**DR B.R. AMBEDKAR NATIONAL INSTITUTE OF**

**TECHNOLOGY JALANDHAR**



LAB FILEOFInformation Security SystemsCSX-426

Session: Jan - May 2021

**SUBMITTED TO-**

Dr. Samayveer Singh

Assistant Professor

CSE Department

**SUBMITTED BY-**

Ankit Goyal

Roll No.- 17103011

Group - G-1

Branch - CSE

**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No** | **Table of contents** | **Page Numbers** | **Remarks** |
| **1.** | Implementation of Basic and Extended Euclidean Algorithm | 3 |  |
| **2.** | Implementation Shift Cipher and Mono-Alphabetic Cipher Algorithm | 6 |  |
| **3**. | Implementation of Playfair and Hill Cipher Algorithm | 9 |  |
| **4.** | Implementation of Affine and Autokey Cipher Algorithm | 15 |  |
| **5.** | Implement DES algorithm (Encryption and Decryption) | 19 |  |
| **6.** | Implementation AES(Advanced Encryption Standard) Algorithm | 26 |  |
| **7.** | Implement the following Modern Block Ciphers techniques.1) Electronic Codebook (ECB) Mode 2) Cipher Block Chaining (CBC) Mode 3) Cipher Feedback (CFB) Mode 4) Output Feedback (OFB) Mode 5) Counter (CTR) Mode | 36 |  |
| 8. | Implementation Miller-Rabin Premality Test and Chinese Remainder Theorem | 48 |  |
| 9. | Implement the RSA Algorithm | 52 |  |
| 10. | Implement ElGamal cryptosystem | 54 |  |
| 11. | To Implement Diffie-Hellman Key Exchange Algorithm | 57 |  |
| 12. | Implement MD5 Hash Algorithm | 59 |  |

**Experiment -1**

**Aim:** Implementation of Basic and Extended Euclidean Algorithm.

**Theory:**

**Basic Euclidean Algorithm:**The algorithm is based on the below facts.

* If we subtract a smaller number from a larger (we reduce a larger number), GCD doesn’t change. So if we keep subtracting repeatedly the larger of two, we end up with GCD.
* Now instead of subtraction, if we divide the smaller number, the algorithm stops when we find remainder 0.

**Extended Euclidean Algorithm:**

Extended Euclidean algorithm also finds integer coefficients x and y such that:

**ax + by = gcd(a, b)**

The extended Euclidean algorithm updates results of gcd(a, b) using the results calculated by recursive call gcd(b%a, a).

**Program:**

**Basic Euclidean Algorithm:**

#include <bits/stdc++.h>

using namespace std;

int gcd(int a, int b)

{

if (a == 0)

return b;

return gcd(b % a, a);

}

int main()

{

int a,b;

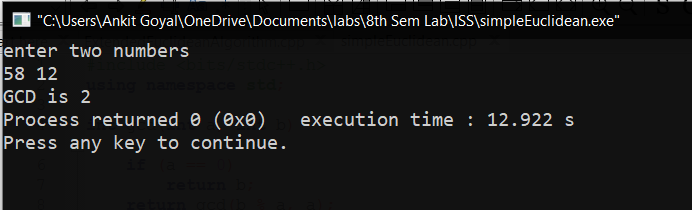
cout<<"enter two numbers\n";

cin>>a>>b;

cout << "GCD is " << gcd(a, b);

}

**Output:**



**Extended Euclidean Algorithm:**

#include <bits/stdc++.h>

using namespace std;

int gcdExtended(int a, int b, int \*x, int \*y)

{

if (a == 0)

{

\*x = 0;

\*y = 1;

return b;

}

int x1, y1;

int gcd = gcdExtended(b%a, a, &x1, &y1);

\*x = y1 - (b/a) \* x1;

\*y = x1;

return gcd;

}

int main()

{

int a,b,x, y;

cout<<"enter two numbers\n";

cin>>a>>b;

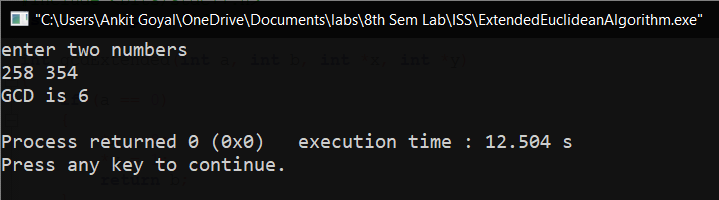
int g = gcdExtended(a, b, &x, &y);

cout << "GCD is " << g << endl;

return 0;

}

**Output:**



**Experiment -2**

**Aim:** Implementation Shift Cipher and Mono-Alphabetic Cipher Algorithm.

**Theory:**

**Shift cipher:**

In this cipher, the characters of the string is converted to encrypted form using shift cipher. This is one of the part of mono-alphabetic cipher, where each letter is substituted by another letter which is the encrypted form of the original character. The example of this type of cipher is Caeser cipher. In caeser cipher, a shift of three is used to encrypt the original data. Meanwhile, these type of ciphers are not very good at encrypting the data, and can be decrypted easily.

**Mono-alphabetic cipher:**

Mono alphabetic cipher is a substitution cipher in which for a given key, the cipher alphabet for each plain alphabet is fixed throughout the encryption process. For example, if ‘A’ is encrypted as ‘D’, for any number of occurrences in that plaintext, ‘A’ will always get encrypted to ‘D’. And it is different from Poly-alphabetic cipher, as it has fixed character for all occurrence. But in poly-alphabetic, there is different encrypted character for different occurrence of the same character.

Shift cipher is one of the way of mono-alphabetic cipher. However, here using a different approach to cipher the text.

Using an encryption string, and using that for decryption as well.

The Encryption string is: qwertyuioplkjhgfdsazxcvbnm

The Decryption string is: abcdefghijklmnopqrstuvwxyz

**Code:**

**Shift cipher:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

string s;

int k;

int i;

cout<<"Input the string :";

cin>>s;

cout<<"Input cipher key : ";

cin>>k;

int n = s.length();

/// Encryption ==>

string encrypted = s;

for(i=0;i<n;i++)

{

if(s[i]>='a' && s[i]<='z')

encrypted[i] = (s[i]-'a'+k)%26 + 'a';

else

encrypted[i] = (s[i]-'A'+k)%26 + 'A';

}

cout<<"Encrypted : "<<encrypted<<"\n";

/// Decryption ==>

string decrypted = encrypted;

for(i=0;i<n;i++)

{

if(s[i]>='a' && s[i]<='z')

decrypted[i] = (encrypted[i]-'a'-k+26)%26 + 'a';

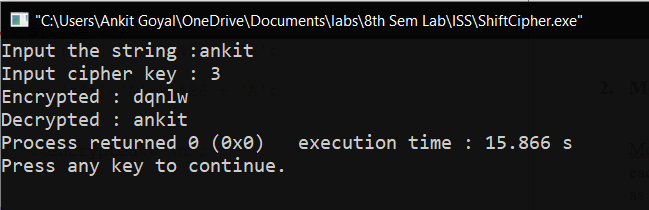
else

decrypted[i] = (encrypted[i]-'A'-k+26)%26 + 'A';

}

cout<<"Decrypted : "<<decrypted;

}



**Mono-alphabetic cipher:**

#include <bits/stdc++.h>

using namespace std;

int main()

{

string EncryptionCipher = "qwertyuioplkjhgfdsazxcvbnm";

string DecryptionCipher = "sxvqcponhmlkzyijadregwbuft";

string s;

int i;

cout<<"Input the string :";

cin>>s;

int n = s.length();

/// Encryption ==>

string encrypted = s;

for(i=0;i<n;i++)

{

if(s[i]>='a' && s[i]<='z')

encrypted[i] = EncryptionCipher[s[i]-'a'];

else

encrypted[i] = EncryptionCipher[s[i]-'A']-'a'+'A';

}

cout<<"Encrypted : "<<encrypted<<"\n";

/// Decryption ==>

string decrypted = encrypted;

for(i=0;i<n;i++)

{

if(s[i]>='a' && s[i]<='z')

decrypted[i] = DecryptionCipher[encrypted[i]-'a'];

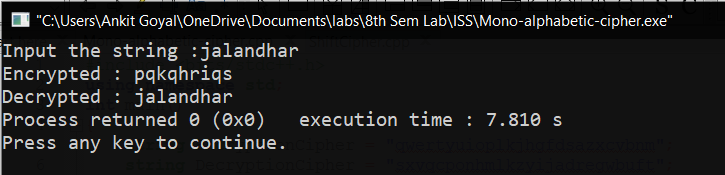
else

decrypted[i] = DecryptionCipher[encrypted[i]-'A']-'a'+'A';

}

cout<<"Decrypted : "<<decrypted;

}



**Experiment -3**

**Aim:** Implementation of Playfair and Hill Cipher Algorithm.

**Theory:**

**Playfair Cipher:**

In this scheme, pairs of letters are encrypted, instead of single letters as in the case of simple substitution cipher.

Initally, a key table is created. The key table is a 5x5 grid of alphabets that acts as the key for encrypting the plaintext. Each of the 25 alphabets must be unique and one letter of the alphabet (usually J) is omitted from the table, as we need only 25 alphabets instead of 26. If the text contains J, then it is replaced by I.

Process of playfair cipher

* A plaintext is split into pairs of two letters (digraphs). If there is an odd number of letters, a Z letter is added to the last letter.
* The rules of encryption are
  + If both the letters are in the same column, take the letter below each one.
  + If both are in same row, then taking right of each one.
  + Else, form a rectangle with the two letters and take the horizontal opposite corner of the rectangle.
* The decryption can also take place by doing these rules in reverse.

