NAME: Gabu Siddharth Merambhai

Enrollment number: 180170107030

Subject code: 3170720

Subject name: Information Security

Semester: 7

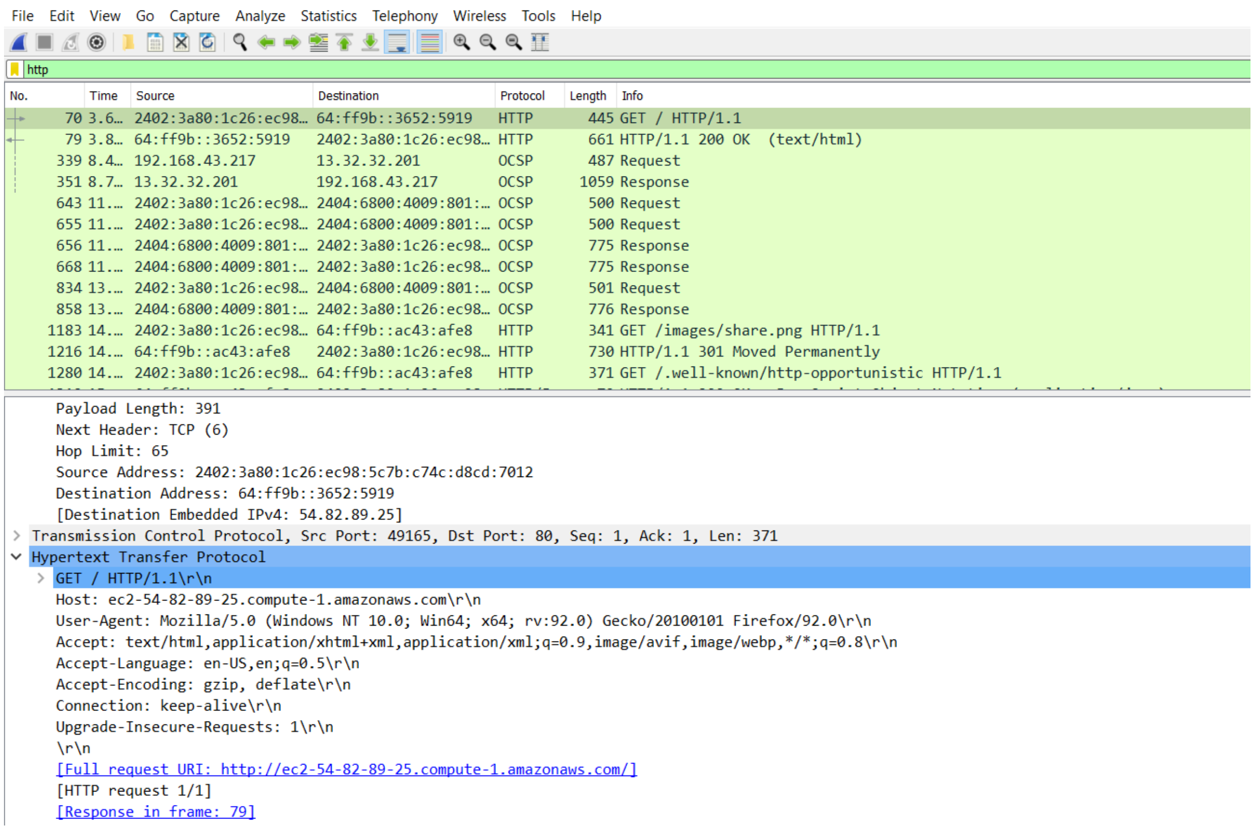
Index

Practical – 1

AIM: Study network analysis tool Wireshark. Filter TCP, UDP, ICMP, packet format, and extract OSI model format.

HTTP

Start up the Ethereal packet sniffer, as described in the Introductory lab (but don't yet begin packet capture). Enter "http" (just the letters, not the quotation marks) in the display-filter-specification window, so that only captured HTTP messages will be displayed later in the packet-listing window. (We're only interested in the HTTP protocol here, and don't want to see the clutter of all captured packets).


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
http

```
No.   Time  Source                Destination           Protocol Length Info
   70 3.6…  2402:3a80:1c26:ec98…  64:ff9b::3652:5919    HTTP      445 GET / HTTP/1.1
   79 3.8…  64:ff9b::3652:5919    2402:3a80:1c26:ec98…  HTTP      661 HTTP/1.1 200 OK  (text/html)
  339 8.4…  192.168.43.217        13.32.32.201          OCSP      487 Request
  351 8.7…  13.32.32.201          192.168.43.217        OCSP     1059 Response
  643 11.…  2402:3a80:1c26:ec98…  2404:6800:4009:801:…  OCSP      500 Request
  655 11.…  2402:3a80:1c26:ec98…  2404:6800:4009:801:…  OCSP      500 Request
  656 11.…  2404:6800:4009:801:…  2402:3a80:1c26:ec98…  OCSP      775 Response
  668 11.…  2404:6800:4009:801:…  2402:3a80:1c26:ec98…  OCSP      775 Response
  834 13.…  2402:3a80:1c26:ec98…  2404:6800:4009:801:…  OCSP      501 Request
  858 13.…  2404:6800:4009:801:…  2402:3a80:1c26:ec98…  OCSP      776 Response
 1183 14.…  2402:3a80:1c26:ec98…  64:ff9b::ac43:afe8    HTTP      341 GET /images/share.png HTTP/1.1
 1216 14.…  64:ff9b::ac43:afe8    2402:3a80:1c26:ec98…  HTTP      730 HTTP/1.1 301 Moved Permanently
 1280 14.…  2402:3a80:1c26:ec98…  64:ff9b::ac43:afe8    HTTP      371 GET /.well-known/http-opportunistic HTTP/1.1
```

```
    Payload Length: 391
    Next Header: TCP (6)
    Hop Limit: 65
    Source Address: 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012
    Destination Address: 64:ff9b::3652:5919
    [Destination Embedded IPv4: 54.82.89.25]
> Transmission Control Protocol, Src Port: 49165, Dst Port: 80, Seq: 1, Ack: 1, Len: 371
v Hypertext Transfer Protocol
  > GET / HTTP/1.1\r\n
    Host: ec2-54-82-89-25.compute-1.amazonaws.com\r\n
    User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:92.0) Gecko/20100101 Firefox/92.0\r\n
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=0.8\r\n
    Accept-Language: en-US,en;q=0.5\r\n
    Accept-Encoding: gzip, deflate\r\n
    Connection: keep-alive\r\n
    Upgrade-Insecure-Requests: 1\r\n
    \r\n
    [Full request URI: http://ec2-54-82-89-25.compute-1.amazonaws.com/]
    [HTTP request 1/1]
    [Response in frame: 79]
```

Fig 1.1 - HTTP

1. Is your browser running HTTP version 1.0 or 1.1? What version of HTTP is the server running?

Browser runs with HTTP 1.1
Server runs with HTTP 1.0

2. What languages (if any) does your browser indicate that it can accept to the server?

Accept-Language: en-us\r\n

3. What is the IP address of your computer? Of the gaia.cs.umass.edu server?

Internet Protocol, Src: 192.168.21.17 (192.168.21.17),
Dst: 128.119.245.12 (128.119.245.12)

4. What is the status code returned from the server to your browser?
Response Code: 200

5. When was the HTML file that you are retrieving last modified at the server?
Last-Modified: Mon, 07 Dec 2009 09:32:02 GMT\r\n

6. How many bytes of content are being returned to your browser?
Content-Length: 126\r\n

7. By inspecting the raw data in the packet content window, do you see any headers within the data that are not displayed in the packet-listing window? If so, name one.
No all of the headers can be found in the raw data

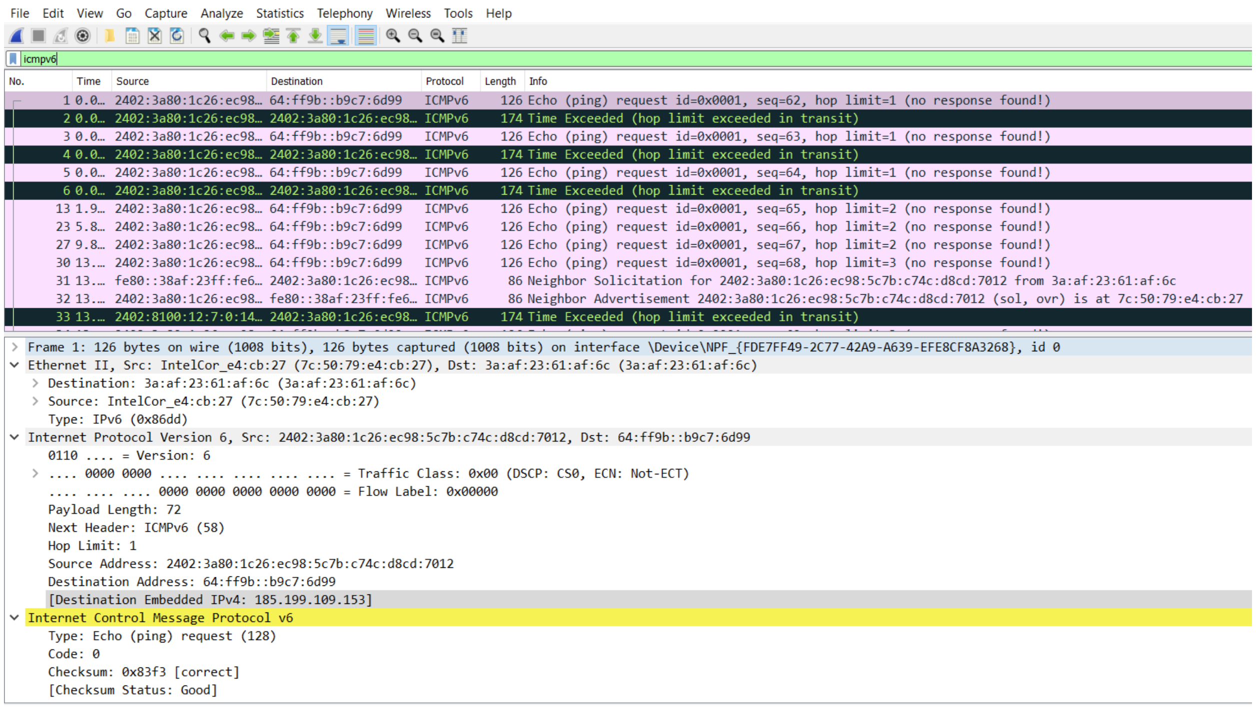
8. Inspect the contents of the first HTTP GET request from your browser to the server. Do you see an "IF-MODIFIED-SINCE" line in the HTTP GET?
No

