**Cryptography & Network Security Lab**

**Assignments**

Name- Aniket Tatoba Mane

PRN-2020BTECS00020

Batch – B7

**ASSIGNMENT NO: 1**

Title: Encryption and Decryption using Caesar cipher.

Aim: To Study and Implement Encryption and Decryption using Caesar cipher

Technique.

Caesar Cipher:

Caesar Cipher, also known as the Shift Cipher, is one of the simplest and oldest

encryption techniques used to secure information.

It is a type of substitution cipher where each letter in the plaintext is shifted a

certain number of places down or up the alphabet.

The number of positions a letter is shifted is determined by a key.

Encryption:

In Encryption, input is a Plain text and output is a Cipher text.

Step 1: Choose a secret key (a positive integer).

Step 2: Take the plaintext message you want to encrypt.

Step 3: Shift each letter in the message forward in the alphabet by the key

positions.

Step 4: Non-alphabetical characters remain unchanged.

Step 5: The result is the ciphertext, the encrypted message.

Decryption:

In Decryption, input is a Cipher text and output is a Plain text.

Step 1: Have the same key used for encryption.

Step 2: Take the ciphertext (the encrypted message).

Step 3: Shift each letter in the ciphertext backward in the alphabet by the key

positions.

Step 4: Non-alphabetical characters remain unchanged.

Step 5: The result is the plaintext, the original message.

Code:

#include <bits/stdc++.h>

using namespace std;

char shift\_char(char c, int shift, char op)

{

    if (isalpha(c) && op == 'e')

    {

        char base = islower(c) ? 'a' : 'A';

        return char((c - base + shift) % 26 + base);

    }

    else if (isalpha(c) && op == 'd')

    {

        char base = islower(c) ? 'a' : 'A';

        return char((c - base - shift + 26) % 26 + base);

    }

    return c;

}

string encrypt\_text(string text, int key)

{

    string encrypted = "";

    for (char c : text)

    {

        encrypted += shift\_char(c, key, 'e');

    }

    return encrypted;

}

string decrypt\_text(string text, int key)

{

    string decrypted = "";

    for (char c : text)

    {

        decrypted += shift\_char(c, key, 'd');

    }

    return decrypted;

}

int main(int argc, char const \*argv[])

{

    int choice, key;

    string text;

    cout << "Enter choice: ";

    cout << endl

         << "1. Encrypt | 2. Decrypt" << endl;

    cin >> choice;

    cin.get();

    if (choice == 1)

    {

        cout << "enter text: ";

        getline(cin, text);

        cout << "Enter key: ";

        cin >> key;

        string result = encrypt\_text(text, key);

        cout << "encrypted text: " << result << endl;

    }

    else if (choice == 2)

    {

        cout << "enter encrypted text: ";

        getline(cin, text);

        cout << "Enter key: ";

        cin >> key;

        string result = decrypt\_text(text, key);

        cout << "decrypted text: " << result << endl;

    }

    else

    {

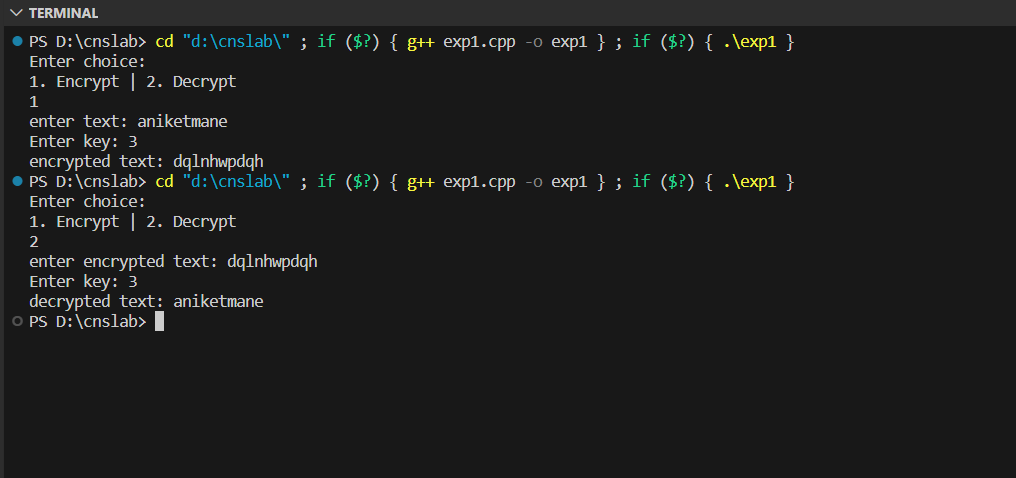
        cout << "not a valid choice!" << endl;

    }

    return 0;

}

Output:



**ASSIGNMENT NO: 2**

Title: Encryption and Decryption using Transposition cipher.

Aim: To Study and Implement Encryption and Decryption using Transposition

cipher technique.

Transposition Cipher:

**1. Railfence Transposition**

The Rail Fence Transposition Cipher, also known as the Zigzag Cipher, is a

simple columnar transposition cipher technique.

It involves arranging the plaintext characters in a zigzag pattern across

multiple rows, known as "rails," and then reading them off row by row to

create the encrypted message.

While this cipher is easy to understand and implement, it lacks strong security

and is mainly used for educational purposes or simple puzzles.

Encryption:

- Choose the number of rails (rows) for the zigzag pattern.

- Write the message diagonally across the rails, moving up and down.

- Read the characters row by row to form the encrypted message.

Decryption:

- Create the zigzag pattern with the chosen number of rails.

- Leave blank spaces in the pattern for characters to be placed.

- Fill in the blanks with the encrypted characters, row by row.

- Read the characters diagonally to retrieve the original message.

Advantages:

- Easy to understand and implement.

- Provides basic encryption and breaks up character repetition.

Disadvantages:

- Not secure against modern cryptanalysis.

- Security depends on the number of rails, making it less practical for strong

encryption.

Code:

#include <bits/stdc++.h>

using namespace std;

string format(string &str)

{

    for (char c : str)

    {

        if (isalpha(c))

        {

            c += tolower(c);

        }

    }

    return str;

}

string encrypt(string &plain, int key)

{

    vector<vector<char>> matrix(key);

    int rowNumber = 0;

    int flag = 1;

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

    {

        matrix[rowNumber].push\_back(plain[i]);

        rowNumber += flag;

        if (rowNumber == 0)

            flag = 1;

        if (rowNumber == key - 1)

            flag = -1;

    }

    string cipher;

    for (int i = 0; i < key; i++)

    {

        for (int j = 0; j < matrix[i].size(); j++)

            cipher += matrix[i][j];

    }

    return cipher;

}

string decrypt(string &cipher, int key)

{

    vector<vector<int>> matrixDecry(key);

    int rowNumber = 0;

    int flag = 1;

    int n = cipher.length();

    for (int i = 0; i < n; i++)

    {

        matrixDecry[rowNumber].push\_back(i);

        rowNumber += flag;

        if (rowNumber == (key - 1))

            flag = -1;

        if (rowNumber == 0)

            flag = 1;

    }

    vector<int> mapping;

    for (int i = 0; i < key; i++)

    {

        for (int j = 0; j < matrixDecry[i].size(); j++)

            mapping.push\_back(matrixDecry[i][j]);

    }

    map<int, char> m;

    for (int i = 0; i < n; i++)

        m[mapping[i]] = cipher[i];

    string plain;

    for (int i = 0; i < n; i++)

        plain += m[i];

    return plain;

}

int main()

{

    int choice;

    cout << "1. Encrypt\n2. Decrypt\nEnter your choice: ";

    cin >> choice;

    cin.get();

    if (choice == 1)

    {

        string plain;

        int key;

        cout << "\nEnter plain text: ";

        getline(cin, plain);

        plain = format(plain);

        cout << "\nEnter key: integer value(depth): ";

        cin >> key;

        string cipher = encrypt(plain, key);

        cout << "\nEncrypted text is : " << cipher << endl;

    }

    else if (choice == 2)

    {

        string cipher;

        int key;

        cout << "\nEnter cipher text: ";

        getline(cin, cipher);

        cipher = format(cipher);

        cout << "\nEnter key: integer value(depth): ";

        cin >> key;

        string plain = decrypt(cipher, key);

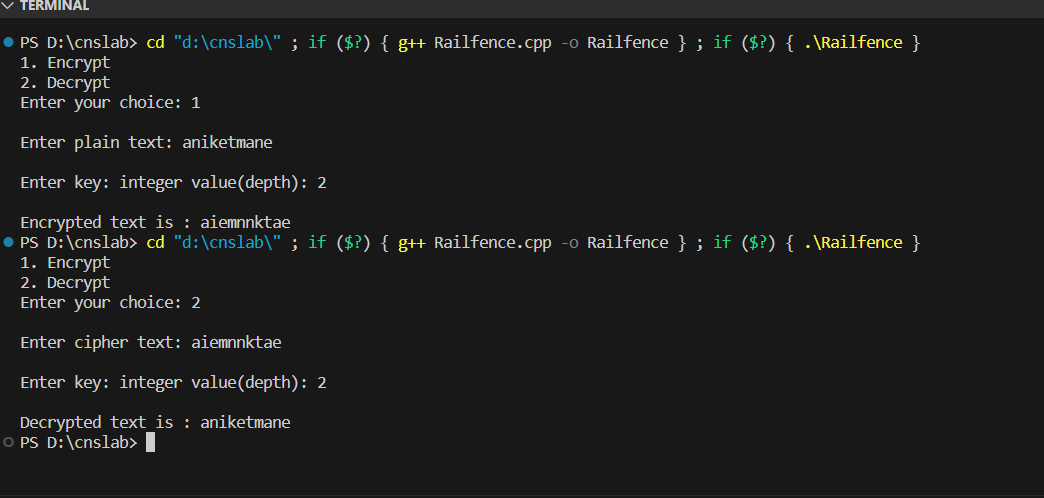
        cout << "\nDecrypted text is : " << plain << endl;

    }

    return 0;

}

Output:



2. Row/Columnar Transposition

The Columnar Transposition Cipher is a more advanced transposition cipher

technique that involves reordering the characters of a message based on a

chosen keyword or key phrase.

It provides a higher level of security compared to simpler ciphers like the Rail

Fence Cipher. Here is how the Columnar Transposition Cipher works:

Encryption:

- Choose a keyword or key phrase. The unique characters of the keyword

determine the order of columns in the transposition grid.

- Write the message row by row into a grid, using the keyword to determine

the order of columns.

- Read the characters column by column to obtain the encrypted message.

Decryption:

- Use the keyword to determine the order of columns in the transposition

grid.

- Write the encrypted message into the grid column by column.

- Read the characters row by row to retrieve the original plaintext.

