Cryptography and Network Security Lab

Assignment No. 1

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

---

1. Caesar Cipher

The **Caesar Cipher** is a simple substitution cipher where each letter in the plaintext is shifted by a certain number of positions down the alphabet. Named after Julius Caesar, who used it to communicate with his officials, this cipher replaces each letter with the letter that is a fixed number of positions away. For example, with a shift of 3, 'A' would become 'D', 'B' becomes 'E', and so on. It's a monoalphabetic cipher, meaning each letter corresponds to only one letter in the ciphertext.

Example:

- Plaintext: "HELLO"
- Shift: 3
- Ciphertext: "KHOOR"

Weakness: The Caesar Cipher is vulnerable to frequency analysis and brute-force attacks since it has only 25 possible shifts.

```cpp
#include <bits/stdc++.h>
#include <string>

using namespace std;

// Function to encrypt the text using Caesar Cipher
string caesarCipherEncrypt(string text, int shift)
{
    string result = "";

    // Traverse the text
    for (int i = 0; i < text.length(); i++)
```

```
    {
        char ch = text[i];

        // Encrypt uppercase letters
        if (isupper(ch))
        {
            result += char(int(ch + shift - 65) % 26 + 65);
        }
        // Encrypt lowercase letters
        else if (islower(ch))
        {
            result += char(int(ch + shift - 97) % 26 + 97);
        }
        // If it's not an alphabet, keep it unchanged
        else
        {
            result += ch;
        }
    }

    return result;
}

// Function to decrypt the text using Caesar Cipher
string caesarCipherDecrypt(string text, int shift)
{
    string result = "";

    // Traverse the text
    for (int i = 0; i < text.length(); i++)
    {
        char ch = text[i];

        // Decrypt uppercase letters
        if (isupper(ch))
        {
            result += char(int(ch - shift - 65 + 26) % 26 + 65);
        }
        // Decrypt lowercase letters
        else if (islower(ch))
        {
            result += char(int(ch - shift - 97 + 26) % 26 + 97);
        }
        // If it's not an alphabet, keep it unchanged
        else
        {
            result += ch;
        }
```

```
    }

    return result;
}

int main()
{
    string text;
    int shift;

    cout << "Enter the text to be encrypted: ";
    getline(cin, text);

    cout << "Enter the shift value: ";
    cin >> shift;

    string encryptedText = caesarCipherEncrypt(text, shift);
    string decryptedText = caesarCipherDecrypt(encryptedText, shift);

    cout << "Encrypted Text: " << encryptedText << endl;
    cout << "Decrypted Text: " << decryptedText << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1> cd "c:\Users\Sourabh Pa
t 1\" ; if ($?) { g++ Caeser_Cipher.cpp -o Caeser_Cipher } ; if ($?) { .\Caeser_Cipher }
Enter the text to be encrypted: Sourabh
Enter the shift value: 3
Encrypted Text: Vrxudek
Decrypted Text: Sourabh
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1>
```

2. Hill Cipher

The **Hill Cipher** is a polygraphic substitution cipher that uses linear algebra to encrypt a block of letters (usually two or three) at once. The cipher uses a matrix (key) to transform each block of plaintext into ciphertext. The plaintext letters are represented as vectors, and the encryption process involves multiplying these vectors by the key matrix.

To decrypt the message, the inverse of the key matrix is used. The strength of the Hill Cipher lies in the use of matrix mathematics, which adds complexity to the encryption.

Example:

- Plaintext: "ACT" (represented as vectors)
- Key matrix: A 2x2 matrix like [[3, 3], [2, 5]]
- Ciphertext: The result of matrix multiplication