**Hill Cipher:**

This is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. The characters are mapped by the scheme as A with 0, B with 1, … , Z with 25.

To encrypt a message, each block of n letters is multiplied by an invertible n\*n matrix, against modulo 26. To decrypt a message, each block is multiplied by the inverse of the matrix used for encryption. The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n\*n matrix (modulo 26).

Here is the code for n = 3.

Hence, there will be the plaintext of only 3 length.

**Code:**

#include<bits/stdc++.h>

using namespace std;

pair<int,int> findInTable(char table[5][5], char x)

{

for(int i=0;i<5;i++)

{

for(int j=0;j<5;j++)

{

if(table[i][j]==x)

{

pair<int,int>ans = make\_pair(i,j);

return ans;

}

}

}

}

int main()

{

string key = "monarchy";

int i,j;

/// Generating table

char table[5][5];

bool arr[26];

memset(arr,false,sizeof(arr));

int index = 0;

for(i=0;i<key.length();i++)

{

if(arr[key[i]-'a']==false && key[i]!='j')

{

table[index/5][index%5] = key[i];

arr[key[i]-'a'] = true;

index++;

}

}

for(i=0;i<26;i++)

{

if('a'+i=='j') continue;

if(arr[i]==false)

{

table[index/5][index%5] = 'a'+i;

arr[i] = true;

index++;

}

}

cout<<"5x5 Square Table : \n";

for(i=0;i<5;i++)

{

for(j=0;j<5;j++) cout<<table[i][j]<<" ";

cout<<"\n";

}

cout<<endl;

string originalText;

cout<<"Enter PlainText : ";

cin>>originalText;

int len = originalText.length();

/// Converting to lowercase(if in uppercase)

for(i=0;i<len;i++)

{

if(originalText[i]>='A' && originalText[i]<='Z')

{

originalText[i] = originalText[i]-'A'+'a';

}

}

if(len%2==1)

{

originalText += 'z';

len++;

}

string EncryptedText="";

for(i=0;i<len;i+=2)

{

char a1 = originalText[i];

char a2 = originalText[i+1];

if(a1=='j') a1='i';

else if(a2=='j') a2 = 'i';

pair<int,int>p1 = findInTable(table, a1);

pair<int,int>p2 = findInTable(table, a2);

if(p1.second==p2.second)

{

EncryptedText += table[(p1.first+1)%5][p1.second];

EncryptedText += table[(p2.first+1)%5][p2.second];

}

else if(p1.first==p2.first)

{

EncryptedText += table[p1.first][(p1.second+1)%5];

EncryptedText += table[p1.first][(p2.second+1)%5];

}

else

{

EncryptedText += table[p1.first][p2.second];

EncryptedText += table[p2.first][p1.second];

}

}

cout<<"The Encrypted String is : "<<EncryptedText<<endl;

string DecryptedText = "";

for(i=0;i<len;i+=2)

{

char a1 = EncryptedText[i];

char a2 = EncryptedText[i+1];

pair<int,int>p1 = findInTable(table,a1);

pair<int,int>p2 = findInTable(table,a2);

if(p1.second==p2.second)

{

DecryptedText += table[(p1.first-1 + 5)%5][p1.second];

DecryptedText += table[(p2.first-1 + 5)%5][p2.second];

}

else if(p1.first==p2.first)

{

DecryptedText += table[p1.first][(p1.second-1+5)%5];

DecryptedText += table[p2.first][(p2.second-1+5)%5];

}

else

{

DecryptedText += table[p1.first][p2.second];

DecryptedText += table[p2.first][p1.second];

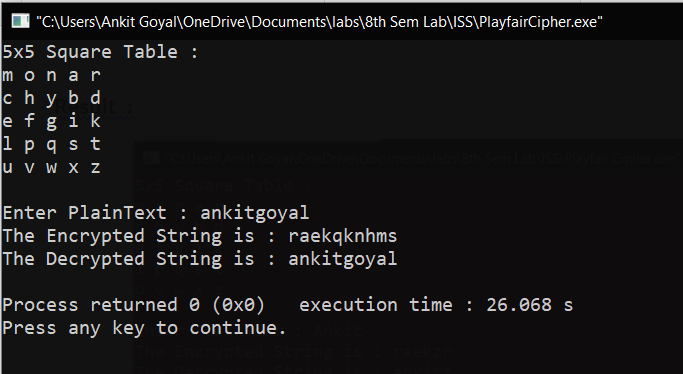
}

}

cout<<"The Decrypted String is : "<<DecryptedText<<endl;

}

**Result:**



**Code:**

#include <bits/stdc++.h>

using namespace std;

string lowercase(string s)

{

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

{

if(s[i]>='A' && s[i]<='Z')

{

s[i] = s[i]-'A'+'a';

}

}

return s;

}

int main()

{

string key = "GYBNQKURP";

key = lowercase(key);

int table[3][3],i,j;

for(i=0;i<3;i++)

for(j=0;j<3;j++)

table[i][j] = key[i\*3+j]-'a';

cout<<"Encryption Table : \n";

for(i=0;i<3;i++)

{

for(j=0;j<3;j++)

cout<<table[i][j]<<" ";

cout<<"\n";

}

cout<<endl;

cout<<"Decryption Table : \n";

/// Decryption

int DecryptTable[3][3];

for(i = 0; i < 3; i++){

for(j = 0; j < 3; j++)

{

DecryptTable[i][j] = (-1\*((table[(j+1)%3][(i+1)%3]\*table[(j+2)%3][(i+2)%3]) - (table[(j+1)%3][(i+2)%3] \* table[(j+2)%3][(i+1)%3]))%26+26)%26;

cout<<DecryptTable[i][j]<<" ";

}

cout<<"\n";

}

cout<<endl;

string OriginalString;

cout<<"Enter string of length 3 : ";

cin>>OriginalString;

OriginalString = lowercase(OriginalString);

int OriginalArray[3];

for(i=0;i<3;i++)

{

OriginalArray[i] = OriginalString[i] - 'a';

}

int EncryptedArray[3];

string EncryptedText = "";

for(i=0;i<3;i++)

{

EncryptedArray[i] = (OriginalArray[0]\*table[i][0] + OriginalArray[1]\*table[i][1] + OriginalArray[2]\*table[i][2])%26;

EncryptedText += 'a'+EncryptedArray[i];

}

cout<<"Encryped Text : "<<EncryptedText;

cout<<endl;

string DecryptedString = "";

for(i=0;i<3;i++)

{

DecryptedString += 'a' + (EncryptedArray[0]\*DecryptTable[i][0] + EncryptedArray[1]\*DecryptTable[i][1] + EncryptedArray[2]\*DecryptTable[i][2])%26;

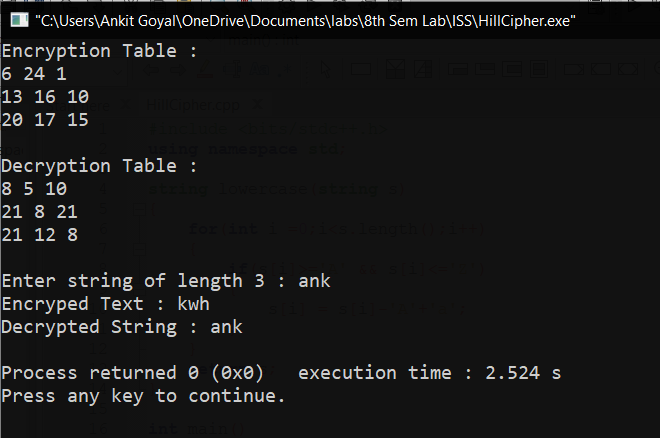
}

cout<<"Decrypted String : "<<DecryptedString;

cout<<endl;

}

**Output:**



**Experiment -4**

**Aim:** Implementation of Affine and Autokey Cipher Algorithm.

**Theory:**

1. **Affine Cipher:**

In this cipher, there are 2 types of keys.

In Additive cipher, the characters are shifted with the key interval to form the encrypted message. In multiplicative cipher, the characters are multiplied with the key to form the new encrypted message. But in affine cipher, there are two keys, of which one behave as additive key, and the other as multiplicative key.

The general Encryption formula for the key :

**New character = ((old character number \* key1) + key2 )%mod;**

The Decryption formula has a key changed, is as :

**New character = (((old character – key2+mod)%mod)\*key3)%mod;**

Where key3 is modular multiplicative inverse of key1 🡺 (key1\*key3)%mod = 1;

**Code :**

#include <bits/stdc++.h>

using namespace std;

string lowercase(string s)

{

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

{

if(s[i]>='A' && s[i]<='Z')

s[i] = s[i]-'A'+'a';

}

return s;

}

int main()

{

string plaintext;

int key1, key2;

cout<<"Enter Plaintext : ";

cin>>plaintext;

cout<<"Enter key(multiplicative and additive ) : ";

cin>>key1>>key2;

cout<<"\n";

plaintext = lowercase(plaintext);

/// Encryption

for(int i = 0;i<plaintext.length();i++)

{

plaintext[i] = 'a' + ((plaintext[i]-'a')\*key1+key2)%26;

}

cout<<"Encrypted String : "<<plaintext<<"\n";

/// Decryption

/// For inverse of key1, we need to find some other key

/// (key1\*x)mod26=(1)mod26

int i = 1;

while((key1\*i)%26!=1)

i++;

int key3 = i;

for(int i = 0;i<plaintext.length();i++)

{

plaintext[i] = 'a'+(((plaintext[i]-'a'-key2+26)%26)\*key3)%26;

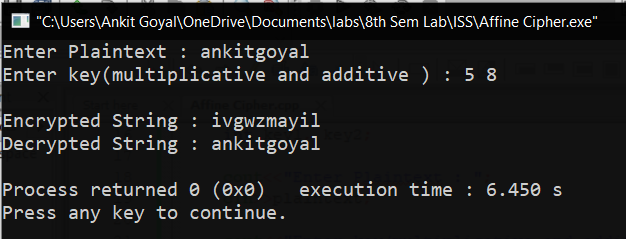
}

cout<<"Decrypted String : "<<plaintext;

cout<<"\n";

}

**Result :**



1. **Autokey Cipher**

This is a polyalphabetic substitution cipher. This cipher is same as vigenere cipher, but the key generation in this and vigenere ciphers are different. In this cipher, keytext is used as it is available, and then the plaintext itself is used to generate the encrypted text.   
  