ICMP and Trace route:

Traceroute is implemented in different ways in Unix/Linux and in Windows. In Unix/Linux, the source sends a series of UDP packets to the target destination using an unlikely destination port number; in Windows, the source sends a series of ICMP packets to the target destination. For both operating systems, the program sends the first packet with TTL=1, the second packet with TTL=2, and so on. Recall that a router will decrement a packet's TTL value as the packet passes through the router. When a packet arrives at a router with TTL=1, the router sends an ICMP error packet back to the source. In the following, we'll use the native Windows tracert program.

```
C:\Windows\system32>tracert -h 5 www.collegeek.com

Tracing route to collegeek.com [64:ff9b::b9c7:6d99]
over a maximum of 5 hops:

  1    47 ms     2 ms     2 ms  2402:3a80:1c26:ec98::af
  2     *        *        *     Request timed out.
  3    77 ms    34 ms    21 ms  2402:8100:12:7:0:14:0:211
  4     *        *        *     Request timed out.
  5     *        *        *     Request timed out.

Trace complete.
```

From this figure we see that for each TTL value, the
source program sends three probe packets. Traceroute displays the RTTs(round trip time) for each of the probe packets, as well as the IP address.


File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmpv6

```
No.   Time  Source                Destination           Protocol  Length Info
    1 0.0…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=62, hop limit=1 (no response found!)
    2 0.0…  2402:3a80:1c26:ec98…  2402:3a80:1c26:ec98…  ICMPv6    174 Time Exceeded (hop limit exceeded in transit)
    3 0.0…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=63, hop limit=1 (no response found!)
    4 0.0…  2402:3a80:1c26:ec98…  2402:3a80:1c26:ec98…  ICMPv6    174 Time Exceeded (hop limit exceeded in transit)
    5 0.0…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=64, hop limit=1 (no response found!)
    6 0.0…  2402:3a80:1c26:ec98…  2402:3a80:1c26:ec98…  ICMPv6    174 Time Exceeded (hop limit exceeded in transit)
   13 1.9…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=65, hop limit=2 (no response found!)
   23 5.8…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=66, hop limit=2 (no response found!)
   27 9.8…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=67, hop limit=2 (no response found!)
   30 13.…  2402:3a80:1c26:ec98…  64:ff9b::b9c7:6d99    ICMPv6    126 Echo (ping) request id=0x0001, seq=68, hop limit=3 (no response found!)
   31 13.…  fe80::38af:23ff:fe6…  2402:3a80:1c26:ec98…  ICMPv6     86 Neighbor Solicitation for 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012 from 3a:af:23:61:af:6c
   32 13.…  2402:3a80:1c26:ec98…  fe80::38af:23ff:fe6…  ICMPv6     86 Neighbor Advertisement 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012 (sol, ovr) is at 7c:50:79:e4:cb:27
   33 13.…  2402:8100:12:7:0:14…  2402:3a80:1c26:ec98…  ICMPv6    174 Time Exceeded (hop limit exceeded in transit)

> Frame 1: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface \Device\NPF_{FDE7FF49-2C77-42A9-A639-EFE8CF8A3268}, id 0
v Ethernet II, Src: IntelCor_e4:cb:27 (7c:50:79:e4:cb:27), Dst: 3a:af:23:61:af:6c (3a:af:23:61:af:6c)
  > Destination: 3a:af:23:61:af:6c (3a:af:23:61:af:6c)
  > Source: IntelCor_e4:cb:27 (7c:50:79:e4:cb:27)
    Type: IPv6 (0x86dd)
v Internet Protocol Version 6, Src: 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012, Dst: 64:ff9b::b9c7:6d99
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 72
    Next Header: ICMPv6 (58)
    Hop Limit: 1
    Source Address: 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012
    Destination Address: 64:ff9b::b9c7:6d99
    [Destination Embedded IPv4: 185.199.109.153]
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0x83f3 [correct]
    [Checksum Status: Good]
```

Fig 1.2 - ICMP

1) List the different protocols that appear in the protocol column in the unfiltered packet-listing window in step 7 above.

   Answer: IPX SAP, NBNS, NBIPX, ARP, TCP, SMP, HTTP, BOOTP, ICMP, UDP

2) How long did it take from when the HTTP GET message was sent until the HTTP OK reply was received? (By default, the value of the Time column in the packetlisting window is the amount of time, in seconds, since Ethereal tracing began. To display the Time field in time-of-day format, select the Ethereal *View* pull down menu, then select Time *Display Format*, then select *Time-of-day*.)

   Answer:

```
190   3.956   2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     64:ff9b::3652:5919     HTTP   435
      GET /chevron-down-solid.svg HTTP/1.1
259   4.212   64:ff9b::3652:5919     2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     HTTP/XML
      764     HTTP/1.1 200 OK
192   3.959   2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     64:ff9b::3652:5919     HTTP   429
      GET /adjust-solid.svg HTTP/1.1
256   4.206   64:ff9b::3652:5919     2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     HTTP/XML
      897     HTTP/1.1 200 OK
172   3.805   64:ff9b::3652:5919     2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     HTTP   661
      HTTP/1.1 200 OK  (text/html)
141   3.562   2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012     64:ff9b::3652:5919     HTTP   445
      GET / HTTP/1.1
```

3) What is the Internet address of the gaia.cs.umass.edu (also known as wwwnet. cs.umass.edu)? What is the Internet address of your computer?

   Answer:

   128.119.245.12

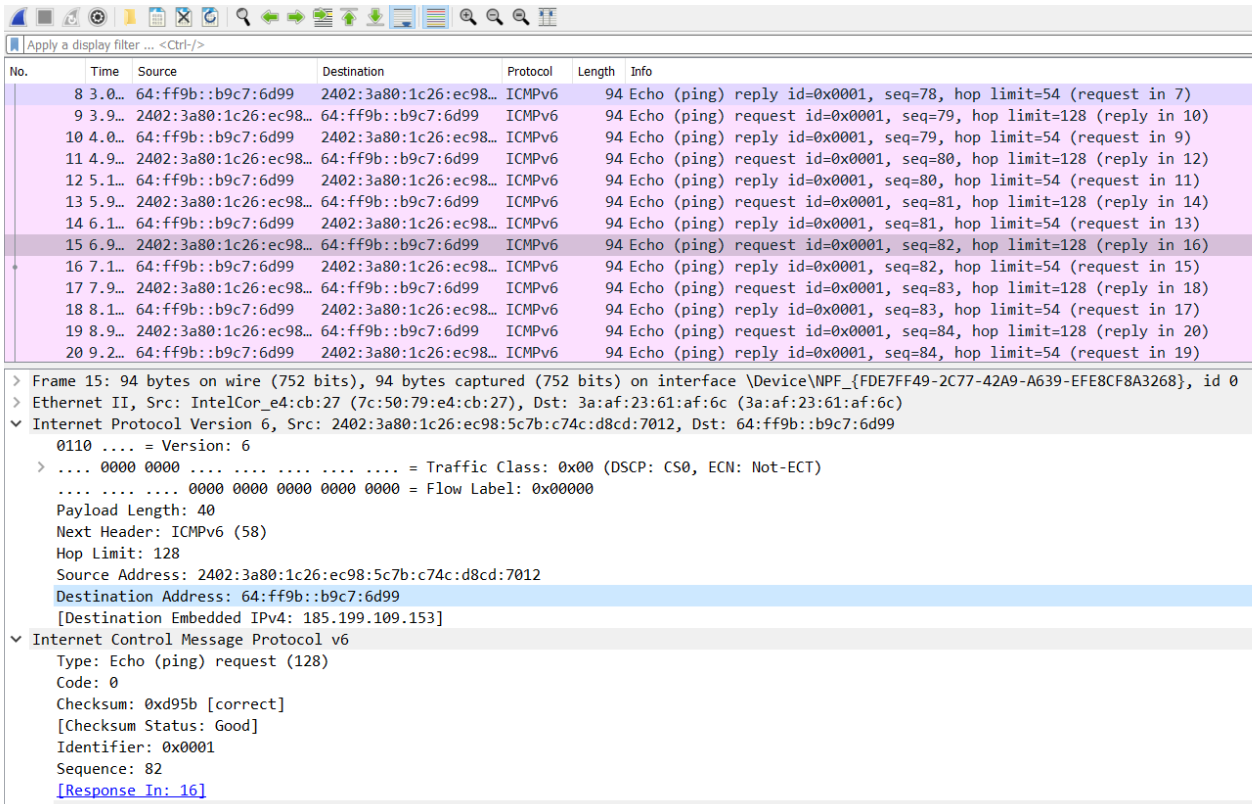
   192.168.43.217

4) Print the two HTTP messages displayed in above.