Weakness: It requires an invertible matrix key, and the method can be broken if enough plaintext-ciphertext pairs are known.

```
#include <iostream>
#include <vector>
using namespace std;

vector<vector<int>> keyMatrix = {{3, 3}, {2, 5}};

vector<int> multiplyMatrix(vector<int> vec)
{
    vector<int> result(2);
    result[0] = (keyMatrix[0][0] * vec[0] + keyMatrix[0][1] * vec[1]) % 26;
    result[1] = (keyMatrix[1][0] * vec[0] + keyMatrix[1][1] * vec[1]) % 26;
    return result;
}

string hillEncrypt(string text)
{
    string result = "";
    for (int i = 0; i < text.length(); i += 2)
    {
        vector<int> vec = {text[i] - 'A', text[i + 1] - 'A'};
        vector<int> resVec = multiplyMatrix(vec);
        result += (char)(resVec[0] + 'A');
        result += (char)(resVec[1] + 'A');
    }
    return result;
}

int main()
{
    string text = "HELP";
    string encrypted = hillEncrypt(text);
    cout << "Plaintext: " << text << endl;
    cout << "Encrypted: " << encrypted << endl;
    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1> cd "c:\User
t 1\" ; if ($?) { g++ Hill_Cipher.cpp -o Hill_Cipher } ; if ($?) { .\Hill_Ciph
Plaintext: Cryptography
Encrypted: XPXJFUBJDNZU
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1>
```

3. Playfair Cipher

The **Playfair Cipher** is a digraph substitution cipher, meaning it encrypts two letters at a time. The cipher uses a 5x5 grid of letters (with 'I' and 'J' typically sharing a space) to replace each pair of letters in the plaintext with another pair based on their positions in the grid.

The rules for encryption depend on whether the letters appear in the same row, column, or different rows/columns. If in the same row, each letter is replaced with the one to its right; if in the same column, with the one below it; and if neither, the two letters form a rectangle and are replaced by the letters on the same row in the corners of the rectangle.

Example:

- Key: "MONARCHY" (to form the grid)
- Plaintext: "HELLO"
- Ciphertext: "GCNVF"

Weakness: It is vulnerable to frequency analysis of digraphs and can be broken with known-plaintext attacks.

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

// Function to remove duplicates from a string
string removeDuplicates(string str)
{
    string result;
    for (char c : str)
    {
        if (result.find(c) == string::npos)
        {
            result += c;
        }
    }
    return result;
}
```

```
// Function to prepare the 5x5 key matrix
void generateKeyMatrix(string key, char keyMatrix[5][5])
{
    key = removeDuplicates(key);                                    // Remove
duplicates from the key
    key.erase(remove(key.begin(), key.end(), 'J'), key.end()); // Remove 'J'

    vector<bool> used(26, false);
    used['J' - 'A'] = true; // Treat 'I' and 'J' as the same letter

    int index = 0;
    for (char c : key)
    {
        if (!used[c - 'A'])
        {
            keyMatrix[index / 5][index % 5] = c;
            used[c - 'A'] = true;
            index++;
        }
    }

    // Fill the remaining spaces with other letters
    for (char c = 'A'; c <= 'Z'; c++)
    {
        if (!used[c - 'A'])
        {
            keyMatrix[index / 5][index % 5] = c;
            index++;
        }
    }
}

// Function to find the position of a letter in the key matrix
void findPosition(char keyMatrix[5][5], char c, int &row, int &col)
{
    if (c == 'J')
        c = 'I'; // Treat 'I' and 'J' as the same letter
    for (int i = 0; i < 5; i++)
    {
        for (int j = 0; j < 5; j++)
        {
            if (keyMatrix[i][j] == c)
            {
                row = i;
                col = j;
                return;
            }
        }
```

```
    }
}

// Function to encrypt a pair of characters
string encryptPair(char keyMatrix[5][5], char a, char b)
{
    int row1, col1, row2, col2;
    findPosition(keyMatrix, a, row1, col1);
    findPosition(keyMatrix, b, row2, col2);

    if (row1 == row2)
    {
        // Same row, shift columns right
        return string(1, keyMatrix[row1][(col1 + 1) % 5]) +
keyMatrix[row2][(col2 + 1) % 5];
    }
    else if (col1 == col2)
    {
        // Same column, shift rows down
        return string(1, keyMatrix[(row1 + 1) % 5][col1]) + keyMatrix[(row2 +
1) % 5][col2];
    }
    else
    {
        // Rectangle swap
        return string(1, keyMatrix[row1][col2]) + keyMatrix[row2][col1];
    }
}

