Cryptography, Network and Security

Assignment 1

1. Perform encryption, decryption using the following substitution techniques

a. Ceaser cipher,

b. playfair cipher

c. Hill Cipher

d. Vigenere cipher

Code:

a.

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

// Function to encrypt text using Caesar Cipher
string caesarEncrypt(string text, int s)
{
    string result = "";

    for (int i = 0; i < text.length(); i++)
    {
        if (text[i] == ' ')
        {
            result += ' ';
            continue;
        }

        if (isupper(text[i]))
        {
            result += char(int(text[i] + s - 65) % 26 + 65);
        }

        else
        {
            result += char(int(text[i] + s - 97) % 26 + 97);
        }
    }
    return result;
}

// Function to decrypt text using Caesar Cipher
string caesarDecrypt(string text, int s)
```

```
{
    return caesarEncrypt(text, 26 - s);
}

int main()
{
    cout << "Enter the text to be encrypted: ";
    string text;
    getline(cin, text);
    cout << "Enter the shift value: ";
    int s;
    cin >> s;

    cout << "Original Text: " << text << endl;
    string encrypted = caesarEncrypt(text, s);
    cout << "Encrypted Text: " << encrypted << endl;
    cout << "Decrypted Text: " << caesarDecrypt(encrypted, s) << endl;

    return 0;
}
```

b.

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;

char keyMatrix[5][5];

// Function to remove duplicates from a string
string removeDuplicates(string str) {
    string result;
    bool used[26] = {false};
    for (char c : str) {
        if (!used[c - 'a']) {
            used[c - 'a'] = true;
            result += c;
        }
    }
    return result;
}

// Function to create the Playfair key matrix
void generateKeyMatrix(string key) {
    key = removeDuplicates(key);
    key.erase(remove(key.begin(), key.end(), 'j'), key.end());
```

```
    bool used[26] = {false};
    int k = 0;
    for (char c : key) {
        used[c - 'a'] = true;
    }
    key += "abcdefghiklmnopqrstuvwxyz";

    int index = 0;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            while (used[key[index] - 'a']) index++;
            keyMatrix[i][j] = key[index];
            used[key[index] - 'a'] = true;
        }
    }
}

// Function to find the position of a character in the key matrix
void findPosition(char c, int &row, int &col) {
    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 preprocess the plaintext (handle repeated characters, make
pairs)
string preprocessPlaintext(string text) {
    text.erase(remove(text.begin(), text.end(), ' '), text.end());
    for (int i = 0; i < text.length(); i += 2) {
        if (i + 1 == text.length()) {
            text += 'x';
        } else if (text[i] == text[i + 1]) {
            text.insert(i + 1, 1, 'x');
        }
    }
    return text;
}

// Function to encrypt plaintext using Playfair Cipher
string playfairEncrypt(string plaintext, string key) {
    generateKeyMatrix(key);
    plaintext = preprocessPlaintext(plaintext);
```

```
    string encrypted = "";

    for (int i = 0; i < plaintext.length(); i += 2) {
        int r1, c1, r2, c2;
        findPosition(plaintext[i], r1, c1);
        findPosition(plaintext[i + 1], r2, c2);

        if (r1 == r2) {
            encrypted += keyMatrix[r1][(c1 + 1) % 5];
            encrypted += keyMatrix[r2][(c2 + 1) % 5];
        } else if (c1 == c2)
            encrypted += keyMatrix[(r1 + 1) % 5][c1];
            encrypted += keyMatrix[(r2 + 1) % 5][c2];
        } else {
            encrypted += keyMatrix[r1][c2];
            encrypted += keyMatrix[r2][c1];
        }
    }
    return encrypted;
}

// Function to decrypt ciphertext using Playfair Cipher
string playfairDecrypt(string ciphertext, string key) {
    generateKeyMatrix(key);
    string decrypted = "";

    for (int i = 0; i < ciphertext.length(); i += 2) {
        int r1, c1, r2, c2;
        findPosition(ciphertext[i], r1, c1);
        findPosition(ciphertext[i + 1], r2, c2);

        if (r1 == r2) {
            decrypted += keyMatrix[r1][(c1 + 4) % 5];
            decrypted += keyMatrix[r2][(c2 + 4) % 5];
        } else if (c1 == c2) {
            decrypted += keyMatrix[(r1 + 4) % 5][c1];
            decrypted += keyMatrix[(r2 + 4) % 5][c2];
        } else
            decrypted += keyMatrix[r1][c2];
            decrypted += keyMatrix[r2][c1];
        }
    }
    return decrypted;
}

int main() {
    cout<<"Enter the single word key: ";
    string key;
```

```
    cin>>key;

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

    cout << "Original Text: " << plaintext << endl;

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

    string decrypted = playfairDecrypt(encrypted, key);
    cout << "Decrypted Text: " << decrypted << endl;

    return 0;
}
```