ICMP and Ping

The ping command is in c:\windows\system32, so type either “ping –n 10 hostname” or “c:\windows\system32\ping –n 10 hostname” in the MS-DOS command line (without quotation marks), where hostname is a host on another continent. If you’re outside of Asia, you may want to enter www.ust.hk for the Web server at Hong Kong University of Science and Technology. The argument “-n 10” indicates that 10 ping messages should be sent. Then run the Ping program by typing return.

```
C:\Windows\system32>ping -n 10 collegeek.com

Pinging collegeek.com [64:ff9b::b9c7:6d99] with 32 bytes of data:
Reply from 64:ff9b::b9c7:6d99: time=377ms
Reply from 64:ff9b::b9c7:6d99: time=178ms
Reply from 64:ff9b::b9c7:6d99: time=191ms
Reply from 64:ff9b::b9c7:6d99: time=208ms
Reply from 64:ff9b::b9c7:6d99: time=225ms
Reply from 64:ff9b::b9c7:6d99: time=238ms
Reply from 64:ff9b::b9c7:6d99: time=253ms
Reply from 64:ff9b::b9c7:6d99: time=272ms
Reply from 64:ff9b::b9c7:6d99: time=288ms
Reply from 64:ff9b::b9c7:6d99: time=305ms

Ping statistics for 64:ff9b::b9c7:6d99:
    Packets: Sent = 10, Received = 10, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 178ms, Maximum = 377ms, Average = 253ms
```



```
Apply a display filter ... <Ctrl-/>
No.   Time   Source              Destination         Protocol  Length  Info
    8 3.0…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=78, hop limit=54 (request in 7)
    9 3.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=79, hop limit=128 (reply in 10)
   10 4.0…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=79, hop limit=54 (request in 9)
   11 4.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=80, hop limit=128 (reply in 12)
   12 5.1…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=80, hop limit=54 (request in 11)
   13 5.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=81, hop limit=128 (reply in 14)
   14 6.1…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=81, hop limit=54 (request in 13)
   15 6.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=82, hop limit=128 (reply in 16)
   16 7.1…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=82, hop limit=54 (request in 15)
   17 7.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=83, hop limit=128 (reply in 18)
   18 8.1…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=83, hop limit=54 (request in 17)
   19 8.9…  2402:3a80:1c26:ec98… 64:ff9b::b9c7:6d99  ICMPv6    94 Echo (ping) request id=0x0001, seq=84, hop limit=128 (reply in 20)
   20 9.2…  64:ff9b::b9c7:6d99  2402:3a80:1c26:ec98… ICMPv6    94 Echo (ping) reply id=0x0001, seq=84, hop limit=54 (request in 19)

> Frame 15: 94 bytes on wire (752 bits), 94 bytes captured (752 bits) on interface \Device\NPF_{FDE7FF49-2C77-42A9-A639-EFE8CF8A3268}, id 0
> Ethernet II, Src: IntelCor_e4:cb:27 (7c:50:79:e4:cb:27), Dst: 3a:af:23:61:af:6c (3a:af:23:61:af:6c)
v Internet Protocol Version 6, Src: 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012, Dst: 64:ff9b::b9c7:6d99
    0110 .... = Version: 6
  > .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (DSCP: CS0, ECN: Not-ECT)
    .... .... .... 0000 0000 0000 0000 0000 = Flow Label: 0x00000
    Payload Length: 40
    Next Header: ICMPv6 (58)
    Hop Limit: 128
    Source Address: 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012
    Destination Address: 64:ff9b::b9c7:6d99
    [Destination Embedded IPv4: 185.199.109.153]
v Internet Control Message Protocol v6
    Type: Echo (ping) request (128)
    Code: 0
    Checksum: 0xd95b [correct]
    [Checksum Status: Good]
    Identifier: 0x0001
    Sequence: 82
    [Response In: 16]
```

Fig 1.3 – PING

DNS nslookup

The syntax is :

nslookup –option1 –option2 host-to-find dns-server

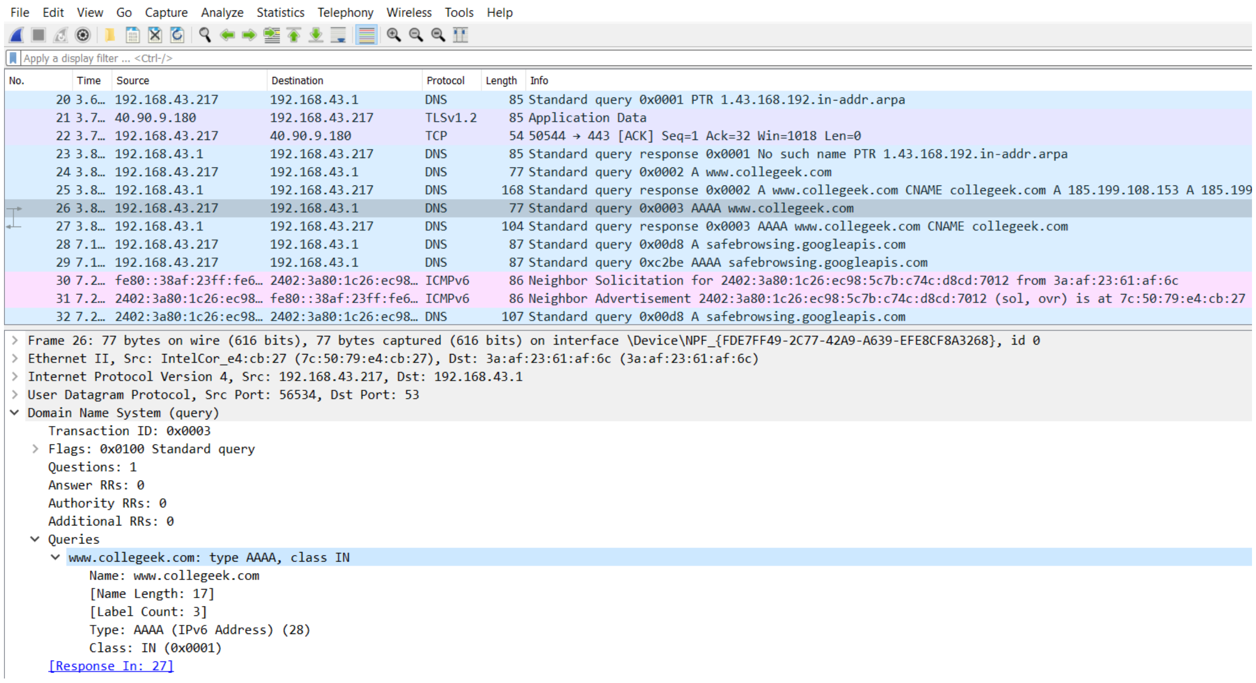
Try these commands :

nslookup www.mit.edu

nslookup –type=NS mit.edu

nslookup www.aiit.or.kr bitsy.mit.edu

```
C:\Windows\system32>nslookup www.collegeek.com
Server:  UnKnown
Address:  192.168.43.1

Non-authoritative answer:
Name:    collegeek.com
Addresses:  185.199.108.153
          185.199.111.153
          185.199.110.153
          185.199.109.153
Aliases:  www.collegeek.com
```



```
No.   Time  Source              Destination         Protocol  Length Info
   20 3.6…  192.168.43.217      192.168.43.1        DNS          85 Standard query 0x0001 PTR 1.43.168.192.in-addr.arpa
   21 3.7…  40.90.9.180         192.168.43.217      TLSv1.2      85 Application Data
   22 3.7…  192.168.43.217      40.90.9.180         TCP          54 50544 → 443 [ACK] Seq=1 Ack=32 Win=1018 Len=0
   23 3.8…  192.168.43.1        192.168.43.217      DNS          85 Standard query response 0x0001 No such name PTR 1.43.168.192.in-addr.arpa
   24 3.8…  192.168.43.217      192.168.43.1        DNS          77 Standard query 0x0002 A www.collegeek.com
   25 3.8…  192.168.43.1        192.168.43.217      DNS         168 Standard query response 0x0002 A www.collegeek.com CNAME collegeek.com A 185.199.108.153 A 185.199
   26 3.8…  192.168.43.217      192.168.43.1        DNS          77 Standard query 0x0003 AAAA www.collegeek.com
   27 3.8…  192.168.43.1        192.168.43.217      DNS         104 Standard query response 0x0003 AAAA www.collegeek.com CNAME collegeek.com
   28 7.1…  192.168.43.217      192.168.43.1        DNS          87 Standard query 0x00d8 A safebrowsing.googleapis.com
   29 7.1…  192.168.43.217      192.168.43.1        DNS          87 Standard query 0xc2be AAAA safebrowsing.googleapis.com
   30 7.2…  fe80::38af:23ff:fe6… 2402:3a80:1c26:ec98… ICMPv6     86 Neighbor Solicitation for 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012 from 3a:af:23:61:af:6c
   31 7.2…  2402:3a80:1c26:ec98… fe80::38af:23ff:fe6… ICMPv6     86 Neighbor Advertisement 2402:3a80:1c26:ec98:5c7b:c74c:d8cd:7012 (sol, ovr) is at 7c:50:79:e4:cb:27
   32 7.2…  2402:3a80:1c26:ec98… 2402:3a80:1c26:ec98… DNS       107 Standard query 0x00d8 A safebrowsing.googleapis.com

> Frame 26: 77 bytes on wire (616 bits), 77 bytes captured (616 bits) on interface \Device\NPF_{FDE7FF49-2C77-42A9-A639-EFE8CF8A3268}, id 0
> Ethernet II, Src: IntelCor_e4:cb:27 (7c:50:79:e4:cb:27), Dst: 3a:af:23:61:af:6c (3a:af:23:61:af:6c)
> Internet Protocol Version 4, Src: 192.168.43.217, Dst: 192.168.43.1
> User Datagram Protocol, Src Port: 56534, Dst Port: 53
v Domain Name System (query)
    Transaction ID: 0x0003
  > Flags: 0x0100 Standard query
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  v Queries
    v www.collegeek.com: type AAAA, class IN
        Name: www.collegeek.com
        [Name Length: 17]
        [Label Count: 3]
        Type: AAAA (IPv6 Address) (28)
        Class: IN (0x0001)
    [Response In: 27]
```

Fig 1.4 – nslookup DNS

ARP protocol

ARP protocol typically maintains a cache of IP-to-Ethernet address translation pairs on your comnputer The *arp* command (in both MSDOS and Linux/Unix) is used to view and manipulate the contents of this cache the *arp* command is used to view andmanipulate the ARP cache contents, while the ARP protocol defines the format and meaning of the messages sent and received, and defines the actions taken on message transmission and receipt.