// Function to prepare the plaintext by handling pairs
string prepareText(string text)
{
    string result;
    for (size_t i = 0; i < text.length(); i++)
    {
        result += toupper(text[i]);
        if (i + 1 < text.length() && toupper(text[i]) == toupper(text[i + 1]))
        {
            result += 'X'; // Insert 'X' between duplicate letters in a pair
        }
    }
    if (result.length() % 2 != 0)
    {
        result += 'X'; // Append 'X' if the length of the text is odd
    }
    return result;
}
```

```cpp
// Function to encrypt the plaintext using Playfair cipher
string playfairEncrypt(string plaintext, string key)
{
    char keyMatrix[5][5];
    generateKeyMatrix(key, keyMatrix);

    plaintext = prepareText(plaintext);
    string cipherText = "";

    for (size_t i = 0; i < plaintext.length(); i += 2)
    {
        cipherText += encryptPair(keyMatrix, plaintext[i], plaintext[i + 1]);
    }

    return cipherText;
}

int main()
{
    string plaintext, key;
    cout << "Enter the plaintext: ";
    getline(cin, plaintext);
    cout << "Enter the key: ";
    getline(cin, key);

    string cipherText = playfairEncrypt(plaintext, key);
    cout << "Encrypted Text: " << cipherText << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1> cd "c:\Users\Sour
t 1\" ; if ($?) { g++ Play_Fair_Cipher.cpp -o Play_Fair_Cipher } ; if ($?) { .\Play_
Enter the plaintext: Sourabh
Enter the key: Playfair
Encrypted Text: TQTSBCGC
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1>
```

4. Vigenère Cipher

The **Vigenère Cipher** is a polyalphabetic substitution cipher that uses a keyword to determine the shift for each letter in the plaintext. Each letter in the plaintext is shifted by a different amount based on the corresponding letter in the keyword. The key is repeated as needed to match the length of the plaintext.

Unlike the Caesar Cipher, which uses a constant shift, the Vigenère Cipher uses multiple shifts, making it more resistant to simple frequency analysis. However, it can still be broken with techniques like the Kasiski examination.

Example:

- Plaintext: "ATTACKATDAWN"
- Key: "LEMON"
- Ciphertext: "LXFOPVEFRNHR"

Weakness: While stronger than the Caesar Cipher, the Vigenère Cipher is vulnerable to modern cryptanalysis techniques.

These ciphers represent important milestones in cryptography, ranging from simple to more complex encryption methods.

```cpp
#include <iostream>
#include <string>

using namespace std;

// Function to generate the key in a cyclic manner until its length equals the length of the text
string generateKey(string text, string key)
{
    int textLength = text.length();
    int keyLength = key.length();

    for (int i = 0; i < textLength - keyLength; i++)
    {
        key += key[i % keyLength];
    }
    return key;
}

// Function to encrypt the plaintext using Vigenère Cipher
string vigenereEncrypt(string text, string key)
{
    string cipherText = "";

    for (int i = 0; i < text.length(); i++)
    {
        // Convert letters to numbers, A=0, B=1, ..., Z=25
        char x = (text[i] + key[i]) % 26;
        // Convert numbers back to letters
        x += 'A';
        cipherText += x;
    }
```

```
    return cipherText;
}

// Function to decrypt the ciphertext using Vigenère Cipher
string vigenereDecrypt(string cipherText, string key)
{
    string plainText = "";

    for (int i = 0; i < cipherText.length(); i++)
    {
        // Convert letters to numbers, A=0, B=1, ..., Z=25
        char x = (cipherText[i] - key[i] + 26) % 26;
        // Convert numbers back to letters
        x += 'A';
        plainText += x;
    }

    return plainText;
}

int main()
{
    string text, keyword;

    cout << "Enter the plaintext (uppercase letters only): ";
    getline(cin, text);

    cout << "Enter the keyword (uppercase letters only): ";
    getline(cin, keyword);

    // Generate the key in a cyclic manner
    string key = generateKey(text, keyword);

    // Encrypt the text
    string cipherText = vigenereEncrypt(text, key);
    cout << "Encrypted Text: " << cipherText << endl;