Advantages:

- Offers stronger security compared to simpler ciphers.

- Security depends on the length and uniqueness of the keyword.

Disadvantages:

- Can be vulnerable to attacks if the keyword is short or easily guessed.

- May require additional padding characters for messages that don't fit

evenly into the grid.

Code:

#include <bits/stdc++.h>

using namespace std;

string alpha\_lower(string text)

{

    for (char c : text)

    {

        if (isalnum(c))

        {

            c = tolower(c);

        }

    }

    return text;

}

string encrypt(string text, string key)

{

    map<char, vector<char>> mp;

    int cnt = 0;

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

    {

        if (cnt == key.size())

            cnt = 0;

        mp[key[cnt++]].push\_back(text[i]);

    }

    string encrypted;

    for (auto i : mp)

    {

        for (auto j : i.second)

        {

            encrypted += j;

        }

    }

    return encrypted;

}

string decrypt(string cipher, string key)

{

    map<int, int> map1;

    int common = cipher.size() / key.size();

    int extra = cipher.size() % key.size();

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

    {

        if (i < extra)

            map1[i] = common + 1;

        else

            map1[i] = common;

    }

    map<int, vector<char>> map2;

    int start = 0;

    string sortedKey = key;

    sort(sortedKey.begin(), sortedKey.end());

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

    {

        for (int j = 0; j < key.size(); j++)

        {

            if (sortedKey[i] == key[j])

            {

                for (int k = 0; k < map1[j]; k++)

                {

                    map2[key[j]].push\_back(cipher[start++]);

                }

            }

        }

    }

    string plain;

    vector<int> counters(key.size(), 0);

    while (plain.size() < cipher.size())

    {

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

        {

            if (counters[i] < map1[i])

                plain += map2[key[i]][counters[i]++];

        }

    }

    return plain;

}

int main()

{

    int choice;

    cout << "Enter choice: ";

    cout << endl

         << "1. Encrypt | 2. Decrypt" << endl;

    cin >> choice;

    cin.get();

    if (choice == 1)

    {

        string text, key;

        cout << "\nEnter text: ";

        getline(cin, text);

        text = alpha\_lower(text);

        cout << "\nEnter key: ";

        getline(cin, key);

        alpha\_lower(key);

        string cipher = encrypt(text, key);

        cout << "\nEncrypted text is : " << cipher << endl;

    }

    else if (choice == 2)

    {

        string cipher, key;

        cout << "\nEnter cipher text: ";

        getline(cin, cipher);

        cipher = alpha\_lower(cipher);

        cout << "\nEnter key: ";

        getline(cin, key);

        alpha\_lower(key);

        string text = decrypt(cipher, key);

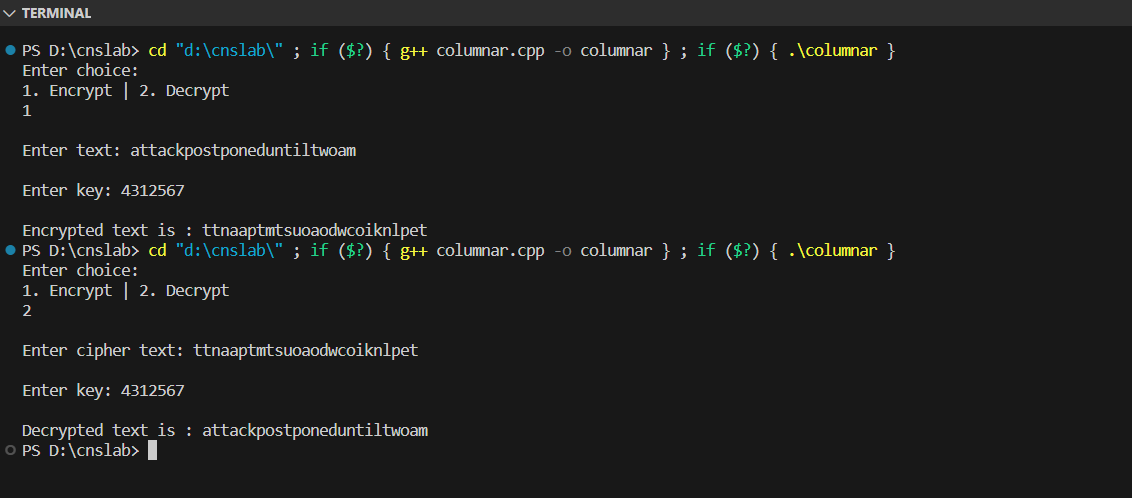
        cout << "\nDecrypted text is : " << text << endl;

    }

    return 0;

}

Output:



Conclusion: Studied and implemented encryption and decryption of

Transposition cipher.

**ASSIGNMENT NO: 3**

Title: Encryption and Decryption using Playfair cipher.

Aim: To Study and Implement Encryption and Decryption using Playfair cipher

technique.

**Playfair Cipher:**

The Playfair Cipher Technique is a substitution cipher that encrypts pairs of

characters (digraphs) from the plaintext using a 5x5 key square matrix.

The matrix is constructed from a keyword, with duplicate letters removed and the

keyword letters placed at the beginning.

Encryption involves applying rules based on the positions of the letters within the

key square.

If the letters are in the same row, column, or form a rectangle, they are replaced

by specific neighbouring letters.

Encryption:

Divide the plaintext into digraphs.

Apply rules based on the positions of letters in the key square to replace each

digraph.

Decryption:

Divide the ciphertext into digraphs.

Apply the rules in reverse to each digraph to retrieve the original plaintext.

Advantages:

Enhanced security due to digraphs and key square usage.

Reduces susceptibility to frequency analysis.

Key square generation is straightforward using a keyword.

Disadvantages:

Complexity increases with handling various cases (same row, column, rectangle).

Security depends on the keyword and arrangement of the key square.

Code:

#include <iostream>

#include <string>

using namespace std;

const int SIZE = 5;

void generateMatrix(string key, char matrix[SIZE][SIZE])

{

    string keyWithoutDuplicates = "";

    bool used[26] = {false};

    for (char c : key)

    {

        if (c != 'J' && !used[c - 'A'])

        {

            keyWithoutDuplicates += c;

            used[c - 'A'] = true;

        }

    }

    int index = 0;

    for (int i = 0; i < SIZE; i++)

    {

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

        {

            if (index < keyWithoutDuplicates.length())

            {

                matrix[i][j] = keyWithoutDuplicates[index++];

            }

            else

            {

                for (char c = 'A'; c <= 'Z'; c++)

                {

                    if (c != 'J' && !used[c - 'A'])

                    {

                        matrix[i][j] = c;

                        used[c - 'A'] = true;

                        break;

                    }

                }

            }

        }

    }

}

void findPosition(char matrix[SIZE][SIZE], char c, int &row, int &col)

{

    if (c == 'J')

        c = 'I';

    for (int i = 0; i < SIZE; i++)

    {

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

        {

            if (matrix[i][j] == c)

            {

                row = i;

                col = j;

                return;

            }

        }

    }

}

string encryptDigraph(char matrix[SIZE][SIZE], char a, char b)

{

    int rowA, colA, rowB, colB;

    findPosition(matrix, a, rowA, colA);

    findPosition(matrix, b, rowB, colB);

    if (rowA == rowB)

    {

        return string(1, matrix[rowA][(colA + 1) % SIZE]) + string(1,

                                                                   matrix[rowB][(colB + 1) % SIZE]);

    }

    if (colA == colB)

    {

        return string(1, matrix[(rowA + 1) % SIZE][colA]) + string(1,

                                                                   matrix[(rowB + 1) % SIZE][colB]);

    }

    return string(1, matrix[rowA][colB]) + string(1, matrix[rowB][colA]);

}

string decryptDigraph(char matrix[SIZE][SIZE], char a, char b)

{

    int rowA, colA, rowB, colB;

    findPosition(matrix, a, rowA, colA);

    findPosition(matrix, b, rowB, colB);

    if (rowA == rowB)

    {

        return string(1, matrix[rowA][(colA - 1 + SIZE) % SIZE]) + string(1,

                                                                          matrix[rowB][(colB - 1 + SIZE) % SIZE]);

    }

    if (colA == colB)

    {

        return string(1, matrix[(rowA - 1 + SIZE) % SIZE][colA]) + string(1,

                                                                          matrix[(rowB - 1 + SIZE) % SIZE][colB]);

    }

    return string(1, matrix[rowA][colB]) + string(1, matrix[rowB][colA]);

}

int main()

{

    string key, text;

    char matrix[SIZE][SIZE];

    cout << "Enter the key (uppercase, excluding J): ";

    cin >> key;

    generateMatrix(key, matrix);

    cout << "Enter the text (uppercase, without spaces): ";

    cin >> text;

    string result;

    char choice;

    cout << "Encrypt (E) or Decrypt (D)? ";

    cin >> choice;

    if (choice == 'E' || choice == 'e')

    {

        string preparedText = "";

        for (int i = 0; i < text.length(); i += 2)

        {

            if (i + 1 < text.length())

            {

                if (text[i] == text[i + 1])

                {

                    preparedText += text[i];

                    preparedText += 'X';

                    i--;

                }

                else

                {

                    preparedText += text.substr(i, 2);

                }

            }

            else

            {

                preparedText += text[i];

                preparedText += 'X';

            }

        }

        for (int i = 0; i < preparedText.length(); i += 2)

        {

            char a = preparedText[i];

            char b = preparedText[i + 1];

            result += encryptDigraph(matrix, a, b);

        }

    }

    else if (choice == 'D' || choice == 'd')

    {

        for (int i = 0; i < text.length(); i += 2)

        {

            char a = text[i];

            char b = text[i + 1];

            result += decryptDigraph(matrix, a, b);

        }

        string cleanedText = "";

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

        {

            if (result[i] != 'X')

            {

                cleanedText += result[i];

            }

        }

        result = cleanedText;

    }

    else

    {

        cout << "Invalid choice. Please enter 'E' for Encrypt or 'D' for Decrypt" << endl;

        return 1;

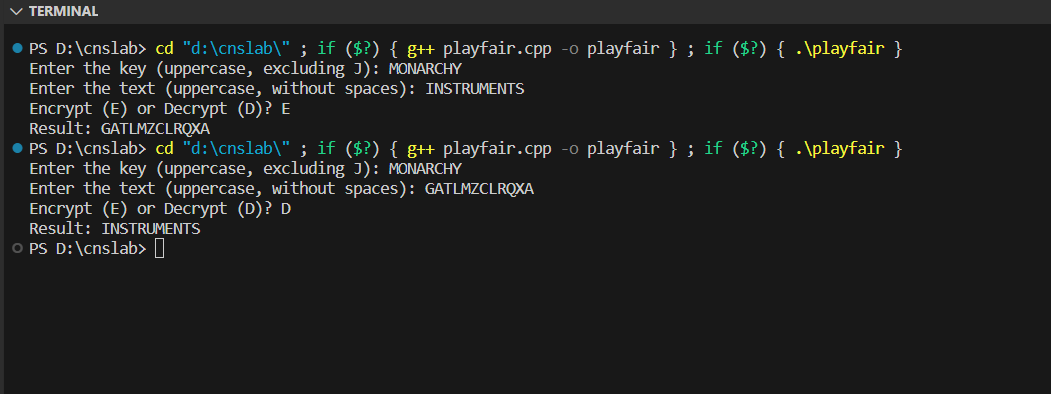
    }

    cout << "Result: " << result << endl;

    return 0;

}

Output:



Conclusion: Studied and implemented encryption and decryption of Playfair

cipher.