The contents of the ARP cache on your computer

```
C:\Windows\system32>arp -a

Interface: 192.168.43.217 --- 0x11
  Internet Address      Physical Address      Type
  192.168.43.1          3a-af-23-61-af-6c     dynamic
  192.168.43.255        ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  224.0.0.252           01-00-5e-00-00-fc     static
  239.255.255.250       01-00-5e-7f-ff-fa     static
  255.255.255.255       ff-ff-ff-ff-ff-ff     static
```

Content in wireshark,

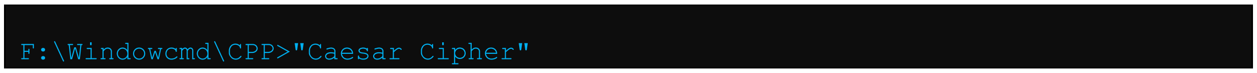
```
1.773   3a:af:23:61:af:6c     IntelCor_e4:cb:27     ARP    42     192.168.43.1 is at 3a:af:23:61:af:6c
1.761   IntelCor_e4:cb:27     3a:af:23:61:af:6c     ARP    42     Who has 192.168.43.1? Tell 192.168.43.217
```

Practical -2

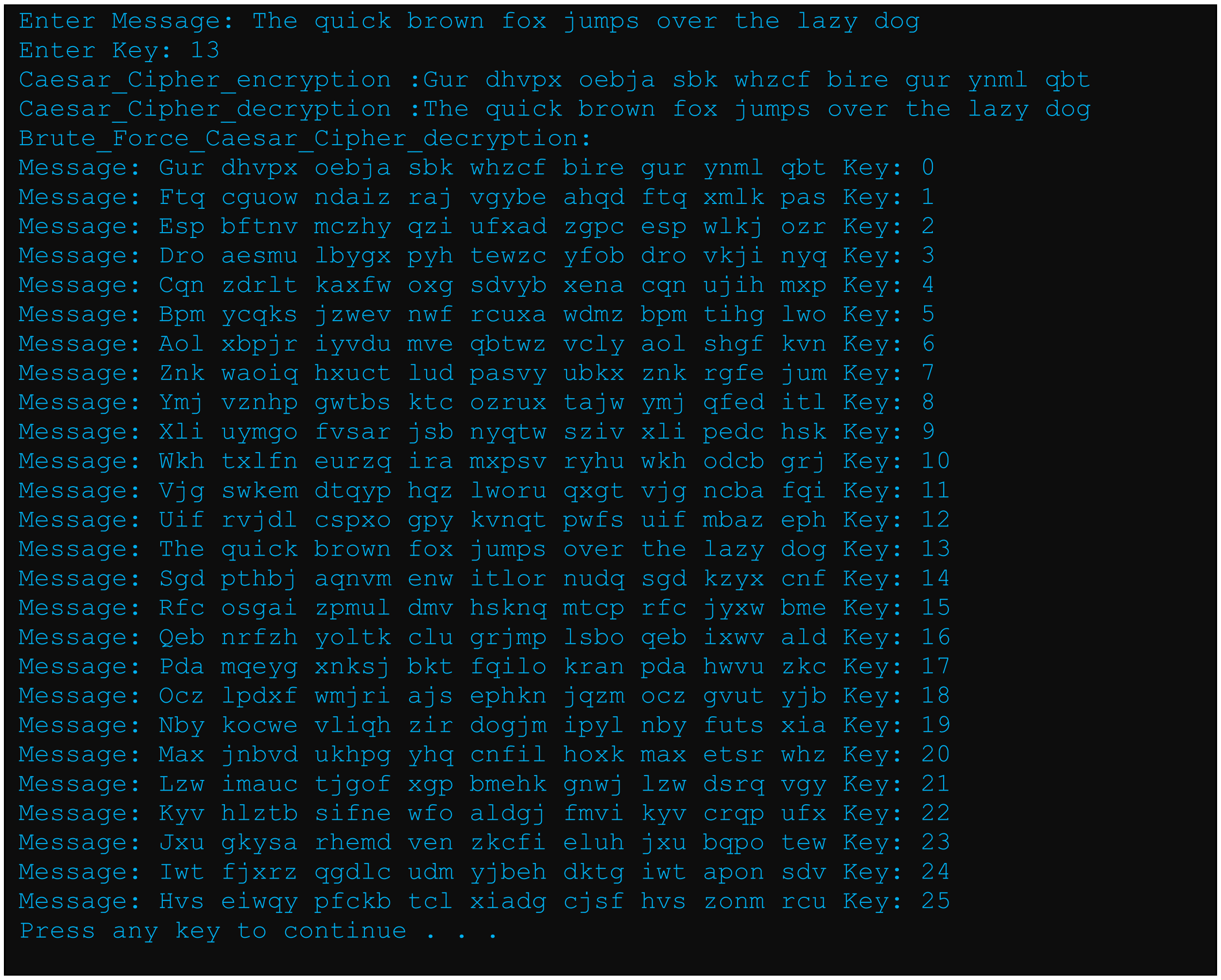
AIM: Implement Ceaser cipher encryption and decryption processes.

```
//============================================================================
// Name        : Caesar Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*
The Caesar cipher involves replacing each letter of the alphabet with the
letter standing three places further down the alphabet.

shift may be of any amount, so that the general Caesar algorithm is
C = E(k, p) = (p + k) mod 26
where k takes on a value in the range 1 to 25.
The decryption algorithm is simply
p = D(k, C) = (C - k) mod 26
*/
//============================================================================


#include <iostream>
#include <cctype>
#include <cmath>
#include <string>
using namespace std;

string Caesar_Cipher_encryption(string message,int k){
	int l = message.length();
	string ency = "";
	for(int i=0;i<l;++i){
		if(message[i]==' '){
			ency+=' ';
			continue;
		}
		if(isupper(message[i])){
			ency += 'A' + ((message[i] - 'A') + k) % 26;
		}
		else{
			ency += 'a' + ((message[i] - 'a') + k) % 26;
		}
	}
	return ency;
}
string Caesar_Cipher_decryption(string message,int k){
	int l = message.length();
```

```
        string ency = "";
        for(int i=0;i<l;++i){
                if(message[i]==' '){
                        ency+=' ';
                        continue;
                }
                if(isupper(message[i])){
                        int temp = (message[i] - 'A') - k;
                        ency += 'A' +  (temp<0?(temp+26):temp)  % 26;
                }
                else{
                        int temp = (message[i] - 'a') - k;
                        ency += 'a' +  (temp<0?(temp+26):temp) % 26;
                }
        }
        return ency;
}

string Brute_Force_Caesar_Cipher_decryption(string message){
        string dcy="";
        for(int i=0;i<26;++i){
                dcy+="Message: "+Caesar_Cipher_decryption(message,i)+" Key: 
"+to_string(i)+"\n";
        }
        return dcy;
}

int main(){

        string s;
        int key = 19;
        cout<<"Enter Message: ";
        getline(cin,s);
        cout<<"Enter Key: ";
        cin>>key;
        string p = Caesar_Cipher_encryption(s,key);
        cout<<"Caesar_Cipher_encryption :"<<p<<"\n";
        cout<<"Caesar_Cipher_decryption :"<<Caesar_Cipher_decryption(p,key)<<"\n";
        cout<<"Brute_Force_Caesar_Cipher_decryption:\n"<<Brute_Force_Caesar_Cipher_decr
yption(p);
        system("pause");
        return 0;
}
```



```
F:\Windowcmd\CPP>"Caesar Cipher"
```



```
Enter Message: The quick brown fox jumps over the lazy dog
Enter Key: 13
Caesar_Cipher_encryption :Gur dhvpx oebja sbk whzcf bire gur ynml qbt
Caesar_Cipher_decryption :The quick brown fox jumps over the lazy dog
Brute_Force_Caesar_Cipher_decryption:
Message: Gur dhvpx oebja sbk whzcf bire gur ynml qbt Key: 0
Message: Ftq cguow ndaiz raj vgybe ahqd ftq xmlk pas Key: 1
Message: Esp bftnv mczhy qzi ufxad zgpc esp wlkj ozr Key: 2
Message: Dro aesmu lbygx pyh tewzc yfob dro vkji nyq Key: 3
Message: Cqn zdrlt kaxfw oxg sdvyb xena cqn ujih mxp Key: 4
Message: Bpm ycqks jzwev nwf rcuxa wdmz bpm tihg lwo Key: 5
Message: Aol xbpjr iyvdu mve qbtwz vcly aol shgf kvn Key: 6
Message: Znk waoiq hxuct lud pasvy ubkx znk rgfe jum Key: 7
Message: Ymj vznhp gwtbs ktc ozrux tajw ymj qfed itl Key: 8
Message: Xli uymgo fvsar jsb nyqtw sziv xli pedc hsk Key: 9
Message: Wkh txlfn eurzq ira mxpsv ryhu wkh odcb grj Key: 10
Message: Vjg swkem dtqyp hqz lworu qxgt vjg ncba fqi Key: 11
Message: Uif rvjdl cspxo gpy kvnqt pwfs uif mbaz eph Key: 12
Message: The quick brown fox jumps over the lazy dog Key: 13
Message: Sgd pthbj aqnvm enw itlor nudq sgd kzyx cnf Key: 14
Message: Rfc osgai zpmul dmv hsknq mtcp rfc jyxw bme Key: 15
Message: Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald Key: 16
Message: Pda mqeyg xnksj bkt fqilo kran pda hwvu zkc Key: 17
Message: Ocz lpdxf wmjri ajs ephkn jqzm ocz gvut yjb Key: 18
Message: Nby kocwe vliqh zir dogjm ipyl nby futs xia Key: 19
Message: Max jnbvd ukhpg yhq cnfil hoxk max etsr whz Key: 20
Message: Lzw imauc tjgof xgp bmehk gnwj lzw dsrq vgy Key: 21
Message: Kyv hlztb sifne wfo aldgj fmvi kyv crqp ufx Key: 22
Message: Jxu gkysa rhemd ven zkcfi eluh jxu bqpo tew Key: 23
Message: Iwt fjxrz qgdlc udm yjbeh dktg iwt apon sdv Key: 24
Message: Hvs eiwqy pfckb tcl xiadg cjsf hvs zonm rcu Key: 25
Press any key to continue . . .
```