The encryption formula is   
**new character = (key character + old character)%mod;**

The Decryption formula is

**New character = (old character – key character)%mod;**

This is more secure than vigenere cipher as well.

**Code :**

#include <bits/stdc++.h>

using namespace std;

string lowercase(string s)

{

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

{

if(s[i]>='A' && s[i]<='Z')

s[i] = s[i]-'A'+'a';

}

return s;

}

int main()

{

string plaintext;

string key;

cout<<"Enter Plaintext : ";

cin>>plaintext;

cout<<"Enter key : ";

cin>>key;

cout<<"\n";

plaintext = lowercase(plaintext);

key = lowercase(key);

int n = plaintext.size();

if(n>key.length()) key += plaintext;

/// Encryption

for(int i=0;i<plaintext.length();i++)

plaintext[i] = 'a' + ((plaintext[i]-'a') + (key[i]-'a'))%26;

cout<<"Encryption String : "<<plaintext<<"\n";

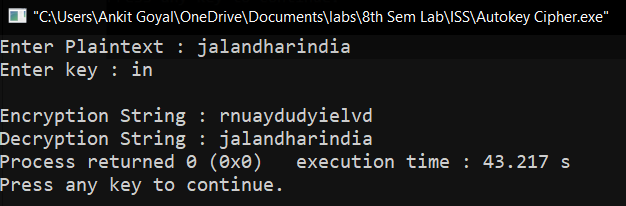
for(int i=0;i<plaintext.length();i++)

plaintext[i] = 'a' + (plaintext[i]- key[i]+26)%26;

cout<<"Decryption String : "<<plaintext;

}

**Result :**



**Experiment -5**

**Aim:** Implement DES algorithm (Encryption and Decryption)

**Theory:** DES is a block cipher, and encrypts data in blocks of size of 64 bit each, means 64 bits of plain text goes as the input to DES, which produces 64 bits of cipher text. The same algorithm and key are used for encryption and decryption, with minor differences. The key length is 56 bits.

**Code :**

#include<bits/stdc++.h>

using namespace std;

string hex2bin(string s)

{

unordered\_map<char, string> mp;

mp['0'] = "0000";

mp['1'] = "0001";

mp['2'] = "0010";

mp['3'] = "0011";

mp['4'] = "0100";

mp['5'] = "0101";

mp['6'] = "0110";

mp['7'] = "0111";

mp['8'] = "1000";

mp['9'] = "1001";

mp['A'] = "1010";

mp['B'] = "1011";

mp['C'] = "1100";

mp['D'] = "1101";

mp['E'] = "1110";

mp['F'] = "1111";

string bin = "";

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

bin += mp[s[i]];

}

return bin;

}

string bin2hex(string s)

{

unordered\_map<string, string> mp;

mp["0000"] = "0";

mp["0001"] = "1";

mp["0010"] = "2";

mp["0011"] = "3";

mp["0100"] = "4";

mp["0101"] = "5";

mp["0110"] = "6";

mp["0111"] = "7";

mp["1000"] = "8";

mp["1001"] = "9";

mp["1010"] = "A";

mp["1011"] = "B";

mp["1100"] = "C";

mp["1101"] = "D";

mp["1110"] = "E";

mp["1111"] = "F";

string hex = "";

for (int i = 0; i < s.length(); i += 4) {

string ch = "";

ch += s[i];

ch += s[i + 1];

ch += s[i + 2];

ch += s[i + 3];

hex += mp[ch];

}

return hex;

}

string permute(string k, int\* arr, int n)

{

string per = "";

for (int i = 0; i < n; i++) {

per += k[arr[i] - 1];

}

return per;

}

string shift\_left(string k, int shifts)

{

string s = "";

for (int i = 0; i < shifts; i++) {

for (int j = 1; j < 28; j++) {

s += k[j];

}

s += k[0];

k = s;

s = "";

}

return k;

}

string xor\_(string a, string b)

{

string ans = "";

for (int i = 0; i < a.size(); i++) {

if (a[i] == b[i]) {

ans += "0";

}

else {

ans += "1";

}

}

return ans;

}

string encrypt(string pt, vector<string> rkb, vector<string> rk)

{

pt = hex2bin(pt);

int initial\_perm[64] = { 58, 50, 42, 34, 26, 18, 10, 2,

60, 52, 44, 36, 28, 20, 12, 4,

62, 54, 46, 38, 30, 22, 14, 6,

64, 56, 48, 40, 32, 24, 16, 8,

57, 49, 41, 33, 25, 17, 9, 1,

59, 51, 43, 35, 27, 19, 11, 3,

61, 53, 45, 37, 29, 21, 13, 5,

63, 55, 47, 39, 31, 23, 15, 7 };

pt = permute(pt, initial\_perm, 64);

cout << "After initial permutation: " << bin2hex(pt) << endl;

string left = pt.substr(0, 32);

string right = pt.substr(32, 32);

cout << "After splitting: L0=" << bin2hex(left)

<< " R0=" << bin2hex(right) << endl;

int exp\_d[48] = { 32, 1, 2, 3, 4, 5, 4, 5,

6, 7, 8, 9, 8, 9, 10, 11,

12, 13, 12, 13, 14, 15, 16, 17,

16, 17, 18, 19, 20, 21, 20, 21,

22, 23, 24, 25, 24, 25, 26, 27,

28, 29, 28, 29, 30, 31, 32, 1 };

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

int per[32] = { 16, 7, 20, 21,

29, 12, 28, 17,

1, 15, 23, 26,

5, 18, 31, 10,

2, 8, 24, 14,

32, 27, 3, 9,

19, 13, 30, 6,

22, 11, 4, 25 };

cout << endl;

for (int i = 0; i < 16; i++) {

string right\_expanded = permute(right, exp\_d, 48);

string x = xor\_(rkb[i], right\_expanded);

string op = "";

for (int i = 0; i < 8; i++) {

int row = 2 \* int(x[i \* 6] - '0') + int(x[i \* 6 + 5] - '0');

int col = 8 \* int(x[i \* 6 + 1] - '0') + 4 \* int(x[i \* 6 + 2] - '0') + 2 \* int(x[i \* 6 + 3] - '0') + int(x[i \* 6 + 4] - '0');

int val = s[i][row][col];

op += char(val / 8 + '0');

val = val % 8;

op += char(val / 4 + '0');

val = val % 4;

op += char(val / 2 + '0');

val = val % 2;

op += char(val + '0');

}

op = permute(op, per, 32);

x = xor\_(op, left);

left = x;

if (i != 15) {

swap(left, right);

}

cout << "Round " << i + 1 << " " << bin2hex(left) << " "

<< bin2hex(right) << " " << rk[i] << endl;

}

string combine = left + right;

int final\_perm[64] = { 40, 8, 48, 16, 56, 24, 64, 32,

39, 7, 47, 15, 55, 23, 63, 31,

38, 6, 46, 14, 54, 22, 62, 30,

37, 5, 45, 13, 53, 21, 61, 29,

36, 4, 44, 12, 52, 20, 60, 28,

35, 3, 43, 11, 51, 19, 59, 27,

34, 2, 42, 10, 50, 18, 58, 26,

33, 1, 41, 9, 49, 17, 57, 25 };

string cipher = bin2hex(permute(combine, final\_perm, 64));

return cipher;

}

int main()

{

string pt, key;

cout<<"Enter plain text(in hexadecimal): ";

cin>>pt;

cout<<"Enter key(in hexadecimal): ";

cin>>key;

key = hex2bin(key);

int keyp[56] = { 57, 49, 41, 33, 25, 17, 9,

1, 58, 50, 42, 34, 26, 18,

10, 2, 59, 51, 43, 35, 27,

19, 11, 3, 60, 52, 44, 36,

63, 55, 47, 39, 31, 23, 15,

7, 62, 54, 46, 38, 30, 22,

14, 6, 61, 53, 45, 37, 29,

21, 13, 5, 28, 20, 12, 4 };

key = permute(key, keyp, 56);

int shift\_table[16] = { 1, 1, 2, 2,

, 2, 2, 2,

1, 2, 2, 2,

2, 2, 2, 1 };

int key\_comp[48] = { 14, 17, 11, 24, 1, 5,

3, 28, 15, 6, 21, 10,

23, 19, 12, 4, 26, 8,

16, 7, 27, 20, 13, 2,

41, 52, 31, 37, 47, 55,

30, 40, 51, 45, 33, 48,

44, 49, 39, 56, 34, 53,

46, 42, 50, 36, 29, 32 };

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

string right = key.substr(28, 28);

vector<string> rkb;

vector<string> rk;

for (int i = 0; i < 16; i++) {

left = shift\_left(left, shift\_table[i]);

right = shift\_left(right, shift\_table[i]);

string combine = left + right;

string RoundKey = permute(combine, key\_comp, 48);

rkb.push\_back(RoundKey);

rk.push\_back(bin2hex(RoundKey));

