Table of Contents

Problem No: 01

Problem Name: A Program for Encryption and Decryption Using Additive Cipher .

Objectives: The main objectives of this experiments are given below :

- To understand the concept of Additive Cipher (also known as Caesar Cipher with a shift value).
- To implement encryption and decryption techniques using Additive Cipher in C/C++.
- To observe the working of the cipher by encrypting and decrypting a given message.
- To analyze the security aspects and limitations of the Additive Cipher.

Theory :

The Additive Cipher, also known as the Caesar Cipher, is a type of substitution cipher where each letter in the plaintext is shifted by a fixed number of positions in the alphabet. The key is the number of positions to shift, and both encryption and decryption use modular arithmetic to wrap around the alphabet.

Algorithm:

Encryption:

1. Choose a key K.
2. For each character P in the plaintext:
    - Convert P to its position in the alphabet.
    - Compute C = (P + K) mod 26 (for letters 'A' to 'Z').
    - Convert C back to a letter.
3. The result is the ciphertext.

Decryption:

1. Use the same key K.
2. For each character C in the ciphertext:
    - Convert C to its position in the alphabet.
    - Compute P = (C - K + 26) mod 26.
    - Convert P back to a letter.
3. The result is the original plaintext

Program Code:

```
int main()
{  int ciphercode , key;;
   char cipherarr[100];
    char p[]={'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'};
    char c[]={'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'};
    string plaintext,ciphertext,mainCipher;
  cout<<"Enter the plaintext : ";
 getline(cin, plaintext);
cout<<"Enter the key : ";
    cin>>key;
int len = plaintext.length();
 for(int i=0;i<len;i++){
     for(int j=0;j<26;j++){
          if(plaintext[i]==p[j]){
              ciphercode = (j+key)%26;
             break;
          }
       }
       if(plaintext[i]!=' '){
          mainCipher+=c[ciphercode];
          ciphertext+=c[ciphercode];
       }
```

```
        else
            ciphertext+=' '
    }
    cout<<endl<<"Cipher : "<<mainCipher<<endl;
      int plaincode;
      string newPlaintext;
      for(int i=0;i<ciphertext.length();i++){
        for(int j=0;j<26;j++){
            if(ciphertext[i]==c[j]){
                if(j-key<0)
                    plaincode = (j-key+26)%26;
                else
plaincode = (j-key)%26;
                break;
            }
        }
        if(ciphertext[i]!=' ')
            newPlaintext+=p[plaincode];
        else
            newPlaintext+=' ';
      }
    cout<<endl<<"Decrypted Plaintext : "<<newPlaintext<<endl;return 0;
}
```

Input & Output :

Enter the plaintext : how are you

Enter the key:15

Ciphertext: WDLPGTNDJ

Decrypted Plaintext : how are you

Result & Discussion:

The C++ program successfully encrypts and decrypts the message using the Additive Cipher. The input message "Hello World " with a shift value of 3 was encrypted to "KHOORZRUOG". The decryption process accurately recovered the original message.

The experiment demonstrates the working of the Additive Cipher for both encryption and decryption. While it provides a clear understanding of basic substitution ciphers, it is evident that its security is limited, making it ineffective for real-world applications. However, it serves as an excellent introduction to cryptography concepts and helps in understanding more advanced techniques. From this experiment we can learn about the following concept:

- **Working of the Additive Cipher:** The cipher shifts each character by a fixed number of positions in the alphabet. For example, with a key of 3, 'H' is shifted to 'K', 'E' to 'H', and so on.
- **Security Concerns:** The Additive Cipher is simple to implement but not secure due to its vulnerability to brute-force attacks. Since there are only 25 possible shift values (keys), an attacker can try all possible keys to decrypt the message without knowing the actual key.
- **Handling of Non-Alphabetic Characters:** Non-alphabetic characters such as spaces, punctuation, and numbers are not affected by the cipher. This maintains the message's readability while focusing encryption only on alphabetic characters.
- **Case Sensitivity:** The program maintains the case of the original message. Uppercase letters remain uppercase, and lowercase letters remain lowercase during encryption and decryption.
- **Limitation:** Due to its simplicity, the Additive Cipher is not suitable for modern encryption needs. More complex encryption algorithms, such as AES, offer much stronger security.

Problem No: 02

Problem Name: A Program for Encryption and Decryption Using Multiplicative Cipher .

Objectives: The main objectives of this experiment is given below:

- To implement the encryption and decryption techniques of the Multiplicative Cipher in C/C++.
- To understand the basic working of the Multiplicative Cipher in cryptography.
- To analyze the encryption and decryption of a message using a multiplicative key.

Theory :

The Multiplicative Cipher is a type of substitution cipher where each character in the plaintext is multiplied by a fixed key value (which is a positive integer), and the result is taken modulo 26 (for 26 letters of the alphabet).

Algorithm :

Encryption :

1. Input the plaintext message and a valid key.
2. For each character in the plaintext:
    - Convert the character to its corresponding alphabetic position.
    - Apply the encryption formula: $C = (P \times K) \bmod 26$.
    - Convert the result back to the alphabet.
3. Display the encrypted message.

Decryption :

1. Input the ciphertext and the same key used for encryption.
2. Calculate the modular inverse of the key.
3. For each character in the ciphertext:
    - Convert the character to its corresponding alphabetic position.
    - Apply the decryption formula: $P = (C \times K^{-1}) \bmod 26$.
    - Convert the result back to the alphabet.
4. Display the decrypted message.