Practical – 3

AIM: Implement monoalphabetic cipher encryption and decryption processes.

```
//============================================================================
// Name        : Monoalphabetic Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*

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

If, instead, the "cipher" line can be any permutation of the 26 alphabetic characters,
then there are 26! or greater than 4 * 10^26 possible keys. This is 10 orders of magnitude
greater than the key space for DES and would seem to eliminate brute-force
techniques for cryptanalysis.

Such an approach is referred to as a monoalphabetic
substitution cipher, because a single cipher alphabet (mapping from plain alphabet
to cipher alphabet) is used per message.
*/
//============================================================================

#include<iostream>
using namespace std;

string Monoalphabetic_Cipher_encryption(string message){
	int n = message.length(),temp;
	string key="FDSAVCXZGHJKLREWQTYUIOPMNB";
	string ency="";
	for(int i=0;i<n;++i){
		temp = message[i]-'a';
		ency+=key[temp];
	}
	return ency;
}

string Monoalphabetic_Cipher_decryption(string message){
	int n = message.length(),temp;
	string key = "dzfboaijuklmxyvwqncrtepgsh";
	string ency="";
	for(int i=0;i<n;++i){
		temp = message[i]-'A';
```

```
			ency+=key[temp];
		}
		return ency;
}
int main(){
		string s;
		cout<<"Enter Message: ";
		getline(cin,s);
		string p = Monoalphabetic_Cipher_encryption(s);
		cout<<"Monoalphabetic_Cipher_encryption :"<<p<<"\n";
		cout<<"Monoalphabetic_Cipher_decryption: 
"<<Monoalphabetic_Cipher_decryption(p)<<"\n";
		system("pause");
		return 0;
}
```

```
F:\Windowcmd\CPP>"Monoalphabetic Cipher"
Enter Message: thequickbrownfoxjumpsoverthelazydog
Monoalphabetic_Cipher_encryption :UZVQIGSJDTEPRCEMHILWYEOVTUZVKFBNAEX
Monoalphabetic_Cipher_decryption: thequickbrownfoxjumpsoverthelazydog
Press any key to continue . . .
```

Practical – 4

AIM: Implement polyalphabetic cipher encryption and decryption processes.

```
//============================================================================
// Name        : Vigenère Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*

polyalphabetic ciphers
is the Vigenère cipher. In this scheme, the set of related monoalphabetic substitution
rules consists of the 26 Caesar ciphers with shifts of 0 through 25. Each cipher is
denoted by a key letter, which is the ciphertext letter that substitutes for the plaintext
letter a.

Thus, the first letter of the key is added to the first letter of the plaintext, mod 26,
the second letters are added, and so on through the first m letters of the plaintext.
For the next m letters of the plaintext, the key letters are repeated. This process
continues until all of the plaintext sequence is encrypted. A general equation of the
encryption process is

                                    C[i] = (p[i] + k[i mod m]) mod 26

Similarly, decryption is a generalization of above,

                                    p[i] = (C[i] - k[i mod m]) mod 26

key : deceptive

key    : deceptivedeceptivedeceptive
Message: wearediscoveredsaveyourself
Cipher : zicvtwqngrzgvtwavzhcqyglmgj
*/
//============================================================================

#include<iostream>
#include <cctype>
using namespace std;

string Vigenere_Cipher_encryption(string message,string key){
        int n = message.length(),m = key.length();
        int k=0;
        string ency="";
```

```
    for(int i=0;i<n;++i,++k){
        if(message[i]==' '){
            ency+=' ';
            --k;
            continue;
        }
        ency += 'a' + ( (message[i] -'a') + (key[(k%m)] - 'a') ) % 26;
    }
    return ency;
}

string Vigenere_Cipher_decryption(string message,string key){
    int n = message.length(),m = key.length();
    int k=0,temp;
    string ency="";
    for(int i=0;i<n;++i,++k){
        if(message[i]==' '){
            ency+=' ';
            --k;
            continue;
        }
        temp = (message[i] -'a') - (key[(k%m)] - 'a');
        ency += 'a' + (temp<0?(temp+26):temp) % 26;
    }
    return ency;
}
int main(){
    string s;
    string key="deceptive";
    cout<<"Enter Message: ";
    getline(cin,s);
    string p = Vigenere_Cipher_encryption(s,key);
    cout<<"Vigenere_Cipher_encryption :"<<p<<"\n";
    cout<<"Vigenere_Cipher_decryption: "<<Vigenere_Cipher_decryption(p,key)<<"\n";
    system("pause");
    return 0;
}
```

```
F:\Windowcmd\CPP>"Vigenere Cipher"
Enter Message: the quick brown fox jumps over the lazy dog
Vigenere_Cipher_encryption :wlg ujbkf fusyr uhf eyptu skxz olh pcdn wwb
Vigenere_Cipher_decryption: the quick brown fox jumps over the lazy dog
Press any key to continue . . .
```

Practical – 5

AIM: Implement play fair cipher encryption and decryption processes.

```
//============================================================================
// Name        : Playfair Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*

The Playfair algorithm is based on the use of a 5 * 5 matrix of letters constructed
using a keyword.
*/
//============================================================================

#include<iostream>
#include<cstring>
#include <cctype>
using namespace std;

// 5 * 5 matrix of letters constructed using a key
void assign_matrix(char matrix[][5],string key){
	int n = key.length(),j=0,i=0;

	bool arr[26] = {true};
	char ch[26];
	memset(arr,true,sizeof(arr));

	for(i=0;i<n;++i){
		ch[i]=key[i];
		int t = key[i] - 'a';
		arr[t]=false;
	}
	j=i;
	for(int i=0;i<26;++i){
		if(arr[i]){
			arr[i]=false;
			char temp='a'+i;
			if(temp=='j')continue;
			ch[j]=temp;
			++j;
		}
	}
	int t=0;
```

```
        for(i=0;i<5;++i){
                for(j=0;j<5;++j){
                        matrix[i][j]=ch[t];
                        ++t;
                }
        }
}
//in st ru me nt s ->length = 11;
//01 23 45 67 89 10
string Playfair_Cipher_encryption(char matrix[][5],string message){
        string origin = message;
        int n = message.length();
        int k=0;
        string ency="";
        bool is_odd = false;

        // check odd?
        if(n%2!=0){
                is_odd=true;
        }


        while(true){
                //on the spot change 'j' to 'i'
                if(k<n && message[k]=='j'){
                        message[k]='i';
                        if(k+1<n && message[k+1]=='j')
                                message[k+1]='i';
                }
                else if(k+1<n && message[k+1]=='j'){
                        message[k+1]='i';
                        if(k+2<n && message[k+2]=='j')
                                message[k+2]='i';
                }
                // both character is same
                else if(message[k]==message[k+1]){
                        string temp1 = message.substr(0,k+1);
                        string temp2 = message.substr(k+1);
                        if(message[k]!='x'){
                                message = temp1 + 'x' + temp2;
                                n= message.length();
                                if(n%2!=0){
                                        is_odd=true;
                                }else{is_odd=false;}
                        }
                        else{
                                message = temp1 + 'z' + temp2;
                                n= message.length();
                                if(n%2!=0){
                                        is_odd=true;
                                }else{is_odd=false;}
                        }
                }
                // at last,if length is odd than add 'z' if it is not last character.
                else if(is_odd && k==n-1){
                        if(message[n-1]!='z')message+='z';
                        else message+='x';
                        n=message.length();
                }
                else{
                        int x1,y1,x2,y2;
                        for(int i=0;i<5;++i){
                                for(int j=0;j<5;++j){
                                        if(message[k]==matrix[i][j]){
                                                x1=i;
                                                y1=j;
                                        }
                                        if(message[k+1]==matrix[i][j]){
                                                x2=i;
                                                y2=j;
                                        }
                                }
                        }

                        // both letter in same column
                        if(y1==y2){
                                ency+=matrix[((x1+1)%5)][y1];
                                ency+=matrix[((x2+1)%5)][y1];
                        }
                        // both letter in same row
                        else if(x1==x2){
                                ency+=matrix[x1][((y1+1)%5)];
                                ency+=matrix[x1][((y2+1)%5)];
                        }
                        // if above is not true
                        else{
                                ency+=matrix[x1][y2];
                                ency+=matrix[x2][y1];
                        }
                        k+=2;
                }
                if(k>=n)break;
```

```
		}
		return ency;
}

string Playfair_Cipher_decryption(char matrix[][5],string message){
		int x1,y1,x2,y2,k=0;
		string ency="";
		int n = message.length();
		while(true){
				for(int i=0;i<5;++i){
						for(int j=0;j<5;++j){
								if(message[k]==matrix[i][j]){
										x1=i;
										y1=j;
								}
								if(message[k+1]==matrix[i][j]){
										x2=i;
										y2=j;
								}
						}
				}

				// both letter in same column
				if(y1==y2){
						ency+=matrix[((x1-1)<0?(x1-1+5):(x1-1))][y1];
						ency+=matrix[((x2-1)<0?(x2-1+5):(x2-1))][y1];
				}
				// both letter in same row
				else if(x1==x2){
						ency+=matrix[x1][((y1-1)<0?(y1-1+5):(y1-1))];
						ency+=matrix[x1][((y2-1)<0?(y2-1+5):(y2-1))];
				}
				// if above is not true
				else{
						ency+=matrix[x1][y2];
						ency+=matrix[x2][y1];
				}
				k+=2;
				if(k>=n)break;
		}
		return ency;
}

int main(){
		char matrix[5][5];
		string key = "monarchy";
```

```
        string s;
        cout<<"Enter Message: ";
        getline(cin,s);

        assign_matrix(matrix,key);
        for(int i=0;i<5;++i){
                for(int j=0;j<5;++j){
                        cout<<matrix[i][j]<<" ";
                }
                cout<<"\n";
        }
        string p = Playfair_Cipher_encryption(matrix,s);
        cout<<"Playfair_Cipher_encryption: "<<p<<"\n";
        cout<<"Playfair_Cipher_decryption: "<<Playfair_Cipher_decryption(matrix,p);
        return 0;
}
```