    // Decrypt the text
    string decryptedText = vigenereDecrypt(cipherText, key);
    cout << "Decrypted Text: " << decryptedText << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1> cd "c:\Users\
t 1\" ; if ($?) { g++ Vigenere_Cipher.cpp -o Vigenere_Cipher } ; if ($?) { .\Vig
Enter the plaintext (uppercase letters only): SOURABH
Enter the keyword (uppercase letters only): CNS
Encrypted Text: UBMTNTJ
Decrypted Text: SOURABH
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 1>
```

Cryptography and Network Security Lab

Assignment No. 2

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

---

1. Rail Fence Cipher

The **Rail Fence Cipher** is a form of transposition cipher that rearranges the letters of the plaintext to create ciphertext. The method involves writing the plaintext in a zigzag pattern across multiple "rails" (lines) and then reading the letters off row by row.

How It Works:

1. **Determine the Number of Rails**: Decide how many rails (lines) you want to use.
2. **Write the Plaintext**: Write the plaintext diagonally down and up across the rails. When you reach the bottom rail, go back up to the top rail, continuing this zigzag pattern.
3. **Read Off the Ciphertext**: After filling the rails, read the letters from each rail sequentially to form the ciphertext.

Example:

- **Plaintext**: "HELLO WORLD"
- **Number of Rails**: 3
- **Ciphertext**: "HOR ELWLD"

Weakness: The Rail Fence Cipher is relatively easy to break with frequency analysis, especially if the number of rails is known.

```cpp
#include <iostream>
#include <string>
#include <vector>

using namespace std;

// Function to encrypt using the Rail Fence Cipher
string railFenceEncrypt(string text, int key)
{
    if (key == 1)
        return text; // Special case where key is 1 (no encryption)

    vector<string> rail(key);
    int direction = -1;
    int row = 0;

    for (char c : text)
    {
        rail[row] += c;

        // Change direction when you reach the top or bottom rail
        if (row == 0 || row == key - 1)
        {
            direction *= -1;
        }

        row += direction;
    }

    // Combine all the rails to form the ciphertext
    string cipherText;
    for (const string &line : rail)
    {
        cipherText += line;
    }

    return cipherText;
}

// Function to decrypt using the Rail Fence Cipher
string railFenceDecrypt(string cipherText, int key)
{
    if (key == 1)
        return cipherText; // Special case where key is 1 (no decryption)

    vector<int> railLength(key, 0);
    int direction = -1;
    int row = 0;
```

```
    // First determine the length of each rail
    for (char c : cipherText)
    {
        railLength[row]++;
        if (row == 0 || row == key - 1)
        {
            direction *= -1;
        }
        row += direction;
    }

    // Now, populate the rails with the cipher text
    vector<string> rail(key);
    int index = 0;
    for (int i = 0; i < key; i++)
    {
        rail[i] = cipherText.substr(index, railLength[i]);
        index += railLength[i];
    }

    // Reconstruct the original text by reading in zigzag order
    string plainText;
    row = 0;
    direction = -1;
    int railPos[key] = {0};

    for (int i = 0; i < cipherText.length(); i++)
    {
        plainText += rail[row][railPos[row]++];
        if (row == 0 || row == key - 1)
        {
            direction *= -1;
        }
        row += direction;
    }

    return plainText;
}

int main()
{
    string text;
    int key;

    cout << "Enter the plaintext: ";
    getline(cin, text);
    cout << "Enter the key (number of rails): ";
```

```
    cin >> key;

    string cipherText = railFenceEncrypt(text, key);
    cout << "Encrypted Text: " << cipherText << endl;