Program Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
   int key,ciphercode;
   string plaintext,ciphertext, mainCipher;
  char p[]={'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'};
   char c[]={'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'};
cout<<"Enter the plaintext : ";
getline(cin, plaintext);
cout<<"Enter the key : ";
   cin>>key;
   int len = plaintext.length();
  // check multiplicative is possible or not
   if(__gcd(26,key)!=1)
   {
      cout<<"Invalid key!"<<endl<<"Please Enter a Valid key: ";
      cin>>key;
   }
 // Encryption
   for(int i=0;i<len;i++){
      for(int j=0;j<26;j++){
         if(plaintext[i]==p[j]){
            ciphercode = (j*key)%26;
  break;
         }
      }
```

```
        if(plaintext[i]!=' '){
            ciphertext+=c[ciphercode];
            mainCipher+=c[ciphercode];
        }
       else
            ciphertext+=' ';
    }
cout<<endl<<"Cipher : "<<mainCipher<<endl;
  // Decryption
    int q, r1=26, r2=key, r, t1=0, t2=1, t, inverk;
//finding inverse key
       while(r2!=0){
            q=r1/r2;
            r=r1%r2;
            t=t1-(q*t2);
            r1=r2;
            r2=r;
            t1=t2;
            t2=t;
           }
     inverk = t1+t2;
             cout<<"inverse key: "<<inverk;
       // decryption
        string newPlaintext;
        int plaincode;
        for(int i=0;i<ciphertext.length();i++){
        for(int j=0;j<26;j++){
            if(ciphertext[i]==c[j]){
```

```
            plaincode = (j*inverk)%26;
            break;
          }
       }
       if(ciphertext[i]!=' ')
          newPlaintext+=p[plaincode];
       else
          newPlaintext+=' ';
   }
   cout<<endl<<"Decrypted Plaintext : "<<newPlaintext<<endl;
return 0;
```

Output:

Enter the plaintext : hello world

Enter the key : 4

Invalid key!

Please Enter a Valid key: 7

Cipher : XCZZUYUPZV

Decrypted Plaintext : hello world

Result & Discussion:

The program successfully encrypted and decrypted the message using the Multiplicative Cipher. For instance, the input message "Hello World " with a multiplicative key of 7 was encrypted to " XCZZUYUPZV " and correctly decrypted back to "Hello World".

This experiment successfully implemented encryption and decryption using the Multiplicative Cipher in C++. While the cipher demonstrates the basics of multiplicative encryption, its limitations in key selection and security make it impractical for use in modern encryption systems. However, it provides a valuable learning opportunity for understanding cryptographic principles.

The Multiplicative Cipher encrypts each character by multiplying its position with a fixed key. The key must have a modular inverse under modulo 26 to ensure decryption. The program handled alphabetic characters, preserving non-alphabetic characters like spaces.

The cipher's security is limited due to its restricted key space—only keys that are co-prime with 26 can be used. While it is stronger than the Additive Cipher, the Multiplicative Cipher remains vulnerable to brute-force attacks due to its relatively small key space. The need for modular inverse calculations also introduces some complexity, but the program correctly manages this by handling invalid keys.

Problem No: 03

Problem Name: A Program for Encryption and Decryption Using Affine Cipher.

Objectives: The main objectives of this experiment is given below:

- To understand the working of the Affine Cipher.
- To implement the Affine Cipher for encryption and decryption in C/C++.
- To analyze the correctness and security of the Affine Cipher.
- To perform encryption and decryption of a message using the Affine Cipher and examine the results.

Theory :

The Affine Cipher is a type of substitution cipher where each letter in the plaintext is mapped to its numeric equivalent, encrypted using a mathematical function, and then converted back to a letter. The encryption function is a combination of two basic ciphers: multiplication and addition.

Algorithm :

Encryption:

1. Input the plaintext message and keys A and B.
2. For each character in the message:
    - Convert the character to its numeric equivalent.
    - Apply the encryption formula: $C=(A*P+ B) \bmod 26$.
    - Convert back to the character.
3. Output the encrypted message.

Decryption:

1. Input the ciphertext and keys a and b.
2. Find the modular inverse of a modulo 26.
3. For each character in the ciphertext:
    - Apply the decryption formula: $P=A^{-1}\cdot(Ci-B) \bmod 26$.
    - Convert back to the character.
4. Output the decrypted message.

Program Code:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
  int key1, key2,ciphercode;
 string plaintext,ciphertext,mainCipher, T, CipherText;
char p[]={'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'};
 char c[]={'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'};
cout<<"Enter the plaintext : ";
getline(cin, plaintext);
cout<<"Enter two key for Affine Cipher : ";
  cin>>key1>>key2;
 int len = plaintext.length();
  while(true){
      if(__gcd(26,key1)!=1)
      {
         cout<<"Invalid key1 !"<<endl<<"Please Enter a Valid key1: ";
          cin>>key1;
      }
      else
          break;
   }
```

```
// Affine Cipher Encryption
  for(int i=0;i<len;i++){
      for(int j=0;j<26;j++){
         if(plaintext[i]==p[j]){
             ciphercode = ((j*key1)+key2)%26;
            break;
         }
      }
      if(plaintext[i]!=' ')
      {
         ciphertext+=c[ciphercode];
         CipherText+=c[ciphercode];
      }
      else
         CipherText+=' ';
   }
cout<<endl<<"Affine Cipher Text : "<<ciphertext<<endl;
 // way to finding the inverse key
      int inverk, inverseCount = 1;
      while(true){
         if((key1*inverseCount)%26 == 1){
            inverk = inverseCount;
            break;
```

```
        }
        inverseCount++;
      }
 cout<<"inverse key: "<<inverk;
 int plaincode;
  string newPlainText="";
  for(int i=0; i<CipherText.length(); i++){
     for(int j=0;j<26;j++){
        if(CipherText[i]==c[j]){
           if(j-key2<0)
              plaincode = ((j-key2+26)*inverk)%26;
           else
              plaincode = ((j-key2)*inverk)%26;
           break;
        }    }
     if(CipherText[i]!=' ')
     newPlainText += p[plaincode];
     else
     newPlainText += ' ';
 }
cout<<endl<<"Affine Decrypted Plain Text : "<<newPlainText<<endl;
return 0;
}
```

Output:

Enter The Plaintext:my name is suraiya

Enter The Two key:7 2

The Ciphertext:IOPCIEGYYMRCGOC

The Plaintext:my name is suraiya

Result & Discussion:

The Affine Cipher program successfully encrypts and decrypts the given message using the specified keys.

The Affine Cipher has successfully implemented, and the encryption and decryption results were correct. Though educational, the cipher is limited in practical cryptographic applications due to security concerns.

From this experiment we can learn about the following concept:

- **Affine Cipher Application**: The encryption and decryption processes correctly apply the Affine Cipher formulas. A coprime a ensures successful decryption.
- **Modular Inverse**: Decryption depends on finding the modular inverse of a. If it doesn't exist, decryption is impossible.
- **Security**: While the Affine Cipher is stronger than basic substitution ciphers, it remains vulnerable to frequency analysis and brute-force attacks.
- **Case Handling**: The program preserves both uppercase and lowercase characters, providing case-sensitive encryption.
- **Non-Alphabetic Characters**: Non-alphabetic characters remain unaffected.

