INFORMATION AND NETWORK SECURITY (2170709)

LAB MANUAL

JEEL ANRUTIA
160470107002
VVPEC CE SEM-7

Table of Contents

Practical - 1

Aim: Introduction to Information and Network Security

CRYPTOGRAPHY

Human being from ages had two inherent needs − (a) to communicate and share information and (b) to communicate selectively. These two needs gave rise to the art of coding the messages in such a way that only the intended people could have access to the information. Unauthorized people could not extract any information, even if the scrambled messages fell in their hand.

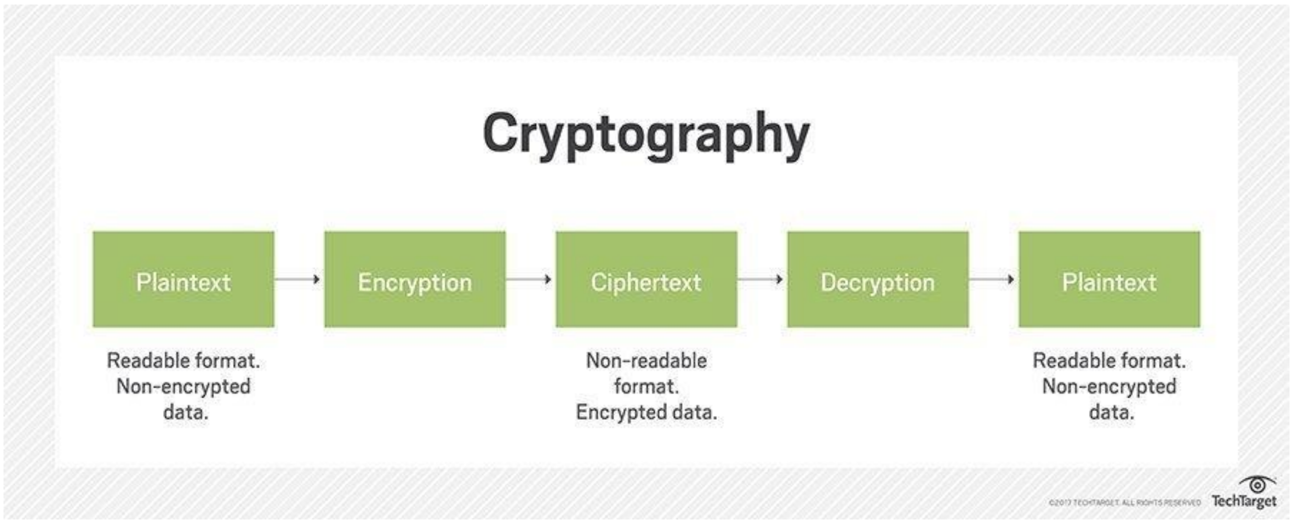
The art and science of concealing the messages to introduce secrecy in information security is recognized as cryptography.

The word ‘cryptography’ was coined by combining two Greek words, ‘Krypto’ meaning hidden and ‘graphene’ meaning writing.

Cryptography involves creating written or generated codes that allow information to be kept secret. Cryptography converts data into a format that is unreadable for an unauthorized user, allowing it to be transmitted without unauthorized entities decoding it back into a readable format, thus compromising the data.

Information security uses cryptography on several levels. The information cannot be read without a key to decrypt it. The information maintains its integrity during transit and while being stored. Cryptography also aids in nonrepudiation. This means that the sender and the delivery of a message can be verified.

Cryptography is also known as cryptology.


Cryptography
Plaintext
Encryption
Ciphertext
Decryption
Plaintext
Readable format. Non-encrypted data.
Non-readable format. Encrypted data.
Readable format. Non-encrypted data.
TechTarget

Security Services of Cryptography

The primary objective of using cryptography is to provide the following four fundamental information security services. Let us now see the possible goals intended to be fulfilled by cryptography.

Confidentiality

Confidentiality is the fundamental security service provided by cryptography. It is a security service that keeps the information from an unauthorized person. It is sometimes referred to as **privacy** or **secrecy**.

Confidentiality can be achieved through numerous means starting from physical securing to the use of mathematical algorithms for data encryption.

Data Integrity

It is security service that deals with identifying any alteration to the data. The data may get modified by an unauthorized entity intentionally or accidently. Integrity service confirms that whether data is intact or not since it was last created, transmitted, or stored by an authorized user.

Data integrity cannot prevent the alteration of data, but provides a means for detecting whether data has been manipulated in an unauthorized manner.

Authentication

Authentication provides the identification of the originator. It confirms to the receiver that the data received has been sent only by an identified and verified sender.

Authentication service has two variants −

- **Message authentication** identifies the originator of the message without any regard router or system that has sent the message.

- **Entity authentication** is assurance that data has been received from a specific entity, say a particular website.

Apart from the originator, authentication may also provide assurance about other parameters related to data such as the date and time of creation/transmission.

Non-repudiation

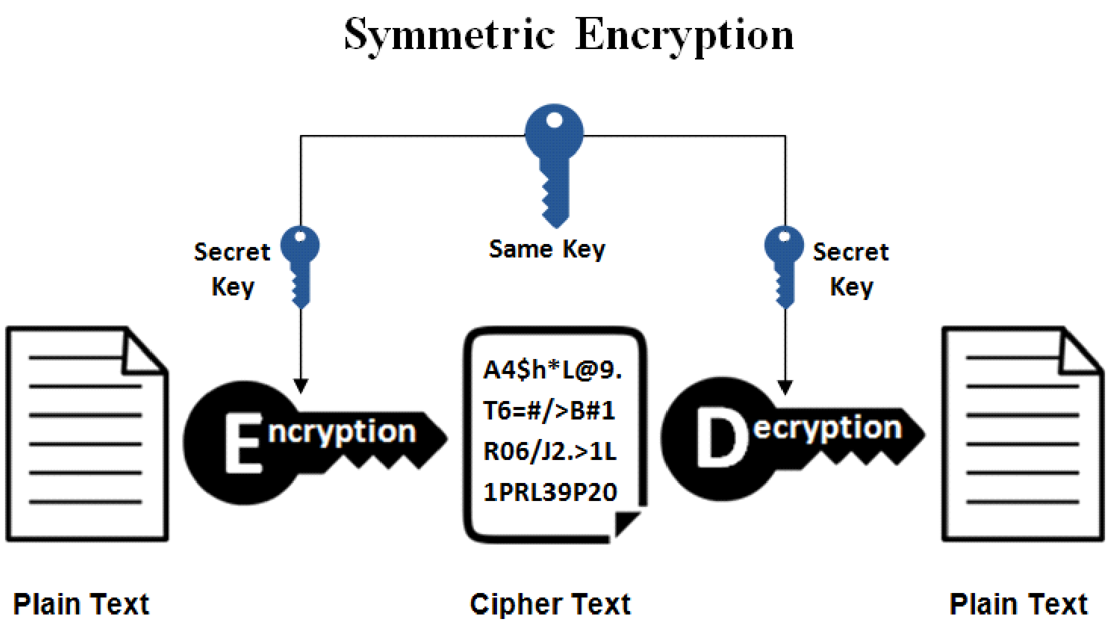
It is a security service that ensures that an entity cannot refuse the ownership of a previous commitment or an action. It is an assurance that the original creator of the data cannot deny the creation or transmission of the said data to a recipient or third party.