}

cout << "\nEncryption:\n\n";

string cipher = encrypt(pt, rkb, rk);

cout << "\nCipher Text: " << cipher << endl;

cout << "\nDecryption\n\n";

reverse(rkb.begin(), rkb.end());

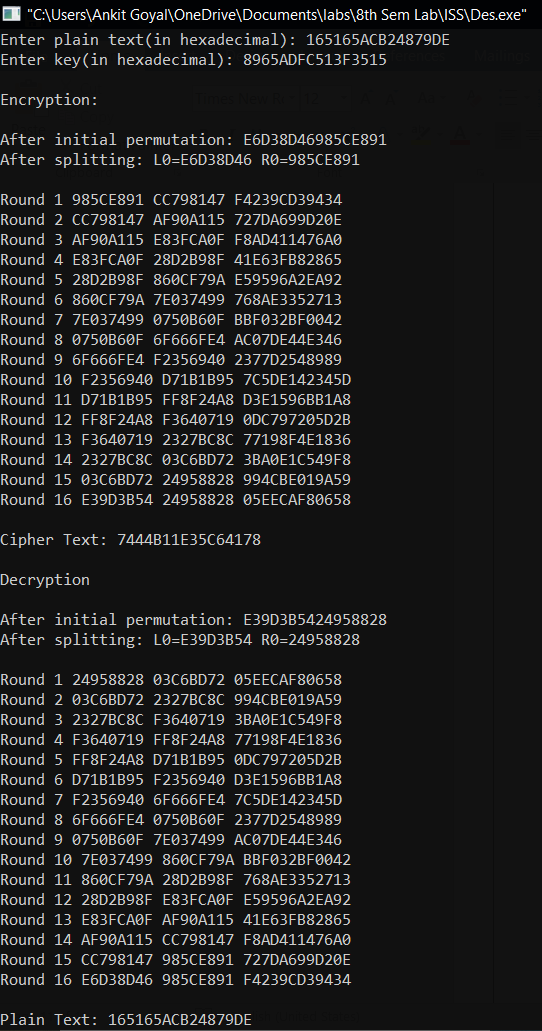
reverse(rk.begin(), rk.end());

string text = encrypt(cipher, rkb, rk);

cout << "\nPlain Text: " << text << endl;

}

**Result:**



**Experiment -6**

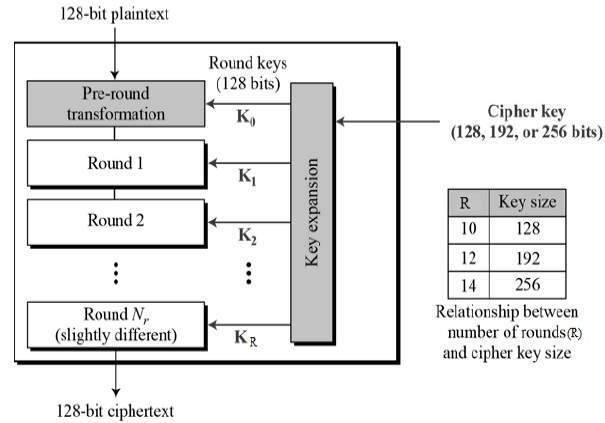
**Aim:** Implementation AES(Advanced Encryption Standard) Algorithm.

**Theory:**

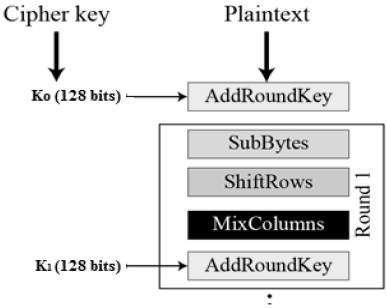
AES is an iterative rather than Feistel cipher. It is based on ‘substitution–permutation network’. It comprises of a series of linked operations, some of which involve replacing inputs by specific outputs (substitutions) and others involve shuffling bits around (permutations).

Interestingly, AES performs all its computations on bytes rather than bits. Hence, AES treats the 128 bits of a plaintext block as 16 bytes. These 16 bytes are arranged in four columns and four rows for processing as a matrix.

The schematic of AES structure is given in the following illustration –



Encryption :



***Sub Byte*** : 16 Input bytes are substituted using a lookup table.

***Shift Rows*** : Each of the four rows of the matrix is shifted to the left. First row is not shifted, second row is shifted by one position, third row by two position and fourth row by three positions.

***Mix columns*** : We multiply the resultant matrix to another given fixed matrix.

***Add Round Key*** : The 16 bytes of the matrix are now considered as 128 bits and are XORed to the 128 bits of the round key.

**Code :**

#include <bits/stdc++.h>

using namespace std;

typedef bitset<8> bytes;

typedef bitset<32> word;

const int Nr = 10;

const int Nk = 4;

bytes S\_Box[16][16] = {

{0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76},

{0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0},

{0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15},

{0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75},

{0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84},

{0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF},

{0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8},

{0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2},

{0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73},

{0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB},

{0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79},

{0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08},

{0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A},

{0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E},

{0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF},

{0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16}

};

bytes Inv\_S\_Box[16][16] = {

{0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E, 0x81, 0xF3, 0xD7, 0xFB},

{0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4, 0xDE, 0xE9, 0xCB},

{0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E},

{0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25},

{0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92},

{0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84},

{0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06},

{0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B},

{0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73},

{0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E},

{0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B},

{0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4},

{0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F},

{0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF},

{0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61},

{0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D}

};

word Rcon[10] = {0x01000000, 0x02000000, 0x04000000, 0x08000000, 0x10000000,

0x20000000, 0x40000000, 0x80000000, 0x1b000000, 0x36000000};

void SubBytes(bytes mtx[4\*4])

{

for(int i=0; i<16; ++i)

{

int row = mtx[i][7]\*8 + mtx[i][6]\*4 + mtx[i][5]\*2 + mtx[i][4];

int col = mtx[i][3]\*8 + mtx[i][2]\*4 + mtx[i][1]\*2 + mtx[i][0];

mtx[i] = S\_Box[row][col];

}

}

void ShiftRows(bytes mtx[4\*4])

{

//The second line circle moves one bit to the left

bytes temp = mtx[4];

for(int i=0; i<3; ++i)

mtx[i+4] = mtx[i+5];

mtx[7] = temp;

//The third line circle moves two places to the left

for(int i=0; i<2; ++i)

{

temp = mtx[i+8];

mtx[i+8] = mtx[i+10];

mtx[i+10] = temp;

}

//The fourth line moves three left circles

temp = mtx[15];

for(int i=3; i>0; --i)

mtx[i+12] = mtx[i+11];

mtx[12] = temp;

}

bytes GFMul(bytes a, bytes b) {

bytes p = 0;

bytes hi\_bit\_set;

for (int counter = 0; counter < 8; counter++) {

if ((b & bytes(1)) != 0) {

p ^= a;

}

hi\_bit\_set = (bytes) (a & bytes(0x80));

a <<= 1;

if (hi\_bit\_set != 0) {

a ^= 0x1b; /\* x^8 + x^4 + x^3 + x + 1 \*/

}

b >>= 1;

}

return p;

}

void MixColumns(bytes mtx[4\*4])

{

bytes arr[4];

for(int i=0; i<4; ++i)

{

for(int j=0; j<4; ++j)

arr[j] = mtx[i+j\*4];

mtx[i] = GFMul(0x02, arr[0]) ^ GFMul(0x03, arr[1]) ^ arr[2] ^ arr[3];

mtx[i+4] = arr[0] ^ GFMul(0x02, arr[1]) ^ GFMul(0x03, arr[2]) ^ arr[3];

mtx[i+8] = arr[0] ^ arr[1] ^ GFMul(0x02, arr[2]) ^ GFMul(0x03, arr[3]);

mtx[i+12] = GFMul(0x03, arr[0]) ^ arr[1] ^ arr[2] ^ GFMul(0x02, arr[3]);

}

}

void AddRoundKey(bytes mtx[4\*4], word k[4])

{

for(int i=0; i<4; ++i)

{

word k1 = k[i] >> 24;

word k2 = (k[i] << 8) >> 24;

word k3 = (k[i] << 16) >> 24;

word k4 = (k[i] << 24) >> 24;

mtx[i] = mtx[i] ^ bytes(k1.to\_ulong());

mtx[i+4] = mtx[i+4] ^ bytes(k2.to\_ulong());

mtx[i+8] = mtx[i+8] ^ bytes(k3.to\_ulong());

mtx[i+12] = mtx[i+12] ^ bytes(k4.to\_ulong());

}

}

void InvSubBytes(bytes mtx[4\*4])

{

for(int i=0; i<16; ++i)

{

int row = mtx[i][7]\*8 + mtx[i][6]\*4 + mtx[i][5]\*2 + mtx[i][4];

int col = mtx[i][3]\*8 + mtx[i][2]\*4 + mtx[i][1]\*2 + mtx[i][0];

mtx[i] = Inv\_S\_Box[row][col];

}

}

void InvShiftRows(bytes mtx[4\*4])

{

//The second line circle moves one bit to the right

bytes temp = mtx[7];

for(int i=3; i>0; --i)

mtx[i+4] = mtx[i+3];

mtx[4] = temp;

//The third line circle moves two to the right

for(int i=0; i<2; ++i)

{

temp = mtx[i+8];

mtx[i+8] = mtx[i+10];

mtx[i+10] = temp;

}

//Fourth line circle moves three to the right

temp = mtx[12];

for(int i=0; i<3; ++i)

mtx[i+12] = mtx[i+13];

mtx[15] = temp;