Problem No : 04

Problem Name : A Program for Encryption and Decryption Using Monoalphabetic Substitution Cipher .

Objectives: The main objectives are :

- To implement encryption and decryption using a Monoalphabetic Substitution Cipher in C/C++.
- To understand and apply the concept of key-based substitution in message encryption.
- To test the program for correctness by encrypting and decrypting a message.

Theory :

A Monoalphabetic Substitution Cipher is a type of substitution cipher in which each letter in the plaintext is replaced by a corresponding letter from a fixed key alphabet. In this cipher, the substitution for each character is fixed for the entire message, meaning that a given letter in the plaintext will always map to the same letter in the ciphertext.

Algorithm :

Encryption:

1. Input the plaintext and the substitution key (mapping between alphabets).
2. For each character in the plaintext:
    - Replace it with the corresponding character from the substitution key.
    - If it's an uppercase letter, replace it using the uppercase section of the key.
    - If it's a lowercase letter, replace it using the lowercase section.
3. Output the encrypted message.

Decryption:

1. Input the ciphertext and the substitution key.
2. For each character in the ciphertext:
    - Find the corresponding character in the substitution key and reverse the substitution.
    - Use the uppercase or lowercase section of the key based on the case of the character.
3. Output the decrypted message.

Program Code:

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

int main()
{
   char p[]={'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'};
   char c[]={'N','O','A','T','R','B','E','C','F','U','X','D','Q','G','Y','L','K','H','V','I','J','M','P','Z','S','W'};
   string plaintext, newPlaintext, ciphertext="", CipherText="";

cout<<"Enter the plaintext : ";
 getline(cin, plaintext);



//

   int len = plaintext.length();

//encryption
   int cipher;
   for(int i=0; i<len; i++){
      for (int j = 0; j < 26; ++j) {
         if (plaintext[i]==p[j]) {
            cipher = j;
         }
      }
      if(plaintext[i]!=' '){
         ciphertext+=c[cipher];
         CipherText+=c[cipher];
```

```
        }
        else{
            CipherText+=' ';
        }

    }

    cout<<"Ciphertext : "<<ciphertext<<endl;

    // decryption
    int plain;
    for(int i=0; i<len; i++){
        for (int j = 0; j < 26; ++j) {
            if (CipherText[i]==c[j]) {
                plain = j;
            }
        }
        if(plaintext[i]!=' '){
            newPlaintext+=p[plain];

        }
        else{
            newPlaintext+=' ';
        }

    }
    cout<<"Decrypted plaintext : "<<newPlaintext<<endl;

return 0;
}
```

Output:

Enter the plaintext : this message is easy to encrypt but hard to find the key

Ciphertext : ICFVQRVVNERFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS

Decrypted plaintext : this message is easy to encrypt but hard to find the key

Result & Discussion:

The Monoalphabetic Substitution Cipher program successfully encrypts and decrypts the input message based on the user-specified key.

The Monoalphabetic Substitution Cipher was successfully implemented, with encryption and decryption functioning correctly. While easy to use, this cipher is susceptible to cryptanalysis, making it unsuitable for strong encryption.

From this experiment we can learn about the following concept:

Key Mapping: Each letter in the plaintext is replaced according to the substitution key, ensuring that both encryption and decryption follow the correct mapping.

Security: Monoalphabetic substitution ciphers are easy to implement but are vulnerable to frequency analysis attacks since the same letter in plaintext always maps to the same letter in ciphertext.

Efficiency: The program handles both uppercase and lowercase letters and leaves non-alphabetic characters unchanged during encryption and decryption.

Usability: The program requires a 26-character key, where each letter of the alphabet is mapped uniquely.

Problem No : 05

Problem Name : A Program for Encryption and Decryption Using Keyless Transposition Cipher .

Objectives: The main objectives of this experiment are given below :

- To implement the Keyless Transposition Cipher for encryption and decryption in C/C++.
- To understand the process of rearranging characters of the message without using a key for secure communication.
- To verify the correctness of encryption and decryption using test inputs.

Theory :

A Keyless Transposition Cipher is a type of cipher where the letters of the plaintext are rearranged based on a fixed pattern, rather than substituted. In keyless transposition, there is no key involved; the pattern of rearrangement is predetermined (e.g., rearranging characters by splitting even and odd indexed characters). For example, in a basic keyless transposition, the characters at even positions are grouped together followed by characters at odd positions.

Algorithm:

Encryption:

1. Input the plaintext message.
2. Remove any spaces from the input.
3. Group all characters at even positions together followed by characters at odd positions.
4. Output the resulting ciphertext.

Decryption:

1. Input the ciphertext.
2. Divide the ciphertext into two halves.
3. Interleave characters from the two halves to reconstruct the original message.
4. Output the resulting decrypted message.

Program Code:

```
#include<bits/stdc++.h>
using namespace std;
int main(){
    int keySize, count=0;
   string plaintext,PlainText="", newPlaintext="", ciphertext="";
   char p[]={'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'};
   char c[]={'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'};
   // input plaintext
    cout<<"Give a plaintext : ";
    getline(cin, plaintext);
    int len = plaintext.length();
    //remove space
     for(int i=0; i<len; i++){
       if(plaintext[i]!=' '){
           PlainText+=plaintext[i];
       }
   }
   int length = PlainText.length();
//encryption :
  for(int i=0; i<length; i=i+2){
   ciphertext+=c[PlainText[i]-'A'-32];
   }
  for(int i=1; i<length; i=i+2){
       ciphertext+=c[PlainText[i]-'A'-32];
   }
cout<<"Ciphertext : "<<ciphertext;
// decryption
```