Non-repudiation is a property that is most desirable in situations where there are chances of a dispute over the exchange of data. For example, once an order is placed electronically, a purchaser cannot deny the purchase order, if non-repudiation service was enabled in this transaction.

Symmetric Key Encryption

The encryption process where **same keys are used for encrypting and decrypting** the information is known as Symmetric Key Encryption.

The study of symmetric cryptosystems is referred to as **symmetric cryptography**. Symmetric cryptosystems are also sometimes referred to as **secret key cryptosystems**.



The salient features of cryptosystem based on symmetric key encryption are −

- Persons using symmetric key encryption must share a common key prior to exchange of information.
- Keys are recommended to be changed regularly to prevent any attack on the system.
- A robust mechanism needs to exist to exchange the key between the communicating parties. As keys are required to be changed regularly, this mechanism becomes expensive and cumbersome.
- Length of Key (number of bits) in this encryption is smaller and hence, process of encryption-decryption is faster than asymmetric key encryption.

Challenge of Symmetric Key Cryptosystem

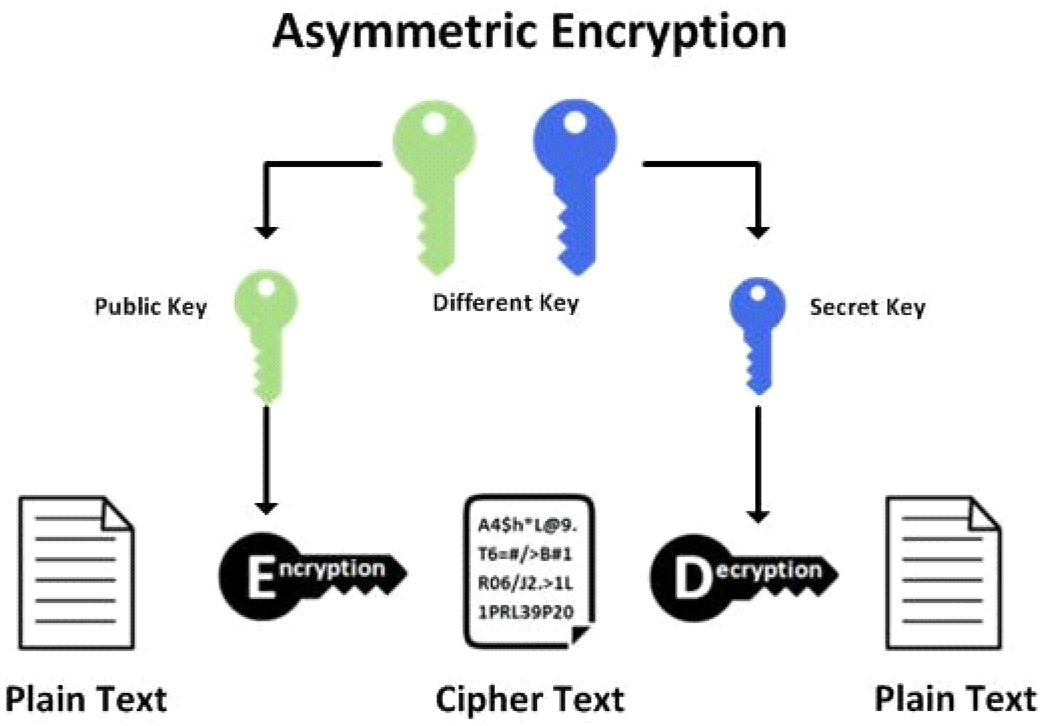
There are two restrictive challenges of employing symmetric key cryptography.

- **Key establishment** − Before any communication, both the sender and the receiver need to agree on a secret symmetric key. It requires a secure key establishment mechanism in place.
- **Trust Issue** − Since the sender and the receiver use the same symmetric key, there is an implicit requirement that the sender and the receiver 'trust' each other. For example, it may happen that the receiver has lost the key to an attacker and the sender is not informed.

These two challenges are highly restraining for modern day communication. Today, people need to exchange information with non-familiar and non-trusted parties. For example, a communication between online seller and customer. These limitations of symmetric key encryption gave rise to asymmetric key encryption schemes.

Asymmetric Key Encryption

The encryption process where **different keys are used for encrypting and decrypting the information** is known as Asymmetric Key Encryption. Though the keys are different, they are mathematically related and hence, retrieving the plaintext by decrypting ciphertext is feasible. The process is depicted in the following illustration −



The salient features of this encryption scheme are as follows −

- Every user in this system needs to have a pair of dissimilar keys, **private key** and **public key**. These keys are mathematically related − when one key is used for encryption, the other can decrypt the ciphertext back to the original plaintext.
- It requires to put the public key in public repository and the private key as a well-guarded secret. Hence, this scheme of encryption is also called **Public Key Encryption**.
- Though public and private keys of the user are related, it is computationally not feasible to find one from another. This is a strength of this scheme.
- When *Host1* needs to send data to *Host2*, he obtains the public key of *Host2* from repository, encrypts the data, and transmits.
- *Host2* uses his private key to extract the plaintext.
- Length of Keys (number of bits) in this encryption is large and hence, the process of encryption-decryption is slower than symmetric key encryption.
- Processing power of computer system required to run asymmetric algorithm is higher.

Symmetric cryptosystems are a natural concept. In contrast, public-key cryptosystems are quite difficult to comprehend.

Relation between Encryption Schemes

A summary of basic key properties of two types of cryptosystems is given below −

	Symmetric Cryptosystems	**Public Key Cryptosystems**

Relation between Keys	Same	Different, but mathematically related
Encryption Key	Symmetric	Public
Decryption Key	Symmetric	Private

Due to the advantages and disadvantage of both the systems, symmetric key and public-key cryptosystems are often used together in the practical information security systems.

Difference Between Symmetric and Asymmetric Encryption

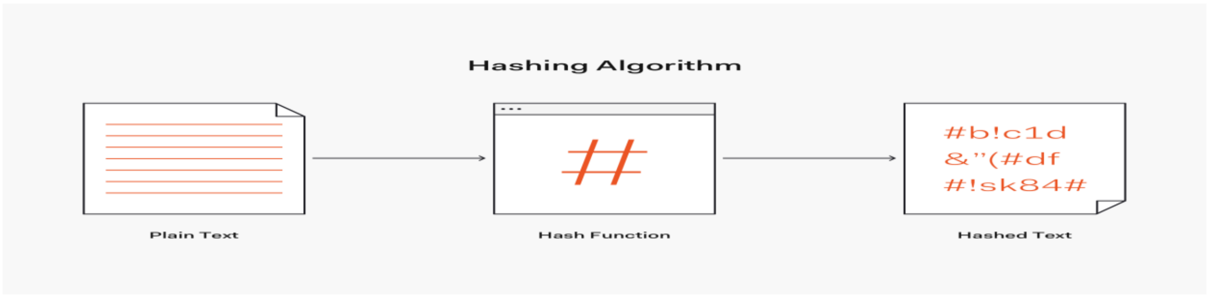
- Symmetric encryption uses a single key that needs to be shared among the people who need to receive the message while asymmetrical encryption uses a pair of public key and a private key to encrypt and decrypt messages when communicating.
- Symmetric encryption is an old technique while asymmetric encryption is relatively new.
- Asymmetric encryption was introduced to complement the inherent problem of the need to share the key in symmetrical encryption model, eliminating the need to share the key by using a pair of public-private keys.
- Asymmetric encryption takes relatively more time than the symmetric encryption.