    string decryptedText = railFenceDecrypt(cipherText, key);
    cout << "Decrypted Text: " << decryptedText << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 2> cd "c:\Users\S
t 2\" ; if ($?) { g++ Rail_fence_cipher.cpp -o Rail_fence_cipher } ; if ($?) { .\
Enter the plaintext: sourabh
Enter the key (number of rails): 5
Encrypted Text: souhrba
Decrypted Text: sourabh
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 2>
```

2. Row Column Transformation Cipher

The **Row Column Transformation Cipher** is a transposition cipher that encrypts the plaintext by rearranging the characters based on a specified row and column structure. This method typically involves creating a grid or matrix to organize the plaintext letters and then applying a specific transformation to generate the ciphertext.

How It Works:

1. **Choose a Key**: Select a key that will dictate the arrangement of the rows and columns.
2. **Create a Matrix**: Write the plaintext into a matrix (grid) based on the number of rows and columns defined by the key. If the plaintext doesn't fill the matrix completely, you may need to pad it with a filler character (like 'X').
3. **Rearrange Based on the Key**: The key determines the order in which the columns (or rows) will be read to form the ciphertext. For example, if the key is a number representing the order of columns, you would read the columns in that order.

Example:

- **Plaintext**: "HELLO WORLD"
- **Key**: 3 (indicating a 3-column matrix)

- **Reading Order**: Based on the key, the columns are read in a specific order (e.g., 2nd column, 1st column, 3rd column).
- **Ciphertext**: "EOLHROLD"

Weakness: Like other transposition ciphers, the Row Column Transformation Cipher can be vulnerable to frequency analysis and can be broken if the arrangement pattern is known.

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>

using namespace std;

// Row and Column Transformation Encryption
string rowColumnEncrypt(string text, vector<int> key)
{
    int n = key.size();
    int paddedLen = text.length() + (n - text.length() % n) % n;
    text.append(paddedLen - text.length(), 'X');

    vector<string> grid;
    for (int i = 0; i < text.length(); i += n)
    {
        grid.push_back(text.substr(i, n));
    }

    string encryptedText;
    for (int i : key)
    {
        for (const string &row : grid)
        {
            encryptedText += row[i - 1];
        }
    }

    return encryptedText;
}

// Row and Column Transformation Decryption
string rowColumnDecrypt(string cipher, vector<int> key)
{
    int n = key.size();
    int numRows = cipher.length() / n;

    vector<string> grid(numRows, string(n, ' '));
    int index = 0;

    for (int i : key)
```

```
    {
        for (int j = 0; j < numRows; ++j)
        {
            grid[j][i - 1] = cipher[index++];
        }
    }

    string decryptedText;
    for (const string &row : grid)
    {
        decryptedText += row;
    }

    while (decryptedText.back() == 'X')
    {
        decryptedText.pop_back();
    }

    return decryptedText;
}

int main()
{
    string plaintext = "Sourabh Patil";
    vector<int> key = {3, 1, 4, 2};

    string encryptedText = rowColumnEncrypt(plaintext, key);
    string decryptedText = rowColumnDecrypt(encryptedText, key);

    cout << "\nRow and Column Transformation:\nEncrypted: " << encryptedText
<< "\nDecrypted: " << decryptedText << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 2> cd "c:\User
t 2\" ; if ($?) { g++ Row_column_transformation_cipher.cpp -o Row_column_trans
mation_cipher }

Row and Column Transformation:
Encrypted: uhtXSaPlr iXobaX
Decrypted: Sourabh Patil
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 2>
```

Cryptography and Network Security Lab

Assignment No. 3

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

---

1. Euclidean Algorithm

The **Euclidean Algorithm** is a method for finding the greatest common divisor (GCD) of two integers. The GCD is the largest positive integer that divides both numbers without leaving a remainder. The algorithm is based on the principle that the GCD of two numbers also divides their difference.

How It Works:

1. Given two integers a and b (where a≥b), divide a by b and find the remainder r.
2. Replace a with b and b with r.
3. Repeat the process until b becomes zero. The last non-zero remainder is the GCD.

Example:

- Find the GCD of 48 and 18.
- The GCD is 6, the last non-zero remainder.

```
#include <bits/stdc++.h>

using namespace std;

// Function to compute the GCD of two numbers using the Euclidean algorithm
int gcd(int a, int b)
{
    while (b != 0)
    {
        int remainder = a % b;
```

```
        a = b;
        b = remainder;
    }
    return a;
}

int main()
{
    int a, b;

    cout << "Enter two integers: ";
    cin >> a >> b;

    int result = gcd(a, b);
    cout << "GCD of " << a << " and " << b << " is: " << result << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 3> cd "c:\Users\Sou
ignment 3\" ; if ($?) { g++ tempCodeRunnerFile.cpp -o tempCodeRunnerFile } ; if ($?
Enter two integers: 10 15
GCD of 10 and 15 is: 5
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 3>
```

2. Extended Euclidean Algorithm

The **Extended Euclidean Algorithm** not only computes the GCD of two integers a and b, but it also finds integers x and y such that:

ax+by=GCD(a,b)

This is particularly useful in number theory, especially in solving linear Diophantine equations and finding modular inverses.