ASIGNMENT NO: 4

Title: Encryption and Decryption using Vigenere cipher.

Aim: To Study and Implement Encryption and Decryption using Vigenere cipher

technique.

Vigenere Cipher:

The Vigenere Cipher Technique is a polyalphabetic substitution cipher that adds

an extra layer of complexity to encryption by using a keyword or key phrase to

determine the shifts applied to the plaintext letters.

Unlike monoalphabetic ciphers, where each letter is replaced with a fixed

substitution, the Vigenere Cipher employs multiple alphabets with different

shifts, making it more secure against frequency analysis.

Key Setup:

Choose a keyword or key phrase.

Replicate the keyword to match the length of the plaintext, repeating it as needed.

Convert the keyword letters to their corresponding numerical values (A=0, B=1,

..., Z=25).

Encryption:

Divide the plaintext into individual letters and convert them to numerical values.

For each letter, determine the shift value using the corresponding keyword letter.

Shift the plaintext letter by the calculated shift value (mod 26).

Convert the shifted numerical value back to a letter to create the ciphertext.

Decryption:

Divide the ciphertext into individual letters and convert them to numerical values.

For each letter, determine the shift value using the corresponding keyword letter.

Reverse the shift (subtract the shift value, mod 26).

Convert the shifted numerical value back to a letter to retrieve the original

plaintext.

Advantages:

Stronger security due to polyalphabetic nature and keyword-driven shifts.

Reduces susceptibility to frequency analysis.

Key space increases with keyword length, enhancing security.

Disadvantages:

Vulnerable to key length repetition for shorter keywords.

Security can weaken if the keyword is short or predictable.

Code:

#include <bits/stdc++.h>

using namespace std;

string alpah\_lower(string str)

{

    for (char c : str)

    {

        if (isalpha(c))

        {

            c += tolower(c);

        }

    }

    return str;

}

string encrypt(string text, string key)

{

    string cipher;

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

    {

        int val = text[i] - 'a' + key[i % (key.size())] - 'a';

        cipher += ('a' + (val % 26));

    }

    return cipher;

}

string decrypt(string cipher, string key)

{

    string text;

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

    {

        int val = cipher[i] - 'a' - (key[i % (key.size())] - 'a');

        text += ('a' + (val + 26) % 26);

    }

    return text;

}

int main()

{

    int choice;

    cout << "1. Encrypt\n2. Decrypt\nEnter your choice: ";

    cin >> choice;

    cin.get();

    if (choice == 1)

    {

        string plain, key;

        cout << "\nEnter plain text: ";

        getline(cin, plain);

        plain = alpah\_lower(plain);

        cout << "\nEnter key: ";

        getline(cin, key);

        string cipher = encrypt(plain, key);

        cout << "\nEncrypted text is : " << cipher << endl;

    }

    else if (choice == 2)

    {

        string cipher, key;

        cout << "\nEnter cipher text: ";

        getline(cin, cipher);

        cipher = alpah\_lower(cipher);

        cout << "\nEnter key: ";

        getline(cin, key);

        string plain = decrypt(cipher, key);

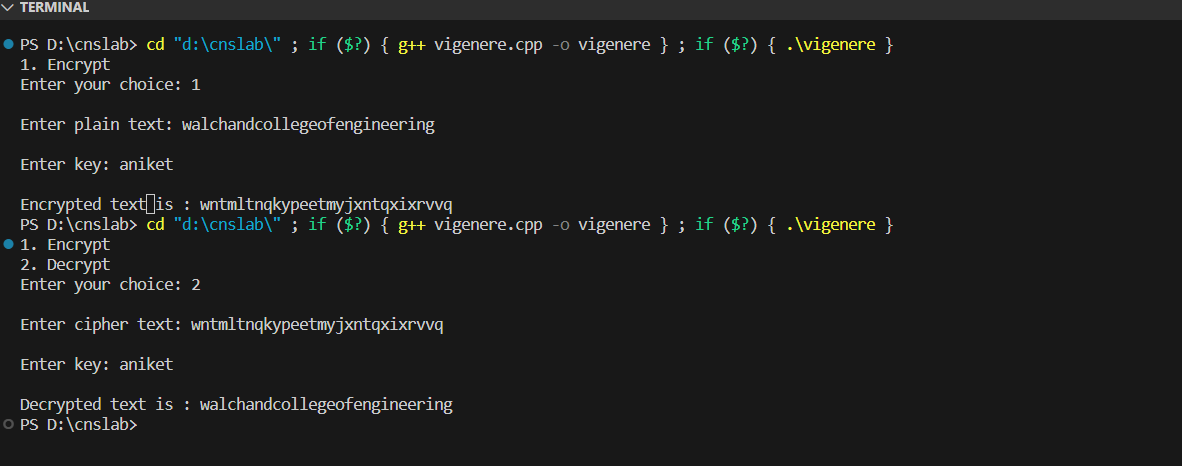
        cout << "\nDecrypted text is : " << plain << endl;

    }

    return 0;

}

Output:



Conclusion: Studied and implemented encryption and decryption of Vigenere

cipher.

DES Algorithm:

The Data Encryption Standard (DES) is a symmetric-key block cipher algorithm

that was widely used for secure data encryption before being replaced by more

secure algorithms like AES (Advanced Encryption Standard). Here are the steps

involved in the DES algorithm:

1. Key Generation:

• Start with a 64-bit encryption key. However, only 56 bits of this key are

used directly; the remaining 8 bits are used for error detection and do not

affect the encryption process.

• Perform a key permutation, called the "Initial Permutation (IP)", to create

two 28-bit subkeys, one for each half of the data.

2. Data Preparation:

• The plaintext message is divided into 64-bit blocks.

• An initial permutation, similar to the one used for the key, is applied to the

64-bit plaintext block.

3. Initial Round:

• The 64-bit plaintext block is divided into two 32-bit halves, referred to as

the "left half" (L0) and the "right half" (R0).

• The right half (R0) is expanded to 48 bits using an expansion permutation

(E-box).

• The expanded right half (E(R0)) is then XORed with the 48-bit subkey

generated from the key in the first round (K1).

• The result is passed through the S-boxes (Substitution Boxes), which

substitute 48 bits with 32 bits according to predefined tables.

• The output from the S-boxes is subjected to a permutation (P-box).

• The output from the P-box is then XORed with the left half (L0).

• Swap the left half (L0) and the right half (R0) to prepare for the next round.

4. Iterative Rounds:

• DES uses a total of 16 rounds with each round using a different 48-bit

subkey (K2 to K16) derived from the original key using a process that

involves key rotation and permutation.

• For rounds 2 to 16, the same process as the initial round is repeated,

including the expansion, S-box substitution, permutation, and XOR

operations.

• After each round, the left and right halves are swapped.

5. Final Round:

• After the 16th round, the left and right halves are swapped one more time.

6. Final Permutation:

• A final permutation, the inverse of the initial permutation, is applied to the

data. This effectively reverses the initial permutation and produces the 64-

bit ciphertext block.

7. Repeat for Multiple Blocks:

• If there are additional 64-bit plaintext blocks to encrypt, repeat the above

steps for each block using the same key.

8. Decryption:

• To decrypt the ciphertext, the same steps are applied in reverse order using

the same key schedule. The ciphertext is first subjected to the initial

permutation, and then the rounds are applied in reverse order using the

subkeys in reverse order.

Advantages:

1. Standardization: A widely accepted encryption standard.

2. Efficiency: Designed for efficient implementation.

3. Symmetric Key: Uses the same key for encryption and decryption.

4. Proven Security (in its time): Considered secure when initially introduced.

5. Compact: Uses small key and block sizes.

6. Historical Significance: Paved the way for modern cryptography.

Disadvantages:

1. Small Key Size: 56-bit key vulnerable to brute-force attacks.

2. Outdated Security: Ineffective against modern threats.

3. Cryptanalysis Vulnerabilities: Susceptible to certain attack methods.

4. Fixed Block Size: Limited adaptability to varying data sizes.

5. No Longer Recommended: Replaced by more secure encryption standards.

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, 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";

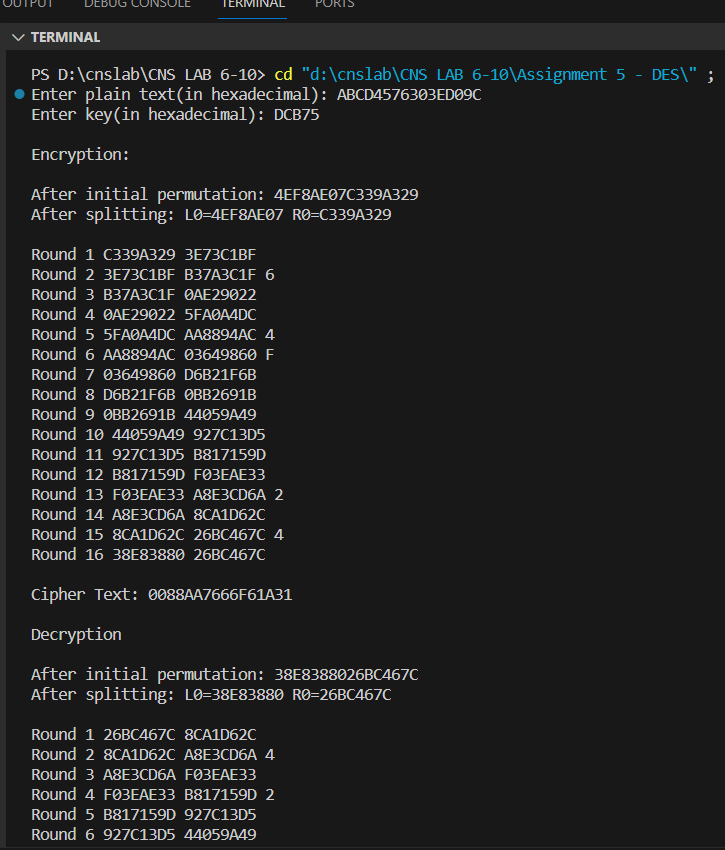
    reverse(rkb.begin(), rkb.end());

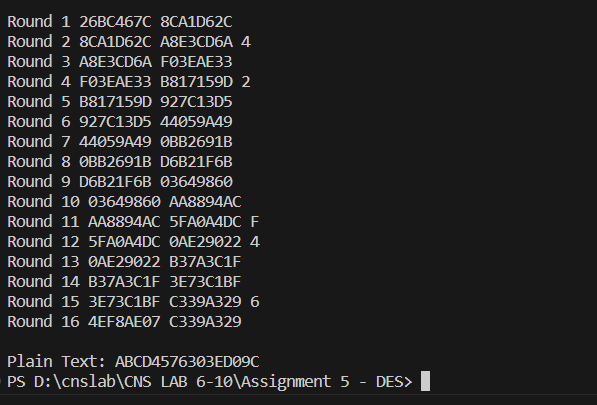
    reverse(rk.begin(), rk.end());

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

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

}





**Advanced Encryption Standard:**

The Advanced Encryption Standard (AES) is a widely used symmetric

encryption algorithm that provides strong data security. Here are the algorithm

steps, advantages, and disadvantages of AES:

Algorithm:

• The AES algorithm begins with a key expansion phase, where the original

encryption key is expanded into a set of round keys. These round keys are

used in the subsequent encryption rounds.