c.

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

// Function to multiply matrices
vector<int> matrixMultiplication(vector<vector<int>> key, vector<int>
textVec, int n)
{
    vector<int> result(n);
    for (int i = 0; i < n; i++)
    {
        result[i] = 0;
        for (int j = 0; j < n; j++)
        {
            result[i] += key[i][j] * textVec[j];
        }
        result[i] = result[i] % 26;
    }
    return result;
}

// Function to encrypt using Hill cipher
string hillEncrypt(string message, vector<vector<int>> key)
{
    int n = key.size();
    vector<int> textVec(n);

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

```
    {
        textVec[i] = message[i] - 'A';
    }

    vector<int> cipherVec = matrixMultiplication(key, textVec, n);

    string cipherText = "";
    for (int i = 0; i < n; i++)
    {
        cipherText += cipherVec[i] + 'A';
    }

    return cipherText;
}

int main()
{
    string message = "ACT";
    vector<vector<int>> key = {{6, 24, 1}, {13, 16, 10}, {20, 17, 15}};

    cout << "Original Text: " << message << endl;
    string encrypted = hillEncrypt(message, key);
    cout << "Encrypted Text: " << encrypted << endl;

    return 0;
}
```

d.

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

// Function to generate key to match length of text
string generateKey(string text, string key)
{
    int x = text.size();
    for (int i = 0;; i++)
    {
        if (x == i)
            i = 0;
        if (key.size() == text.size())
            break;
        key.push_back(key[i]);
    }
    return key;
}
```

```
// Function to encrypt using Vigenere Cipher
string vigenereEncrypt(string text, string key)
{
    string encryptedText;
    for (int i = 0; i < text.size(); i++)
    {
        char x = (text[i] + key[i]) % 26;
        x += 'A';
        encryptedText.push_back(x);
    }
    return encryptedText;
}

// Function to decrypt using Vigenere Cipher
string vigenereDecrypt(string encryptedText, string key)
{
    string decryptedText;
    for (int i = 0; i < encryptedText.size(); i++)
    {
        char x = (encryptedText[i] - key[i] + 26) % 26;
        x += 'A';
        decryptedText.push_back(x);
    }
    return decryptedText;
}

int main()
{
    cout<<"Enter the text to be encrypted: ";
    string text;
    getline(cin>>ws, text);

    cout<<"Enter the key: ";
    string key;
    cin>>key;

    key = generateKey(text, key);
    string encrypted = vigenereEncrypt(text, key);
    cout << "Original Text: " << text << endl;
    cout << "Encrypted Text: " << encrypted << endl;
    cout << "Decrypted Text: " << vigenereDecrypt(encrypted, key) <<
endl;

    return 0;
}
```

Cryptography, Network and Security

Assignment 2

2. Perform encryption and decryption using following transposition techniques

a. Rail fence

b. row and Column Transformation

Code:

a.

```cpp
#include <iostream>
#include <string>
using namespace std;

// Function to encrypt using Rail Fence Cipher
string railFenceEncrypt(string plaintext, int rails)
{
    string matrix[rails];

    int row = 0;
    bool down = true;

    for (int i = 0; i < plaintext.length(); i++)
    {
        matrix[row].push_back(plaintext[i]);

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }
```

```
    string ciphertext = "";
    for (int i = 0; i < rails; i++)
    {
        ciphertext += matrix[i];
    }
    return ciphertext;
}