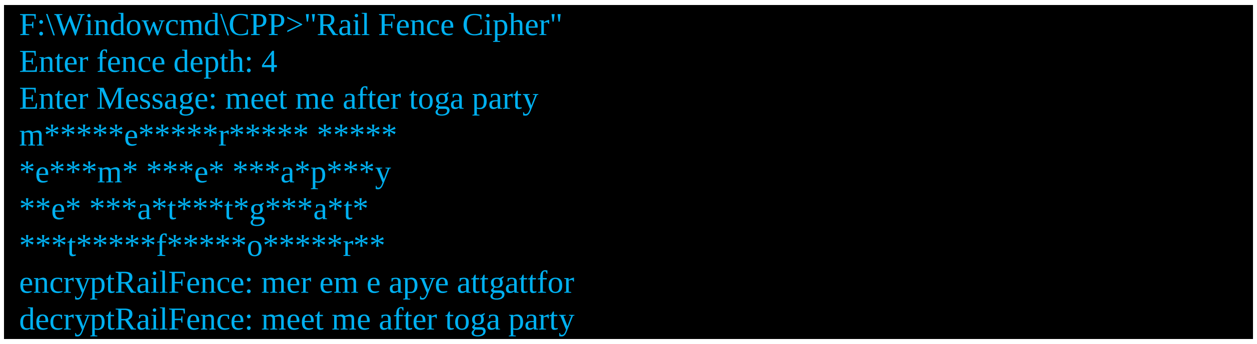
```
F:\Windowcmd\CPP>"Playfair Cipher"
Enter Message: thequickbrownfoxjumpsoverthelazydog
m o n a r
c h y b d
e f g i k
l p q s t
u v w x z
Playfair_Cipher_encryption: pdglxededanvogavexolpaufdzcfsmwdhrkw
Playfair_Cipher_decryption: thequickbrownfoxiumpsoverthelazydogz
F:\Windowcmd\CPP>"Playfair Cipher"
Enter Message: xxeroxusingzzmachine
m o n a r
c h y b d
e f g i k
l p q s t
u v w x z
Playfair_Cipher_encryption: zuuimnzvxsyquzurmbbfmg
Playfair_Cipher_decryption: xzxeroxusingzxzmachine
```

Practical – 6

AIM: Implement rail fence cipher encryption and decryption processes

```
//============================================================================
// Name        : Rail Fence Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*

the Rail Fence technique, in which the plaintext is
written down as a sequence of diagonals and then read off as a sequence of
rows.For example, to encipher the message "meet me after the toga party" with
a railfence of depth 2,

we write the following:
m e m a t r h t g p r y
 e t e f e t e o a a t

The encrypted message is
mematrhtgpryetefeteoaat

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

// function to encrypt a message
string encryptRailFence(string text, int key){
      // create the matrix to cipher plain text
      // key = rows , length(text) = columns
      int n = text.length();
      char rail[key][n];

      // filling the rail matrix to distinguish filled
      // spaces from blank ones
      for (int i=0; i < key; i++)
            for (int j = 0; j < n; j++)
                  rail[i][j] = '\n';

      // to find the direction
      bool dir_down = false;
      int row = 0, col = 0;

      for (int i=0; i < n; i++)
      {
            // check the direction of flow
            // reverse the direction if we've just
            // filled the top or bottom rail
            if (row == 0 || row == key-1)
                  dir_down = !dir_down;

            // fill the corresponding alphabet
            rail[row][col++] = text[i];

            // find the next row using direction flag
            dir_down?row++ : row--;
      }

      //now we can construct the cipher using the rail matrix
      string result;
      for (int i=0; i < key; i++){
            for (int j=0; j < n; j++){
                //cout<<((rail[i][j]=='*')?'n':rail[i][j]);
                  if (rail[i][j]!='\n')
                        result.push_back(rail[i][j]);
            }
            //cout<<"\n";
      }
      return result;
}

string decryptRailFence(string cipher, int key){
      // create the matrix to cipher plain text
      // key = rows , length(text) = columns
      int n = cipher.length();
      char rail[key][n];

      // filling the rail matrix to distinguish filled
      // spaces from blank ones
      for (int i=0; i < key; i++)
            for (int j=0; j < n; j++)
                  rail[i][j] = '\n';

      // to find the direction
      bool dir_down;

      int row = 0, col = 0;

      // mark the places with '*'
      for (int i=0; i < n; i++)
      {
            // check the direction of flow
            if (row == 0)
                  dir_down = true;
            if (row == key-1)
                  dir_down = false;

            // place the marker
            rail[row][col++] = '*';

            // find the next row using direction flag
            dir_down?row++ : row--;
      }

      // now we can construct the fill the rail matrix
      int index = 0;
      for (int i=0; i<key; i++)
            for (int j=0; j<n; j++)
                  if (rail[i][j] == '*' && index<n)
```

```
                        rail[i][j] = cipher[index++];

        // now read the matrix in zig-zag manner to construct
        // the resultant text
        string result;

        row = 0, col = 0;
        for (int i=0; i< n; i++)
        {
                // check the direction of flow
                if (row == 0)
                        dir_down = true;
                if (row == key-1)
                        dir_down = false;

                // place the marker
                if (rail[row][col] != '*')
                        result.push_back(rail[row][col++]);

                // find the next row using direction flag
                dir_down?row++: row--;
        }
        return result;
}

//driver program to check the above functions
int main(){
     string message;
     int fence_depth;
     cout<<"Enter fence depth: ";
     cin>>fence_depth; // key for row technique encryption
       cout<<"Enter Message: ";
     cin.ignore(numeric_limits<streamsize>::max(),'\n'); // discards the input
buffer
       getline(cin,message); // get message    string ency =
encryptRailFence(message,fence_depth);
     cout<<"encryptRailFence: "<<ency<<"\n";
     cout<<"decryptRailFence: "<<decryptRailFence(ency,fence_depth);
       return 0;
}
```



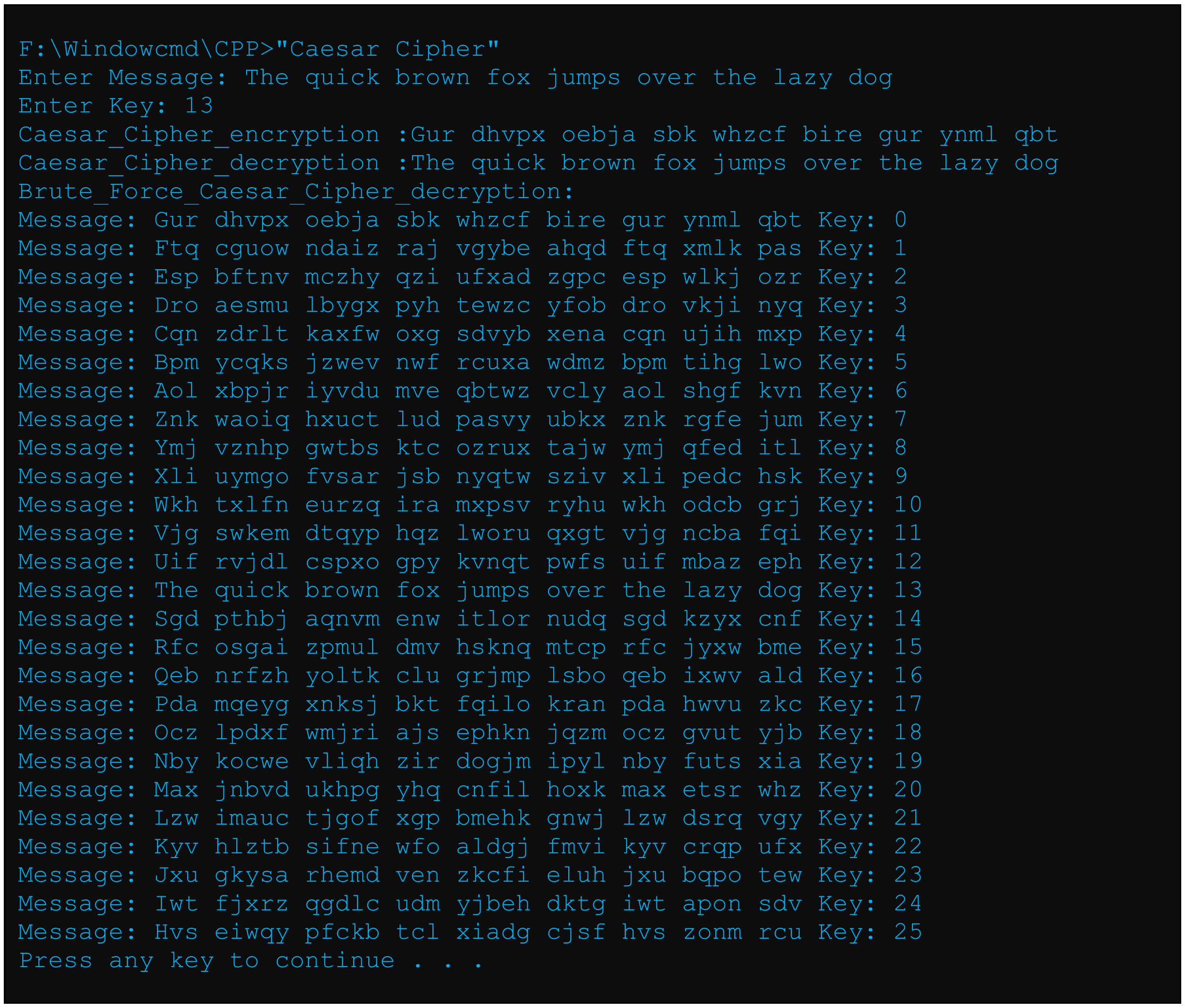
```
F:\Windowcmd\CPP>"Rail Fence Cipher"
Enter fence depth: 4
Enter Message: meet me after toga party
m*****e*****r***** *****
*e***m* ***e* ***a*p***y
**e* ***a*t***t*g***a*t*
***t*****f*****o*****r**
encryptRailFence: mer em e apye attgattfor
decryptRailFence: meet me after toga party
```