```
    int half;
    if(length%2==0){
        half = length/2;
    }
    else{
        half = (length/2)+1;
    }
cout<<endl<<"Half length of plaintext: "<<half<<endl;
    for(int i=0; i<half; i++){
        newPlaintext+=p[32-('a'-ciphertext[i])];
        if((i+half)<length){
            newPlaintext+=p[32-('a'-ciphertext[i+half])];
        }
 }
  cout<<"Plaintext : "<< newPlaintext;
 return 0;
}
```

Output:

Give a plaintext : meet me at the park

Ciphertext : MEMATEAKETETHPR

Half length of plaintext: 8

Plaintext : meet me at the park

Result & Discussion:

The Keyless Transposition Cipher program successfully encrypts and decrypts the input message by rearranging the characters based on their positions.

The Keyless Transposition Cipher has successfully implemented, with the program correctly encrypting and decrypting text based on the predetermined character arrangement. Although simple, this cipher can be used as part of more complex encryption systems to enhance security. In the keyless transposition cipher, the plaintext is rearranged based on a fixed pattern without the use of a key. The characters at even and odd positions are grouped separately during encryption.

- **Efficiency**: This cipher is simple and quick to implement, as it only involves rearranging characters, making it computationally efficient.
- **Security**: While transposition ciphers change the order of characters, they do not obscure the frequency of letters, making them susceptible to cryptanalysis if used alone. Combining with other methods enhances security.
- **Usability**: This program efficiently removes spaces from input and handles both uppercase and lowercase characters.

Problem No : 06

Problem Name : A Program for Encryption and Decryption Using Keyed Transposition Cipher.

Objectives: The main objectives of this experiment are given below :

- To implement encryption and decryption using a Keyed Transposition Cipher in C/C++.
- To learn how reordering the characters of a message based on a given key strengthens encryption.
- To test the program with various inputs and verify the correctness of encryption and decryption.

Theory:

A Keyed Transposition Cipher is a type of transposition cipher where the characters of the plaintext are rearranged according to a key. The key is a word or string of characters, and each character in the key determines the column or order in which to arrange the characters in the plaintext.

Algorithm :

Encryption:

1. Input the plaintext and the key.
2. Remove spaces from the plaintext.
3. Create a grid by filling the plaintext into rows, where the number of columns is equal to the length of the key.
4. Rearrange the columns according to the alphabetical order of the characters in the key.
5. Read off the ciphertext by concatenating the rearranged columns.

Decryption:

1. Input the ciphertext and the key.
2. Determine the number of rows based on the ciphertext length and key length.
3. Fill the columns in the grid based on the alphabetical order of the key.
4. Rearrange the columns back to their original order using the key.
5. Read off the decrypted plaintext row by row.

Program Code :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
 string plaintext,PlainText="", newPlaintext="", ciphertext="";
   char p[]={'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'};
   char c[]={'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'};
   cout<<"Give a plaintext : ";
   getline(cin, plaintext);
   int block;
   cout<<"Enter block size : ";
   cin>>block;
   int keys[block];
   cout<<"Enter keys : ";
   for(int i=0; i<block; i++){
   int tmp;
      cin>>tmp;
      keys[i]=tmp-1;
   }
int len = plaintext.length();
   for(int i=0; i<len; i++){
      if(plaintext[i]!=' '){
         PlainText+=plaintext[i];
      }
   }
   int length = PlainText.length();
   int rows;
   if(length%block==0){
```

```
    rows=length/block;
  }
  else{
    rows=(length/block)+1;
    for(int i=1; i<block; i++){
      PlainText+='z';
    }
  }
  for(int i=0; i<rows; i++){
   for(int j=0; j<block; j++){
      int t=j;
      j=keys[t];
      ciphertext+= c[PlainText[(i*block)+j]-'A'-32];
      j=t;
    }
  }
cout<<endl<<"Ciphertext : "<<ciphertext<< endl;
for(int i=0; i<rows; i++){
    for(int j=0; j<block; j++){
      int t;
      for(int l=0;l<block; l++){
        if(j==keys[l]){
          t=l;
        }
      }
      newPlaintext+= p[32-('a'-ciphertext[(i*block)+t])];
    }
  }
```

```
cout<<"Plaintext : ";
   for(int i=0; i<length; i++){
      cout<<newPlaintext[i];
   }
}
```

Output:

Enter the plaintext:enemy attacks tonight

Enter block size:5

Enter keys:3 1 4 5 2

The Ciphertext is:EEMYNTAACTTKONSHITZG

The Decrypted plaintext is:enemy attacks tonight

Result & Discussion :

The Keyed Transposition Cipher program successfully encrypts and decrypts the input message using the provided key.The Keyed Transposition Cipher has successfully implemented, and both encryption and decryption processes worked as expected. While the cipher strengthens encryption by rearranging characters based on a key, it can still be vulnerable to attacks like pattern recognition, especially if the key is short or predictable.

In this experiment we can learn :

Encryption: The plaintext is organized into columns based on the key, and the columns are rearranged according to the alphabetical order of the key.

Decryption: The decryption process reverses the column rearrangement based on the original key order, restoring the message.

Key Importance: The security of the transposition cipher depends on the key. The same key must be used for both encryption and decryption.

Efficiency: This method is straightforward to implement but provides moderate security, as it only scrambles the positions of characters without changing their values.

Problem No : 07

Problem Name : A Program for Encryption and Decryption Using Autokey Cipher.

Objectives:

The main objectives of this experiment is shown below :

- To implement encryption and decryption using the Autokey Cipher in C/C++.
- To understand and apply the concept of key generation using plaintext in the encryption process.
- To verify the correct implementation of the Autokey Cipher by testing encryption and decryption.

Theory :

The Autokey Cipher is a polyalphabetic substitution cipher that uses a keyword to encrypt a message, but unlike traditional polyalphabetic ciphers like the Vigenère cipher, the Autokey Cipher utilizes the plaintext itself as part of the key. This approach enhances security by making the key dynamic and dependent on the content of the message.

The Autokey cipher is a polyalphabetic substitution cipher, similar to the Vigenère cipher, but with a key that is extended by the plaintext itself, making it more secure against frequency analysis.

Steps for Encryption:

1. **Generate the Key**: Start with a keyword, then append the plaintext (without the first letter) to the end of the keyword to form the full key.
2. **Encrypt**: Each plaintext letter is shifted based on the corresponding letter in the key. The shift is determined by the position of the key letter in the alphabet.

Decryption:

The decryption process involves using the keyword to decrypt the first few letters, after which the plaintext letters are used to decrypt the rest.

The Autokey cipher improves upon the Vigenère cipher by reducing repeated patterns in the key, making cryptanalysis more difficult. However, it still has vulnerabilities if the keyword or plaintext structure is predictable.

Algorithm :

Encryption:

1. Input the plaintext message and the initial key.
2. Concatenate the plaintext to the key (excluding the last characters) to form the full key for encryption.
3. For each character in the plaintext:
    - Convert the character to its numeric equivalent (A = 0, B = 1, ..., Z = 25).
    - Convert the corresponding character from the key.
    - Apply the encryption formula:

      $$C=(P+K) \bmod 26,$$

      where P is the plaintext character and K is the key character.
4. Convert the result back to a character.
5. Output the ciphertext.