// Function to decrypt using Rail Fence Cipher
string railFenceDecrypt(string ciphertext, int rails)
{
    string matrix[rails];

    int length = ciphertext.length();
    int row = 0, index = 0;
    bool down = true;

    bool mark[length][rails];
    for (int i = 0; i < rails; i++)
        for (int j = 0; j < length; j++)
            mark[j][i] = false;

    for (int i = 0; i < length; i++)
    {
        mark[i][row] = true;

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }

    for (int i = 0; i < rails; i++)
    {
```

```
        for (int j = 0; j < length; j++)
        {
            if (mark[j][i])
            {
                matrix[i].push_back(ciphertext[index++]);
            }
        }
    }

    row = 0;
    down = true;
    string plaintext = "";
    for (int i = 0; i < length; i++)
    {
        plaintext += matrix[row][0];
        matrix[row].erase(matrix[row].begin());

        if (down)
        {
            row++;
            if (row == rails)
            {
                row = rails - 2;
                down = false;
            }
        }
        else
        {
            row--;
            if (row == -1)
            {
                row = 1;
                down = true;
            }
        }
    }

    return plaintext;
}

int main()
{
    cout<<"Enter the text to be encrypted: ";
    string plaintext;
    getline(cin, plaintext);

    int rails = 3;
```

```
    string encrypted = railFenceEncrypt(plaintext, rails);
    cout << "Encrypted Text (Rail Fence): " << encrypted << endl;

    string decrypted = railFenceDecrypt(encrypted, rails);
    cout << "Decrypted Text (Rail Fence): " << decrypted << endl;

    return 0;
}
```

b.

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

// Function to generate the column order based on the key
vector<int> getColumnOrder(string key)
{
    vector<pair<char, int>> keyWithIndices;
    for (int i = 0; i < key.size(); i++)
    {
        keyWithIndices.push_back({key[i], i});
    }

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

    vector<int> order;
    for (auto &p : keyWithIndices)
    {
        order.push_back(p.second);
    }

    return order;
}

// Function to encrypt using Row and Column Transformation Cipher
string rowColumnEncrypt(string plaintext, string key)
{
    int columns = key.size();
    int rows = (plaintext.size() + columns - 1) / columns;

    while (plaintext.size() < rows * columns)
    {
        plaintext += 'X';
    }
```

```
    vector<vector<char>> matrix(rows, vector<char>(columns));
    int index = 0;
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            matrix[i][j] = plaintext[index++];
        }
    }

    vector<int> columnOrder = getColumnOrder(key);

    string ciphertext = "";
    for (int col : columnOrder)
    {
        for (int i = 0; i < rows; i++)
        {
            ciphertext += matrix[i][col];
        }
    }
    return ciphertext;
}

// Function to decrypt using Row and Column Transformation Cipher
string rowColumnDecrypt(string ciphertext, string key)
{
    int columns = key.size();
    int rows = (ciphertext.size() + columns - 1) / columns;

    vector<vector<char>> matrix(rows, vector<char>(columns));

    vector<int> columnOrder = getColumnOrder(key);

    int index = 0;
    for (int col : columnOrder)
    {
        for (int i = 0; i < rows; i++)
        {
            matrix[i][col] = ciphertext[index++];
        }
    }

    string plaintext = "";
    for (int i = 0; i < rows; i++)
    {
        for (int j = 0; j < columns; j++)
        {
            plaintext += matrix[i][j];
```

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

    return plaintext;
}

int main()
{
    string plaintext;
    cout << "Enter the text to be encrypted: ";
    getline(cin>>ws, plaintext);
    string key;
    cout << "Enter the key: ";
    cin >> key;

    string encrypted = rowColumnEncrypt(plaintext, key);
    cout << "Encrypted Text (Row-Column): " << encrypted << endl;

    string decrypted = rowColumnDecrypt(encrypted, key);
    cout << "Decrypted Text (Row-Column): " << decrypted << endl;

    return 0;
}
```

Cryptography, Network and Security

Assignment 3

3. Implementation of Euclidean and Extended Euclidean Algorithm

Code:

```cpp
#include <iostream>
#include <string>
using namespace std;

// Euclidean Algorithm to find GCD
int euclideanGCD(int a, int b)
{
    while (b != 0)
    {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// Extended Euclidean Algorithm to find GCD and coefficients x, y
int extendedEuclidean(int a, int b, int &x, int &y)
{
    if (b == 0)
    {
        x = 1;
        y = 0;
        return a;
    }