How It Works:

1. Perform the steps of the Euclidean algorithm while keeping track of coefficients xxx and yyy.
2. Start with initial values:
3. During each iteration of the Euclidean algorithm, update the coefficients using the relationship:

```cpp
#include <iostream>

using namespace std;

// Function to implement the Extended Euclidean Algorithm
int extendedGCD(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

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

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

    return gcd;
}

int main()
{
    int a, b;
    int x, y;

    cout << "Enter two integers: ";
    cin >> a >> b;

    int gcd = extendedGCD(a, b, x, y);
    cout << "GCD of " << a << " and " << b << " is: " << gcd << endl;
    cout << "Coefficients x and y are: " << x << " and " << y << endl;
    cout << "Verification: " << a << " * " << x << " + " << b << " * " << y <<
" = " << gcd << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 3> cd "c:\Users
ignment 3\" ; if ($?) { g++ Extended_Euclidean_algo.cpp -o Extended_Euclidean_a

Enter two integers: 20 25
GCD of 20 and 25 is: 5
Coefficients x and y are: -1 and 1
Verification: 20 * -1 + 25 * 1 = 5
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 3>
```

Cryptography and Network Security Lab

Assignment No. 4

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

Chinese Remainder Theorem (CRT)

The **Chinese Remainder Theorem** states that if you have a system of simultaneous congruences with pairwise coprime moduli, then there exists a unique solution modulo the product of the moduli.

```cpp
#include <bits/stdc++.h>

using namespace std;

// Function to perform the Extended Euclidean Algorithm to find modular inverses
int extendedGCD(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

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

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

    return gcd;
```

```cpp
}

// Function to find the modular inverse of a under modulo m
int modInverse(int a, int m)
{
    int x, y;
    int g = extendedGCD(a, m, x, y);
    if (g != 1)
    {
        return -1; // Modular inverse doesn't exist
    }
    else
    {
        return (x % m + m) % m;
    }
}

// Function to apply the Chinese Remainder Theorem
int chineseRemainderTheorem(vector<int> num, vector<int> rem)
{
    int k = num.size(); // Number of equations
    int N = 1;

    // Compute product of all numbers
    for (int i = 0; i < k; i++)
    {
        N *= num[i];
    }

    int result = 0;

    // Apply the formula x = sum(ai * Ni * Mi) % N
    for (int i = 0; i < k; i++)
    {
        int Ni = N / num[i];
        int Mi = modInverse(Ni, num[i]);
        result += rem[i] * Ni * Mi;
    }

    return result % N;
}

int main()
{
    vector<int> num = {3, 5, 7}; // Moduli
    vector<int> rem = {2, 3, 2}; // Remainders

    int result = chineseRemainderTheorem(num, rem);
```

```
    cout << "The solution x is: " << result << endl;

    return 0;
}
```

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 4> cd "c:\Users\Sourabh Pat
t 4\" ; if ($?) { g++ Chinese_remainder_theorem.cpp -o Chinese_remainder_theorem } ; if ($
The solution x is: 23
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 4>
```

Cryptography and Network Security Lab

Assignment No. 5

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

Data Encryption Standard (DES)

The **Data Encryption Standard (DES)** is a symmetric-key block cipher that was adopted as a federal standard in the United States in 1977. It encrypts data in fixed-size blocks of 64 bits using a 56-bit key.

Key Features:

- **Symmetric Encryption**: The same key is used for both encryption and decryption.
- **Block Cipher**: Operates on fixed-size blocks of data (64 bits).
- **Feistel Structure**: DES uses a Feistel network, dividing the block into two halves and applying multiple rounds of transformation.
- **Rounds**: It consists of 16 rounds of processing, including substitution and permutation steps, to increase security.

Security:

- DES was considered secure for many years; however, advances in computing power have made it vulnerable to brute-force attacks.
- It was officially withdrawn as a standard by the National Institute of Standards and Technology (NIST) in 2005 in favor of more secure algorithms like **AES (Advanced Encryption Standard)**.