• Initial Round (Add Round Key): The plaintext data is divided into blocks.

In the initial round, the round key is added to the plaintext using a bitwise

XOR operation.

• AES operates in a fixed number of rounds, which depends on the key size.

For AES-128, there are 10 rounds; for AES-192, there are 12 rounds; and for

AES-256, there are 14 rounds. Each round consists of the following

operations:

- SubBytes: Substitutes each byte in the block with a corresponding

byte from the S-Box, a fixed substitution table.

- ShiftRows: Shifts the rows of the block to the left by different offsets.

- MixColumns: Mixes the columns of the block using a mathematical

transformation.

- AddRoundKey: Adds the round key to the block using a bitwise

XOR operation.

• In the final round, the SubBytes, ShiftRows, and MixColumns operations

are performed, followed by adding the final round key.

• The result after the final round is the ciphertext.

Advantages:

• AES is widely considered to be highly secure and is used by governments,

organizations, and individuals for protecting sensitive information.

• AES is a fast and efficient encryption algorithm, making it suitable for a

wide range of applications, including secure communication and data

storage.

• AES has been standardized by the National Institute of Standards and

Technology (NIST) and is widely adopted and accepted in the security

community.

• AES supports multiple key lengths (128, 192, and 256 bits), allowing users

to choose the level of security they need.

• AES has withstood extensive cryptanalysis and has not been found to have

any significant vulnerabilities.

Disadvantages:

• AES is a symmetric encryption algorithm, which means that both the sender

and receiver need to possess the same key. Secure key distribution can be a

challenge, especially over insecure channels.

• While AES is highly secure, it may not be the best choice for all encryption

scenarios. For example, it may not be suitable for public-key encryption or

digital signatures, which require asymmetric cryptography.

• While AES is generally efficient, it can introduce some performance

overhead, particularly for very resource-constrained devices or when using

large key sizes.

• Like many cryptographic algorithms, AES can be vulnerable to side-channel

attacks (e.g., timing or power analysis attacks) if not properly implemented

and protected.

Code:

from block\_cipher import BlockCipher, BlockCipherWrapper

from block\_cipher import MODE\_ECB, MODE\_CBC, MODE\_CFB, MODE\_OFB, MODE\_CTR

def print\_str\_into\_AES\_matrix(str):

    characters = ' '.join([str[i:i+2] for i in range(0, len(str), 2)]).split()

    matrix = [[characters[i] for i in range(j, j + 4)] for j in range(0, len(characters), 4)]

    for col in range(4):

        for row in range(4):

            print(matrix[row][col], end=" ")

        print()

\_\_all\_\_ = [

    'new', 'block\_size', 'key\_size',

    'MODE\_ECB', 'MODE\_CBC', 'MODE\_CFB', 'MODE\_OFB', 'MODE\_CTR'

]

SBOX = (

    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,

)

INV\_SBOX = (

    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,

)

round\_constants = (0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80, 0x1b, 0x36)

block\_size = 16

key\_size = None

def new(key, mode, IV=None, \*\*kwargs) -> BlockCipherWrapper:

    if mode in (MODE\_CBC, MODE\_CFB, MODE\_OFB) and IV is None:

        raise ValueError("This mode requires an IV")

    cipher = BlockCipherWrapper()

    cipher.block\_size = block\_size

    cipher.IV = IV

    cipher.mode = mode

    cipher.cipher = AES(key)

    if mode == MODE\_CFB:

        cipher.segment\_size = kwargs.get('segment\_size', block\_size \* 8)

    elif mode == MODE\_CTR:

        counter = kwargs.get('counter')

        if counter is None:

            raise ValueError("CTR mode requires a callable counter object")

        cipher.counter = counter

    return cipher

class AES(BlockCipher):

    def \_\_init\_\_(self, key: bytes):

        self.key = key

        self.Nk = len(self.key) // 4  # words per key

        if self.Nk not in (4, 6, 8):

            raise ValueError("Invalid key size")

        self.Nr = self.Nk + 6

        self.Nb = 4  # words per block

        self.state: list[list[int]] = []

        # raise NotImplementedError

        # key schedule

        self.w: list[list[int]] = []

        for i in range(self.Nk):

            self.w.append(list(key[4\*i:4\*i+4]))

        for i in range(self.Nk, self.Nb\*(self.Nr+1)):

            tmp: list[int] = self.w[i-1]

            q, r = divmod(i, self.Nk)

            if not r:

                tmp = self.sub\_word(self.rot\_word(tmp))

                tmp[0] ^= round\_constants[q-1]

            elif self.Nk > 6 and r == 4:

                tmp = self.sub\_word(tmp)

            self.w.append(

                [a ^ b for a, b in zip(self.w[i-self.Nk], tmp)]

            )

    def encrypt\_block(self, block: bytes) -> bytes:

        self.set\_state(block)

        self.add\_round\_key(0)

        print("\nInitial:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        for r in range(1, self.Nr):

            print(f"\nRound {r}:")

            self.sub\_bytes()

            print("After SubBytes:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.shift\_rows()

            print("After ShiftRows:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.mix\_columns()

            print("After MixColumns:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.add\_round\_key(r)

            print("After AddRoundKey:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

        print(f"\nFinal Round {r+1}:")

        self.sub\_bytes()

        print("After SubBytes:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        self.shift\_rows()

        print("After ShiftRows:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        self.add\_round\_key(self.Nr)

        print("After AddRoundKey:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        return self.get\_state()

    def decrypt\_block(self, block: bytes) -> bytes:

        self.set\_state(block)

        print("\nInitial:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        self.add\_round\_key(self.Nr)

        for r in range(self.Nr-1, 0, -1):

            print(f"\nRound {r}:")

            self.inv\_shift\_rows()

            print("After Inverse ShiftRows:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.inv\_sub\_bytes()

            print("After Inverse SubBytes:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.add\_round\_key(r)

            print("After AddRoundKey:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

            self.inv\_mix\_columns()

            print("After Inverse MixColumns:")

            print\_str\_into\_AES\_matrix(self.get\_state().hex())

        print(f"\nFinal Round {r}:")

        self.inv\_shift\_rows()

        print("After Inverse ShiftRows:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        self.inv\_sub\_bytes()

        print("After Inverse SubBytes:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        self.add\_round\_key(0)

        print("After AddRoundKey:")

        print\_str\_into\_AES\_matrix(self.get\_state().hex())

        return self.get\_state()

    @staticmethod

    def rot\_word(word: list[int]):

        # for key schedule

        return word[1:] + word[:1]

    @staticmethod

    def sub\_word(word: list[int]):

        # for key schedule

        return [SBOX[b] for b in word]

    def set\_state(self, block: bytes):

        self.state = [

            list(block[i:i+4])

            for i in range(0, 16, 4)

        ]

    def get\_state(self) -> bytes:

        return b''.join(

            bytes(col)

            for col in self.state

        )

    def add\_round\_key(self, r: int):

        round\_key = self.w[r\*self.Nb:(r+1)\*self.Nb]

        for col, word in zip(self.state, round\_key):

            for row\_index in range(4):

                col[row\_index] ^= word[row\_index]

    def mix\_columns(self):

        for i, word in enumerate(self.state):

            new\_word = []

            for j in range(4):

                # element wise cl mul with constants 2, 3, 1, 1

                value = (word[0] << 1)

                value ^= (word[1] << 1) ^ word[1]

                value ^= word[2] ^ word[3]

                # polynomial reduction in constant time

                value ^= 0x11b & -(value >> 8)

                new\_word.append(value)

                # rotate word in order to match the matrix multiplication

                word = self.rot\_word(word)

            self.state[i] = new\_word

    def inv\_mix\_columns(self):

        for i, word in enumerate(self.state):

            new\_word = []

            for j in range(4):

                # element wise cl mul with constants 0xe, 0xb, 0xd, 0x9

                value = (word[0] << 3) ^ (word[0] << 2) ^ (word[0] << 1)

                value ^= (word[1] << 3) ^ (word[1] << 1) ^ word[1]

                value ^= (word[2] << 3) ^ (word[2] << 2) ^ word[2]

                value ^= (word[3] << 3) ^ word[3]

                # polynomial reduction in constant time

                value ^= (0x11b << 2) & -(value >> 10)

                value ^= (0x11b << 1) & -(value >> 9)

                value ^= 0x11b & -(value >> 8)

                new\_word.append(value)

                # rotate word in order to match the matrix multiplication

                word = self.rot\_word(word)

            self.state[i] = new\_word

    def shift\_rows(self):

        for row\_index in range(4):

            row = [

                col[row\_index] for col in self.state

            ]

            row = row[row\_index:] + row[:row\_index]

            for col\_index in range(4):

                self.state[col\_index][row\_index] = row[col\_index]

    def inv\_shift\_rows(self):

        for row\_index in range(4):

            row = [

                col[row\_index] for col in self.state

            ]

            row = row[-row\_index:] + row[:-row\_index]

            for col\_index in range(4):

                self.state[col\_index][row\_index] = row[col\_index]

    def sub\_bytes(self):

        for col in self.state:

            for row\_index in range(4):

                col[row\_index] = SBOX[col[row\_index]]

    def inv\_sub\_bytes(self):

        for col in self.state:

            for row\_index in range(4):

                col[row\_index] = INV\_SBOX[col[row\_index]]

    def print\_state(self):