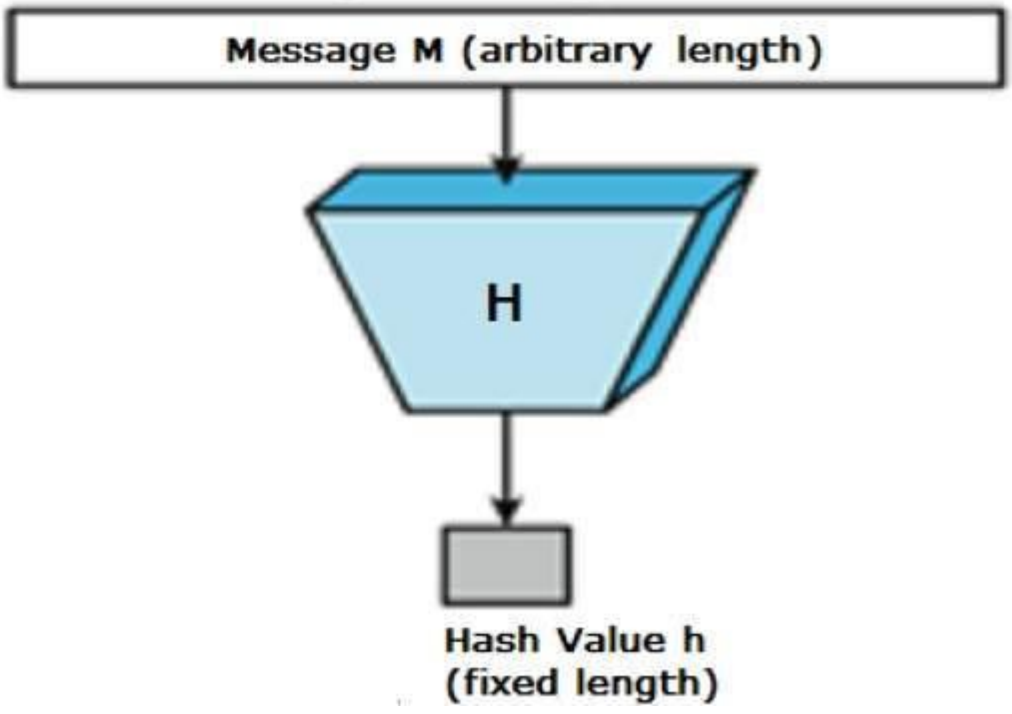
Hashing

Hash functions are extremely useful and appear in almost all information security applications.

A hash function is a mathematical function that converts a numerical input value into another compressed numerical value. The input to the hash function is of arbitrary length but output is always of fixed length.

Values returned by a hash function are called **message digest** or simply **hash values**.





Features of Hash Functions

The typical features of hash functions are −

- **Fixed Length Output (Hash Value)**
    - Hash function coverts data of arbitrary length to a fixed length. This process is often referred to as **hashing the data**.
    - In general, the hash is much smaller than the input data, hence hash functions are sometimes called **compression functions**.
    - Since a hash is a smaller representation of a larger data, it is also referred to as a **digest**.
    - Hash function with n bit output is referred to as an **n-bit hash function**. Popular hash functions generate values between 160 and 512 bits.
- **Efficiency of Operation**
    - Generally for any hash function h with input x, computation of h(x) is a fast operation.
    - Computationally hash functions are much faster than a symmetric encryption.

Properties of Hash Functions

In order to be an effective cryptographic tool, the hash function is desired to possess following properties −

- **Pre-Image Resistance**

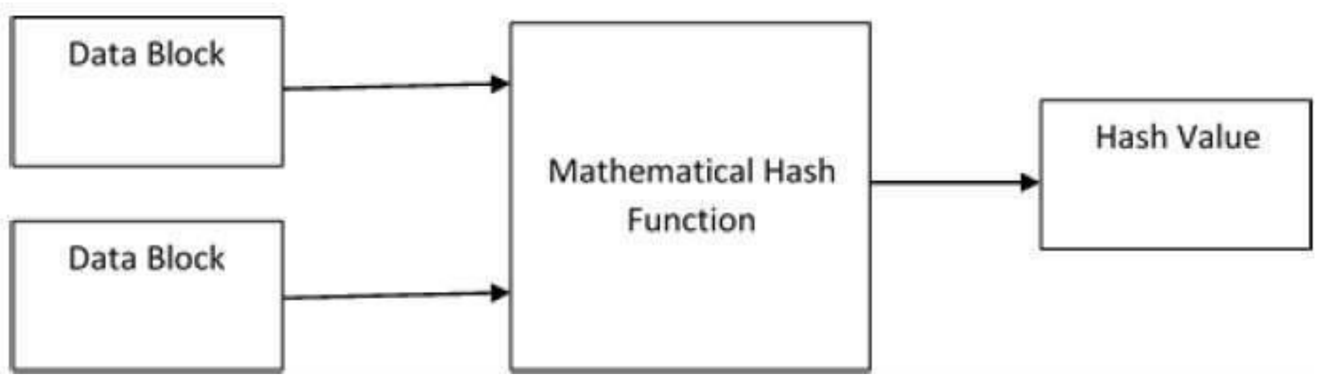
- This property means that it should be computationally hard to reverse a hash function.
- In other words, if a hash function h produced a hash value z, then it should be a difficult process to find any input value x that hashes to z.
- This property protects against an attacker who only has a hash value and is trying to find the input.

- **Second Pre-Image Resistance**
  - This property means given an input and its hash, it should be hard to find a different input with the same hash.
  - In other words, if a hash function h for an input x produces hash value h(x), then it should be difficult to find any other input value y such that h(y) = h(x).
  - This property of hash function protects against an attacker who has an input value and its hash, and wants to substitute different value as legitimate value in place of original input value.
- **Collision Resistance**
  - This property means it should be hard to find two different inputs of any length that result in the same hash. This property is also referred to as collision free hash function.
  - In other words, for a hash function h, it is hard to find any two different inputs x and y such that h(x) = h(y).
  - Since, hash function is compressing function with fixed hash length, it is impossible for a hash function not to have collisions. This property of collision free only confirms that these collisions should be hard to find.
  - This property makes it very difficult for an attacker to find two input values with the same hash.
  - Also, if a hash function is collision-resistant **then it is second pre-image resistant.**

Design of Hashing Algorithms

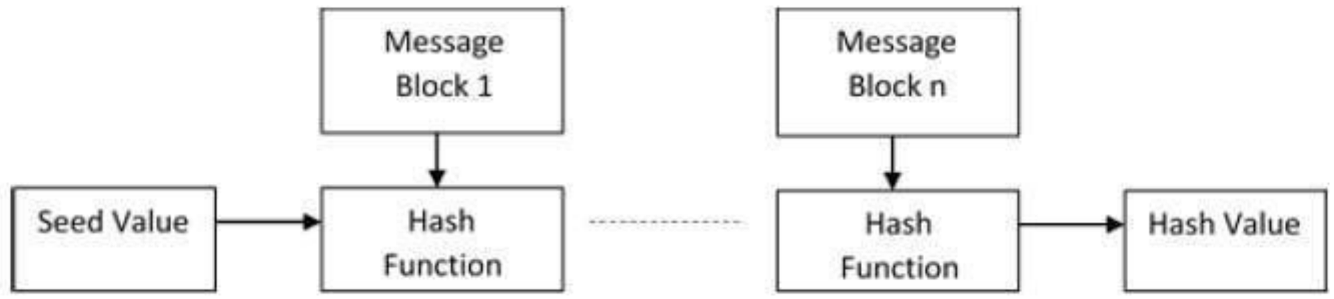
At the heart of a hashing is a mathematical function that operates on two fixed-size blocks of data to create a hash code. This hash function forms the part of the hashing algorithm.