Decryption:

1. Input the ciphertext and the initial key.
2. For each character in the ciphertext:
    - Convert the character to its numeric equivalent (A = 0, B = 1, ..., Z = 25).
    - Use the initial key and the previous decrypted characters as part of the decryption key.
    - Apply the decryption formula:

      $$P=(C-K+26) \bmod 26 ;$$

      where C is the ciphertext character and K is the key character.
3. Convert the result back to a character.
4. Output the decrypted plaintext.

Program Code :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
   int key,ciphercode;
   string plaintext,ciphertext, mainCipher, newPlaintext;
 char p[]={'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'};
   char c[]={'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'};
cout<<"Enter the plaintext : ";
getline(cin, plaintext);
 cout<<"Enter the initial key value : ";
   cin>>key;
   int len = plaintext.length();
int pre=0;
// Encryption
   for(int i=0;i<len;i++){
      for(int j=0;j<26;j++){
         if(plaintext[i]==p[j]){
            if(i==0)
            {
               ciphercode = (j+key)%26;
               pre = j;
            }
            else{
               ciphercode = (j+pre)%26;
               pre=j;
            }
            break;
```

```
        }
      }
      if(plaintext[i]!=' '){
        ciphertext+=c[ciphercode];
        mainCipher+=c[ciphercode];
      }
else
        ciphertext+=' ';
   }
cout<<endl<<"Ciphertext : "<<mainCipher<<endl;
 // decryption
   int pree, plaincode;;
   for(int i=0;i<len;i++){
      for(int j=0;j<26;j++){
        if(ciphertext[i]==c[j]){
          for(int k=0; k<26; k++){
            if(i>0 && plaintext[i-1]!=' ' && plaintext[i-1]==p[k]){
              pree= k;
            }
            else if(i>0 && plaintext[i-1]==' ' && plaintext[i-2]==p[k]){
              pree= k;
            }
          }
          if(i==0)
          {
            if(j-key<0)
              plaincode = (j-key+26)%26;
            else
```

```
                plaincode = (j-key)%26;
            }
            else{
               if(j-pree<0)
                  plaincode = (j-pree+26)%26;
               else
                  plaincode = (j-pree)%26;
            }
            break;
         }
      }
      if(plaintext[i]!=' '){
         newPlaintext+=p[plaincode];
      }
else
         newPlaintext+=' ';
   }
cout<<endl<<"Plaintext : "<<newPlaintext<<endl;
 return 0;
}
```

Output:

Enter the plaintext : attack is today

Enter the initial key value : 12

Ciphertext : MTMTCMSALHRDY

Plaintext : attack is today

Result & Discussion:

The Autokey Cipher program successfully encrypts and decrypts the given plaintext using the Autokey Cipher method.

The Autokey Cipher has implemented successfully, and the program correctly encrypts and decrypts the given input. The use of the plaintext to extend the key enhances the security of the cipher but still requires careful consideration of its limitations.

In this experiment , we can learn the following :

Key Generation: The key is extended using the plaintext itself, creating a dynamic and unique key for each encryption.

Security: The Autokey Cipher enhances the security compared to a basic Vigenère Cipher by using the plaintext to modify the key, reducing the predictability of the cipher.

Efficiency: The program works efficiently for uppercase letters and ensures the correct transformation of both encryption and decryption.

Limitation: The Autokey Cipher, while stronger than a static key cipher, still remains vulnerable to some cryptanalysis techniques like known-plaintext attacks.

Problem No : 08

Problem Name : A Program for Encryption and Decryption Using Vigenère Cipher

Objectives :

The main objectives of the experiment is given below :

- To implement encryption and decryption using the Vigenère Cipher in C/C++.
- To apply the Vigenère Cipher on a message using a keyword for repetitive shifting.
- To verify the accuracy of encryption and decryption processes with user inputs.

Theory:

The Vigenère cipher is a polyalphabetic substitution cipher that uses a keyword to encrypt the plaintext. Unlike simple substitution ciphers, which use a single alphabet for all letters, the Vigenère cipher applies different Caesar shifts to each letter based on the keyword.

Steps for Encryption:

1. **Generate a Vigenère Table**: A 26x26 grid where each row represents a Caesar cipher shift.
2. **Keyword Expansion**: The keyword is repeated or truncated to match the length of the plaintext.
3. **Encrypt**: For each letter in the plaintext, find the intersection of the row corresponding to the plaintext letter and the column corresponding to the keyword letter. This gives the cipher letter.

Decryption:

To decrypt, use the same keyword and reverse the shift. For each cipher letter, find the corresponding row (based on the keyword) and identify the plaintext letter from the first column of that row.

The Vigenère cipher is more secure than simple substitution ciphers as it disguises letter frequencies by using multiple shifting alphabets. However, it is vulnerable to cryptanalysis if the keyword is short or repeated frequently.

Algorithm :

Encryption:

1. Input the plaintext and keyword.
2. Repeat the keyword to match the length of the plaintext.
3. For each character in the plaintext:
    - Convert it to its numeric equivalent (A = 0, B = 1, ..., Z = 25).
    - Shift the character based on the corresponding letter of the keyword using the formula:
      $C_i = (P_i + K_i) \bmod 26$, where:
        - C_i is the encrypted character,
        - P_i is the plaintext character,
        - K_i is the corresponding keyword character.
4. Output the ciphertext.

Decryption:

1. Input the ciphertext and keyword.
2. Repeat the keyword to match the length of the ciphertext.
3. For each character in the ciphertext:
    - Convert it to its numeric equivalent.
    - Reverse the shift using the following formula :

      $$P_i = (C_i - K_i + 26) \bmod 26$$, where:

        - P_i is the decrypted character.
4. Output the decrypted text.

Program Code :