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

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

    return gcd;
}

int main()
{
    int a, b;

    cout << "Enter two numbers to find GCD and coefficients: ";
    cin >> a >> b;
```

```
    int gcd = euclideanGCD(a, b);
    cout << "GCD using Euclidean Algorithm: " << gcd << endl;

    int x, y;
    int gcd_ext = extendedEuclidean(a, b, x, y);
    cout << "GCD using Extended Euclidean Algorithm: " << gcd_ext <<
endl;
    cout << "Coefficients: x = " << x << ", y = " << y << endl;

    return 0;
}
```

Cryptography, Network and Security

Assignment 4

4. Implementation of Chinese Remainder Theorem (CRT)

Code:

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

// Function to compute the GCD and the coefficients x and y for the
equation ax + by = gcd(a, b)
int extendedEuclidean(int a, int b, int &x, int &y) {
    if (b == 0) {
        x = 1;
        y = 0;
        return a;
    }
    int x1, y1;
    int gcd = extendedEuclidean(b, a % b, x1, y1);
    x = y1;
    y = x1 - (a / b) * y1;
    return gcd;
}

// Function to find modular inverse using Extended Euclidean Algorithm
int modInverse(int a, int m) {
    int x, y;
    int g = extendedEuclidean(a, m, x, y);
    if (g != 1) {
        cout << "Inverse doesn't exist!";
        return -1;
    } else {
        return (x % m + m) % m;
    }
}

// Function to solve the system of congruences using the Chinese
Remainder Theorem
int chineseRemainder(vector<int> num, vector<int> rem, int n) {
    int prod = 1;
    for (int i = 0; i < n; i++) {
        prod *= num[i];
    }

    int result = 0;
```

```
    for (int i = 0; i < n; i++) {
        int pp = prod / num[i];
        int inv = modInverse(pp, num[i]);
        result += rem[i] * inv * pp;
    }

    return result % prod;
}

int main() {
    // System of equations:
    // x ≡ rem[0] (mod num[0])
    // x ≡ rem[1] (mod num[1])
    // x ≡ rem[2] (mod num[2])

    vector<int> num = {3, 4, 5};
    vector<int> rem = {2, 3, 1};
    int n = num.size();

    int result = chineseRemainder(num, rem, n);
    cout << "x is " << result << endl;

    return 0;
}
```

Cryptography, Network and Security

Assignment 5

Apply DES algorithm for practical applications

Code:

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

using namespace std;

// Define the initial permutation table
int initial_permutation[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};

// Define the final permutation table
int final_permutation[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};

// Dummy function for round function and key schedule (simplified for
demonstration)
bitset<32> round_function(bitset<32> right, bitset<48> key)
{
    return right ^ bitset<32>(key.to_string().substr(0, 32));
}

// DES encryption function
bitset<64> DES_encrypt(bitset<64> plaintext, bitset<64> key)
{
    bitset<64> permuted_text;

    for (int i = 0; i < 64; i++)
    {
        permuted_text[63 - i] = plaintext[64 - initial_permutation[i]];
    }

    bitset<32> left = permuted_text.to_ullong() >> 32;
    bitset<32> right = permuted_text.to_ullong();

    bitset<48> round_key = bitset<48>(key.to_string().substr(0, 48));
```

```
    for (int i = 0; i < 2; i++)
    {
        bitset<32> new_right = left ^ round_function(right, round_key);
        left = right;
        right = new_right;
    }

    bitset<64> combined((left.to_ullong() << 32) | right.to_ullong());
    bitset<64> ciphertext;
    for (int i = 0; i < 64; i++)
    {
        ciphertext[63 - i] = combined[64 - final_permutation[i]];
    }

    return ciphertext;
}

int main()
{
    bitset<64>
plaintext(string("0000000100100011010001010110011111000100110101011110011
111101111"));
    bitset<64>
key(string("0001001100110100010101110111100110011011101111001101111111110
001"));

    bitset<64> ciphertext = DES_encrypt(plaintext, key);

    cout << "Ciphertext: " << ciphertext << endl;
    return 0;
}
```

Cryptography, Network and Security

Assignment 6

Apply AES algorithm for practical applications

Code:

```
#include <iostream>
#include <iomanip>
#include <cstdint>

using namespace std;

const int Nb = 4;
const int Nk = 4;
const int Nr = 10;

uint8_t s_box[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};

// Rijndael's Galois field multiplication for MixColumns
uint8_t gmul(uint8_t a, uint8_t b)
{
    uint8_t p = 0;
    uint8_t hi_bit_set;
    for (int i = 0; i < 8; i++)
    {
        if (b & 1)
        {
            p ^= a;
        }
        hi_bit_set = a & 0x80;
        a <<= 1;
        if (hi_bit_set)
        {
            a ^= 0x1b;
        }
        b >>= 1;
    }
    return p;
}

// Function to substitute bytes using the S-box
void subBytes(uint8_t state[4][4])
```

```
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            state[i][j] = s_box[state[i][j]];
        }
    }
}

// Function to shift rows in the state
void shiftRows(uint8_t state[4][4])
{
    uint8_t temp;

    temp = state[1][0];
    for (int i = 0; i < 3; i++)
    {
        state[1][i] = state[1][i + 1];
    }
    state[1][3] = temp;

    temp = state[2][0];
    state[2][0] = state[2][2];
    state[2][2] = temp;
    temp = state[2][1];
    state[2][1] = state[2][3];
    state[2][3] = temp;

    temp = state[3][3];
    for (int i = 3; i > 0; i--)
    {
        state[3][i] = state[3][i - 1];
    }
    state[3][0] = temp;
}

// Function to mix columns in the state matrix
void mixColumns(uint8_t state[4][4])
{
    uint8_t temp[4];
    for (int i = 0; i < 4; i++)
    {
        temp[0] = gmul(state[0][i], 2) ^ gmul(state[1][i], 3) ^
state[2][i] ^ state[3][i];
        temp[1] = state[0][i] ^ gmul(state[1][i], 2) ^ gmul(state[2][i],
3) ^ state[3][i];
```

```
        temp[2] = state[0][i] ^ state[1][i] ^ gmul(state[2][i], 2) ^
gmul(state[3][i], 3);
        temp[3] = gmul(state[0][i], 3) ^ state[1][i] ^ state[2][i] ^
gmul(state[3][i], 2);

        state[0][i] = temp[0];
        state[1][i] = temp[1];
        state[2][i] = temp[2];
        state[3][i] = temp[3];
    }
}

// Function to add round key to state
void addRoundKey(uint8_t state[4][4], uint8_t roundKey[4][4])
{
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            state[i][j] ^= roundKey[i][j];
        }
    }
}

// Simplified key expansion and encryption example
void AES_encrypt(uint8_t input[16], uint8_t key[16])
{
    uint8_t state[4][4];
    uint8_t roundKey[4][4];

    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            state[j][i] = input[i * 4 + j];
        }
    }

    for (int i = 0; i < 16; i++)
    {
        roundKey[i % 4][i / 4] = key[i];
    }
    addRoundKey(state, roundKey);

    for (int round = 0; round < 9; round++)
    {
        subBytes(state);
        shiftRows(state);
```

```
        mixColumns(state);
        addRoundKey(state, roundKey);
    }

    subBytes(state);
    shiftRows(state);
    addRoundKey(state, roundKey);

    cout << "Ciphertext: ";
    for (int i = 0; i < 4; i++)
    {
        for (int j = 0; j < 4; j++)
        {
            cout << hex << setw(2) << setfill('0') << (int)state[j][i] <<
" ";
        }
    }
    cout << endl;
}

int main()
{
    uint8_t plaintext[16] = {0x32, 0x43, 0xf6, 0xa8, 0x88, 0x5a, 0x30,
0x8d, 0x31, 0x31, 0x98, 0xa2, 0xe0, 0x37, 0x07, 0x34};
    uint8_t key[16] = {0x2b, 0x7e, 0x15, 0x16, 0x28, 0xae, 0xd2, 0xa6,
0xab, 0xf7, 0xcf, 0x15, 0x88, 0x09, 0xcf, 0x4f};

    AES_encrypt(plaintext, key);

    return 0;
}
```