}

void InvMixColumns(bytes mtx[4\*4])

{

bytes arr[4];

for(int i=0; i<4; ++i)

{

for(int j=0; j<4; ++j)

arr[j] = mtx[i+j\*4];

mtx[i] = GFMul(0x0e, arr[0]) ^ GFMul(0x0b, arr[1]) ^ GFMul(0x0d, arr[2]) ^ GFMul(0x09, arr[3]);

mtx[i+4] = GFMul(0x09, arr[0]) ^ GFMul(0x0e, arr[1]) ^ GFMul(0x0b, arr[2]) ^ GFMul(0x0d, arr[3]);

mtx[i+8] = GFMul(0x0d, arr[0]) ^ GFMul(0x09, arr[1]) ^ GFMul(0x0e, arr[2]) ^ GFMul(0x0b, arr[3]);

mtx[i+12] = GFMul(0x0b, arr[0]) ^ GFMul(0x0d, arr[1]) ^ GFMul(0x09, arr[2]) ^ GFMul(0x0e, arr[3]);

}

}

word Word(bytes& k1, bytes& k2, bytes& k3, bytes& k4)

{

word result(0x00000000);

word temp;

temp = k1.to\_ulong(); // K1

temp <<= 24;

result |= temp;

temp = k2.to\_ulong(); // K2

temp <<= 16;

result |= temp;

temp = k3.to\_ulong(); // K3

temp <<= 8;

result |= temp;

temp = k4.to\_ulong(); // K4

result |= temp;

return result;

}

word RotWord(word& rw)

{

word high = rw << 8;

word low = rw >> 24;

return high | low;

}

word SubWord(word& sw)

{

word temp;

for(int i=0; i<32; i+=8)

{

int row = sw[i+7]\*8 + sw[i+6]\*4 + sw[i+5]\*2 + sw[i+4];

int col = sw[i+3]\*8 + sw[i+2]\*4 + sw[i+1]\*2 + sw[i];

bytes val = S\_Box[row][col];

for(int j=0; j<8; ++j)

temp[i+j] = val[j];

}

return temp;

}

void KeyExpansion(bytes key[4\*Nk], word w[4\*(Nr+1)])

{

word temp;

int i = 0;

while(i < Nk)

{

w[i] = Word(key[4\*i], key[4\*i+1], key[4\*i+2], key[4\*i+3]);

++i;

}

i = Nk;

while(i < 4\*(Nr+1))

{

temp = w[i-1];

if(i % Nk == 0) {

word r=RotWord(temp);

w[i] = w[i-Nk] ^ SubWord(r) ^ Rcon[i/Nk-1];

}

else

w[i] = w[i-Nk] ^ temp;

++i;

}

}

void encrypt(bytes in[4\*4], word w[4\*(Nr+1)])

{

word key[4];

for(int i=0; i<4; ++i)

key[i] = w[i];

AddRoundKey(in, key);

for(int round=1; round<Nr; ++round)

{

SubBytes(in);

ShiftRows(in);

MixColumns(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*round+i];

AddRoundKey(in, key);

}

SubBytes(in);

ShiftRows(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*Nr+i];

AddRoundKey(in, key);

}

void decrypt(bytes in[4\*4], word w[4\*(Nr+1)])

{

word key[4];

for(int i=0; i<4; ++i)

key[i] = w[4\*Nr+i];

AddRoundKey(in, key);

for(int round=Nr-1; round>0; --round)

{

InvShiftRows(in);

InvSubBytes(in);

for(int i=0; i<4; ++i)

key[i] = w[4\*round+i];

AddRoundKey(in, key);

InvMixColumns(in);

}

InvShiftRows(in);

InvSubBytes(in);

for(int i=0; i<4; ++i)

key[i] = w[i];

AddRoundKey(in, key);

}

int main()

{

bytes key[16] = {0x2b, 0x7e, 0x15, 0x16,

0x28, 0xae, 0xd2, 0xa6,

0xab, 0xf7, 0x15, 0x88,

0x09, 0xcf, 0x4f, 0x3c};

bytes plain[16] = {0x32, 0x88, 0x31, 0xe0,

0x43, 0x5a, 0x31, 0x37,

0xf6, 0x30, 0x98, 0x07,

0xa8, 0x8d, 0xa2, 0x34};

//Output key

cout << "The key is:";

for(int i=0; i<16; ++i)

cout << hex << key[i].to\_ulong() << " ";

cout << endl;

word w[4\*(Nr+1)];

KeyExpansion(key, w);

//Output plaintext to be encrypted

cout << endl << "Plaintext to be encrypted:"<<endl;

for(int i=0; i<16; ++i)

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

}

cout << endl;

//Encryption, output ciphertext

encrypt(plain, w);

cout << "Encrypted ciphertext:"<<endl;

for(int i=0; i<16; ++i)

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

}

cout << endl;

//Decrypt, output plaintext

decrypt(plain, w);

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

for(int i=0; i<16; ++i)

{

cout << hex << plain[i].to\_ulong() << " ";

if((i+1)%4 == 0)

cout << endl;

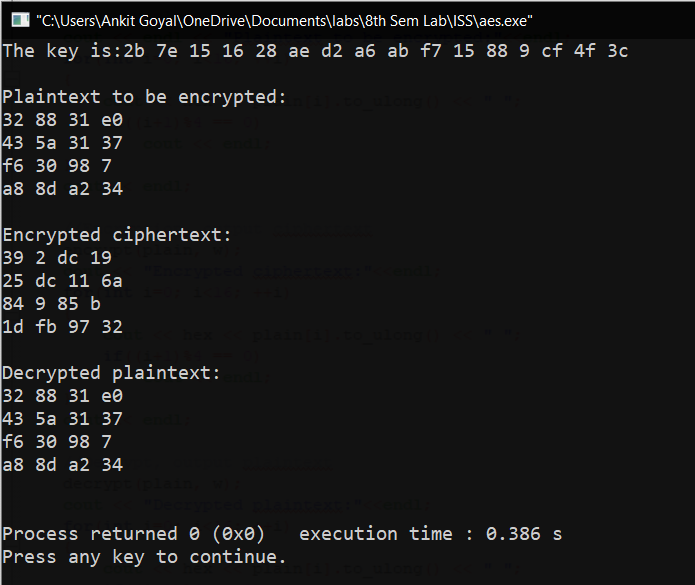
}

cout << endl;

return 0;

}

**Output:**



**Experiment -7**

**Aim**: Implement the following Modern Block Ciphers techniques.  
1) Electronic Codebook (ECB) Mode  
2) Cipher Block Chaining (CBC) Mode  
3) Cipher Feedback (CFB) Mode  
4) Output Feedback (OFB) Mode

5) Counter (CTR) Mode

1. **Electronic Codebook (ECB) Mode:**

**Code:**

#include<bits/stdc++.h>

using namespace std;

string generateKey(string key, int x)

{

for (int i = 0; ; i++)

{

if (x == i)

i = 0;

if (key.size() == x)

break;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key)

{

string cipher\_text;

for (int i = 0; i < str.size(); i++)

{

char x = (str[i] + key[i]) %26;

x += 'a';

cipher\_text.push\_back(x);

}

return cipher\_text;

}

int main()

{

int n;

cout<<"Enter the value of n(size of each block) : ";

cin>>n;

string plain, cipher="";

cout<<"Enter the plain text : ";

cin>>plain;

string key;

cout<<"Enter the key for vigenere cipher :";

cin>>key;

key = generateKey(key,n);

cout<<"key "<<key<<"\n";

int blocks;

if(plain.length()%n!=0)

{

int k= (plain.length()/n) \* n;

int g= plain.length()-k;

g=n-g;

for(int i=0; i<g; ++i)

plain.append("z");

}

blocks= plain.length()/n;

for(int i=0;i<blocks; ++i)

{ string tp= plain.substr(i\*n, i\*n+n);

string ci= cipherText(tp,key);

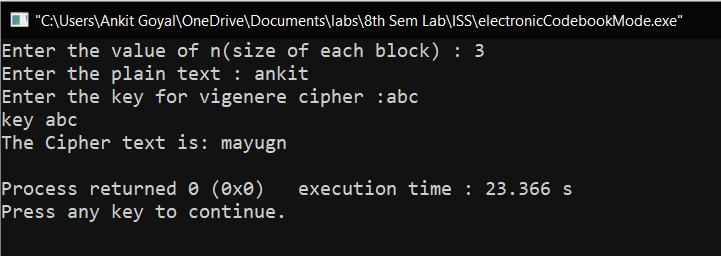
cipher.append(ci);

}

cout<<"The Cipher text is: "<<cipher<<"\n";

}

**Output:**



1. **Cipher Block Chaining (CBC) Mode**

**Code:**

#include<bits/stdc++.h>

using namespace std;

string xor\_operation(string a, string b)

{

string ans="";

int n=a.length();

for(int i=0; i<n; ++i)

{

char k= ((a[i]^b[i])%26 )+'a';

ans+=k;

}

return ans;

}

string generateKey(string key, int x)

{

for (int i = 0; ; i++)

{

if (x == i)

i = 0;

if (key.size() == x)

break;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key)

{

string cipher\_text;

for (int i = 0; i < str.size(); i++)

{

char x = (str[i] + key[i]) %26;

x += 'a';

cipher\_text.push\_back(x);

}

return cipher\_text;

}

int main()

{

int n;

cout<<"Enter the value of n(size of each block) : ";

cin>>n;

string plain, cipher="";

cout<<"Enter the plain text : ";

cin>>plain;

string key;

cout<<"Enter the key for vigenere cipher :";

cin>>key;

key = generateKey(key,n);

int blocks;

if(plain.length()%n!=0)

