DELHI TECHNOLOGICAL UNIVERSITY

CS305 –: Information Network & Security Lab File

Submitted by:
YUG BATHLA
23/CS/479
CSE1 (A1) (G3)

Submitted to:

INDEX

S.NO	EXPERIMENT	DATE
1	To implement Caesar cipher encryption Encryption: Replace each plaintext letter with one a Fixed number of places down the alphabet. Decryption: Replace each cipher text letter with one a fixed number of places up the alphabet.	7/8/25
2	To implement Monoalphabetic decryption. Encrypting and Decrypting works exactly the same for all monoalphabetic ciphers. Encryption/Decryption: Every letter in the alphabet is represented by exactly one other letter in the key.	14/8/25
3	To implement Play fair cipher encryption-decryption.	21/8/25
4	To implement Polyalphabetic cipher encryption decryption. Encryption/Decryption: Based on substitution, using multiple substitution Alphabets	28/8/25
5	To implement Hill- cipher encryption decryption	4/9/25
6	To implement S-DES sub key Generation	11/9/25
7	To implement Diffie-hallman key exchange algorithm	9/10/25
8	To implement RSA encryption-decryption.	
9	Write a program to generate SHA-1 hash.	
10	Implement a digital signature algorithm	

EXPERIMENT – 1

Aim: To implement Caesar cipher encryption.
Encryption: Replace each plaintext letter with one a Fixed number of places down the alphabet.
Decryption: Replace each cipher text letter with one a fixed number of places up the alphabet.

Theory:

The **Caesar cipher** is one of the simplest and oldest **encryption techniques**, named after Julius Caesar, who used it in his private correspondence. It is a **substitution cipher** in which each letter in the plaintext is shifted a fixed number of positions down or up the alphabet.

1. **Encryption:**
   - Each letter of the plaintext is replaced by a letter located **a fixed number of positions down the alphabet**.
   - The fixed number is called the **key** or **shift**.
   - For example, with a shift of 3:
   - `Plaintext:  A B C D E ...`
   - `Ciphertext: D E F G H ...`

   Encryption Formula:

   ```
   C = (P + K) mod 26
   ```

   - `C` = Ciphertext letter (numeric value 0–25)
   - `P` = Plaintext letter (numeric value 0–25)
   - `K` = Key (shift value)
2. **Decryption:**
   - To retrieve the original text, each letter in the ciphertext is shifted **up by the same key value**.
   - **Decryption Formula:** `P = (C - K + 26) mod 26`
   - `P` = Original plaintext letter
   - `C` = Ciphertext letter
   - `K` = Key (shift value)

Characteristics

- **Symmetric Cipher:**
  The same key is used for encryption and decryption.
- **Alphabetic Substitution:**
  Only letters are substituted; spaces and punctuation may remain unchanged.
- **Fixed Shift:**
  All letters are shifted by the same amount.
-

Source code:

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

// Encrypt a message using Caesar Cipher
string encrypt(string text, int shift) {
    string result = "";

    for (char c : text) {
        if (isupper(c)) {
            result += char((c - 'A' + shift) % 26 + 'A');
        } else if (islower(c)) {
            result += char((c - 'a' + shift) % 26 + 'a');
        } else {
            result += c; // leave non-alphabet characters unchanged
        }
    }

    return result;
}

// Decrypt a message using Caesar Cipher
string decrypt(string text, int shift) {
    string result = "";

    for (char c : text) {
        if (isupper(c)) {
            result += char((c - 'A' - shift + 26) % 26 + 'A');
        } else if (islower(c)) {
            result += char((c - 'a' - shift + 26) % 26 + 'a');
        } else {
            result += c;
        }
    }

    return result;
}

int main() {
    string text;
    int shift, choice;

    cout << "Caesar Cipher\n";
    cout << "1. Encrypt\n2. Decrypt\nChoose (1 or 2): ";
    cin >> choice;
    cin.ignore(); // ignore newline character left in input buffer

    cout << "Enter text: ";
    getline(cin, text);

    cout << "Enter shift value (e.g., 3): ";
```

```
    cin >> shift;

    if (choice == 1) {
        cout << "Encrypted Text: " << encrypt(text, shift) << endl;
    } else if (choice == 2) {
        cout << "Decrypted Text: " << decrypt(text, shift) << endl;
    } else {
        cout << "Invalid choice.\n";
    }

    return 0;
}
```

Output:

```
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding stuff/ins lab
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Caesar Cipher
1. Encrypt
2. Decrypt
Choose (1 or 2): 1
Enter text: i am happy
Enter shift value (e.g., 3): 3
Encrypted Text: l dp kdssb
```

```
yug@yugs-MacBook-Air ins lab  % cd "/Use
&& "/Users/yug/coding stuff/ins lab /"te
Caesar Cipher
1. Encrypt
2. Decrypt
Choose (1 or 2): 2
Enter text: l dp kdssb
Enter shift value (e.g., 3): 3
Decrypted Text: i am happy
```

EXPERIMENT – 2

Aim: To implement Monoalphabetic decryption. Encrypting and Decrypting works exactly the same for all monoalphabetic ciphers.
Encryption/Decryption: Every letter in the alphabet is represented by exactly one other letter in the key.

Theory:

A **Monoalphabetic Cipher** is a **substitution cipher** in which each letter of the plaintext is replaced by exactly **one unique letter** of the alphabet according to a **fixed key**. Unlike Caesar cipher, the shift doesn't have to be uniform; any mapping between plaintext and ciphertext letters is allowed.

Working Principle

1. **Encryption:**
    - Each plaintext letter is replaced by its corresponding letter in the key.
    - Example Key Mapping:
    - `Plain:  A B C D E F G H I J ...`
    - `Cipher: Q W E R T Y U I O P ...`
    - Plaintext "HELLO" → Ciphertext "ITSSG"
2. **Decryption:**
    - To decrypt, each ciphertext letter is replaced with its corresponding plaintext letter using the **reverse mapping**.

Characteristics

- **Fixed substitution:** Each letter has exactly **one corresponding ciphertext letter**.
- **Symmetric key:** Same key is required for encryption and decryption.
- **Alphabet-only substitution:** Typically, only letters are encrypted; numbers and punctuation remain unchanged.

Advantages

- Simple to implement and understand.
- Offers more variability than Caesar cipher.

Limitations

- Vulnerable to **frequency analysis**, as the mapping is static.
- Less secure for large texts without additional techniques.

Applications

- Used historically for confidential messages.
- Educational purposes for learning basic cryptography.

Source code:

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

// Function to build mapping from standard to key alphabet
unordered_map<char, char> buildMap(const string& from, const string& to) {
    unordered_map<char, char> map;
    for (int i = 0; i < 26; ++i) {
        map[from[i]] = to[i];
    }
    return map;
}

// Function to encrypt or decrypt text
string monoalphabeticCipher(const string& text, const unordered_map<char, char>&
map) {
    string result = "";
    for (char c : text) {
        if (isupper(c)) {
            result += toupper(map.at(tolower(c)));
        } else if (islower(c)) {
            result += map.at(c);
        } else {
            result += c; // Keep non-alphabetic characters unchanged
        }
    }
    return result;
}

int main() {
    string plainAlphabet = "abcdefghijklmnopqrstuvwxyz";

    // Monoalphabetic key (must be a permutation of 26 unique letters)
    string keyAlphabet =     "qwertyuiopasdfghjklzxcvbnm"; // key

    // Build encrypt and decrypt maps
    unordered_map<char, char> encryptMap = buildMap(plainAlphabet, keyAlphabet);
    unordered_map<char, char> decryptMap = buildMap(keyAlphabet, plainAlphabet);

    int choice;
    string input;

    cout << "Monoalphabetic Cipher\n";
    cout << "1. Encrypt\n2. Decrypt\nChoose (1 or 2): ";
    cin >> choice;
```

```
    cin.ignore(); // flush newline

    cout << "Enter text: ";
    getline(cin, input);

    if (choice == 1) {
        cout << "Encrypted Text: " << monoalphabeticCipher(input, encryptMap) <<
endl;
    } else if (choice == 2) {
        cout << "Decrypted Text: " << monoalphabeticCipher(input, decryptMap) <<
endl;
    } else {
        cout << "Invalid choice." << endl;
    }

    return 0;
}
```

Output:

```
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding stu
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Monoalphabetic Cipher
1. Encrypt
2. Decrypt
Choose (1 or 2): 1
Enter text: abcdefghijk
Encrypted Text: qwertyuiopa
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding stu
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Monoalphabetic Cipher
1. Encrypt
2. Decrypt
Choose (1 or 2): 2
Enter text: qwertyuiop
Decrypted Text: abcdefghij
```

EXPERIMENT – 3

Aim: To implement Play fair cipher encryption-decryption.

