt
2. Decrypt

Enter your choice: 1

Enter plaintext: 2
Enter public key:

e = 7

n = 527
Ciphertext : 128

In [5]: runfile('D:/CNS Lab/rsa.py', wdir='D:/CNS Lab')
1. Encrypt
2. Decrypt

Enter your choice: 2

Enter ciphertext: 128
Enter private key:

d = 343

n = 527
Plaintext:  2
```

For english character:

```
#Iterative Function to calculate(x ^ n) % p in O(logy)
```

```
def power(x, y, p):
  res = 1 # Initialize result

  x = x % p # Update x if it is more than or equal to p

  while (y > 0):
#If y is odd, multiply x with result
    if (y & 1):
      res = (res * x) % p

#y must be even now
    y = y >> 1 # y = y/2
    x = (x * x) % p

  return res

def encrypt(plaintext, e, n):
  return power(plaintext, e, n)

def decrypt(ciphertext, d, n):
  return power(ciphertext, d, n)

def main():

  print('1. Encrypt')
  print('2. Decrypt')

  choice = int(input('Enter your choice: '))

  if choice == 1:

    text = input('Enter plaintext: ')
    print('Enter public key:')
    e = int(input('e = '))
    n = int(input('n = '))
    print('Ciphertext :')
    for i in range(0, len(text)):
        plaintext= ord(text[i])
        ciphertext = encrypt(plaintext, e, n)
        print(ciphertext,end="")
```

```
  else:
    ciphertext = int(input('Enter ciphertext: '))

    print('Enter private key:')
    d = int(input('d = '))
    n = int(input('n = '))

    plaintext = decrypt(ciphertext, d, n)
    print("Plaintext: ",plaintext)

main()
```

```
In [5]: runfile('D:/CNS Lab/rsa.py', wdir='D:/CNS Lab')
1. Encrypt
2. Decrypt

Enter your choice: 1

Enter plaintext: sweety
Enter public key:

e = 7

n = 527
Ciphertext :
21119333391417
```

RSA Key Generator:-

```
def isPrime(n):
  if n < 2:
    return False
```

```
  if n != 2 and n % 2 == 0:
    return False

  for i in range(3, int(n ** 0.5) + 1, 2):
    if n % i == 0:
      return False

  return True

def gcd(a, b):
  r1 = a
  r2 = b

  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2

    r1 = r2
    r2 = r

  return r1

def multiplicativeInverse(a, b):
  r1 = a
  r2 = b
  t1 = 0
  t2 = 1

  while r2 != 0:
    q = r1 // r2
    r = r1 - q * r2
    t = t1 - q * t2

    r1 = r2
    r2 = r
    t1 = t2
    t2 = t

  if t1 > 0:
```

```
    return t1
  else:
    return a + t1

def generateKeys():
  p = 0
  q = 0

  while True:
    p = int(input('Enter large prime p: '))

    if isPrime(p):
      break
    else:
      print('Given number is not prime')

  while True:
    q = int(input('Enter large prime q: '))

    if p == q:
      print('q can not be equal to p')
    elif isPrime(q):
      break
    else:
      print('Given number is not prime')

  n = p * q
  phi = (p - 1) * (q - 1)

  e = 0
  d = 0

  while True:
    e = int(input(f'Enter number e coprime to {phi}: '))

    if gcd(e, phi) == 1:
      d = multiplicativeInverse(phi, e)

      if e != d:
        break
```

```
      else:
        print('e == d not allowed; Try some other value for e')
    else:
      print('Given number not coprime')

  print(f'Public key: {{{e}, {n}}}')
  print(f'Private key: {{{d}, {n}}}')




generateKeys()
```

```
In [1]: runfile('D:/CNS Lab/rsa_key_generator.py', wdir='D:/CNS Lab')

Enter large prime p: 23

Enter large prime q: 13

Enter number e coprime to 264: 243
Given number not coprime

Enter number e coprime to 264: 245
Public key: {245, 299}
Private key: {125, 299}
```