        # debug function

        for row\_index in range(4):

            print(' '.join(f'{col[row\_index]:02x}' for col in self.state))

        print()

# Main Code

ch = int(input("What do you want to perform?\n1. Encryption\n2. Decryption\n"))

if(ch == 1):

    msg = str(input("Enter the message to be encrypted(16 characters only):\n"))

    if(len(msg) != 16):

        print("Invalid Message size!")

        exit()

    key = str(input("Enter the key for encryption(16 or 24 or 32 characters):\n"))

    key\_length = len(key)

    if (key\_length!=16 and key\_length!=24 and key\_length!=32):

        print("Invalid Key size!")

        exit()

    mode = int(input("Choose the Mode of Operation:\n1. ECB\n2. CBC\n3. CFB\n4. OFB\n5. CTR\n"))

    iv = None

    if mode == 1:

        AES\_MODE = MODE\_ECB

    elif mode == 2:

        AES\_MODE = MODE\_CBC

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 3:

        AES\_MODE = MODE\_CFB

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 4:

        AES\_MODE = MODE\_OFB

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 5:

        AES\_MODE = MODE\_CTR

    else:

        print("Invalid choice !!")

        exit()

    key = bytes.fromhex(key.encode('utf-8').hex())

    plain\_text = bytes.fromhex(msg.encode('utf-8').hex())

    if iv is not None:

        iv = bytes.fromhex(iv.encode('utf-8').hex())

    cipher = new(key, AES\_MODE, IV=iv)

    cipher\_text = cipher.encrypt(plain\_text)

    print(f"\nCiphertext is: {cipher\_text.hex()}")

elif(ch == 2):

    c\_txt = str(input("Enter the ciphertext to be decrypted(16 characters) [in hex format]:\n"))

    if(len(c\_txt) != 32):

        print("Invalid Cipher text size!")

        exit()

    key = str(input("Enter the key for decryption(16 or 24 or 32 characters):\n"))

    key = bytes.fromhex(key.encode('utf-8').hex())

    key\_length = len(key)

    if (key\_length!=16 and key\_length!=24 and key\_length!=32):

        print("Invalid Key size!")

        exit()

    mode = int(input("Choose the Mode of Operation used:\n1. ECB\n2. CBC\n3. CFB\n4. OFB\n5. CTR\n"))

    iv = None

    if mode == 1:

        AES\_MODE = MODE\_ECB

    elif mode == 2:

        AES\_MODE = MODE\_CBC

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 3:

        AES\_MODE = MODE\_CFB

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 4:

        AES\_MODE = MODE\_OFB

        iv = str(input("Enter the Initialization Vector(IV) [16 characters]:\n"))

        if(len(iv) != 16):

            print("Invalid IV size!")

            exit()

    elif mode == 5:

        AES\_MODE = MODE\_CTR

    else:

        print("Invalid choice !!")

        exit()

    if iv is not None:

        iv = bytes.fromhex(iv.encode('utf-8').hex())

    cipher = new(key, AES\_MODE, IV=iv)

    dec\_bytes = cipher.decrypt(bytes.fromhex(c\_txt))

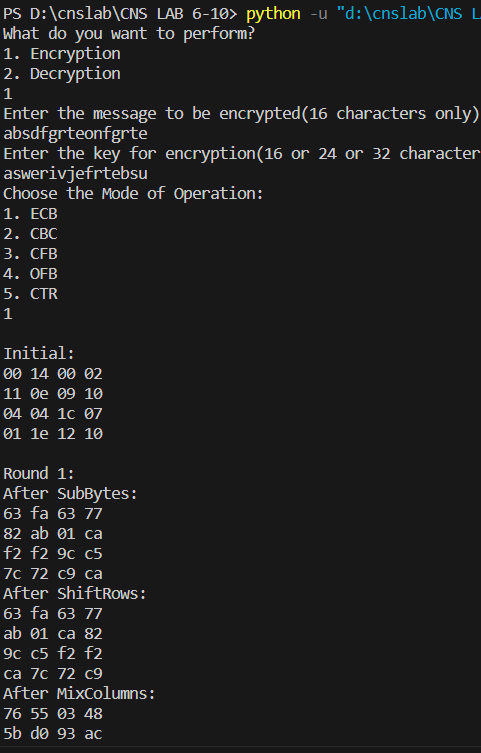
    dec\_txt = dec\_bytes.decode('utf-8')

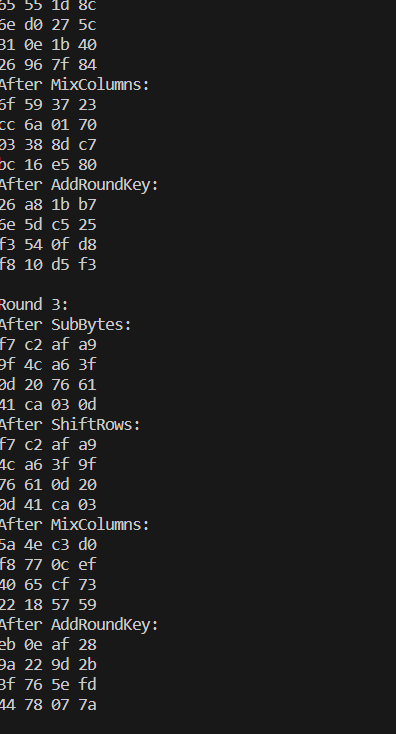
    print(f"\nDecrypted message is: {dec\_txt}")

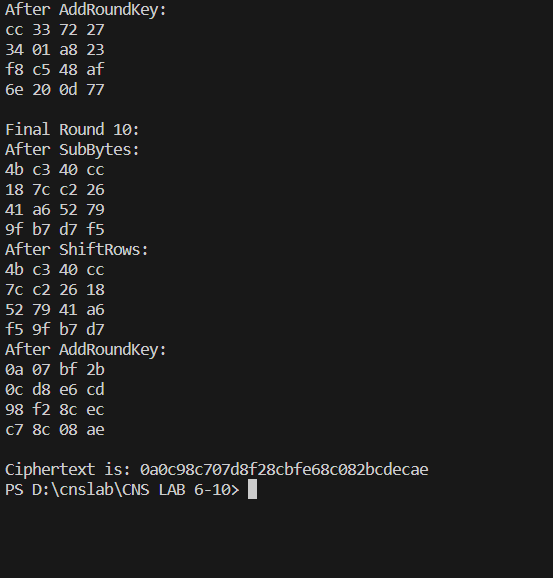
else:

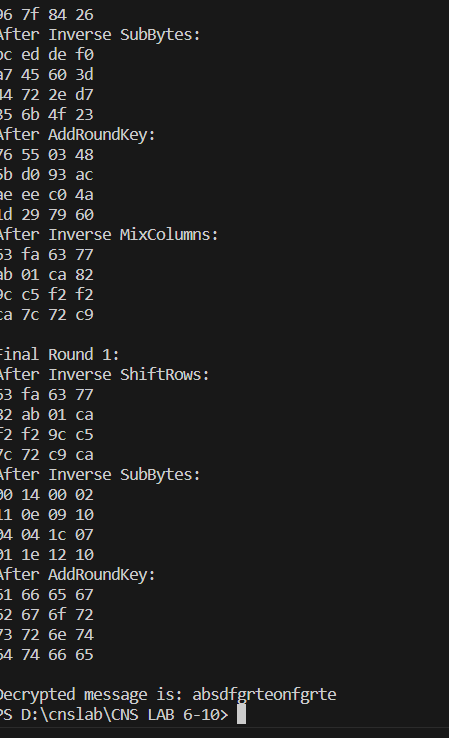
    print("Invalid input!")

output:









**RSA :**

Algorithm:

1. Select Primes: Choose two distinct large prime numbers, p and q.

2. Compute Modulus: Calculate n, where n = p \* q. This becomes the modulus

for the public and private keys.

3. Calculate Euler's Totient: Compute φ(n) where φ(n) = (p - 1)(q - 1).

4. Choose Public Key: Select a number e such that 1 < e < φ(n) and e is coprime

to φ(n).

5. Compute Private Key: Determine d as the modular multiplicative inverse of e

modulo φ(n) (d \* e) mod φ(n) = 1.

Encryption:

6. Represent Message: Represent the message as an integer m, where 0 < m < n.

7. Compute Cipher: Encrypt the message using the public key: \(c = m^e \mod

n\).

Decryption:

8. Compute Message: Decrypt the ciphertext using the private key: \(m = c^d

\mod n\).

Advantages:

• Enables secure transmission of data over insecure networks.

• Uses public and private keys for encryption and decryption, enhancing

security.

• Facilitates the creation and verification of digital signatures for data

integrity and authenticity.

Disadvantages:

• Larger keys are needed for higher security, leading to increased

computational load.

• RSA operations, particularly with larger keys, can be computationally

intensive, affecting performance.

• Weaknesses in random number generation could compromise security.

Code:

from generate\_prime import generate\_prime\_no, is\_prime

# Function to find mod: a^m mod n

def findExpoMod(a, m, n):

    return pow(a, m, n)

def mod\_inverse(a, m):

    m0, x0, x1 = m, 0, 1

    while a > 1:

        q = a // m

        m, a = a % m, m

        x0, x1 = x1 - q \* x0, x0

    return x1 + m0 if x1 < 0 else x1

def gcd(a, h):

    temp = 0

    while (1):

        temp = a % h

        if (temp == 0):

            return h

        a = h

        h = temp

def gen\_keys(p, q):

    n = p\*q

    phi = (p-1)\*(q-1)

    e = 2

    while (e < phi):

        if (gcd(e, phi) == 1):

            break

        else:

            e += 1

    d = mod\_inverse(e, phi)

    print(f"Your Public Key is:\ne = {str(e)}\nn = {str(n)}")

    print(f"Your Private Key is:\nd = {str(d)}\nn = {str(n)}")

def encrypt(message, e, n):

    # Convert alphabetic input to numerical values

    numerical\_message = [ord(char) - ord('A') for char in message.upper()]

    # Encryption: C = (M ^ e) % n

    encrypted\_message = [findExpoMod(char, e, n) for char in numerical\_message]

    return encrypted\_message

def decrypt(encrypted\_message, d, n):

    # Decryption: M = (C ^ d) % n

    decrypted\_numerical\_message = [findExpoMod(

        char, d, n) for char in encrypted\_message]

    # Convert back to alphabetic characters

    decrypted\_message = ''.join(chr(char + ord('A'))