The size of each data block varies depending on the algorithm. Typically the block sizes are from 128 bits to 512 bits. The following illustration demonstrates hash function –



Hashing algorithm involves rounds of above hash function like a block cipher. Each round takes an input of a fixed size, typically a combination of the most recent message block and the output of the last round.

This process is repeated for as many rounds as are required to hash the entire message. Schematic of hashing algorithm is depicted in the following illustration −



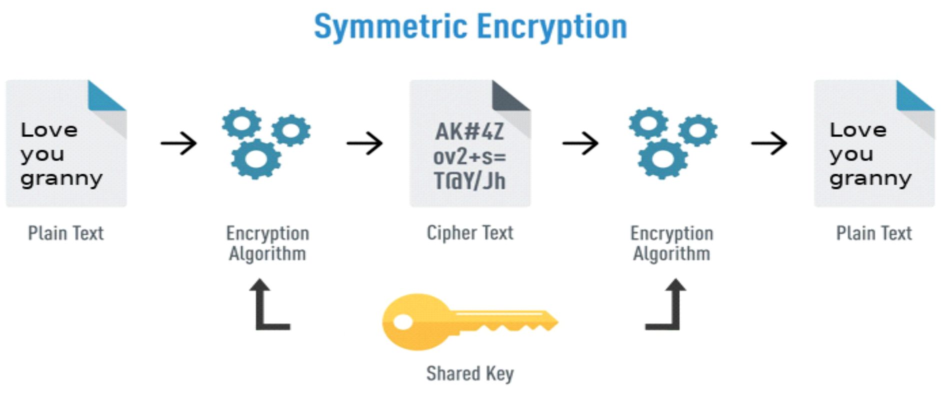
Since, the hash value of first message block becomes an input to the second hash operation, output of which alters the result of the third operation, and so on. This effect, known as an **avalanche** effect of hashing.

Avalanche effect results in substantially different hash values for two messages that differ by even a single bit of data.

Understand the difference between hash function and algorithm correctly. The hash function generates a hash code by operating on two blocks of fixed-length binary data.

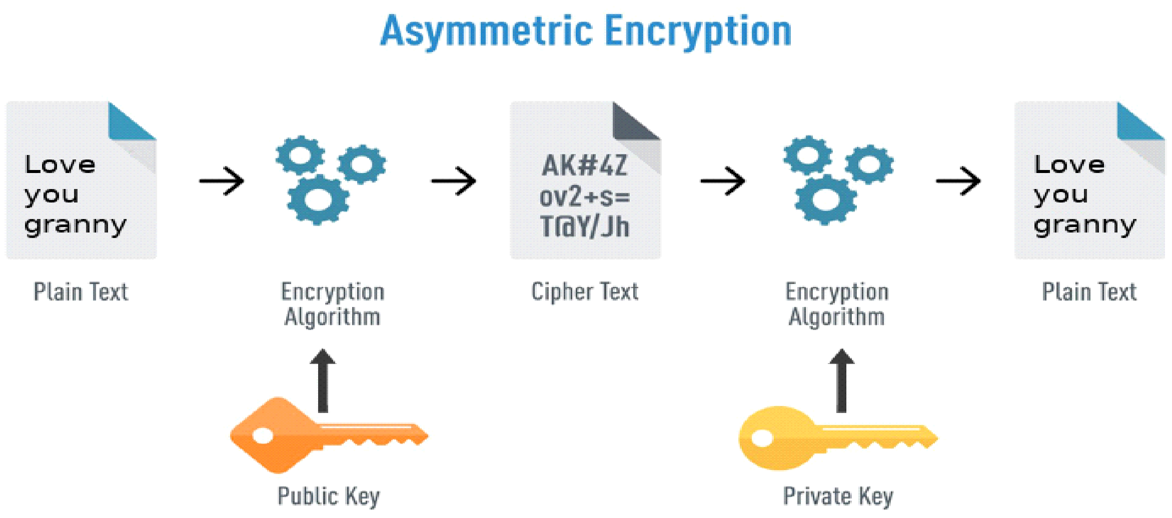
Hashing algorithm is a process for using the hash function, specifying how the message will be broken up and how the results from previous message blocks are chained together.

Private Key Cryptography



•A private key, also known as a secret key, is a variable in cryptography that is used with an algorithm to encrypt and decrypt code.
•Secret keys are only shared with the key's generator, making it highly secure.
•It is a fast process since it uses a single key.
•However, protecting one key creates a key management issue when everyone is using private keys.
•The private key may be stolen or leaked.
•Key management requires prevention of these risks and necessitates changing the encryption key often, and appropriately distributing the key.

Public key Cryptography



•Public-key cryptography, or asymmetric cryptography, is an encryption scheme that uses two mathematically related, but not identical, keys - a public key and a private key.

•Unlike symmetric key algorithms that rely on one key to both encrypt and decrypt, each key performs a unique function.

•The public key is used to encrypt and the private key is used to decrypt.

•It is computationally infeasible to compute the private key based on the public key.

•Because of this, public keys can be freely shared, allowing users an easy and convenient method for encrypting content and verifying digital signatures, and private keys can be kept secret, ensuring only the owners of the private keys can decrypt content and create digital signatures.

•Since public keys need to be shared but are too big to be easily remembered, they are stored on digital certificates for secure transport and sharing.