```
#include<bits/stdc++.h>
using namespace std;
int main(){
 int keys[50], plainCodes[100];
  string keyword, plaintext, PlainText,ciphertext, mainCipher, newPlaintext;
  char p[]={'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'};
  char c[]={'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'};
    cout<<"Enter the plaintext : ";
    getline(cin, plaintext);
   int len = plaintext.length();
  cout<<"Enter the keyword : ";
  cin>>keyword;
 int keylen = keyword.length();
    for(int i=0; i<keylen; i++){
        for(int j=0; j<26; j++){
            if(keyword[i]==c[j]){
                keys[i] = j;
                break;
            }
        }
    }
    cout<<endl<<"key Stream : ";
    for(int i=0; i<keylen; i++){
        cout<<keys[i]<<" ";
    }    for(int i=0; i<len; i++){
        if(plaintext[i]!=' '){
            PlainText+=plaintext[i];
        }
```

```
    }
  int length = PlainText.length();
    for(int i=0; i<length; i++){
        for(int j=0; j<26; j++){
            if(PlainText[i]==p[j]){
                plainCodes[i] = j;
                break;
            }
        }
    }
    // print all P's value
    cout<<endl<<"P's value : ";
    for(int i=0; i<length; i++){
        cout<<plainCodes[i]<<" ";
    }
int cipherCode[100];
    for(int i=0;i<length;i++){
        ciphertext+=c[(plainCodes[i]+keys[i%keylen])%26];
        cipherCode[i]=(plainCodes[i]+keys[i%keylen])%26;
    }
cout<<endl<<"Ciphertext : "<<ciphertext<<endl;
    for(int i=0;i<length;i++){
        if((cipherCode[i]-keys[i%keylen])<0)
            newPlaintext+=p[(cipherCode[i]-keys[i%keylen]+26)%26];

        else
            newPlaintext+=p[(cipherCode[i]-keys[i%keylen])%26];
  }
    newPlaintext+=' ';
 cout<<"Plaintext: "<<newPlaintext<<endl;
```

```
  return 0;
}
```

Output:

Enter the plaintext : life is full of surprises

Enter the keyword : health

key Stream : 7 4 0 11 19 7

P's value : 11 8 5 4 8 18 5 20 11 11 14 5 18 20 17 15 17 8 18 4 18

Ciphertext : SMFPBZMYLWHMZYRAKPZIS

Plaintext: life is full of surprises

Result & Discussion :

The Vigenère Cipher program successfully encrypts and decrypts the input message using the provided keyword, matching the expected results.

The Vigenère Cipher has successfully implemented, and both encryption and decryption were tested with accurate results. This cipher is more secure than simpler substitution ciphers but still has vulnerabilities to more advanced cryptanalysis techniques such as the Kasiski examination.

- **Key Repetition**: The key is repeated to match the length of the plaintext, ensuring that each letter of the plaintext is encrypted based on the corresponding letter of the keyword.
- **Security**: The Vigenère Cipher is more secure than the Caesar Cipher because it uses a sequence of different shifts based on the keyword, making it harder to break using frequency analysis.
- **Case Sensitivity**: This implementation handles only uppercase letters for simplicity. Modifications can be made to support lowercase letters and non-alphabetic characters if needed.
- **Efficiency**: The program efficiently handles both encryption and decryption with linear time complexity based on the size of the input message.

Problem No : 09

Problem Name : A Program for Encryption and Decryption Using Hill Cipher .

Objectives: The main objectives of this experiment are given below:

- To implement the Hill Cipher for encryption and decryption in C/C++.
- To understand matrix-based encryption and its application in cryptography.
- To test the correctness of encryption and decryption using Hill Cipher for a user-defined key matrix.

Theory :

The Hill Cipher is a polyalphabetic substitution cipher based on linear algebra, specifically matrix multiplication. It was invented by Lester S. Hill in 1929 and is notable for its ability to encrypt blocks of letters simultaneously, rather than one letter at a time like simpler ciphers. This method enhances the security of the encryption process. Hill Cipher uses matrix multiplication to encrypt plaintext and decrypt ciphertext .The key is a square matrix that must be invertible in modulo 26 for decryption to be possible.

Algorithm :

Encryption :

1. Input the plaintext and the key matrix (ensure the key matrix is invertible).
2. If the length of the plaintext is not divisible by the matrix size, pad it with filler characters.
3. Convert the plaintext into numerical form, where 'A' = 0, 'B' = 1, ..., 'Z' = 25.
4. Group the plaintext into vectors, each of the size equal to the matrix dimension.
5. Multiply each vector by the key matrix, and take the result modulo 26.
6. Convert the resulting numbers back into characters to get the ciphertext.

Decryption :

1. Calculate the inverse of the key matrix modulo 26.
2. Input the ciphertext and split it into vectors, similar to the encryption process.
3. Multiply each ciphertext vector by the inverse key matrix, and take the result modulo 26.
4. Convert the result back into characters to obtain the plaintext.

Program Code :

```
#include <bits/stdc++.h>
using namespace std;
int modInv(int a) {
    a = ((a % 26) + 26) % 26;
    for (int x = 1; x < 26; x++) {
        if ((a * x) % 26 == 1)
            return x;  }
    return -1; }
int det(int mat[2][2]) {
    return mat[0][0] * mat[1][1] - mat[0][1] * mat[1][0]; }
string decryption(string cipher, int mat[2][2]) {
    string plain = "";
    int deter = det(mat);
    int invDet = modInv(deter);
    int invMat[2][2];
    invMat[0][0] = (mat[1][1] * invDet) % 26;
    invMat[0][1] = (((-1 * (mat[0][1] * invDet) % 26) + 26) % 26);
    invMat[1][0] = (((-1 * (mat[1][0] * invDet) % 26) + 26) % 26);
    invMat[1][1] = (mat[0][0] * invDet) % 26;
    for (int i = 0; i < cipher.size(); i += 2) {
        int a = cipher[i] - 'A';
        int b = cipher[i + 1] - 'A';
        int x = (a * invMat[0][0] + b * invMat[1][0]) % 26;
        int y = (a * invMat[0][1] + b * invMat[1][1]) % 26;
        plain += (x + 'a');
```

```
        plain += (y + 'a');    }
    return plain; }
string encryption(string plain, int mat[2][2]) {
    string cipher = "";
    if (plain.size() % 2 == 1) {
        plain += 'z'; // Padding if length is odd }
    for (int i = 0; i < plain.size(); i += 2) {
        int a = plain[i] - 'a';
        int b = plain[i + 1] - 'a';
        int x = (a * mat[0][0] + b * mat[1][0]) % 26;
        int y = (a * mat[0][1] + b * mat[1][1]) % 26;
        cipher += (x + 'A');
        cipher += (y + 'A');  }
    return cipher; }
int main() {
    string plain;
    int mat[2][2], row = 2, col = 2;
    cout << "Enter the plaintext: ";
    getline(cin, plain); // Allow input with spaces
    plain.erase(remove(plain.begin(), plain.end(), ' '), plain.end());
    cout << "Enter the 2x2 Matrix:\n";
    for (int i = 0; i < row; i++)
        for (int j = 0; j < col; j++)
            cin >> mat[i][j];
    if (det(mat) == 0 || modInv(det(mat)) == -1) {
        cout << "Matrix is not valid\n";
        return 0;   }
    string cipher = encryption(plain, mat);
```