Theory:

The **Playfair Cipher** is a **digraph substitution cipher** invented by Charles Wheatstone (and promoted by Lord Playfair) in 1854.

- Instead of encrypting single letters, **two letters at a time (digraphs)** are encrypted, making it more secure than monoalphabetic ciphers.

Working Principle

1. **Key Table Creation**
   - A **5×5 matrix** is filled with letters of a key (replacing J with I), followed by the remaining letters of the alphabet in order.
   - Example:
   - K E Y W O
   - R D A B C
   - F G H I L
   - M N P Q S
   - T U V X Z
2. **Encryption Rules**
   - Break plaintext into digraphs (pairs of letters). If a digraph has identical letters, insert a filler (like X).
   - For each digraph:
     1. **Same row:** Replace each letter with the letter to its **right** (wrap around at end).
     2. **Same column:** Replace each letter with the letter **below** (wrap around at bottom).
     3. **Rectangle:** Replace each letter with the letter in the **same row but the column of the other letter**.
3. **Decryption Rules**
   - Reverse the encryption rules:
     1. **Same row:** Letter to **left**.
     2. **Same column:** Letter **above**.
     3. **Rectangle:** Swap columns as in encryption.

Characteristics

- Encrypts **pairs of letters (digraphs)** instead of single letters.
- Provides better **security** than monoalphabetic cipher.
- Uses a **5×5 key table**, combining I/J into a single letter.

Advantages

- Harder to break using simple frequency analysis.
- Encrypts more than one letter at a time, improving security.

Limitations

- Slightly complex compared to monoalphabetic cipher.
- Still vulnerable to **modern cryptanalysis**.

Applications

- Historically used for military communication.
- Good for **educational purposes** to demonstrate digraph encryption.