                                for char in decrypted\_numerical\_message)

    return decrypted\_message

# Main Code

ch = int(input("What do you want to perform?\n1. Generate Public & Private Keys\n2. Encryption\n3. Decryption\n"))

if (ch == 1):

    gen\_r = input(

        "Do you want to generate the prime numbers automatically ? [y/n]\n")

    if gen\_r == 'y':

        dig\_p = int(

            input("Enter the number of digits in first prime number(p): "))

        p = generate\_prime\_no(dig\_p)

        dig\_q = int(

            input("Enter the number of digits in second prime number(q): "))

        q = generate\_prime\_no(dig\_q)

        print(f"p = {p}")

        print(f"q = {q}")

        gen\_keys(p, q)

    elif gen\_r == 'n':

        p = int(input("Enter first large prime number(p):\n"))

        if not is\_prime(p):

            print(f"Entered number is not prime!")

            exit()

        q = int(input("Enter second large prime number(q):\n"))

        if not is\_prime(q):

            print(f"Entered number is not prime!")

            exit()

        gen\_keys(p, q)

    else:

        print("Invaild choice!")

        exit()

elif (ch == 2):

    message = input("Enter the message to be encrypted:\n")

    print("Enter the Public Key (e, n):")

    e = int(input("Enter the value of 'e':\n"))

    n = int(input("Enter the value of 'n':\n"))

    encrypted\_message = encrypt(message, e, n)

    print(f"Encrypted message is:\n{' '.join(map(str, encrypted\_message))}")

elif (ch == 3):

    encrypted\_message = list(map(int, input(

        "Enter the list of encrypted values separated by space:\n").split()))

    print("Enter the Private Key (d, n):")

    d = int(input("Enter the value of 'd':\n"))

    n = int(input("Enter the value of 'n':\n"))

    decrypted\_message = decrypt(encrypted\_message, d, n)

    ans = ""

    for a in decrypted\_message:

        if (a < 'A' or a > 'Z'):

            ans += " "

        else:

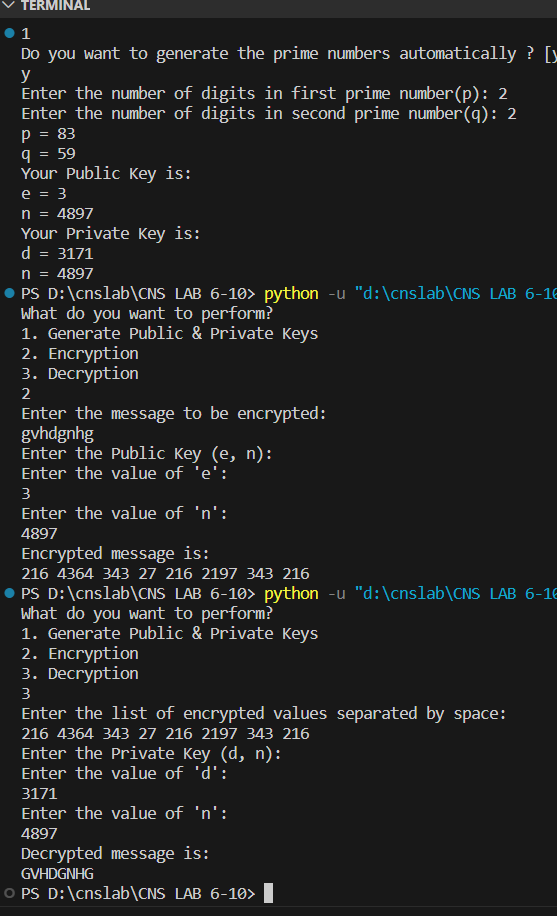
            ans += a

    print(f"Decrypted message is:\n{ans}")

else:

    print("Invalid input!")

Output:



RSA Factorization:

#include <bits/stdc++.h>

using namespace std;

int m = 1e9+7;

pair<long long, long long> factorize(long long n){

for(long long i=2; i\*i<=n; i++){

if(n % i == 0){

return {i,n/i};

}

}

return {0,0};

}

int main()

{

cout<<"Enter long number which is multiplication of 2 prime numbers :";

long long n;

cin>>n;

auto ans = factorize(n);

if(ans.first == 0 && ans.second == 0){

cout<<"Enter number which is multiplication of 2 prime numbers \n";

}else{

cout << 1LL\*n << " is multiplication of "<<ans.first << " and "<<ans.second<<endl;

}

return 0;

}

Output:

**Diffie Hellman using socket programming**

The Diffie-Hellman key exchange is a method used to securely establish a

shared secret key between two parties over an insecure communication channel.

Here is an outline of the Diffie-Hellman algorithm:

1. Setup:

2. Key Exchange:

a. Public Value Calculation:

Alice computes her public value (A): A=ga mod p

Bob computes his public value (B): B=gbmod p

Alice and Bob exchange their calculated public values (A and B).

b. Shared Secret Key Generation:

Alice uses Bob's public value and her private key to compute the

shared secret: s=Ba mod p

Bob uses Alice's public value and his private key to compute the

same shared secret: s=Ab mod p

3. Result: Both Alice and Bob now possess a shared secret key (s) that is

identical.

4. Usage of Shared Key: The shared secret key (s) can be utilized for further

secure communication, such as symmetric encryption or decryption.

Advantages:

• Facilitates secure exchange of cryptographic keys over an insecure

communication channel.

• Uses public and private keys, ensuring a shared secret key without directly

transmitting the private keys.

• Prevents eavesdroppers from deriving the shared secret key, as it's

computationally difficult to deduce from exchanged public values.

Disadvantages:

• Vulnerable to potential man-in-the-middle attacks where an intruder

intercepts and alters the exchanged public keys.

• Diffie-Hellman only provides a method for secure key exchange, lacking

built-in authentication mechanisms to verify the identity of the parties

involved.

• If a private key is compromised, it can compromise the secrecy of the shared

key in subsequent communications.

Code:

Client:

import socket

# Function to find mod: a^m mod n

def findExpoMod(a, m, n):

    # Decimal to binary conversion

    m\_bin = bin(m).replace("0b", "")

    # Convert it into list (individual characters)

    m\_bin\_lst = [int(i) for i in m\_bin]

    # Initialize the list

    a\_lst = [a]

    # Functions to perform operations

    # If next value = 0

    def oneOperation(num):

        return (num\*num) % n

    # If next value = 1

    def twoOperation(num):

        return (a \* oneOperation(num)) % n

    for j in range(len(m\_bin\_lst)):

        if j+1 == len(m\_bin\_lst):

            break

        if(m\_bin\_lst[j+1] == 0):

            a\_lst.append(oneOperation(a\_lst[j]))

        else:

            a\_lst.append(twoOperation(a\_lst[j]))

    return a\_lst[-1]

HOST = 'localhost'

PORT = 12345

client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = (HOST, PORT)  # Server address and port

client\_socket.connect(server\_address)

print(f"Connected to server at: {HOST}:{PORT}")

# Receive the server's public key, q, alpha

received\_Ya = client\_socket.recv(1024)

Ya = int(received\_Ya.decode())

print(f"Received server's public key as: {Ya}")

received\_q = client\_socket.recv(1024)

q = int(received\_q.decode())

print(f"Received large prime as: {q}")

received\_alpha = client\_socket.recv(1024)

alpha = int(received\_alpha.decode())

print(f"Received alpha as: {alpha}")

# Client's Private Key

Xb = int(input(f"Enter the private key for B (Xb) [less than {q}]:\n"))

if Xb >= q:

    print("Private key must be less than choosen prime!")

    exit()

# Client's Public Key

Yb = findExpoMod(alpha, Xb, q)

print(f"Client's Public key is: {Yb}")

# Send this to Server

print("Sending Client's Public Key to Server...")

send\_Yb = str(Yb).encode()

client\_socket.sendall(send\_Yb)

# Receive Server's Shared key

received\_Ks = client\_socket.recv(1024)

Ks = int(received\_Ks.decode())

print(f"Received Server's Shared key as: {Ks}")

# Compute shared key and send to server

Kc = findExpoMod(Ya, Xb, q)

print(f"Client's Shared key is: {Kc}")

print("Sending it to server...")

send\_Kc = str(Kc).encode()

client\_socket.sendall(send\_Kc)

if Kc == Ks:

    print("Both shared keys are equal\nKeys exchanged successfully!")

else:

    print("Both shared keys aren't equal.\nKey exchange failed!")

client\_socket.close()

server:

from generate\_prime import is\_prime, generate\_prime\_no

import socket

# Function to find mod: a^m mod n

def findExpoMod(a, m, n):

    # Decimal to binary conversion

    m\_bin = bin(m).replace("0b", "")

    # Convert it into list (individual characters)

    m\_bin\_lst = [int(i) for i in m\_bin]

    # Initialize the list

    a\_lst = [a]

    # Functions to perform operations

    # If next value = 0

    def oneOperation(num):

        return (num\*num) % n

    # If next value = 1

    def twoOperation(num):

        return (a \* oneOperation(num)) % n

    for j in range(len(m\_bin\_lst)):

        if j+1 == len(m\_bin\_lst):

            break

        if(m\_bin\_lst[j+1] == 0):

            a\_lst.append(oneOperation(a\_lst[j]))

        else:

            a\_lst.append(twoOperation(a\_lst[j]))

    return a\_lst[-1]

def is\_primitive\_root(alpha, q):

    L = []

    for i in range(1, q):

        L.append(findExpoMod(alpha, i, q))

    for i in range(1, q):

        if L.count(i) > 1:

            L.clear()

            return False

        return True

# Initialize Socket

HOST = 'localhost'

PORT = 12345

server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)

server\_address = (HOST, PORT)  # Server address and port

server\_socket.bind(server\_address)

server\_socket.listen(1)

print(f"Server started at: {HOST}:{PORT}")

print("Waiting for a client to connect...")

client\_socket, client\_address = server\_socket.accept()

print("Client connected: ", client\_address)

# DH Key-exchange

# Choose prime no. 'q'

print("Choose a large integer prime number(q):")

gen\_r = input("Do you want to generate the prime number automatically ? [y/n]\n")

if gen\_r == 'y':

    dig\_p = int(input("Enter the number of digits in prime number: "))

    q = generate\_prime\_no(dig\_p)

    print(f"q = {q}")

elif gen\_r == 'n':

    q = int(input("Enter a large prime number:\n"))

    if not is\_prime(q):

        print(f"Entered number is not prime!")

        exit()

else:

    print("Invaild choice!")

    exit()

# Choose primitive root 'alpha'

print("Choose primitive root (alpha):")

gen\_pr = input("Do you want to find the primitive root automatically ? [y/n]\n")

if gen\_pr == 'y':

    for a in range(2, q):

        if is\_primitive\_root(a, q):

            alpha = a

            break

    print(f"Alpha = {alpha}")

elif gen\_pr == 'n':

    alpha = int(input(f"Enter the primiitive root of {q}:\n"))

    if not is\_primitive\_root(alpha, q):

        print(f"This is not the primitive root!")

        exit()

else:

    print("Invaild choice!")

    exit()

# Server's Private key

Xa = int(input(f"Enter the private key for A (Xa) [less than {q}]:\n"))

if Xa >= q:

    print("Private key must be less than choosen prime!")

    exit()

# Server's Public Key

Ya = findExpoMod(alpha, Xa, q)

# Send this data to client

print(f"Server's Public Key is: {Ya}")

print("Sending Public Key to client...")

send\_Ya = str(Ya).encode()

client\_socket.sendall(send\_Ya)

print("Sending choosen large prime to client...")

send\_q = str(q).encode()

client\_socket.sendall(send\_q)

print("Sending primitive root to client...")

send\_alpha = str(alpha).encode()

client\_socket.sendall(send\_alpha)

print("Waiting for Client's Public Key...")