```
cout << "Ciphertext: " << cipher << endl;

cout << "Plain
```

Input & Output:

Enter the plaintext: we live in an insecure world

Enter the 2x2 Matrix:

3 2

5 7

Ciphertext: IUVAFSLDNNLDWMCOTKGMCHEZ

Plaintext: we live in an insecure world

Result and Discussion :

The Hill Cipher program successfully encrypted and decrypted the given message based on the provided key matrix.The Hill Cipher has successfully implemented using matrix multiplication, and both encryption and decryption functions worked as expected. This experiment demonstrated the importance of invertible key matrices for the cipher's reversibility, while padding ensures consistent block size.

From this experiment, we can learn the following :

Matrix Multiplication: The encryption uses matrix multiplication to transform plaintext into ciphertext. Decryption is done by multiplying the ciphertext with the inverse matrix.

Padding: If the plaintext length is not divisible by the matrix dimension, padding (e.g., 'X') is added to make it compatible.

Invertibility: The key matrix must be invertible modulo 26 for decryption to work; otherwise, the ciphertext cannot be decrypted.

Efficiency: The program efficiently handles matrix-based encryption for small text sizes but may require optimization for larger matrix dimensions.

Problem No : 10

Problem Name : A Program for Encryption and Decryption Using Playfair Cipher .

Objectives :

The main objectives of the experiment are given below :

- To implement the Playfair Cipher algorithm for encryption and decryption in C/C++.
- To understand how digraphs are used in the Playfair Cipher.
- To evaluate the program's performance through practical testing.

Theory :

The Playfair cipher is a digraph substitution cipher, meaning it encrypts pairs of letters (digraphs) rather than single letters. It was invented by Sir Charles Wheatstone in 1854 and popularized by Lord Playfair. The encryption process involves creating a 5x5 matrix of letters using a keyword, which omits the letter 'J' (combining it with 'I').

Steps for Encryption:

1. **Generate the Key Square**: A 5x5 matrix is filled with the keyword (without repeating letters) and the remaining letters of the alphabet.
2. **Divide Plaintext into Pairs**: The message is split into pairs of letters. If both letters in a pair are the same, a filler like 'X' is added between them.
3. **Encrypt Using Rules**:
    - If the letters appear in the same row, each is replaced by the letter to its right.
    - If they appear in the same column, each is replaced by the letter below it.
    - If they form a rectangle, each letter is replaced by the letter in its respective row but in the other pair’s column.
4. **Decryption**: The reverse of the encryption process, applying the same key square but shifting positions accordingly to retrieve the original message.

The Playfair cipher improves security by encrypting digraphs instead of single letters, making frequency analysis more challenging.

Algorithm :

Key Generation:

1. **Input the Key**: Accept a keyword from the user.
2. **Create the 5x5 Matrix**:
    - Remove duplicate letters from the key.
    - Replace 'J' with 'I' if present, as the matrix consists of only 25 letters.
    - Fill the matrix with the letters of the keyword first, followed by the remaining letters of the alphabet (A-Z).

Text Preparation:

1. **Input the Plaintext**: Accept the plaintext message from the user.
2. **Format the Plaintext**:
    - Remove spaces and convert to uppercase.
    - Split the text into digraphs (pairs of letters).
    - If a pair contains the same letter (e.g., "LL"), replace the second letter with 'X' (e.g., "LX").
    - If the plaintext has an odd number of characters, append 'X' to the last character.

Encryption:

1. **For each digraph**:
    - Find the positions of the two letters in the matrix.
    - If both letters are in the same row, replace them with the letters immediately to their right (wrap around if needed).
    - If both letters are in the same column, replace them with the letters immediately below (wrap around if needed).
    - If the letters form a rectangle, replace them with letters on the same row but at the opposite corners.

Decryption:

1. **For each digraph**:
    - Find the positions of the two letters in the matrix.

- If both letters are in the same row, replace them with the letters immediately to their left (wrap around if needed).
- If both letters are in the same column, replace them with the letters immediately above (wrap around if needed).
- If the letters form a rectangle, replace them with letters on the same row but at the opposite corners.

Program Code :

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
using namespace std;
char playfairMatrix[5][5];
void createMatrix(string key) {
   string matrixKey = "";
   bool letterPresent[26] = {false};
   letterPresent['J' - 'A'] = true;  // Treat 'I' and 'J' as the same letter
   for (char c : key) {
      if (!letterPresent[toupper(c) - 'A']) {
         matrixKey += toupper(c);
         letterPresent[toupper(c) - 'A'] = true;  }  }
   for (char c = 'A'; c <= 'Z'; ++c) {
      if (!letterPresent[c - 'A']) {
         matrixKey += c;  }  }
   int k = 0;
   for (int i = 0; i < 5; i++) {
      for (int j = 0; j < 5; j++) {
         playfairMatrix[i][j] = matrixKey[k++];  }}}
pair<int, int> findPosition(char c) {
```

```
    if (c == 'J') c = 'I';  // Treat 'I' and 'J' as the same letter
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            if (playfairMatrix[i][j] == c) {
                return {i, j};} }   }
    return {-1, -1};}
string encryptPair(char a, char b) {
    pair<int, int> posA = findPosition(a);
    pair<int, int> posB = findPosition(b);
    if (posA.first == posB.first) {
        return string(1, playfairMatrix[posA.first][(posA.second + 1) % 5]) +
               string(1, playfairMatrix[posB.first][(posB.second + 1) % 5]);
    } else if (posA.second == posB.second) {
        return string(1, playfairMatrix[(posA.first + 1) % 5][posA.second]) +
               string(1, playfairMatrix[(posB.first + 1) % 5][posB.second]);
    } else {
        return string(1, playfairMatrix[posA.first][posB.second]) +
               string(1, playfairMatrix[posB.first][posA.second]); }}
string encrypt(string plaintext, string key) {
    createMatrix(key);
    string formattedText = formatPlaintext(plaintext);
    string ciphertext = "";
    for (size_t i = 0; i < formattedText.length(); i += 2) {
        ciphertext += encryptPair(formattedText[i], formattedText[i + 1]);   }
    return ciphertext;}
int main() {
    string plaintext, key;
    cout << "Enter the plaintext: ";
```

```
    getline(cin, plaintext);
    cout << "Enter the key: ";
    getline(cin, key);
    string ciphertext = encrypt(plaintext, key);
    cout << "Encrypted
}
```