Fig6.1 – Rail Fence Cipher

Practical -7

AIM: Create a program for cryptanalysis using brute-force approach on Ceaser cipher.

```
//==================================================================
// Name        : Caesar Cipher.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*
The Caesar cipher involves replacing each letter of the alphabet with the
letter standing three places further down the alphabet.

shift may be of any amount, so that the general Caesar algorithm is
C = E(k, p) = (p + k) mod 26
where k takes on a value in the range 1 to 25.
The decryption algorithm is simply
p = D(k, C) = (C - k) mod 26
*/
//==================================================================


#include <iostream>
#include <cctype>
#include <cmath>
#include <string>
using namespace std;

string Caesar_Cipher_encryption(string message,int k){
	int l = message.length();
	string ency = "";
	for(int i=0;i<l;++i){
		if(message[i]==' '){
			ency+=' ';
			continue;
		}
		if(isupper(message[i])){
			ency += 'A' + ((message[i] - 'A') + k) % 26;
		}
		else{
			ency += 'a' + ((message[i] - 'a') + k) % 26;
		}
	}
	return ency;
}
string Caesar_Cipher_decryption(string message,int k){
	int l = message.length();
	string ency = "";
	for(int i=0;i<l;++i){
		if(message[i]==' '){
			ency+=' ';
			continue;
		}
		if(isupper(message[i])){
			int temp = (message[i] - 'A') - k;
			ency += 'A' +  (temp<0?(temp+26):temp)  % 26;
		}
		else{
			int temp = (message[i] - 'a') - k;
			ency += 'a' +  (temp<0?(temp+26):temp) % 26;
		}
	}
	return ency;
}

string Brute_Force_Caesar_Cipher_decryption(string message){
	string dcy="";
	for(int i=0;i<26;++i){
		dcy+="Message: "+Caesar_Cipher_decryption(message,i)+" Key: 
"+to_string(i)+"\n";
	}
	return dcy;
}

int main(){

	string s;
	int key = 19;
	cout<<"Enter Message: ";
	getline(cin,s);
	cout<<"Enter Key: ";
	cin>>key;
	string p = Caesar_Cipher_encryption(s,key);
	cout<<"Caesar_Cipher_encryption :"<<p<<"\n";
	cout<<"Caesar_Cipher_decryption :"<<Caesar_Cipher_decryption(p,key)<<"\n";
	cout<<"Brute_Force_Caesar_Cipher_decryption:\n"<<Brute_Force_Caesar_Cipher_decr
yption(p);
	system("pause");
	return 0;
}
```

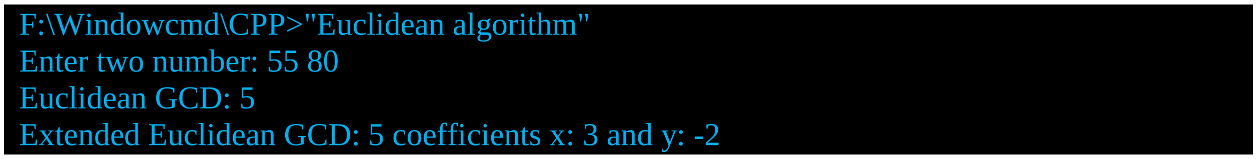


```
F:\Windowcmd\CPP>"Caesar Cipher"
Enter Message: The quick brown fox jumps over the lazy dog
Enter Key: 13
Caesar_Cipher_encryption :Gur dhvpx oebja sbk whzcf bire gur ynml qbt
Caesar_Cipher_decryption :The quick brown fox jumps over the lazy dog
Brute_Force_Caesar_Cipher_decryption:
Message: Gur dhvpx oebja sbk whzcf bire gur ynml qbt Key: 0
Message: Ftq cguow ndaiz raj vgybe ahqd ftq xmlk pas Key: 1
Message: Esp bftnv mczhy qzi ufxad zgpc esp wlkj ozr Key: 2
Message: Dro aesmu lbygx pyh tewzc yfob dro vkji nyq Key: 3
Message: Cqn zdrlt kaxfw oxg sdvyb xena cqn ujih mxp Key: 4
Message: Bpm ycqks jzwev nwf rcuxa wdmz bpm tihg lwo Key: 5
Message: Aol xbpjr iyvdu mve qbtwz vcly aol shgf kvn Key: 6
Message: Znk waoiq hxuct lud pasvy ubkx znk rgfe jum Key: 7
Message: Ymj vznhp gwtbs ktc ozrux tajw ymj qfed itl Key: 8
Message: Xli uymgo fvsar jsb nyqtw sziv xli pedc hsk Key: 9
Message: Wkh txlfn eurzq ira mxpsv ryhu wkh odcb grj Key: 10
Message: Vjg swkem dtqyp hqz lworu qxgt vjg ncba fqi Key: 11
Message: Uif rvjdl cspxo gpy kvnqt pwfs uif mbaz eph Key: 12
Message: The quick brown fox jumps over the lazy dog Key: 13
Message: Sgd pthbj aqnvm enw itlor nudq sgd kzyx cnf Key: 14
Message: Rfc osgai zpmul dmv hsknq mtcp rfc jyxw bme Key: 15
Message: Qeb nrfzh yoltk clu grjmp lsbo qeb ixwv ald Key: 16
Message: Pda mqeyg xnksj bkt fqilo kran pda hwvu zkc Key: 17
Message: Ocz lpdxf wmjri ajs ephkn jqzm ocz gvut yjb Key: 18
Message: Nby kocwe vliqh zir dogjm ipyl nby futs xia Key: 19
Message: Max jnbvd ukhpg yhq cnfil hoxk max etsr whz Key: 20
Message: Lzw imauc tjgof xgp bmehk gnwj lzw dsrq vgy Key: 21
Message: Kyv hlztb sifne wfo aldgj fmvi kyv crqp ufx Key: 22
Message: Jxu gkysa rhemd ven zkcfi eluh jxu bqpo tew Key: 23
Message: Iwt fjxrz qgdlc udm yjbeh dktg iwt apon sdv Key: 24
Message: Hvs eiwqy pfckb tcl xiadg cjsf hvs zonm rcu Key: 25
Press any key to continue . . .
```

Practical – 8

AIM: Implement Euclidean algorithm to find gcd and Extended Euclidian algorithm to find multiplicative inverse.

```
//============================================================================
// Name        : Euclidean algorithm.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*

Given two non-negative integers a and b, we have to find their GCD (greatest
common divisor), i.e. the largest number which is a divisor of both a and b.

When one of the numbers is zero, while the other is non-zero, their greatest
common divisor, by definition, is the second number. When both numbers are
zero, their greatest common divisor is undefined (it can be any arbitrarily
large number), but we can define it to be zero. Which gives us a simple rule:
if one of the numbers is zero, the greatest common divisor is the other
number.

The Euclidean algorithm, discussed below, allows to find the greatest common
divisor of two numbers a and b in O (logmin(a,b)).

While the Euclidean algorithm calculates only the greatest common divisor
(GCD) of two integers a and b, the extended version also finds a way to
represent GCD in terms of a and b, i.e. coefficients x and y

for which:
a*x+b*y=gcd(a,b)

It's important to note, that we can always find such a representation, for
instance gcd(55,80)=5 therefore we can represent 5 as a linear combination
with the terms 55 and 80: 55*3+80*(-2)=5
*/
//============================================================================
#include <iostream>
#include <tuple>
using namespace std;

int gcd (int a, int b) {
    int temp;
    while (b) {
        a %= b;
        temp = a;
        a = b;
        b = temp;
    }
    return a;
}

int gcd(int a, int b, int& x, int& y) {
    x = 1, y = 0;
    int x1 = 0, y1 = 1, a1 = a, b1 = b;
    while (b1) {
        int q = a1 / b1;
        tie(x, x1) = make_tuple(x1, x - q * x1);
        tie(y, y1) = make_tuple(y1, y - q * y1);
        tie(a1, b1) = make_tuple(b1, a1 - q * b1);
    }
    return a1;
}

int main() {
      int num1,num2;
      int x,y;
      cout<<"Enter two number: ";
    cin>>num1>>num2;
    cout<<"Euclidean GCD: "<<gcd(num1,num2)<<"\n";
    cout<<"Extended Euclidean GCD: "<<gcd(num1,num2,x,y)<<" coefficients x:
"<<x<<" and y: "<<y;
}
```



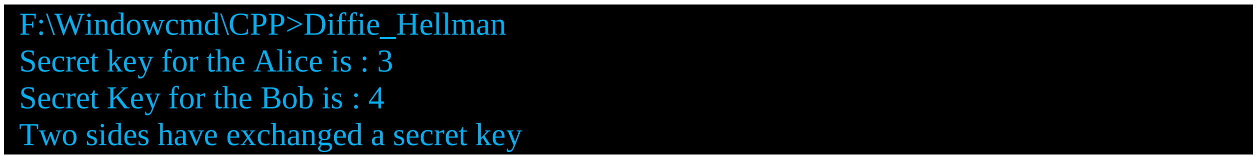
```
F:\Windowcmd\CPP>"Euclidean algorithm"
Enter two number: 55 80
Euclidean GCD: 5
Extended Euclidean GCD: 5 coefficients x: 3 and y: -2
```

Fig 8.1

Practical – 9

AIM: Implement diffie-hellman key exchange algorithm to exchange keys.

```
//============================================================================
// Name        : Diffie_Hellman.cpp
// Author      : SidPro
// Version     : 1.0
// Description :
/*
this is public-key cryptography and is generally
referred to as Diffie-Hellman key exchange

The purpose of the algorithm is to enable two users to securely exchange a
key that can then be used for subsequent symmetric encryption of messages.
The algorithm itself is limited to the exchange of secret values

For any integer b and a primitive root a of prime number p, we can find a
unique exponent i such that b = a^i (mod p) where 0 <=i<=(p - 1)
The exponent i is referred to as the discrete logarithm of b for the base a,
mod p.We express this value as dloga,p(b).

*/
//============================================================================
#include <iostream>
#include <cmath>
using namespace std;


// Power function to return value of a^b(mod q)
long long int power(long long int a, long long int b,long long int q){
    if (b == 1)return a;
    else return (((long long int)pow(a, b)) % q);
}

int main() {
    //there are two publicly known numbers: a prime number q and an integer a
that is a primitive root of q.
      //Suppose the users A and B wish to create a shared key.
    long long int a=2,q=11;

      // Alice generates a private key XA such that XA<q
      long long int XA=8,YA,KA;
      //Alice calculates a public key YA = a^XA mod q
      YA = power(a,XA,q);


      // Bob generates a private key XB such that XB<q
      long long int XB=4,YB,KB;
      // Bob calculates a public key YB = a^XB mod q
      YB = power(a,XB,q);

      // Alice receives Bob’s public key YB in plaintext
      // Alice calculates shared secret key K = YB^XA mod q
      KA = power(YB,XA,q);
      // Bob receives Alice’spublic key YA in plaintext
      // Bob calculates shared secret key K = YA^XB mod q
      KB = power(YA,XB,q);

      /*
      above two calculations produce identical results:
    K = (YB)^XA mod q
      = (a^XB mod q)^XA mod q
      = (a^XB)^XA mod q by the rules of modular arithmetic
      = a^(XB*XA) mod q
      = (a^XA)^XB mod q
      = (a^XA mod q)^XB mod q
      = (YA)^XB mod q
     The result is that the two sides have exchanged a secret value

      */
      cout<<"Secret key for the Alice is : "<<KA<<"\n";
    cout<<"Secret Key for the Bob is : "<<KB<<"\n";
    cout<<"Two sides have exchanged a secret key";

      return 0;
}
```



```
F:\Windowcmd\CPP>Diffie_Hellman
Secret key for the Alice is : 3
Secret Key for the Bob is : 4
Two sides have exchanged a secret key
```

Fig 9.1

Practical - 10

AIM: Study various hashing algorithms for message integrity

Message Digest (MD):

A group of hashing algorithms used in cryptography and developed by Rivest. The term “message digest” refers to a short string or hash value of fixed length that is computed from the longer variable-length message being hashed by the algorithm. The important message digest (MD) algorithms include MD2, MD4, and MD5, all of which produce a 128-bit hash value. The MD algorithms are commonly used to generate a digital signature from a message.