Source code:

```
#include <iostream>
#include <vector>
#include <string>
#include <map>
#include <cctype>

using namespace std;

class PlayfairCipher {
    vector<vector<char>> keyTable;
    map<char, pair<int, int>> pos; // letter -> (row, col)

public:
    PlayfairCipher(string key) {
        createKeyTable(key);
    }

    void createKeyTable(string key) {
        vector<bool> used(26, false);
        keyTable.assign(5, vector<char>(5, ' '));
        string filteredKey;

        // Uppercase, replace J with I, remove duplicates
        for (char c : key) {
            c = toupper(static_cast<unsigned char>(c));
            if (c == 'J') c = 'I';
            if (c < 'A' || c > 'Z') continue;
            if (!used[c - 'A']) {
                used[c - 'A'] = true;
                filteredKey.push_back(c);
            }
        }

        // Add remaining letters
        for (char c = 'A'; c <= 'Z'; c++) {
            if (c == 'J') continue;
```

```
        if (!used[c - 'A']) {
            used[c - 'A'] = true;
            filteredKey.push_back(c);
        }
    }

    // Fill 5x5 table
    int idx = 0;
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 5; j++) {
            keyTable[i][j] = filteredKey[idx];
            pos[filteredKey[idx]] = {i, j};
            idx++;
        }
    }
}

string prepareText(string text, bool forEncryption) {
    string processed;
    for (char c : text) {
        c = toupper(static_cast<unsigned char>(c));
        if (c == 'J') c = 'I';
        if (c >= 'A' && c <= 'Z') processed.push_back(c);
    }

    if (forEncryption) {
        string result;
        for (size_t i = 0; i < processed.size(); i++) {
            result.push_back(processed[i]);
            if (i + 1 == processed.size()) {
                result.push_back('X'); // padding
            } else if (processed[i] == processed[i + 1]) {
                result.push_back('X');
            } else {
                result.push_back(processed[i + 1]);
                i++;
            }
        }
        return result;
    }
    return processed; // For decryption, no digraph processing needed
}

string encrypt(string plaintext) {
    string text = prepareText(plaintext, true);
    string cipher;
    for (size_t i = 0; i < text.size(); i += 2) {
        char a = text[i], b = text[i + 1];
        pair<int, int> p1 = pos[a];
        pair<int, int> p2 = pos[b];
        int r1 = p1.first, c1 = p1.second;
```

```
            int r2 = p2.first, c2 = p2.second;


            if (r1 == r2) { // Same row
                cipher.push_back(keyTable[r1][(c1 + 1) % 5]);
                cipher.push_back(keyTable[r2][(c2 + 1) % 5]);
            } else if (c1 == c2) { // Same column
                cipher.push_back(keyTable[(r1 + 1) % 5][c1]);
                cipher.push_back(keyTable[(r2 + 1) % 5][c2]);
            } else { // Rectangle
                cipher.push_back(keyTable[r1][c2]);
                cipher.push_back(keyTable[r2][c1]);
            }
        }
        return cipher;
    }

    string decrypt(string ciphertext) {
        string text = prepareText(ciphertext, false);
        string plain;
        for (size_t i = 0; i < text.size(); i += 2) {
            char a = text[i], b = text[i + 1];
            auto [r1, c1] = pos[a];
            auto [r2, c2] = pos[b];

            if (r1 == r2) { // Same row
                plain.push_back(keyTable[r1][(c1 + 4) % 5]);
                plain.push_back(keyTable[r2][(c2 + 4) % 5]);
            } else if (c1 == c2) { // Same column
                plain.push_back(keyTable[(r1 + 4) % 5][c1]);
                plain.push_back(keyTable[(r2 + 4) % 5][c2]);
            } else { // Rectangle
                plain.push_back(keyTable[r1][c2]);
                plain.push_back(keyTable[r2][c1]);
            }
        }
        return plain;
    }

    void printKeyTable() {
        for (auto &row : keyTable) {
            for (char c : row) cout << c << ' ';
            cout << "\n";
        }
    }
};

int main() {
    string key, plaintext;

    cout << "Enter key: ";
```

```
    getline(cin, key);

    PlayfairCipher cipher(key);

    cout << "\nKey Table:\n";
    cipher.printKeyTable();

    cout << "\nEnter plaintext: ";
    getline(cin, plaintext);

    string encrypted = cipher.encrypt(plaintext);
    string decrypted = cipher.decrypt(encrypted);

    cout << "\nPlaintext:  " << plaintext;
    cout << "\nEncrypted:  " << encrypted;
    cout << "\nDecrypted:  " << decrypted << "\n";

    return 0;
}
```

Output:

```
Enter key: KEYWORD

Key Table:
K E Y W O
R D A B C
F G H I L
M N P Q S
T U V X Z

Enter plaintext: HELLO WORLD

Plaintext:  HELLO WORLD
Encrypted:  GYIZSCOKCFBU
Decrypted:  HELXLOWORLDX
yug@yugs-MacBook-Air ins lab  %
```

EXPERIMENT – 4

Aim: To implement Polyalphabetic cipher encryption decryption.
Encryption/Decryption: Based on substitution, using multiple substitution Alphabets

Theory:

A **polyalphabetic cipher** uses **multiple substitution alphabets** to encrypt plaintext, unlike monoalphabetic ciphers which use a single substitution.

- Famous example: **Vigenère cipher**.

Working Principle

1. A **key (sequence of letters)** determines which substitution alphabet to use.
2. Each letter of plaintext is encrypted using a **different Caesar shift** according to the key.
3. Encryption formula:
4. `C_i = (P_i + K_i) mod 26`
    - `C_i` = Ciphertext letter
    - `P_i` = Plaintext letter
    - `K_i` = Key letter corresponding to that position
5. Decryption formula:
6. `P_i = (C_i - K_i + 26) mod 26`

Characteristics

- Uses **multiple Caesar shifts**, making frequency analysis harder.
- Symmetric key cipher.
- Plaintext letters are **spread over multiple alphabets**.

Advantages

- More secure than monoalphabetic substitution.
- Reduces predictability of letter frequencies.

Applications

- Vigenère cipher for secure communication.
- Educational demonstration of **polyalphabetic substitution**.