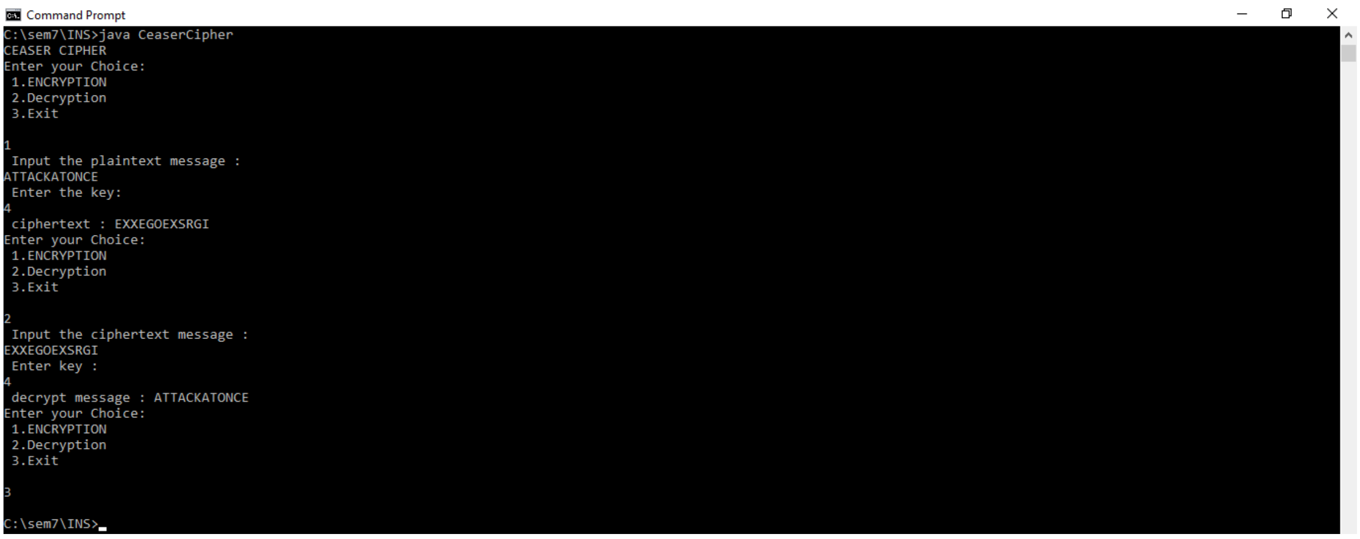
Practical: 2

Aim: Implement Caesar Cipher

```
import java.util.*;

  public class CeaserCipher {
  private static Scanner sc;

      public static void main(String args[]) {

      System.out.print("CEASER CIPHER \n");
      while(true) {
           sc = new Scanner(System.in);
            System.out.println("Enter your Choice: \n 1.ENCRYPTION \n
                2.Decryption \n 3.Exit \n");
           int choice=sc.nextInt();

      switch(choice)
      {
      case 1: System.out.println(" Input the plaintext message : ");
                     String plaintext = sc.next();
                     System.out.println(" Enter the key: ");
                     int key = sc.nextInt();
                     String ciphertext = "";
                     char alphabet;
                     for(int i=0; i < plaintext.length();i++)
                     {

                     alphabet = plaintext.charAt(i);
                     if(alphabet >= 'a' && alphabet <= 'z')
                     {

                          alphabet = (char) (alphabet + key);
                               if(alphabet > 'z')
                               {
                               alphabet = (char) (alphabet+'a'-'z'-1);
                               }
                               ciphertext = ciphertext + alphabet;
                     }
                     else if(alphabet >= 'A' && alphabet <= 'Z')
                     {

                          alphabet = (char) (alphabet + key);
                          if(alphabet > 'Z')
                          {
                               alphabet = (char) (alphabet+'A'-'Z'-1);
                          }
                          ciphertext = ciphertext + alphabet;
                }
                else
                {
                     ciphertext = ciphertext + alphabet;
                }
           }
           System.out.println(" ciphertext : " + ciphertext);
           break;


case 2:    System.out.println(" Input the ciphertext message :");
                String ciphertext1 = sc.next();
                System.out.println(" Enter key : ");
                int key1 = sc.nextInt();
                String decryptMessage = "";
                for(int i=0; i < ciphertext1.length();i++)
                {
                char alphabet1 = ciphertext1.charAt(i);
                     if(alphabet1 >= 'a' && alphabet1 <= 'z')
                     {

                          alphabet1 = (char) (alphabet1 - key1);
                          if(alphabet1 < 'a')
                          {
                          alphabet1 = (char) (alphabet1-'a'+'z'+1);
                          }
                     decryptMessage = decryptMessage + alphabet1;
                     }

                     else if(alphabet1 >= 'A' && alphabet1 <= 'Z')
                     {
                     alphabet1 = (char) (alphabet1 - key1);
                          if (alphabet1 < 'A')
                          {
                          alphabet1 = (char) (alphabet1-'A'+'Z'+1);
                          }
                     decryptMessage = decryptMessage + alphabet1;
                     }
                     else
                     {
                     decryptMessage = decryptMessage + alphabet1;
                     }
                }
      System.out.println(" decrypt message : " + decryptMessage);
                break;

           case 3:System.exit(0);
                break;
      }
```

```
        }
}

}
```

OUTPUT:



Command Prompt

```
C:\sem7\INS>java CeaserCipher
CEASER CIPHER
Enter your Choice:
 1.ENCRYPTION
 2.Decryption
 3.Exit

1
 Input the plaintext message :
ATTACKATONCE
 Enter the key:
4
 ciphertext : EXXEGOEXSRGI
Enter your Choice:
 1.ENCRYPTION
 2.Decryption
 3.Exit

2
 Input the ciphertext message :
EXXEGOEXSRGI
 Enter key :
4
 decrypt message : ATTACKATONCE
Enter your Choice:
 1.ENCRYPTION
 2.Decryption
 3.Exit

3

C:\sem7\INS>_
```