# Receive Client's Public Key

received\_Yb = client\_socket.recv(1024)

Yb = int(received\_Yb.decode())

print(f"Received Public Key of Client: {Yb}")

# Compute shared key and send to client

Ks = findExpoMod(Yb, Xa, q)

print(f"Server's Shared Key is: {Ks}")

print("Sending it to client...")

send\_Ks = str(Ks).encode()

client\_socket.sendall(send\_Ks)

# Receive Client's Shared Key

print("Waiting for Client's Shared Key...")

received\_Kc = client\_socket.recv(1024)

Kc = int(received\_Kc.decode())

print(f"Received Client's Shared Key as: {Kc}")

if Ks == Kc:

    print("Both shared keys are equal\nKeys exchanged successfully!")

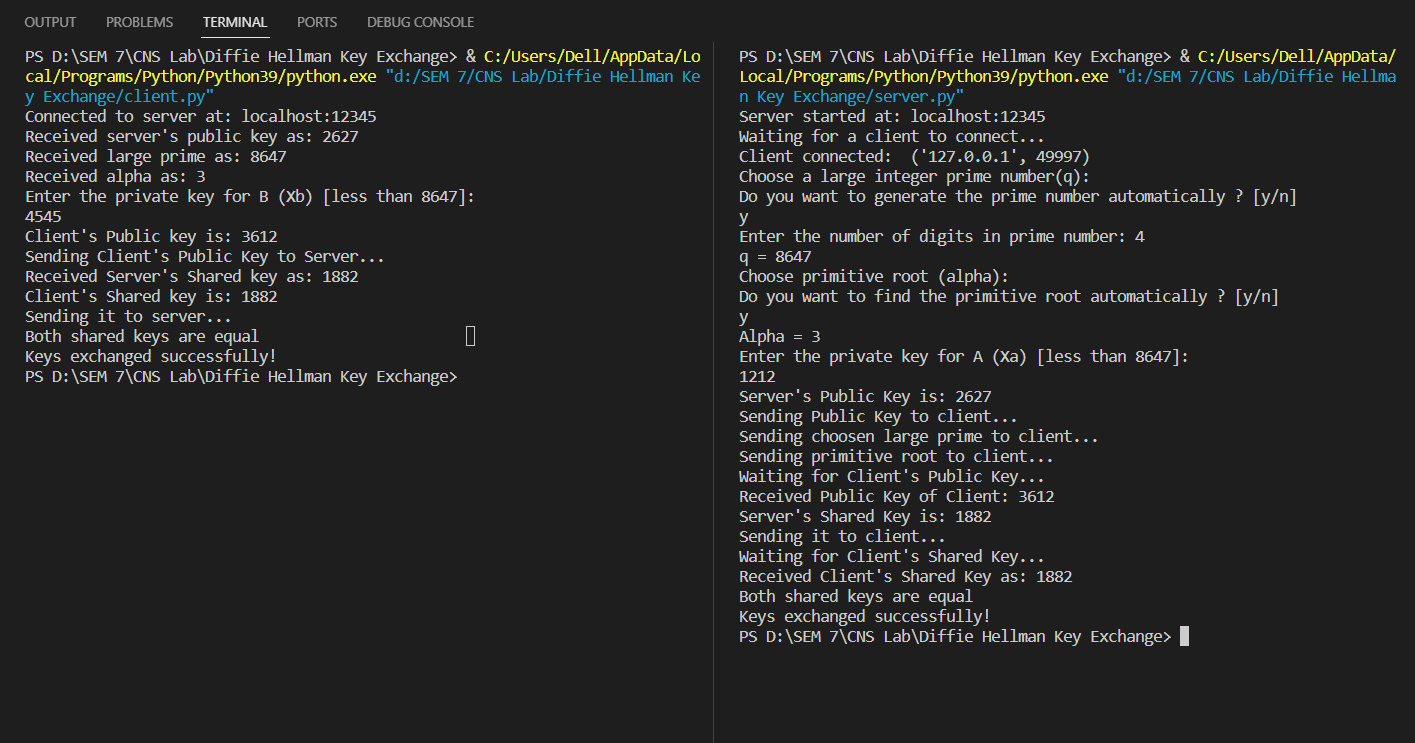
else:

    print("Both shared keys aren't equal.\nKey exchange failed!")

client\_socket.close()

server\_socket.close()

output:



Euclidean Algorithm

The Euclidean Algorithm is a fundamental mathematical algorithm used to find

the greatest common divisor (GCD) of two integers. It has various applications

in number theory, cryptography, and computer science. Here are the algorithm

steps, advantages, and disadvantages of the Euclidean Algorithm:

Algorithm:

• Start with two integers, 'a' and 'b,' where 'a' is greater than or equal to 'b.'

• Divide 'a' by 'b' to obtain a quotient 'q' and a remainder 'r,' such that:

a=b\*q+r

• Set 'a' to 'b' and 'b' to 'r,' then repeat the division and remainder calculation

until 'b' becomes zero.

• The algorithm terminates when 'b' becomes zero. The GCD is then the value

of 'a' at this point.

Advantages:

• The Euclidean Algorithm is highly efficient for finding the GCD of two

integers, and its time complexity is proportional to the number of bits in the

input values.

• It can be applied to both positive and negative integers and is not limited to

just two values; it can find the GCD of multiple integers.

• Mathematical Simplicity: The algorithm is based on simple mathematical

operations (division and remainder), making it easy to understand and

implement.

• Basis for Other Algorithms: The Euclidean Algorithm serves as the basis for

various other algorithms, including the Extended Euclidean Algorithm used

in modular arithmetic and modular inverses.

Disadvantages:

• The algorithm works well for integers, but it may not handle very large

numbers efficiently due to limitations in computational resources and time.

• The Euclidean Algorithm is designed for integers and cannot be directly

applied to non-integer data types.

• If implemented recursively, the algorithm may lead to a stack overflow for

extremely large input values. To avoid this, an iterative version is often

preferred for practical applications.

• The Euclidean Algorithm does not directly factor numbers or provide

information about prime factorization, which may be needed in some

applications.

Code:

#include <bits/stdc++.h>

using namespace std;

int ansS, ansT;

int findGcdExtended(int r1, int r2)

{

    // Base Case

    if (r2 == 0)

    {

        return r1;

    }

    int q = r1 / r2;

    int r = r1 % r2;

    cout << q << " " << r1 << " " << r2 << " " << r << endl;

    return findGcdExtended(r2, r);

}

int main()

{

    int num1, num2;

    cout << "\n Enter 1st number : ";

    cin >> num1;

    cout << "\n Enter 2nd number : ";

    cin >> num2;

    cout<<endl<< "q r1 r2 r" << endl;

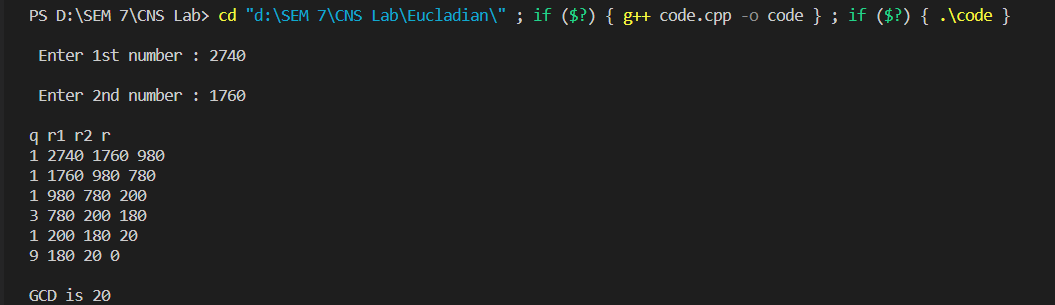
    int gcd = findGcdExtended(num1, num2);

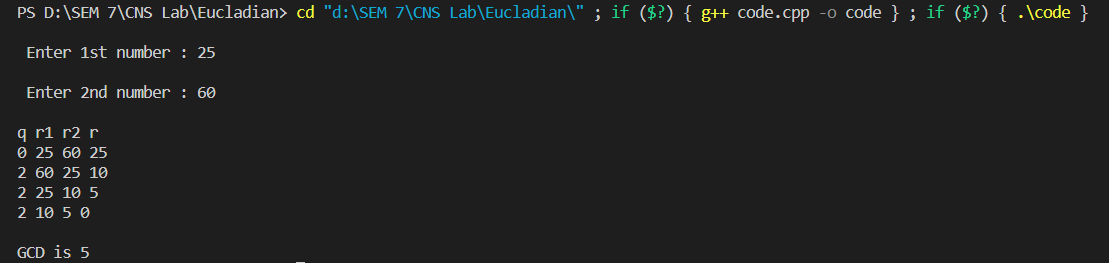
    cout <<endl<< "GCD is " << gcd << endl;

    return 0;

}

**Output:**





Extended Euclidean Algorithm

The Extended Euclidean Algorithm is an extension of the Euclidean Algorithm

that not only calculates the greatest common divisor (GCD) of two integers but

also finds the Bézout coefficients, which are used to compute the modular

inverse of an integer.

Algorithm:

1. Start with two integers 'a' and 'b,' where 'a' is greater than or equal to 'b.'

2. Initialize two sets of variables: 'x0,' 'x1,' 'y0,' and 'y1.' Set 'x0' and 'y1' to 1,

and 'x1' and 'y0' to 0.

3. Divide 'a' by 'b' to obtain a quotient 'q' and a remainder 'r,' such that:

4. a = b \* q + r

5. Update Variables: Update 'a' to 'b' and 'b' to 'r.' Then update the sets of

variables as follows:

• 'x0' becomes 'x1.'

• 'x1' becomes 'x0 - q \* x1.'

• 'y0' becomes 'y1.'

• 'y1' becomes 'y0 - q \* y1.'