{

int k= (plain.length()/n) \* n;

int g= plain.length()-k;

g=n-g;

for(int i=0; i<g; ++i)

plain.append("z");

}

blocks= plain.length()/n;

string x;

for(int i=0;i<blocks; ++i)

{

string tp= plain.substr(i\*n, i\*n+n);

if(i!=0)

{

tp= xor\_operation(tp,x);

}

string ci= cipherText(tp,key);

x=ci;

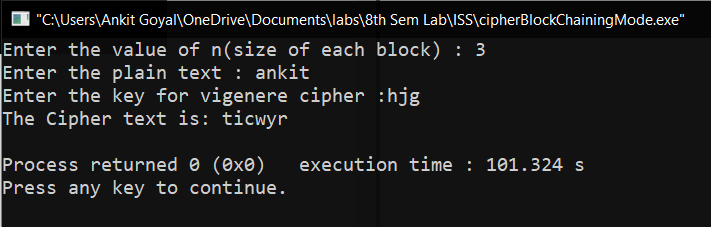
cipher.append(ci);

}

cout<<"The Cipher text is: "<<cipher<<"\n";

}

**Output:**



1. **Cipher Feedback (CFB) Mode**

**Code:**

#include<bits/stdc++.h>

using namespace std;

string xor\_operation(string a, string b)

{

string ans="";

int n=a.length();

for(int i=0; i<n; ++i)

{

char k= ((a[i]^b[i])%26 )+'a';

ans+=k;

}

return ans;

}

string generateKey(string key, int x)

{

for (int i = 0; ; i++)

{

if (x == i)

i = 0;

if (key.size() == x)

break;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key)

{

string cipher\_text;

for (int i = 0; i < str.size(); i++)

{

char x = (str[i] + key[i]) %26;

x += 'a';

cipher\_text.push\_back(x);

}

return cipher\_text;

}

int main()

{

int r;

cout<<"Enter the value of r(size of each block) : ";

cin>>r;

string plain, cipher="", S;

cout<<"Enter the plain text : ";

cin>>plain;

string key;

cout<<"Enter the key for vigenere cipher : ";

cin>>key;

cout<<"Enter the initial value of shift register : ";

cin>>S;

int n=S.length();

key = generateKey(key,n);

cout<<"\nKey : "<<key;

int blocks;

if(plain.length()%r!=0)

{

int k= (plain.length()/r) \* r;

int g= plain.length()-k;

g=r-g;

for(int i=0; i<g; ++i)

plain.append("z");

}

blocks= plain.length()/r;

cout<<"\nBlocks : "<<blocks;

for(int i=0;i<blocks; ++i)

{

string cip=cipherText(S,key);

cip= cip.substr(0,r);

string tp= plain.substr(i\*r, i\*r+r);

tp= xor\_operation(tp,cip);

S=S.substr(r, n);

S.append(tp);

cout<<"\nCipher : "<<tp<<" new S : "<<S<<"\n";

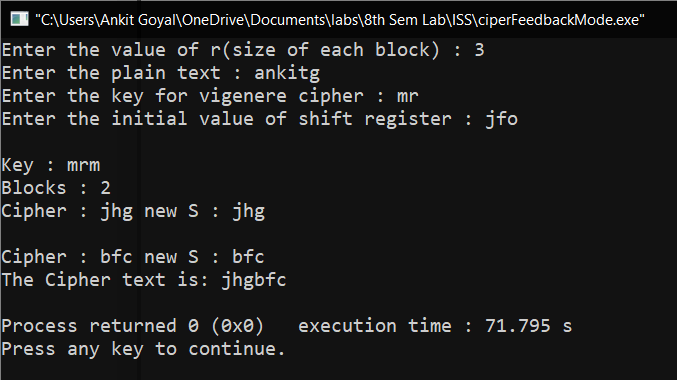
cipher.append(tp);

}

cout<<"The Cipher text is: "<<cipher<<"\n";

}

**Output:**



1. **Output Feedback (OFB) Mode**

**Code:**

#include<bits/stdc++.h>

using namespace std;

string xor\_operation(string a, string b)

{

string ans="";

int n=a.length();

for(int i=0; i<n; ++i)

{

char k= ((a[i]^b[i])%26 )+'a';

ans+=k;

}

return ans;

}

string generateKey(string key, int x)

{

for (int i = 0; ; i++)

{

if (x == i)

i = 0;

if (key.size() == x)

break;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key)

{

string cipher\_text;

for (int i = 0; i < str.size(); i++)

{

char x = (str[i] + key[i]) %26;

x += 'a';

cipher\_text.push\_back(x);

}

return cipher\_text;

}

int main()

{

int r;

cout<<"Enter the value of r(size of each block) : ";

cin>>r;

string plain, cipher="", S;

cout<<"Enter the plain text : ";

cin>>plain;

string key;

cout<<"Enter the key for vigenere cipher : ";

cin>>key;

cout<<"Enter the initial value of shift register : ";

cin>>S;

int n=S.length();

key = generateKey(key,n);

cout<<"\nKey : "<<key;

int blocks;

if(plain.length()%r!=0)

{

int k= (plain.length()/r) \* r;

int g= plain.length()-k;

g=r-g;

for(int i=0; i<g; ++i)

plain.append("z");

}

blocks= plain.length()/r;

cout<<"\nBlocks : "<<blocks;

for(int i=0;i<blocks; ++i)

{

string cip=cipherText(S,key);

cout<<"\nEncrypted Shift Register :"<<cip<<"\n";

cip= cip.substr(0,r);

string tp= plain.substr(i\*r, i\*r+r);

tp= xor\_operation(tp,cip);

S=S.substr(r, n);

S.append(cip);

cout<<"\nCipher : "<<tp<<" new S : "<<S<<"\n";

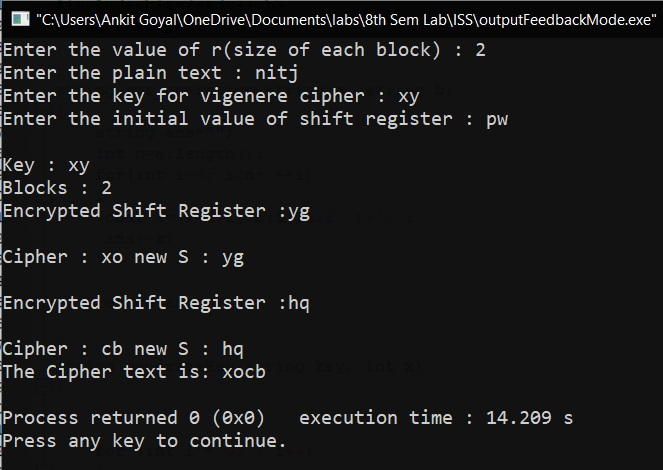
cipher.append(tp);

}

cout<<"The Cipher text is: "<<cipher<<"\n";

}

**Output:**



1. **Counter (CTR) Mode**

**Code:**

#include<bits/stdc++.h>

using namespace std;

string xor\_operation(string a, string b)

{

string ans="";

int n=a.length();

for(int i=0; i<n; ++i)

{

char k= ((a[i]^b[i])%26 )+'a';

ans+=k;

}

return ans;

}

string generateKey(string key, int x)

{

for (int i = 0; ; i++)

{

if (x == i)

i = 0;

if (key.size() == x)

break;

key.push\_back(key[i]);

}

return key;

}

string cipherText(string str, string key)

{

string cipher\_text;

for (int i = 0; i < str.size(); i++)

{

char x = (str[i] + key[i]) %26;

x += 'a';

cipher\_text.push\_back(x);

}

return cipher\_text;

}

int main()

{

int n;

cout<<"Enter the value of n(size of each block) : ";

cin>>n;

string plain, cipher="";

cout<<"Enter the plain text : ";

cin>>plain;

string key;

cout<<"Enter the key for vigenere cipher :";

cin>>key;

key = generateKey(key,n);

int blocks;

if(plain.length()%n!=0)

{

int k= (plain.length()/n) \* n;

int g= plain.length()-k;

g=n-g;

for(int i=0; i<g; ++i)

plain.append("z");

}

blocks= plain.length()/n;

string counter(n,'0');

int count=0;

for(int i=0;i<blocks; ++i)

{

string x = to\_string(count);

counter = counter.substr(0,n-x.length())+x;

cout<<counter<<"\n";

string tp= plain.substr(i\*n, i\*n+n);

string ci= cipherText(counter,key);

tp= xor\_operation(tp,ci);

cipher.append(tp);

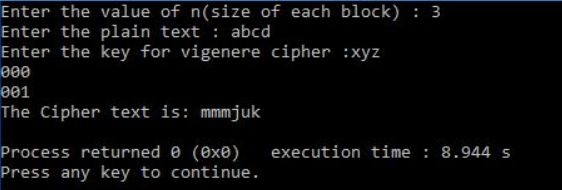
count++;

}

cout<<"The Cipher text is: "<<cipher<<"\n";

}

**Output:**



**Experiment -8**

**Aim:** Implementation Miller-Rabin Premality Test and Chinese Remainder Theorem.

**Theory:**

1. **Miller–Rabin primality test**

**Code:**

#include <bits/stdc++.h>

using namespace std;

int power(int x, unsigned int y, int p)

{

int res = 1;

x = x % p;

while (y > 0)

{

if (y & 1)

res = (res\*x) % p;

y = y>>1;

x = (x\*x) % p;

}

return res;

}

bool miillerTest(int d, int n)