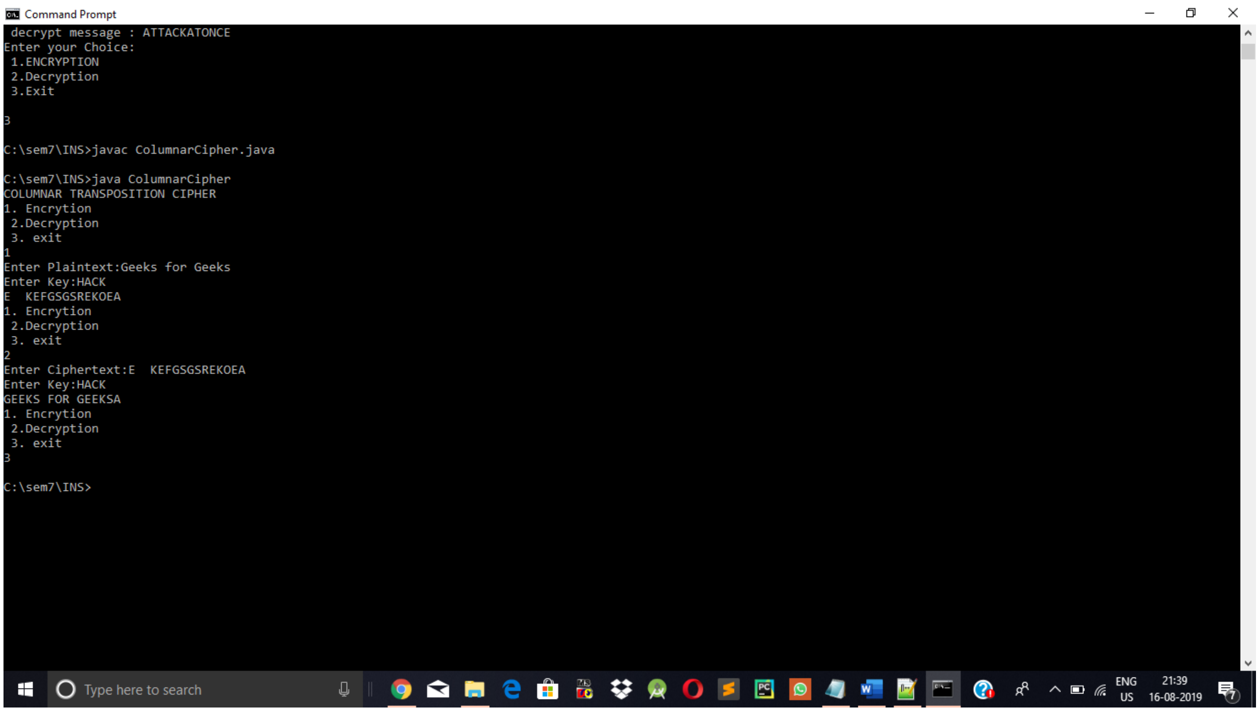
Practical: 3

Aim: Implement Columnar Transposition Cipher

```
import java.util.*;

public class ColumnarCipher {
    private static Scanner sc;

     public static void main(String[] args) {
        sc = new Scanner(System.in);
        String line = System.getProperty("line.separator");
        sc.useDelimiter(line);
        System.out.println("COLUMNAR TRANSPOSITION CIPHER");
      while(true)
      {
        System.out.print("1. Encrytion \n 2.Decryption \n 3. exit 
\n");
        int choice = sc.nextInt();
        switch (choice)
        {
            case 1:
                System.out.print("Enter Plaintext:");
                String text = sc.next();
                System.out.print("Enter Key:");
                String key = sc.next();

                System.out.println(encryptCT(key, 
text).toUpperCase());
                break;

            case 2:
                System.out.print("Enter Ciphertext:");
                text = sc.next();
                System.out.print("Enter Key:");
                key = sc.next();

                System.out.println(decryptCT(key, text));
                break;

            case 3:
               System.exit(0);
               break;
        }
     }
     }

    public static String encryptCT(String key, String text) {
        int[] arrange = arrangeKey(key);

        int lenkey = arrange.length;
        int lentext = text.length();

        int row = (int) Math.ceil((double) lentext / lenkey);

        char[][] grid = new char[row][lenkey];
        int z = 0;
        for (int x = 0; x < row; x++)
        {
            for (int y = 0; y < lenkey; y++)
            {
                if (lentext == z)
                {
                    grid[x][y] = RandomAlpha();
                    z--;
                }
                else
                {
                    grid[x][y] = text.charAt(z);
                }
                z++;
            }
        }
        String enc = "";
        for (int x = 0; x < lenkey; x++)
        {
            for (int y = 0; y < lenkey; y++)
            {
                if (x == arrange[y])
                {
                    for (int a = 0; a < row; a++)
                    {
                        enc = enc + grid[a][y];
                    }
                }
            }
        }
        return enc;
    }

    public static String decryptCT(String key, String text) {
        int[] arrange = arrangeKey(key);
        int lenkey = arrange.length;
        int lentext = text.length();
```

```
        int row = (int) Math.ceil((double) lentext / lenkey);

        String regex = "(?<=\\G.{" + row + "})";
        String[] get = text.split(regex);

        char[][] grid = new char[row][lenkey];

        for (int x = 0; x < lenkey; x++)
        {
            for (int y = 0; y < lenkey; y++)
            {
                if (arrange[x] == y)
                {
                    for (int z = 0; z < row; z++)
                    {
                        grid[z][y] = get[arrange[y]].charAt(z);
                    }
                }
            }
        }

        String dec = "";
        for (int x = 0; x < row; x++)
        {
            for (int y = 0; y < lenkey; y++)
            {
                dec = dec + grid[x][y];
            }
        }

        return dec;
    }

    public static char RandomAlpha()
    {
        Random r = new Random();
        return (char)(r.nextInt(26) + 'a');
    }

    public static int[] arrangeKey(String key)
    {
        //arrange position of grid
        String[] keys = key.split("");
        Arrays.sort(keys);
        int[] num = new int[key.length()];
        for (int x = 0; x < keys.length; x++)
        {
            for (int y = 0; y < key.length(); y++)
            {
                    if (keys[x].equals(key.charAt(y) + ""))
                    {
                        num[y] = x;
                        break;
                    }
                }
            }
            return num;
        }

}
```

OUTPUT:



```
 decrypt message : ATTACKATONCE
Enter your Choice:
 1.ENCRYPTION
 2.Decryption
 3.Exit

3

C:\sem7\INS>javac ColumnarCipher.java

C:\sem7\INS>java ColumnarCipher
COLUMNAR TRANSPOSITION CIPHER
1. Encrytion
 2.Decryption
 3. exit
1
Enter Plaintext:Geeks for Geeks
Enter Key:HACK
E  KEFGSGSREKOEA
1. Encrytion
 2.Decryption
 3. exit
2
Enter Ciphertext:E  KEFGSGSREKOEA
Enter Key:HACK
GEEKS FOR GEEKSA
1. Encrytion
 2.Decryption
 3. exit
3

C:\sem7\INS>
```