Source code:

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

// Function to generate repeating key (ignores non-alphabet characters)
string generateKey(const string &text, const string &key) {
    string newKey;
    int j = 0; // index for key
    for (size_t i = 0; i < text.size(); i++) {
        if (isalpha(text[i])) {
            newKey.push_back(key[j % key.size()]);
            j++;
        } else {
            newKey.push_back(text[i]); // keep spaces/punctuation aligned
        }
    }
    return newKey;
}

// Encrypt function
string encryptText(const string &text, const string &key) {
    string cipher_text;
    for (size_t i = 0; i < text.size(); i++) {
        if (isupper(text[i])) {
            char x = ((text[i] - 'A') + (toupper(key[i]) - 'A')) % 26 + 'A';
            cipher_text.push_back(x);
        } else if (islower(text[i])) {
            char x = ((text[i] - 'a') + (tolower(key[i]) - 'a')) % 26 + 'a';
            cipher_text.push_back(x);
        } else {
            cipher_text.push_back(text[i]); // leave spaces/punctuation
        }
    }
    return cipher_text;
}

// Decrypt function
string decryptText(const string &cipher_text, const string &key) {
    string orig_text;
    for (size_t i = 0; i < cipher_text.size(); i++) {
        if (isupper(cipher_text[i])) {
            char x = ((cipher_text[i] - 'A') - (toupper(key[i]) - 'A') + 26) % 26 +
'A';
            orig_text.push_back(x);
        } else if (islower(cipher_text[i])) {
            char x = ((cipher_text[i] - 'a') - (tolower(key[i]) - 'a') + 26) % 26 +
'a';
            orig_text.push_back(x);
```

```
        } else {
            orig_text.push_back(cipher_text[i]);
        }
    }
    return orig_text;
}

int main() {
    string text, keyword;

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

    cout << "Enter key (letters only): ";
    cin >> keyword;

    string key = generateKey(text, keyword);
    string cipher_text = encryptText(text, key);

    cout << "\nEncrypted Text : " << cipher_text << endl;
    cout << "Decrypted Text : " << decryptText(cipher_text, key) << endl;

    return 0;
}
```

Output:

```
yug@yugs-MacBook-Air ins lab  % cd "/Users/
&& "/Users/yug/coding stuff/ins lab /"tempC
Enter plaintext: HELLO WORLD HIIII
Enter key (letters only): QWERTYUIOPSDASLKJ