MD2 was developed in 1989 for 8-bit encoders. It pads the message to be encoded until it is a multiple of 16 bytes in length, appends a 16-byte checksum, and computes the hash.

MD4 was developed in 1990 for 32-bit encoders. It pads the message to be encoded until it is 56 bytes short of being a multiple of 512 bytes, appends an 8-byte message length value, and iteratively hashes the message in three rounds. MD4 can be broken fairly easily in a dedicated cryptographic attempt. It is implemented in the Microsoft Challenge Handshake Authentication Protocol (MS-CHAP) supported by the Remote Access Service (RAS) on Microsoft Windows NT.

MD5 was developed in 1991 for 32-bit encoders and is an extension of MD4. It uses four rounds of hashing instead of three. It is fairly difficult to crack. Windows NT RAS Client supports MD5-CHAP for connecting to third-party Point-to-Point Protocol (PPP) servers supporting MD5 authentication, but Windows NT RAS Server does not. However, Service Pack 3 for Windows NT provides limited support for MD5-CHAP PPP authentication.

The MessageDigest class provides a method named getInstance(). This method accepts a String variable specifying the name of the algorithm to be used and returns a MessageDigest object implementing the specified algorithm.

```
import java.security.MessageDigest;
import java.util.Scanner;

public class MessageDigestExample {
   public static void main(String args[]) throws Exception{
      //Reading data from user
      Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the message");
        String message = sc.nextLine();

        //Creating the MessageDigest object
        MessageDigest md = MessageDigest.getInstance("SHA-256");

        //Passing data to the created MessageDigest Object
        md.update(message.getBytes());

        //Compute the message digest
        byte[] digest = md.digest();
        System.out.println(digest);

        //Converting the byte array in to HexString format
        StringBuffer hexString = new StringBuffer();

        for (int i = 0;i<digest.length;i++) {
           hexString.append(Integer.toHexString(0xFF & digest[i]));
        }
        System.out.println("Hex format : " + hexString.toString());
    }
}
```

Secure Hash Algorithms:

Secure Hash Algorithms, also known as SHA, are a family of cryptographic functions designed to keep data secured. It works by transforming the data using a hash function: an algorithm that consists of bitwise operations, modular additions, and compression functions. The hash function then produces a fixed-size string that looks nothing like the original. These algorithms are designed to be one-way functions, meaning that once they're transformed into their respective hash values, it's virtually impossible to transform them back into the original data. A few algorithms of interest are SHA-1, SHA-2, and SHA-3, each of which was successively designed with increasingly stronger encryption in response to hacker attacks. SHA-0, for instance, is now obsolete due to the widely exposed vulnerabilities.

A common application of SHA is to encrypting passwords, as the server side only needs to keep track of a specific user's hash value, rather than the actual password. This is helpful in case an attacker hacks the database, as they will only find the hashed functions and not the actual passwords, so if they were to input the hashed value as a password, the hash function will convert it into another string and subsequently deny access. Additionally, SHAs exhibit the avalanche effect, where the modification of very few letters being encrypted causes a big change in output; or conversely, drastically different strings produce similar hash values. This effect causes hash values to not give any information regarding the input string, such as its original length. In addition, SHAs are also used to detect the tampering of data by attackers, where if a text file is slightly changed and barely noticeable, the modified file's hash value will be different than the original file's hash value, and the tampering will be rather noticeable.

Common Attacks on SHA:

Cryptography wouldn't be as quickly developed if it weren't for the attacks that compromise their effectiveness. One of the most common attacks is known as the primeage attack, where pre-computed tables of solutions are used in a brute-force manner in order to crack passwords. The solution against these kinds of attacks is to compose a hash function that would take an attacker an exorbitant amount of resources, such as millions of dollars or decades of work, to find a message corresponding to a given hash value.

Most attacks penetrating SHA-1 are collision attacks, where a non-sensical message produces the same hash value as the original message. Generally, this takes time proportional to 2^ {n/2} to complete, where n is the length of the message. This is the reason the message digests have increased in length from 160-bit digests in SHA-1 to 224- or 256-bit digests in SHA-2.

Other attacks exist that attempt to exploit mathematical properties in order to crack hash functions. Amongst these is the birthday attack, where higher likelihood of collisions are found when using random attacks with a fixed number of letter combinations (see the pigeonhole principle), or the rainbow table attack, where a pre-computed hash table is used to reverse a hash function in order to crack passwords.

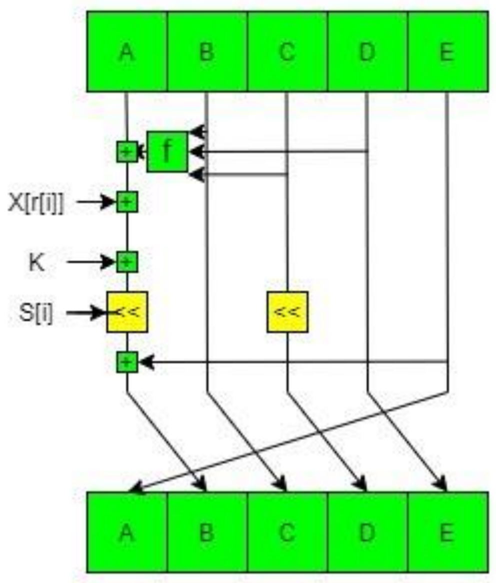
RIPEMD:

RIPEMD(RACE Integrity Primitives Evaluation Message Digest) is a group of hash function which is developed by Hans Dobbertin, Antoon Bosselaers and Bart Preneel in 1992. The development idea of RIPEMD is based on MD4 which in itself is a weak hash function. It is developed to work well with 32-bit processors.Types of RIPEMD:

RIPEMD-128
RIPEMD-160
RIPEMD-256
RIPEMD-320

Working

It is a sub-block of the RIPEMD-160 hash algorithm. The message is processed by compression function in blocks of 512 bits and passed through two streams of this sub-block by using 5 different versions in which the value of constant 'k' is also different.



The first RIPEMD was not considered as a good hash function because of some design flaws which leads to some major security problems one of which is the size of output that is 128 bit which is too small and easy to break. In the next version RIPEMD-128, the design flaw is removed but the output is still 128 bit which makes it less secure.

RIPEMD-160 is the next version which increases the output length to 160 bit and increases the security level of the hash function. This function is designed to work as a replacement for 128-bit hash functions MD4, MD5, and RIPEMD-128.

RIPEMD-256 and RIPEMD-320 are extension of RIPEMD-128 which provide same security as RIPEMD-160 and RIPEMD-128 which is designed for application which prefer large hash value rather than more security level.

Whirlpool Hash Function:

Whirlpool is a cryptographic hash function created by Vincent Rijmen and Paulo S.L.M. Barreto. It was first published in 2000 and revised in 2001 and 2003. It was derived form square and Advanced Encryption Standard. It is a block cipher hash function and designed after square block cipher. It takes less than 2^256 bits length input and convert it in 512 bit hash. The first version of whirlpool is called Whirlpool-0 and changed to Whirlpool-T after its first revision in 2001. In this version the S-box is changed and become easier to use in hardware. In 2002 a vulnerability was founded in the Whirlpool-0's diffusion matrix which was removed by changing the matrix and the name was also changed from Whirlpool-T to Whirlpool.

Whirlpool Logic

Given a message consisting of a sequence of blocks m1, m2... mt, the Whirlpool hash function is expressed as follows:

H0 = initial value
Hi = E (Hi–1, mi)! Hi–1! mi = intermediate value
Ht = hash code value

In terms of the model the encryption key input for each iteration is the intermediate hash value from the previous iteration; the plaintext is the current message block; and the feed forward value is the bitwise XOR of the current message block and the intermediate hash value from the previous iteration. The algorithm takes as input a message with a maximum length of less than 2^256 bits and produces as output a 512-bit message digest. The input is processed in 512-bit blocks.

Step 1: Append padding bits. The message is padded so that its length in bits is an odd multiple of 256. Padding is always added, even if the message is already of the desired length. For example, if the message is 256 * 3 = 768 bits long, it is padded by 512 bits to a length of 256 * 5 = 1280 bits. Thus, the number of padding bits is in the range of 1 to 512.

The padding consists of a single 1-bit followed by the necessary number of 0-bits.

• Step 2: Append length. A block of 256 bits is appended to the message. This block is treated as an unsigned 256-bit integer (most significant byte first) and contains the length in bits of the original message (before the padding).

The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. The expanded message is represented as the sequence of 512-bit blocks m1, m2... mt, so that the total length of the expanded message is t * 512 bits. These blocks are viewed externally as arrays of bytes by sequentially grouping the bits in 8-bit chunks. However, internally, the hash state Hi is viewed as an 8 * 8 matrix of bytes. The transformation between the two is explained subsequently.

• Step 3: Initialize hash matrix. An 8 * 8 matrix of bytes is used to hold intermediate and final results of the hash function. The matrix is initialized as consisting of all 0-bits.

• Step 4: Process message in 512-bit (64-byte) blocks. The heart of the algorithm is the block cipher W.