Practical: 4

Aim: Implement PlayFair Cipher

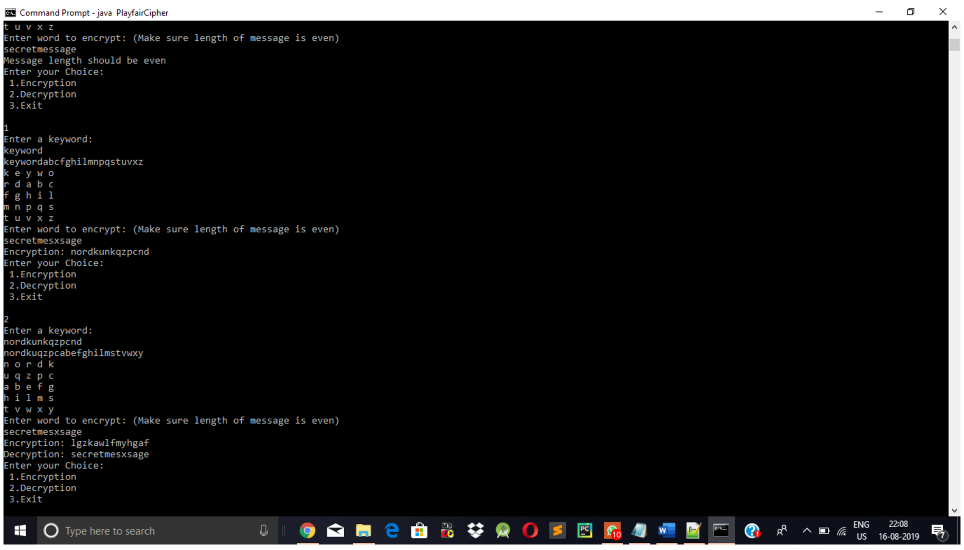
```
Import java.util.Scanner;

   public class PlayfairCipher
   {
       private String KeyWord        = new String();
       private String Key            = new String();
       private char   matrix_arr[][] = new char[5][5];
       static String text = new String();
        private static Scanner sc;

       public void setKey(String k)
       {
           String K_adjust = new String();
           boolean flag = false;
           K_adjust = K_adjust + k.charAt(0);
           for (int i = 1; i < k.length(); i++)
           {
               for (int j = 0; j < K_adjust.length(); j++)
               {
                   if (k.charAt(i) == K_adjust.charAt(j))
                   {
                       flag = true;
                   }
               }
               if (flag == false)
                   K_adjust = K_adjust + k.charAt(i);
               flag = false;
           }
           KeyWord = K_adjust;
       }

       public void KeyGen()
       {
           boolean flag = true;
           char current;
           Key = KeyWord;
           for (int i = 0; i < 26; i++)
           {
               current = (char) (i + 97);
               if (current == 'j')
```

```
                continue;
            for (int j = 0; j < KeyWord.length(); j++)
            {
                if (current == KeyWord.charAt(j))
                {
                    flag = false;
                    break;
                }
            }
            if (flag)
                Key = Key + current;
            flag = true;
        }
        System.out.println(Key);
        matrix();
    }

    private void matrix()
    {
        int counter = 0;
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < 5; j++)
            {
                matrix_arr[i][j] = Key.charAt(counter);
                System.out.print(matrix_arr[i][j] + " ");
                counter++;
            }
            System.out.println();
        }
    }

    private String format(String old_text)
    {
        int i = 0;
        int len = 0;

        len = old_text.length();
        for (int tmp = 0; tmp < len; tmp++)
        {
            if (old_text.charAt(tmp) == 'j')
            {
                text = text + 'i';
            }
            else
                text = text + old_text.charAt(tmp);
        }
```

```
            len = text.length();
            for (i = 0; i < len; i = i + 2)
            {
                if (text.charAt(i + 1) == text.charAt(i))
                {
                    text = text.substring(0, i + 1) + 'x' +
text.substring(i + 1);

                }
            }
            return text;
        }

        private String[] Divid2Pairs(String new_string)
        {
            String Original = format(new_string);
            int size = Original.length();
            if (size % 2 != 0)
            {
                size++;
                Original = Original + 'x';
            }
            String x[] = new String[size / 2];
            int counter = 0;
            for (int i = 0; i < size / 2; i++)
            {
                x[i] = Original.substring(counter, counter + 2);
                counter = counter + 2;
            }
            return x;
        }

        public int[] GetDiminsions(char letter)
        {
            int[] key = new int[2];
            if (letter == 'j')
                letter = 'i';
            for (int i = 0; i < 5; i++)
            {
                for (int j = 0; j < 5; j++)
                {
                    if (matrix_arr[i][j] == letter)
                    {
                        key[0] = i;
                        key[1] = j;
                        break;
                    }
            }
        }
        return key;
    }

    public String encryptMessage(String Source)
    {
        String src_arr[] = Divid2Pairs(Source);
        String Code = new String();
        char one;
        char two;
        int part1[] = new int[2];
        int part2[] = new int[2];
        for (int i = 0; i < src_arr.length; i++)
        {
            one = src_arr[i].charAt(0);
            two = src_arr[i].charAt(1);
            part1 = GetDiminsions(one);
            part2 = GetDiminsions(two);
            if (part1[0] == part2[0])
            {
                if (part1[1] < 4)
                    part1[1]++;
                else
                    part1[1] = 0;
                if (part2[1] < 4)
                    part2[1]++;
                else
                    part2[1] = 0;
            }
            else if (part1[1] == part2[1])
            {
                if (part1[0] < 4)
                    part1[0]++;
                else
                    part1[0] = 0;
                if (part2[0] < 4)
                    part2[0]++;
                else
                    part2[0] = 0;
            }
            else
            {
                int temp = part1[1];
                part1[1] = part2[1];
                part2[1] = temp;
            }
            Code = Code + matrix_arr[part1[0]][part1[1]]
                    + matrix_arr[part2[0]][part2[1]];
        }
        return Code;
    }

    public String decryptMessage(String Code)
    {
        String Original = new String();
        String src_arr[] = Divid2Pairs(Code);
        char one;
        char two;
        int part1[] = new int[2];
        int part2[] = new int[2];
        for (int i = 0; i < src_arr.length; i++)
        {
            one = src_arr[i].charAt(0);
            two = src_arr[i].charAt(1);

            part1 = GetDiminsions(one);
            part2 = GetDiminsions(two);

            if (part1[0] == part2[0])
            {
                if (part1[1] > 0)
                    part1[1]--;
                else
                    part1[1] = 4;

                if (part2[1] > 0)
                    part2[1]--;
                else
                    part2[1] = 4;
            }

            else if (part1[1] == part2[1])
            {
                if (part1[0] > 0)
                    part1[0]--;
                else
                    part1[0] = 4;

                if (part2[0] > 0)
                    part2[0]--;
                else
                    part2[0] = 4;
            }
```

```
                else
                {
                    int temp = part1[1];
                    part1[1] = part2[1];
                    part2[1] = temp;
                }
                Original = Original + matrix_arr[part1[0]][part1[1]] +
matrix_arr[part2[0]][part2[1]];
            }
            return Original;
        }


        public static void main(String[] args)
        {
            PlayfairCipher x = new PlayfairCipher();


            System.out.println("PLAYFAIR CIPHER");
            while(true)
            {
              sc = new Scanner(System.in);
              System.out.println("Enter your Choice: \n 1.Encryption
\n 2.Decryption \n 3.Exit \n");
              int choice = sc.nextInt();
              switch(choice)
              {
              case 1:   System.out.println("Enter a keyword:");
                        String keyword = sc.next();
                        x.setKey(keyword);
                        x.KeyGen();
                        System.out.println("Enter word to encrypt:");
                        String key_input = sc.next();

                             if(text.length() % 2 == 0)
                             {
                                  System.out.println("Encryption: " +
x.encryptMessage(key_input));
                             }
                             else
                             {
                                  String ax = key_input + 'x';
                                  System.out.println(ax);
                                  System.out.println("Encryption: " +
x.encryptMessage(ax));
                             }
                        break;

                case 2: System.out.println("Enter a keyword:");
                        String keyword1 = sc.next();
                        x.setKey(keyword1);
                        x.KeyGen();
                        System.out.println("Enter word to decrypt:");
                        String key_input1 = sc.next();

                        if (text.length() % 2 == 0)
                        {
                                System.out.println("Decryption: "+
x.decryptMessage(key_input1));

                        }


                        break;

                case 3:System.exit(0);
                        break;
                }
            }

        }
    }
```

OUTPUT:



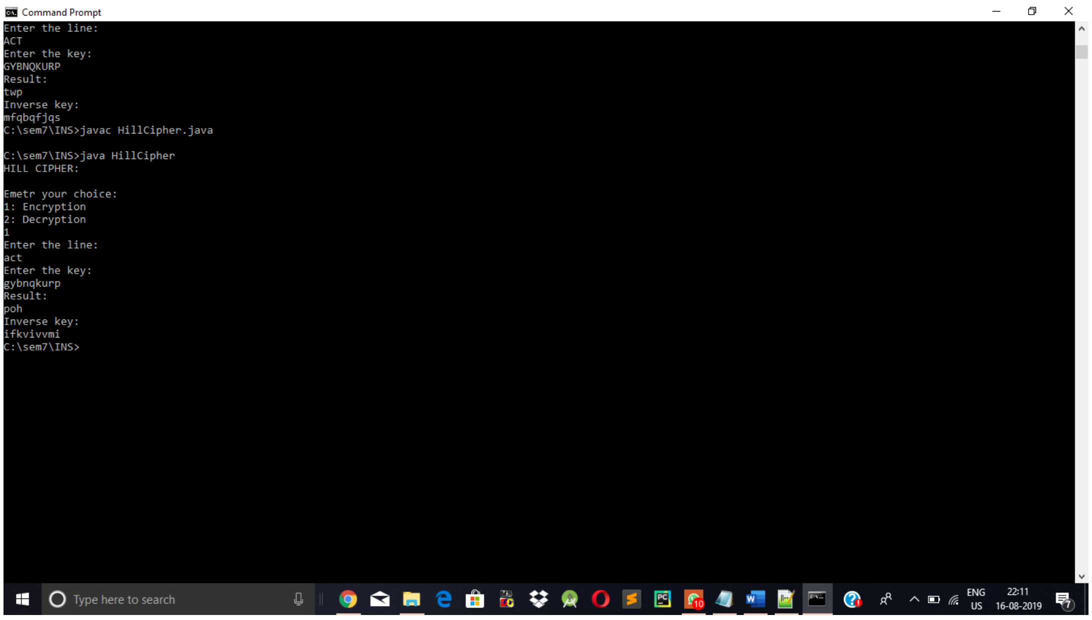
```
t u v x z
Enter word to encrypt: (Make sure length of message is even)
secretmessage
Message length should be even
Enter your Choice:
 1.Encryption
 2.Decryption
 3.Exit

1
Enter a keyword:
keyword
keywordabcfghilmnpqstuvxz
k e y w o
r d a b c
f g h i l
m n p q s
t u v x z
Enter word to encrypt: (Make sure length of message is even)
secretmesxsage
Encryption: nordkunkqzpcnd
Enter your Choice:
 1.Encryption
 2.Decryption
 3.Exit

2
Enter a keyword:
nordkunkqzpcnd
nordkuqzpcabefghilmstvwxy
n o r d k
u q z p c
a b e f g
h i l m s
t v w x y
Enter word to encrypt: (Make sure length of message is even)
secretmesxsage
Encryption: lgzkawlfmyhgaf
Decryption: secretmesxsage
Enter your Choice:
 1.Encryption
 2.Decryption
 3.Exit
```