Encrypted Text : XAPCH UIZZS ZLIAT
Decrypted Text : HELLO WORLD HIIII
```

EXPERIMENT – 5

Aim: To implement Hill- cipher encryption decryption

Theory:

The **Hill cipher** is a **polygraphic substitution cipher** based on **linear algebra**.

- Encrypts plaintext in blocks (vectors) of size **n** using an **n×n key matrix**.
- Developed by Lester Hill in 1929.

Working Principle

1. Represent plaintext letters as **numbers (A=0, B=1, … Z=25)**.
2. Divide plaintext into blocks of size **n**.
3. Encryption formula:
4. `C = (K * P) mod 26`
    - `C` = Ciphertext vector
    - `K` = Key matrix (invertible modulo 26)
    - `P` = Plaintext vector
5. Decryption formula:
6. `P = (K^-1 * C) mod 26`
    - `K^-1` = Modular inverse of key matrix modulo 26

Characteristics

- **Polygraphic cipher**: Encrypts multiple letters at once.
- Requires **invertible key matrix**.
- Provides better security than monoalphabetic ciphers.

Applications

- Classical cryptography education.
- Demonstrates **linear algebra in encryption**.

Source code :

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

// Function to get modulo inverse of a number under mod 26
int modInverse(int a, int m) {
    a = a % m;
    for (int x = 1; x < m; x++) {
        if ((a * x) % m == 1) return x;
    }
```

```
    return -1;
}

// Function to get determinant of matrix (only for 2x2 and 3x3)
int determinant(const vector<vector<int>> &mat, int n) {
    if (n == 2)
        return (mat[0][0]*mat[1][1] - mat[0][1]*mat[1][0]);
    else if (n == 3)
        return (mat[0][0]*(mat[1][1]*mat[2][2] - mat[1][2]*mat[2][1])
                - mat[0][1]*(mat[1][0]*mat[2][2] - mat[1][2]*mat[2][0])
                + mat[0][2]*(mat[1][0]*mat[2][1] - mat[1][1]*mat[2][0]));
    return 0;
}

// Function to get adjoint of 2x2 or 3x3 matrix
vector<vector<int>> adjoint(const vector<vector<int>> &mat, int n) {
    vector<vector<int>> adj(n, vector<int>(n));
    if (n == 2) {
        adj[0][0] = mat[1][1];
        adj[0][1] = -mat[0][1];
        adj[1][0] = -mat[1][0];
        adj[1][1] = mat[0][0];
    } else if (n == 3) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                vector<vector<int>> temp(2, vector<int>(2));
                int r = 0;
                for (int k = 0; k < n; k++) {
                    if (k == i) continue;
                    int c = 0;
                    for (int l = 0; l < n; l++) {
                        if (l == j) continue;
                        temp[r][c] = mat[k][l];
                        c++;
                    }
                    r++;
                }
                adj[j][i] = ((temp[0][0]*temp[1][1] - temp[0][1]*temp[1][0]) *
(((i+j)%2==0)?1:-1));
            }
        }
    }
    return adj;
}

// Function to get inverse of matrix mod 26
vector<vector<int>> inverseMatrix(const vector<vector<int>> &mat, int n) {
    int det = determinant(mat, n);
    det = (det % 26 + 26) % 26;
    int invDet = modInverse(det, 26);
    if (invDet == -1) {
```

```
        throw runtime_error("Key matrix is not invertible mod 26");
    }
    vector<vector<int>> adj = adjoint(mat, n);
    vector<vector<int>> inv(n, vector<int>(n));
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            int val = adj[i][j] * invDet;
            val = (val % 26 + 26) % 26;
            inv[i][j] = val;
        }
    }
    return inv;
}

// Function to multiply matrix and vector mod 26
vector<int> multiply(const vector<vector<int>> &mat, const vector<int> &vec, int n)
{
    vector<int> res(n);
    for (int i = 0; i < n; i++) {
        int sum = 0;
        for (int j = 0; j < n; j++) {
            sum += mat[i][j] * vec[j];
        }
        res[i] = (sum % 26 + 26) % 26;
    }
    return res;
}

string processText(const string &text, int n) {
    string t = text;
    t.erase(remove_if(t.begin(), t.end(), [](char c){ return !isalpha(c); }),
t.end());
    transform(t.begin(), t.end(), t.begin(), ::toupper);
    while (t.size() % n != 0) t += 'X';
    return t;
}

string encrypt(const string &plaintext, const vector<vector<int>> &key, int n) {
    string pt = processText(plaintext, n);
    string ct;
    for (size_t i = 0; i < pt.size(); i += n) {
        vector<int> block(n);
        for (int j = 0; j < n; j++) block[j] = pt[i+j] - 'A';
        vector<int> enc = multiply(key, block, n);
        for (int j = 0; j < n; j++) ct += (enc[j] + 'A');
    }
    return ct;
}

string decrypt(const string &ciphertext, const vector<vector<int>> &key, int n) {
    vector<vector<int>> invKey = inverseMatrix(key, n);
```

```
    string ct = processText(ciphertext, n);
    string pt;
    for (size_t i = 0; i < ct.size(); i += n) {
        vector<int> block(n);
        for (int j = 0; j < n; j++) block[j] = ct[i+j] - 'A';
        vector<int> dec = multiply(invKey, block, n);
        for (int j = 0; j < n; j++) pt += (dec[j] + 'A');
    }
    return pt;
}

int main() {
    int n;
    cout << "Enter size of key matrix (2 or 3): ";
    cin >> n;
    if (n != 2 && n != 3) {
        cout << "Only 2x2 or 3x3 matrices supported.\n";
        return 1;
    }
    vector<vector<int>> key(n, vector<int>(n));
    cout << "Enter key matrix (row-wise):\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> key[i][j];
    cin.ignore();
    string text;
    cout << "Enter text: ";
    getline(cin, text);
    int choice;
    cout << "1. Encrypt\n2. Decrypt\nEnter choice: ";
    cin >> choice;
    try {
        if (choice == 1) {
            string ct = encrypt(text, key, n);
            cout << "Encrypted text: " << ct << endl;
        } else if (choice == 2) {
            string pt = decrypt(text, key, n);
            cout << "Decrypted text: " << pt << endl;
        } else {
            cout << "Invalid choice.\n";
        }
    } catch (const exception &e) {
        cout << "Error: " << e.what() << endl;
    }
    return 0;
}
```