6. Repeat steps 3 and 4 until 'b' becomes zero.

7. At this point, 'a' is the GCD of the original 'a' and 'b,'. x0 and y0 (also called

as Bézout coefficients) can be used to find the modular inverse of 'a' modulo

'b' if 'a' and 'b' are coprime (GCD = 1).

Advantages:

• The Extended Euclidean Algorithm not only finds the GCD of two integers

but also computes the Bézout coefficients, which are useful for solving

linear Diophantine equations and calculating modular inverses.

• It is an efficient algorithm for finding both the GCD and the Bézout

coefficients. Its time complexity is proportional to the number of bits in the

input values.

• The algorithm can handle both positive and negative integers, making it

suitable for various applications in number theory and cryptography.

• The Bézout coefficients obtained from the Extended Euclidean Algorithm

are used in modular arithmetic to find the modular inverse of an integer.

Disadvantages:

• Like the Euclidean Algorithm, the Extended Euclidean Algorithm may not

handle very large numbers efficiently due to computational resource

limitations.

• It is designed for integers and may not be directly applied to non-integer

data types.

• While the algorithm itself is conceptually straightforward, the

implementation can become complex, especially in coding the updates to

the variables, leading to potential programming errors.

• The algorithm is primarily used for finding modular inverses and solving

linear Diophantine equations. It may not be the best choice for other

mathematical operations.

Code:

#include <bits/stdc++.h>

using namespace std;

int ansS, ansT;

int findGcdExtended(int r1, int r2, int s1, int s2, int t1, int t2)

{

    // Base Case

    if (r2 == 0)

    {

        ansS = s1;

        ansT = t1;

        return r1;

    }

    int q = r1 / r2;

    int r = r1 % r2;

    int s = s1 - q \* s2;

    int t = t1 - q \* t2;

    cout << q << " " << r1 << " " << r2 << " " << r << " " << s1 << " " << s2 << " " << s << " " << t1 << " " << t2 << " " << t << endl;

    return findGcdExtended(r2, r, s2, s, t2, t);

}

int main()

{

    int num1, num2;

    cout << "\n Enter 1st number : ";

    cin >> num1;

    cout << "\n Enter 2nd number : ";

    cin >> num2;

    cout<<endl<< "q r1 r2 r s1 s2 s t1 t2 t" << endl;

    int gcd = findGcdExtended(num1, num2, 1, 0, 0, 1);

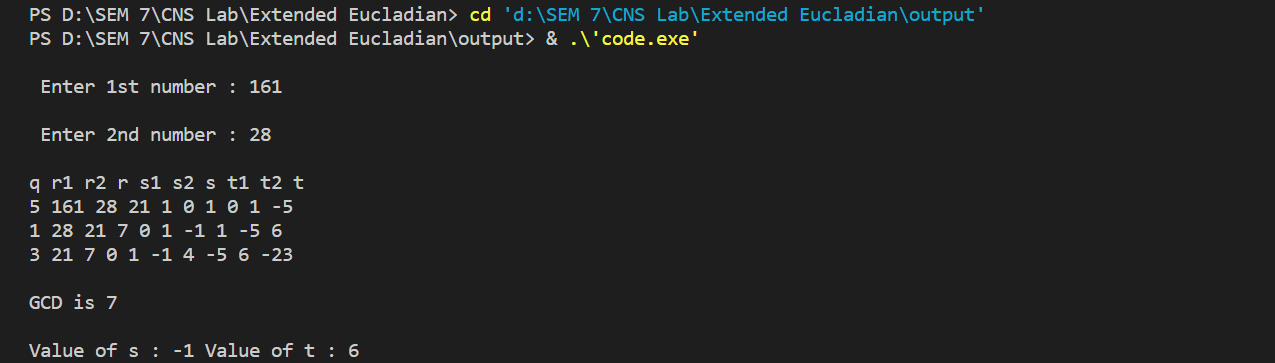
    cout <<endl<< "GCD is " << gcd << endl;

    cout <<endl<< "Value of s : "<<ansS << " " <<"Value of t : "<<ansT << endl;

    return 0;

}

**Output:**



Chinese Remainder Theorem

The Chinese Remainder Theorem (CRT) is a mathematical technique used in

number theory and cryptography. It is primarily used in modular arithmetic to

speed up certain calculations. Here are the algorithm steps, advantages, and

disadvantages of the CRT:

Algorithm:

1. Suppose you have a large integer 'N' that you want to work with and

perform calculations modulo 'N.' You can break this down into simpler

calculations by using the CRT.

2. First, you need to find the prime factorization of 'N.' You express 'N' as a

product of its prime factors: N = p1^e1 \* p2^e2 \* ... \* pk^ek, where p1, p2,

..., pk are distinct prime numbers, and e1, e2, ..., ek are their respective

exponents.

3. For a given number 'x,' you calculate its remainders when divided by each

of the prime factors. This means computing x mod p1, x mod p2, ..., x mod

pk.

4. Inverse Modulus: You then calculate the modular inverse of each prime

factor, which is the number 'y' such that (p1^e1 \* p2^e2 \* ... \* pk^ek) \* y ≡

1 (mod pi) for each prime factor pi.

5. Using the remainders and the modular inverses, you can calculate the value

of 'x' modulo 'N' as follows:

6. x mod N = (x mod p1) \* (p1^e1 \* y1) + (x mod p2) \* (p2^e2 \* y2) + ... + (x

mod pk) \* (pk^ek \* yk).

Advantages:

• CRT allows you to perform modular arithmetic operations more efficiently

by breaking them down into smaller computations.

• You can perform the CRT calculations for different primes in parallel, which

can significantly speed up the process.

• The CRT can reduce the space needed to store large numbers in some

applications.

Disadvantages:

• The CRT algorithm can be complex and requires knowledge of prime

factorization and modular inverses.

• CRT is applicable when you can factor 'N' into its prime factors. In cases

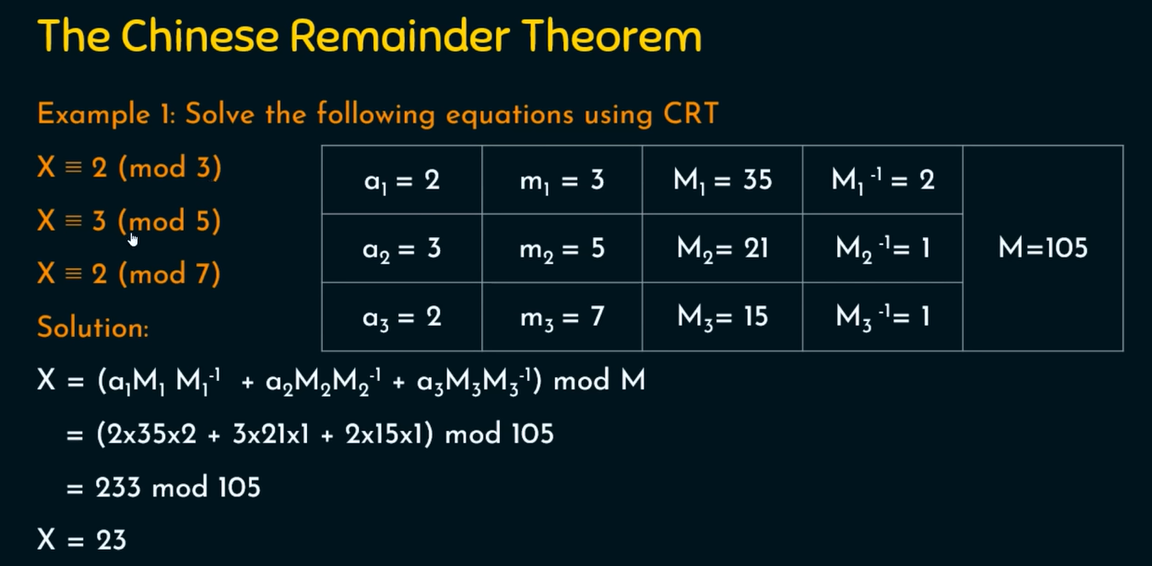
where 'N' is not factorizable, CRT cannot be used.

• The CRT can introduce errors if not implemented carefully due to the risk of

overflow or precision issues in intermediate calculations.

• In cryptography, the CRT can potentially be vulnerable to attacks if not

implemented properly, leading to information leaks.



Code:

#include<iostream>

#include<bits/stdc++.h>

using namespace std;

long long find\_multiplicative\_inverse(long long a, long long b) {

    long long q, r, t1 = 0, t2 = 1, t, main\_a = a;

    while (b > 0) {

        q = a / b;

        r = a % b;

        t = t1 -  (t2 \* q );

        a = b;

        b = r;

        t1 = t2;

        t2 = t;

    }

    if (t1 < 0) {

        t1 += main\_a;

    }

    return t1;

}

int main()

{

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout<<"Chinese Remainder Theorem Problem  \n";

    cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout<<"Suppose that equation needs to be in form of X = a (mod m)\n";

    cout<<"How many equations you want to perfrom : \t";

    int count;

    cin>>count;

     cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    int M=1;

    vector<int> a,m;

    for(int i=0;i<count;i++)

    {

        cout<<"Equation No : \t"<<i+1<<endl;

        cout<<"Enter a :\t";

        int a\_data;

        cin>>a\_data;

        cout<<"Enter m :\t";

        int m\_data;

        cin>>m\_data;

        a.push\_back(a\_data);

        m.push\_back(m\_data);

         cout<<"\n\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

        M=M\*m\_data;

   }

    cout<<"\nValue of M  :\t"<<M<<endl;

    vector<long long > M\_vector,M\_inverse\_vector;

    for(int i=0;i<count;i++)

    {

        M\_vector.push\_back(M/m[i]);

    }

    for(int i=0;i<count;i++)

    {

        M\_inverse\_vector.push\_back(find\_multiplicative\_inverse(m[i],M\_vector[i]));

    }

    long long sum=0;

    for(int i=0;i<count;i++)

    {

        sum+=(a[i] \* M\_vector[i] \* M\_inverse\_vector[i]);

    }

    long long ans=sum%M;

    cout<<"\nAfter calculations :\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    cout << "|\tEq. No\t|\ta[i]\t|\tm[i]\t|\tM[i]\t|\tM\_inverse[i]\t|\n";

    cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    for(int i=0;i<count;i++)

    {

            cout<<"|\t"<<i+1<<"\t|\t"<<a[i]<<"\t|\t"<<m[i]<<"\t|\t"<<M\_vector[i]<<"\t|\t"<<M\_inverse\_vector[i]<<"\t\t|\n";

            cout << "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

    }

    cout<<"\nUsing formula X= E (a[i]\*m[i]\*m^-1[i]) mod M \n";

    cout<<"Value of X is approximate equal to  :  "<<ans;

    return 0;

}

**Output:**