Practical: 5

Aim: Implement Hill Cipher

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class HillCipher
{
    int keymatrix[][];
    int linematrix[];
    int resultmatrix[];
     private static int choice;

    public void divide(String temp, int s)
    {
        while (temp.length() > s)
        {
            String sub = temp.substring(0, s);
            temp = temp.substring(s, temp.length());
            perform(sub);
        }
        if (temp.length() == s)
            perform(temp);
        else if (temp.length() < s)
        {
            for (int i = temp.length(); i < s; i++)
                temp = temp + 'x';
            perform(temp);
        }
    }

    public void perform(String line)
    {
        linetomatrix(line);
        linemultiplykey(line.length());
        result(line.length());
    }

    public void keytomatrix(String key, int len)
    {
        keymatrix = new int[len][len];
        int c = 0;
        for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < len; j++)
            {
                keymatrix[i][j] = ((int) key.charAt(c)) - 97;
                c++;
            }
        }
    }

    public void linetomatrix(String line)
    {
        linematrix = new int[line.length()];
        for (int i = 0; i < line.length(); i++)
        {
            linematrix[i] = ((int) line.charAt(i)) - 97;
        }
    }

    public void linemultiplykey(int len)
    {
        resultmatrix = new int[len];
        for (int i = 0; i < len; i++)
        {
            for (int j = 0; j < len; j++)
            {
                resultmatrix[i] += keymatrix[i][j] *
linematrix[j];
            }
            resultmatrix[i] %= 26;
        }
    }

    public void result(int len)
    {
        String result = "";
        for (int i = 0; i < len; i++)
        {
            result += (char) (resultmatrix[i] + 97);
        }
        System.out.print(result);
    }

    public boolean check(String key, int len)
    {
        keytomatrix(key, len);
        int d = determinant(keymatrix, len);
        d = d % 26;
        if (d == 0)
        {
            System.out
                    .println("Invalid key!!! Key is not invertible
because determinant=0...");
            return false;
        }
        else if (d % 2 == 0 || d % 13 == 0)
        {
            System.out
                    .println("Invalid key!!! Key is not invertible
because determinant has common factor with 26...");
            return false;
        }
        else
        {
            return true;
        }
    }

    public int determinant(int A[][], int N)
    {
        int res;
        if (N == 1)
            res = A[0][0];
        else if (N == 2)
        {
            res = A[0][0] * A[1][1] - A[1][0] * A[0][1];
        }
        else
        {
            res = 0;
            for (int j1 = 0; j1 < N; j1++)
            {
                int m[][] = new int[N - 1][N - 1];
                for (int i = 1; i < N; i++)
                {
                    int j2 = 0;
                    for (int j = 0; j < N; j++)
                    {
                        if (j == j1)
                            continue;
                        m[i - 1][j2] = A[i][j];
                        j2++;
                    }
                }
                res += Math.pow(-1.0, 1.0 + j1 + 1.0) * A[0][j1]
                        * determinant(m, N - 1);
            }
        }
        return res;
    }

    public void cofact(int num[][], int f)
    {
        int b[][], fac[][];
        b = new int[f][f];
        fac = new int[f][f];
        int p, q, m, n, i, j;
        for (q = 0; q < f; q++)
        {
            for (p = 0; p < f; p++)
            {
                m = 0;
                n = 0;
                for (i = 0; i < f; i++)
                {
                    for (j = 0; j < f; j++)
                    {
                        b[i][j] = 0;
                        if (i != q && j != p)
                        {
                            b[m][n] = num[i][j];
                            if (n < (f - 2))
                                n++;
                            else
                            {
                                n = 0;
                                m++;
                            }
                        }
                    }
                }
                fac[q][p] = (int) Math.pow(-1, q + p) *
determinant(b, f - 1);
            }
        }
        trans(fac, f);
    }

    void trans(int fac[][], int r)
    {
        int i, j;
        int b[][], inv[][];
        b = new int[r][r];
        inv = new int[r][r];
        int d = determinant(keymatrix, r);
        int mi = mi(d % 26);
        mi %= 26;
        if (mi < 0)
            mi += 26;
        for (i = 0; i < r; i++)
        {
            for (j = 0; j < r; j++)
            {
                b[i][j] = fac[j][i];
            }
        }
        for (i = 0; i < r; i++)
        {
            for (j = 0; j < r; j++)
            {
                inv[i][j] = b[i][j] % 26;
                if (inv[i][j] < 0)
                    inv[i][j] += 26;
                inv[i][j] *= mi;
                inv[i][j] %= 26;
            }
        }
        System.out.println("\nInverse key:");
        matrixtoinvkey(inv, r);
    }

    public int mi(int d)
    {
        int q, r1, r2, r, t1, t2, t;
        r1 = 26;
        r2 = d;
        t1 = 0;
        t2 = 1;
        while (r1 != 1 && r2 != 0)
        {
            q = r1 / r2;
            r = r1 % r2;
            t = t1 - (t2 * q);
            r1 = r2;
            r2 = r;
            t1 = t2;
            t2 = t;
        }
        return (t1 + t2);
    }

    public void matrixtoinvkey(int inv[][], int n)
    {
        String invkey = "";
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                invkey += (char) (inv[i][j] + 97);
            }
        }
        System.out.print(invkey);
    }
    public static void main(String args[]) throws IOException
    {
        HillCipher obj = new HillCipher();
        BufferedReader in = new BufferedReader(new
InputStreamReader(System.in));

        System.out.println("HILL CIPHER: \n");
        System.out.println("Enter your choice:\n1: Encryption\n2:
Decryption");
        choice = Integer.parseInt(in.readLine());
        System.out.println("Enter the line: ");
        String line = in.readLine();
        System.out.println("Enter the key: ");
        String key = in.readLine();
        double sq = Math.sqrt(key.length());
        if (sq != (long) sq)
            System.out.println("Invalid key length");

        else
        {
            int s = (int) sq;
            if (obj.check(key, s))
            {
                System.out.println("TEXT:");
                obj.divide(line, s);
                obj.cofact(obj.keymatrix, s);
            }
        }
    }
}
```

OUTPUT:



```
Enter the line:
ACT
Enter the key:
GYBNQKURP
Result:
twp
Inverse key:
mfqbqfjqs
C:\sem7\INS>javac HillCipher.java

C:\sem7\INS>java HillCipher
HILL CIPHER:

Emetr your choice:
1: Encryption
2: Decryption
1
Enter the line:
act
Enter the key:
gybnqkurp
Result:
poh
Inverse key:
ifkvivvmi
C:\sem7\INS>
```