Output:

```
yug@yugs-MacBook-Air ins lab  % cd "/Users/
ins lab /"exp5
Enter size of key matrix (2 or 3): 3
Enter key matrix (row-wise):
1 1 1 1 1 1 1 1 1
Enter text: HELLO WORLD
1. Encrypt
2. Decrypt
Enter choice: 1
Encrypted text: WWWVVVQQQXXX
yug@yugs-MacBook-Air ins lab  % cd "/Users/
ins lab /"exp5
Enter size of key matrix (2 or 3): 3
Enter key matrix (row-wise):
1 1 1 1 1 1 1 1 1
Enter text: WWWVVVQQQXXX
1. Encrypt
2. Decrypt
Enter choice: 2
Error: Key matrix is not invertible mod 26
yug@yugs-MacBook-Air ins lab  %
```

EXPERIMENT – 6

Aim: To implement S-DES sub key Generation

Theory:

Simplified Data Encryption Standard (**S-DES**) is a simplified version of the DES encryption algorithm designed for **educational purposes**.

- S-DES operates on **8-bit plaintext** using a **10-bit key**.
- It uses **two 8-bit subkeys (K1 and K2)** generated from the original key.

Key Generation Process

1. **Input 10-bit key**.
2. **Permutation P10**: Rearrange the 10 bits according to a fixed permutation.
3. **Split into two halves**: Left (L) and Right (R), each 5 bits.
4. **Left shift (LS-1)**: Circular left shift on each half.
5. **Permutation P8**: Select 8 bits from the shifted halves to form **K1**.
6. **Left shift (LS-2)**: Circular left shift by 2 bits on each half.
7. **Permutation P8**: Form **K2**.

Characteristics

- Generates **two subkeys** for the encryption rounds.
- Used in the S-DES **Feistel structure** for encryption/decryption.

Applications

- Educational purposes to demonstrate **key scheduling** and Feistel encryption.

Source code:

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

using namespace std;

// Function to permute the key based on the given permutation table
string permute(const string& key, const vector<int>& table) {
    string permutedKey;
    for (int i : table) {
        permutedKey += key[i - 1]; // -1 for zero-based indexing
    }
    return permutedKey;
}

// Function to perform left shift on the given bits
```

```cpp
string leftShift(const string& bits) {
    return bits.substr(1) + bits[0]; // Shift left
}

// Function to generate subkeys K1 and K2 from the given key
void generateSubkeys(const string& key, string& K1, string& K2) {
    // P10 and P8 permutation tables
    vector<int> P10 = {3, 5, 2, 7, 4, 10, 1, 9, 8, 6};
    vector<int> P8 = {6, 3, 7, 4, 8, 5, 10, 9};

    // Step 1: Permute the key using P10
    string permutedKey = permute(key, P10);

    // Step 2: Split the key into two halves
    string leftHalf = permutedKey.substr(0, 5);
    string rightHalf = permutedKey.substr(5, 5);

    // Step 3: Generate K1
    leftHalf = leftShift(leftHalf);
    rightHalf = leftShift(rightHalf);
    K1 = permute(leftHalf + rightHalf, P8);

    // Step 4: Generate K2
    leftHalf = leftShift(leftHalf);
    rightHalf = leftShift(rightHalf);
    K2 = permute(leftHalf + rightHalf, P8);
}

int main() {
    string key;

    // Input 10-bit binary key
    cout << "Enter a 10-bit binary key: ";
    cin >> key;

    // Validate key length
    if (key.length() != 10) {
        cout << "Error: Key must be exactly 10 bits long." << endl;
        return 1;
    }

    // Variables to hold the subkeys
    string K1, K2;

    // Generate subkeys
    generateSubkeys(key, K1, K2);

    // Output the subkeys
    cout << "Subkey K1: " << K1 << endl;
    cout << "Subkey K2: " << K2 << endl;
```

```
    return 0;
}
```

Output:

```
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding st
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Enter a 10-bit binary key: 1110001110
Subkey K1: 11101100
Subkey K2: 10110010
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding st
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Enter a 10-bit binary key: 1111111111
Subkey K1: 11111111
Subkey K2: 11111111
yug@yugs-MacBook-Air ins lab  % cd "/Users/yug/coding st
&& "/Users/yug/coding stuff/ins lab /"tempCodeRunnerFile
Enter a 10-bit binary key: 0101010101
Subkey K1: 00011011
Subkey K2: 01101001
```