Input & Output:

Enter the plaintext:thekeyishiddenunderthepoorpad

Enter the key:guidance

Matrix:

G U I D A

N C E B F

H K L M O

P Q R S T

V W X Y Z

Ciphertext:POCLBXDRLGIYIBCGBGLXPOBILZLTTGIY

Plaintext: thekeyishiddenunderthepoorpadx

Result & Discussion:

The Playfair Cipher program successfully encrypts and decrypts the input plaintext based on the provided key.The Playfair Cipher was successfully implemented, demonstrating effective encryption and decryption. While educational, this cipher is not recommended for secure communications due to its susceptibility to cryptanalysis.

From this experiment, we can learn the following :

- **Matrix Construction**: The program constructs the Playfair matrix correctly by removing duplicates and managing the 'J' character.
- **Digraph Handling**: It effectively forms digraphs, including inserting 'X' where necessary and ensuring that the encryption and decryption processes correctly follow Playfair rules.
- **Efficiency**: The implementation works efficiently for reasonable input lengths, providing a straightforward approach to encryption and decryption.
- **Practicality**: The Playfair cipher is a simple yet effective means of obscuring messages but is vulnerable to frequency analysis and is not secure for modern applications.

Problem No : 11

Problem Name : A Program for Encryption and Decryption Using RSA Public-Key Cryptographic Algorithm .

Objectives : The objectives of this experiment are given below :

- To implement the RSA public-key cryptographic algorithm for encryption and decryption.
- To understand the process of key generation, encryption, and decryption in public-key cryptography.
- To test the algorithm with a sample input and observe the encryption and decryption process.

Theory :

The RSA (Rivest-Shamir-Adleman) Algorithm is one of the most widely used public-key cryptographic algorithms. It is based on the mathematical principle of factoring large prime numbers, which makes it a secure method for encryption and decryption in digital communication.

RSA is an **asymmetric** cryptographic algorithm, which means it uses two different keys .

Algorithm :

Key Generation:

1) Select two prime numbers p and q.
2) Compute n = p * q.
3) Compute Euler's Totient Function φ(n) = (p-1) * (q-1).
4) Choose an integer e such that 1 < e < φ(n) and e is coprime with φ(n).
5) Compute the modular inverse of e, denoted d, such that d * e ≡ 1 (mod φ(n)).
6) The public key is (e, n) and the private key is (d, n).

Encryption:

- Convert the plaintext message M into an integer m such that 0 ≤ m < n.
- Encrypt the message using the public key (e, n): $c=m^e \bmod n$ where c is the ciphertext.

Decryption:

- **Decrypt the ciphertext** using the private key (d, n): $m=c^d \bmod n$.
- Convert the integer m back into the original message.

Program Code :

```
#include <iostream>
#include <cmath>
using namespace std;
int gcd(int a, int b) {
   while (b != 0) {
      int temp = b;
      b = a % b;
      a = temp; }
   return a; }
int modInverse(int e, int phi) {
   for (int d = 1; d < phi; d++) {
      if ((e * d) % phi == 1) {
         return d;   }    }
   return -1;}
long long modExp(long long base, long long exp, long long mod) {
   long long result = 1;
   while (exp > 0) {
      if (exp % 2 == 1) {
         result = (result * base) % mod;   }
      base = (base * base) % mod;
      exp /= 2;   }
   return result;}
int main() {
   int p, q, e;
   cout << "Enter prime number p: ";
   cin >> p;
```

```
    cout << "Enter prime number q: ";
    cin >> q;
    int n = p * q;
    int phi = (p - 1) * (q - 1);
    cout << "Enter public key exponent e (should be coprime with phi): ";
    cin >> e;
    while (gcd(e, phi) != 1) {
        cout << "e is not coprime with phi. Enter a valid e: ";
        cin >> e;
    }
    int d = modInverse(e, phi);
    cout << "\nCalculated Values:" << endl;
    cout << "n (p * q) = " << n << endl;
    cout << "phi (Euler's Totient) = " << phi << endl;
    cout << "d (private key exponent) = " << d << endl;
    cout << "Public Key: (" << e << ", " << n << ")" << endl;
    cout << "Private Key: (" << d << ", " << n << ")" << endl;
    int message;
    cout << "\nEnter a message to encrypt (as an integer): ";
    cin >> message;
    long long encryptedMessage = modExp(message, e, n);
    cout << "Encrypted Message: " << encryptedMessage << endl;
    // Step 6: Decryption
    long long decryptedMessage = modExp(encryptedMessage, d, n);
    cout << "Decrypted Message: " << decryptedMessage << endl;
    return 0;
}
```

Input & Output :

Enter Value of P:13

Enter Value of q:23

Enter The value of e (Coprime with phi):5

Calculated Values:Value of n:299

Value of phi:264

Value of d:53

public key (5,299)

private key (53,299)

Enter message :5

Encryption:135

Decryption:5

Result and Discussion:

The RSA public-key cryptographic algorithm has successfully implemented. The message was encrypted and decrypted accurately using the generated public and private keys.The RSA public-key cryptographic algorithm has implemented successfully, demonstrating how encryption and decryption occur through the use of public and private keys. This experiment highlights the strength of RSA in securing communication, but also points to the computational complexity involved with large key sizes.

From this experiment, we can learn the following :

- **Public and Private Keys**: The RSA algorithm uses two keys: a public key for encryption and a private key for decryption. The security of the system relies on the difficulty of factoring large numbers.
- **Key Generation**: Choosing two large prime numbers ensures that breaking the RSA encryption is computationally infeasible.
- **Message Representation**: The plaintext message must be converted into a numeric format before encryption.
- **Efficiency**: While the algorithm works well for small integers, practical RSA implementations use optimized libraries for larger key sizes.