{

int a = 2 + rand() % (n - 4);

int x = power(a, d, n);

if (x == 1 || x == n-1)

return true;

while (d != n-1)

{

x = (x \* x) % n;

d \*= 2;

if (x == 1) return false;

if (x == n-1) return true;

}

return false;

}

bool isPrime(int n, int k)

{

if (n <= 1 || n == 4) return false;

if (n <= 3) return true;

int d = n - 1;

while (d % 2 == 0)

d /= 2;

for (int i = 0; i < k; i++)

if (!miillerTest(d, n))

return false;

return true;

}

int main()

{

int k;

cout<<"Enter the number of iterations : ";

cin>>k;

int t=1;

while(t)

{

cout<<"\nEnter the number : ";

int n;

cin>>n;

if(isPrime(n,k))

cout<<"The number is prime\n";

else

cout<<"The number is not prime\n";

int x;

cout<<"press 1 for continue, 0 to exit: ";

cin>>x;

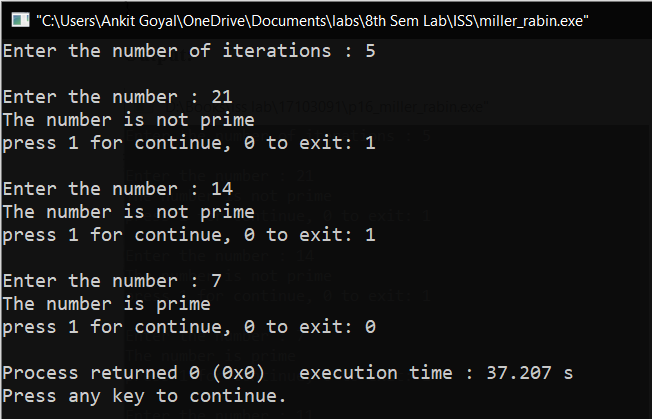
t=x;

}

return 0;

}

**Output:**



1. **Chinese Remainder Theorem**

**Code:**

#include <bits/stdc++.h>

using namespace std;

int inv(int a, int m)

{

int m0 = m, t, q;

int x0 = 0, x1 = 1;

if (m == 1)

return 0;

while (a > 1) {

q = a / m;

t = m;

m = a % m, a = t;

t = x0;

x0 = x1 - q \* x0;

x1 = t;

}

if (x1 < 0)

x1 += m0;

return x1;

}

int findMinX(int num[], int rem[], int k)

{

int prod = 1;

for (int i = 0; i < k; i++)

prod \*= num[i];

int result = 0;

for (int i = 0; i < k; i++) {

int pp = prod / num[i];

result += rem[i] \* inv(pp, num[i]) \* pp;

}

return result % prod;

}

int main(void)

{

int n;

cout<<"Enter the number of equations : ";

cin>>n;

cout<<"\nEnter the numbers and their remainders in each equation:\n ";

int num[n],rem[n];

for(int i=0;i<n; ++i)

{

cin>>num[i]>>rem[i];

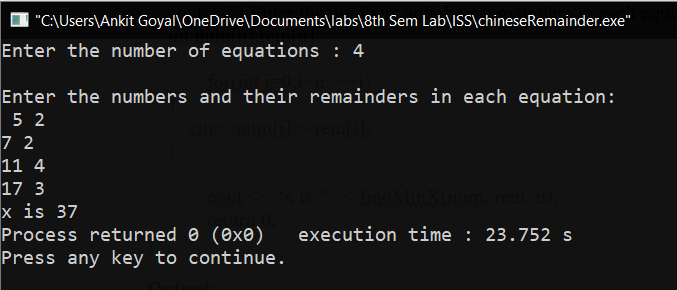
}

cout << "x is " << findMinX(num, rem, n);

return 0;

}

**Output:**



**Experiment -9**

**AIM:** Implement the RSA Algorithm.

**Theory:**

RSA is an asymmetric cryptography algorithm which works on two keys-public key and private key.

**Algorithm :**

Begin

1. Choose two prime numbers p and q.

2. Compute n = p\*q.

3. Calculate phi = (p-1) \* (q-1).

4. Choose an integer e such that 1 < e < phi(n) and gcd(e, phi(n)) = 1; i.e., e and phi(n) are coprime.

5. Calculate d as d ≡ e−1 (mod phi(n)); here, d is the modular multiplicative inverse of e modulo phi(n).

6. For encryption, c = me mod n, where m = original message.

7. For decryption, m = c d mod n.

End

**PROGRAM:**

#include<iostream>

#include<math.h>

using namespace std;

int gcd(int a, int b) {

int t;

while(1) {

t= a%b;

if(t==0)

return b;

a = b;

b= t;

}

}

int main() {

double p = 13;

double q = 11;

double n=p\*q;

double track;

double phi= (p-1)\*(q-1);

double e=7;

while(e<phi) {

track = gcd(e,phi);

if(track==1)

break;

else

e++;

}

double d1=1/e;

double d=fmod(d1,phi);

double message;

cout<<"Enter Message";

cin>>message;

double c = pow(message,e);

double m = pow(c,d);

c=fmod(c,n);

m=fmod(m,n);

cout<<"Original Message = "<<message;

cout<<"\n"<<"p = "<<p;

cout<<"\n"<<"q = "<<q;

cout<<"\n"<<"n = pq = "<<n;

cout<<"\n"<<"phi = "<<phi;

cout<<"\n"<<"e = "<<e;

cout<<"\n"<<"d = "<<d;

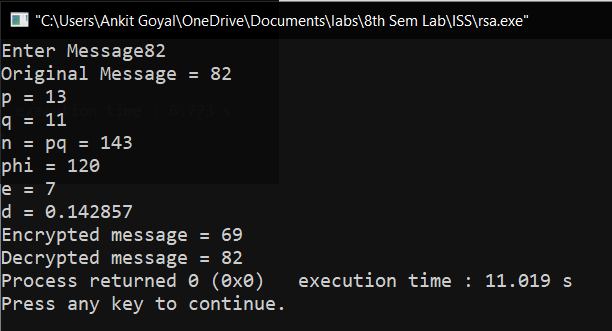
cout<<"\n"<<"Encrypted message = "<<c;

cout<<"\n"<<"Decrypted message = "<<m;

return 0;

}

**OUTPUT :**



**Experiment -10**

**Aim :** To Implement ElGamal cryptosystem.

**Theory:**

In cryptography, the ElGamal encryption system is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie–Hellman key exchange. It was described by Taher Elgamal in 1985. ElGamal encryption is used in the free GNU Privacy Guard software, recent versions of PGP, and other cryptosystems.

**Program:**

import random

from math import pow

a = random.randint(2, 10)

def gcd(a, b):

if a < b:

return gcd(b, a)

elif a % b == 0:

return b;

else:

return gcd(b, a % b)

# Generating large random numbers

def gen\_key(q):

key = random.randint(pow(10, 20), q)

while gcd(q, key) != 1:

key = random.randint(pow(10, 20), q)

return key

# Modular exponentiation

def power(a, b, c):

x = 1

y = a

while b > 0:

if b % 2 == 0:

x = (x \* y) % c;

y = (y \* y) % c

b = int(b / 2)

return x % c

# Asymmetric encryption

def encrypt(msg, q, h, g):

en\_msg = []

k = gen\_key(q)# Private key for sender

s = power(h, k, q)

p = power(g, k, q)

for i in range(0, len(msg)):

en\_msg.append(msg[i])

print("g^k used : ", p)

print("g^ak used : ", s)

for i in range(0, len(en\_msg)):

en\_msg[i] = s \* ord(en\_msg[i])

return en\_msg, p

def decrypt(en\_msg, p, key, q):

dr\_msg = []

h = power(p, key, q)

for i in range(0, len(en\_msg)):

dr\_msg.append(chr(int(en\_msg[i]/h)))

return dr\_msg

# Driver code

def main():

msg = input ("Enter the message to be encrypted: ");

print("Original Message :", msg)

q = random.randint(pow(10, 20), pow(10, 50))

g = random.randint(2, q)

key = gen\_key(q)# Private key for receiver

h = power(g, key, q)

print("g used : ", g)

print("g^a used : ", h)

en\_msg, p = encrypt(msg, q, h, g)

dr\_msg = decrypt(en\_msg, p, key, q)

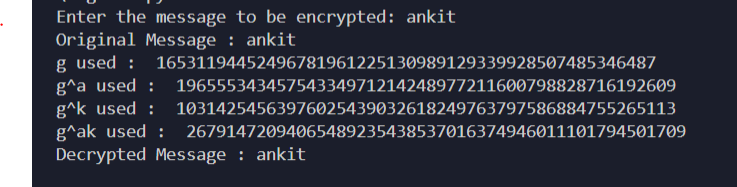
dmsg = ''.join(dr\_msg)

print("Decrypted Message :", dmsg);

# call the main function

main()

**Output:**



**Experiment -11**

**Aim :** To Implement Diffie-Hellman Key Exchange Algorithm.

**Theory:**

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

**Code:**

#include<bits/stdc++.h>

using namespace std;

long long int power(long long int x, long long int y, long long int p)

{

long long int res = 1;

x = x % p;

if (x == 0) return 0;

while (y > 0)

{

if (y & 1)

res = (res\*x) % p;

y = y>>1;

x = (x\*x) % p;

}

return res;

}

int main()

{

long long int P, G, x, a, y, b, ka, kb;

cout<<"enter a prime number: ";

cin>>P;

cout<<"enter primitive root of P: ";

cin>>G;

cout<<"The value of P : "<<P<<"\n";

cout<<"The value of G : "<<G<<"\n\n";

cout<<"enter first private key: ";

cin>>a;

cout<<"The private key a : "<<a<<"\n";

x = power(G, a, P);

cout<<"enter second private key: ";

cin>>b;

cout<<"The private key b : "<<b<<"\n\n";

y = power(G, b, P);

ka = power(y, a, P);

kb = power(x, b, P);