```cpp
#include <iostream>
#include <bitset>
#include <vector>

using namespace std;

// Initial Permutation table for DES
int IP[] = {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};

// Final Permutation table for DES
int FP[] = {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};

bitset<64> DES_encrypt(bitset<64> plaintext)
{
    bitset<64> permutedText;
    for (int i = 0; i < 64; i++)
    {
        permutedText[i] = plaintext[IP[i] - 1];
    }

    bitset<64> cipherText;
    for (int i = 0; i < 64; i++)
    {
        cipherText[i] = permutedText[FP[i] - 1];
    }

    return cipherText;
}

int main()
{
    bitset<64>
plaintext("1100101010110101011101001010101010101010101010101010101010101010");
```

```
    bitset<64> cipherText = DES_encrypt(plaintext);

    // Display the result
    cout << "Encrypted text: " << cipherText << endl;

    return 0;
}
```

Output:

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 5> cd "c:\Users\Sourabh Patil\notes\ly first sem\C
NS\LAB\assignment 5\" ; if ($?) { g++ des.cpp -o des } ; if ($?) { .\des }
Encrypted text: 1100101010110101011101001010101010101010101010101010101010101010
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 5>
```

Cryptography and Network Security Lab

Assignment No. 6

Name : Sourabh Shankar Patil

PRN : 21510045

Batch : B2

Advanced Encryption Standard (AES)

The **Advanced Encryption Standard (AES)** is a widely used symmetric-key block cipher that was established by the National Institute of Standards and Technology (NIST) in 2001. It is the successor to the Data Encryption Standard (DES) and is known for its high level of security and efficiency.

Key Features:

- **Block Size**: AES operates on fixed block sizes of 128 bits.
- **Key Lengths**: AES supports three key lengths: 128, 192, and 256 bits, allowing for different levels of security.
- **Symmetric Encryption**: The same key is used for both encryption and decryption, making key management critical.

Structure:

- **Substitution-Permutation Network (SPN)**: AES uses a series of well-defined transformations that include:
    - **SubBytes**: A non-linear substitution step where each byte is replaced with another byte using a fixed substitution table (S-box).
    - **ShiftRows**: A transposition step where each row of the state is shifted cyclically to the left.
    - **MixColumns**: A mixing operation that combines the bytes of each column.

    - **AddRoundKey**: A key addition step where a round key is combined with the state using bitwise XOR.
- **Rounds**: AES uses a variable number of rounds depending on the key length:
    - 10 rounds for 128-bit keys
    - 12 rounds for 192-bit keys
    - 14 rounds for 256-bit keys

Security:

- AES is considered highly secure and is used in various applications, including data encryption, secure communications, and VPNs.
- It has been extensively analyzed and is resistant to most known cryptographic attacks, making it a standard choice for secure data transmission.

Applications:

- AES is widely implemented in software and hardware across various industries, including banking, telecommunications, and government.
- It is also used in protocols such as SSL/TLS for secure internet communication.

```cpp
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;

// AES S-box table
unsigned char sbox[256] = {
    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};

void AES_encrypt(vector<unsigned char> &state)
{
    for (int i = 0; i < 16; i++)
    {
        state[i] = sbox[state[i]]; // SubBytes operation
    }
}

int main()
{
    vector<unsigned char> plaintext = {0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a,
0x30, 0x8d, 0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34};

    cout << "Plaintext: ";
    for (unsigned char c : plaintext)
    {
        cout << hex << setw(2) << setfill('0') << (int)c << " ";
    }
    cout << endl;

    AES_encrypt(plaintext);

    cout << "Encrypted text: ";
    for (unsigned char c : plaintext)
    {
        cout << hex << setw(2) << setfill('0') << (int)c << " ";
    }
    cout << endl;

    return 0;
}
```

Output:

```
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 6> cd "c:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assi
t 6\" ; if ($?) { g++ aes.cpp -o aes } ; if ($?) { .\aes }
Plaintext: 32 43 f6 a8 88 5a 30 8d 31 31 98 a2 e0 37 07 34
Encrypted text: 23 1a 42 c2 c4 be 04 5d c7 c7 46 3a e1 9a c5 18
PS C:\Users\Sourabh Patil\notes\ly first sem\CNS\LAB\assignment 6>
```