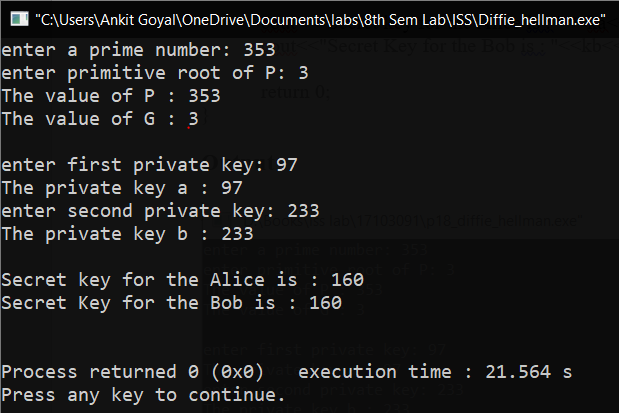
cout<<"Secret key for the Alice is : "<<ka<<"\n";

cout<<"Secret Key for the Bob is : "<<kb<<"\n\n";

return 0;

}

**Output**



**Experiment -12**

**Aim:** Implement MD5 Hash Algorithm.

**Theory:**

The MD5 message-digest algorithm is a widely used hash function producing a 128-bit hash value. Although MD5 was initially designed to be used as a cryptographic hash function, it has been found to suffer from extensive vulnerabilities.

**CODE**:

def newArray(num):

array=[]

for x in range(num):

array.append(0)

return array

def convertToWordArray(string):

lMessageLength=len(string)

lNumberOfWords\_temp1=lMessageLength+8

lNumberOfWords\_temp2=(lNumberOfWords\_temp1-(lNumberOfWords\_temp1%64))/64

lNumberOfWords=int((lNumberOfWords\_temp2+1)\*16)

lWordArray=newArray(lNumberOfWords-1)

lBytePosition=0

lByteCount=0

while lByteCount<lMessageLength:

lWordCount=int((lByteCount-(lByteCount%4))/4)

lBytePosition=(lByteCount%4)\*8

lWordArray[lWordCount]=(lWordArray[lWordCount]|(ord(string[int(lByteCount)])<<lBytePosition))

lByteCount+=1

lWordCount=int((lByteCount-(lByteCount%4))/4)

lBytePosition=(lByteCount%4)\*8

lWordArray[lWordCount]=lWordArray[lWordCount]|(0x80<<lBytePosition)

lWordArray[lNumberOfWords-2]=lMessageLength<<3

lWordArray.append(lMessageLength>>29)

return lWordArray

def F(x,y,z):

return (x & y) | ((~x) & z)

def G(x,y,z):

return (x & z) | (y & (~z))

def H(x,y,z):

return x ^ y ^ z

def I(x,y,z):

return y ^ (x | (~z))

def XX(func, a, b, c, d, x, s, ac):

res=0

res=res+a+func(b,c,d)

res+=x

res+=ac

res=res & 0xffffffff

res=rol(res,s)

res=res & 0xffffffff

res+=b

return res & 0xffffffff

def addu(x,y):

ls=(x & 0xffffffff)+(y & 0xffffffff)

return (((x>>16)+(y>>16)+(ls>>16))<<16)|(ls & 0xffffffff)

def rol(v,s):

return (v<<s)|(v>>(32-s))

def wordToHex(lValue):

wordToHexValue=''

wordToHexValue\_temp=''

for lCount in range(4):

lByte=(lValue>>(lCount\*8)) & 255

wordToHexValue\_temp="0"+format(lByte, 'x')

wordToHexValue=wordToHexValue+wordToHexValue\_temp[-2:]

return wordToHexValue

def md5hash(message):

x=convertToWordArray(message)

a=0x67452301

b=0xEFCDAB89

c=0x98BADCFE

d=0x10325476

xl=len(x)

j=0

while j<xl:

aa=a

bb=b

cc=c

dd=d

a=XX(F,a,b,c,d, x[j+0], 7,0xD76AA478)

d=XX(F,d,a,b,c, x[j+1],12,0xE8C7B756)

c=XX(F,c,d,a,b, x[j+2],17,0x242070DB)

b=XX(F,b,c,d,a, x[j+3],22,0xC1BDCEEE)

a=XX(F,a,b,c,d, x[j+4], 7,0xF57C0FAF)

d=XX(F,d,a,b,c, x[j+5],12,0x4787C62A)

c=XX(F,c,d,a,b, x[j+6],17,0xA8304613)

b=XX(F,b,c,d,a, x[j+7],22,0xFD469501)

a=XX(F,a,b,c,d, x[j+8], 7,0x698098D8)

d=XX(F,d,a,b,c, x[j+9],12,0x8B44F7AF)

c=XX(F,c,d,a,b,x[j+10],17,0xFFFF5BB1)

b=XX(F,b,c,d,a,x[j+11],22,0x895CD7BE)

a=XX(F,a,b,c,d,x[j+12], 7,0x6B901122)

d=XX(F,d,a,b,c,x[j+13],12,0xFD987193)

c=XX(F,c,d,a,b,x[j+14],17,0xA679438E)

b=XX(F,b,c,d,a,x[j+15],22,0x49B40821)

a=XX(G,a,b,c,d, x[j+1], 5,0xF61E2562)

d=XX(G,d,a,b,c, x[j+6], 9,0xC040B340)

c=XX(G,c,d,a,b,x[j+11],14,0x265E5A51)

b=XX(G,b,c,d,a, x[j+0],20,0xE9B6C7AA)

a=XX(G,a,b,c,d, x[j+5], 5,0xD62F105D)

d=XX(G,d,a,b,c,x[j+10], 9,0x2441453)

c=XX(G,c,d,a,b,x[j+15],14,0xD8A1E681)

b=XX(G,b,c,d,a, x[j+4],20,0xE7D3FBC8)

a=XX(G,a,b,c,d, x[j+9], 5,0x21E1CDE6)

d=XX(G,d,a,b,c,x[j+14], 9,0xC33707D6)

c=XX(G,c,d,a,b, x[j+3],14,0xF4D50D87)

b=XX(G,b,c,d,a, x[j+8],20,0x455A14ED)

a=XX(G,a,b,c,d,x[j+13], 5,0xA9E3E905)

d=XX(G,d,a,b,c, x[j+2], 9,0xFCEFA3F8)

c=XX(G,c,d,a,b, x[j+7],14,0x676F02D9)

b=XX(G,b,c,d,a,x[j+12],20,0x8D2A4C8A)

a=XX(H,a,b,c,d, x[j+5], 4,0xFFFA3942)

d=XX(H,d,a,b,c, x[j+8],11,0x8771F681)

c=XX(H,c,d,a,b,x[j+11],16,0x6D9D6122)

b=XX(H,b,c,d,a,x[j+14],23,0xFDE5380C)

a=XX(H,a,b,c,d, x[j+1], 4,0xA4BEEA44)

d=XX(H,d,a,b,c, x[j+4],11,0x4BDECFA9)

c=XX(H,c,d,a,b, x[j+7],16,0xF6BB4B60)

b=XX(H,b,c,d,a,x[j+10],23,0xBEBFBC70)

a=XX(H,a,b,c,d,x[j+13], 4,0x289B7EC6)

d=XX(H,d,a,b,c, x[j+0],11,0xEAA127FA)

c=XX(H,c,d,a,b, x[j+3],16,0xD4EF3085)

b=XX(H,b,c,d,a, x[j+6],23,0x4881D05)

a=XX(H,a,b,c,d, x[j+9], 4,0xD9D4D039)

d=XX(H,d,a,b,c,x[j+12],11,0xE6DB99E5)

c=XX(H,c,d,a,b,x[j+15],16,0x1FA27CF8)

b=XX(H,b,c,d,a, x[j+2],23,0xC4AC5665)

a=XX(I,a,b,c,d, x[j+0], 6,0xF4292244)

d=XX(I,d,a,b,c, x[j+7],10,0x432AFF97)

c=XX(I,c,d,a,b,x[j+14],15,0xAB9423A7)

b=XX(I,b,c,d,a, x[j+5],21,0xFC93A039)

a=XX(I,a,b,c,d,x[j+12], 6,0x655B59C3)

d=XX(I,d,a,b,c, x[j+3],10,0x8F0CCC92)

c=XX(I,c,d,a,b,x[j+10],15,0xFFEFF47D)

b=XX(I,b,c,d,a, x[j+1],21,0x85845DD1)

a=XX(I,a,b,c,d, x[j+8], 6,0x6FA87E4F)

d=XX(I,d,a,b,c,x[j+15],10,0xFE2CE6E0)

c=XX(I,c,d,a,b, x[j+6],15,0xA3014314)

b=XX(I,b,c,d,a,x[j+13],21,0x4E0811A1)

a=XX(I,a,b,c,d, x[j+4], 6,0xF7537E82)

d=XX(I,d,a,b,c,x[j+11],10,0xBD3AF235)

c=XX(I,c,d,a,b, x[j+2],15,0x2AD7D2BB)

b=XX(I,b,c,d,a, x[j+9],21,0xEB86D391)

a=addu(a,aa)

b=addu(b,bb)

c=addu(c,cc)

d=addu(d,dd)

j+=16

return (wordToHex(a)+wordToHex(b)+wordToHex(c)+wordToHex(d)).lower()

message = input ("Enter the message to hash: ")

print (md5hash (message))

**OUTPUT:**

