DELHI TECHNOLOGICAL UNIVERSITY

CS305 –: Information Network & Security Lab File

Submitted by: **Submitted to:**

YUG BATHLA

23/CS/479

CSE1 (A1) (G3)

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.	
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.	
3	To implement Play fair cipher encryption-decryption.	
4	To implement Polyalphabetic cipher encryption decryption. Encryption/Decryption: Based on substitution, using multiple substitution Alphabets	
5	To implement Hill- cipher encryption decryption	
6	To implement S-DES sub key Generation	
7	To implement Diffie-hallman key exchange algorithm	
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)

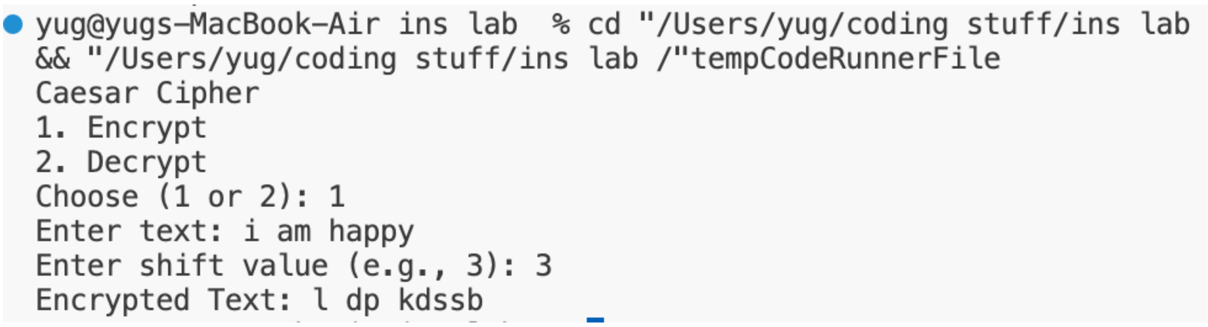
Characteristics

- **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.
-

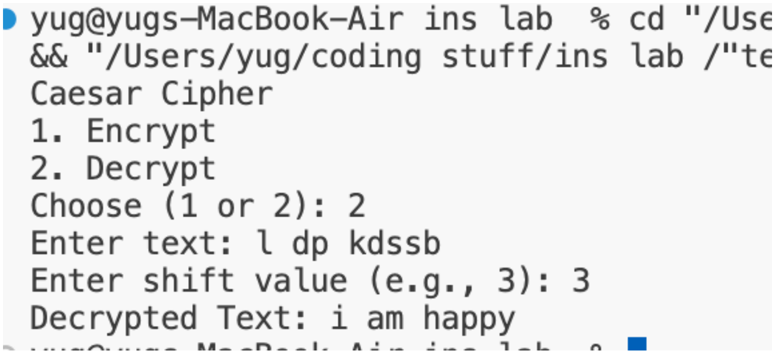
Source code:

```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.

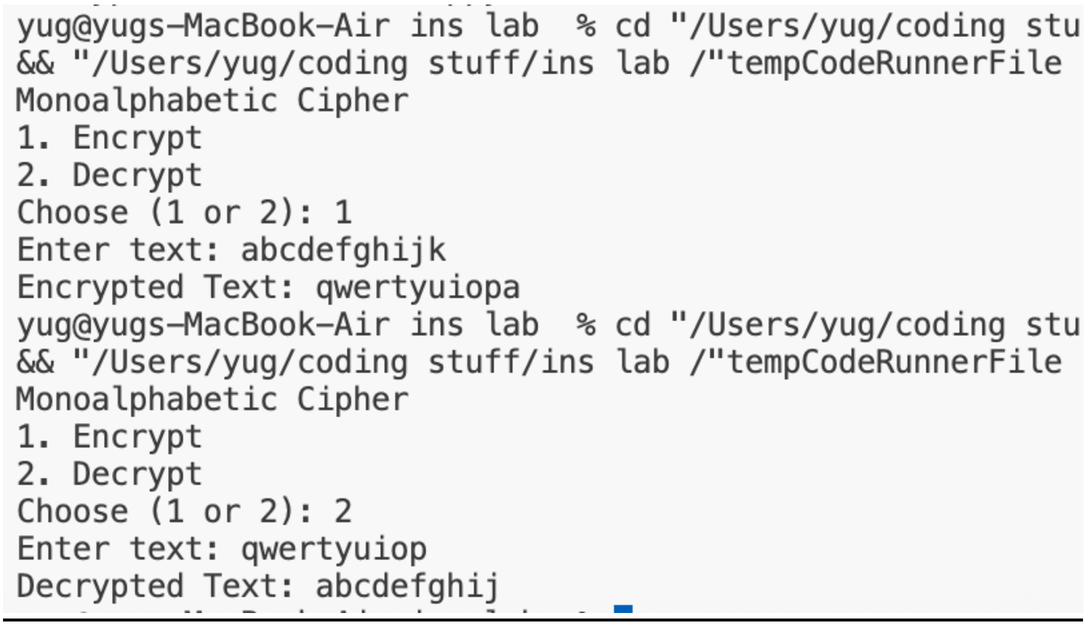
Applications

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

Source code:

```cpp
#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

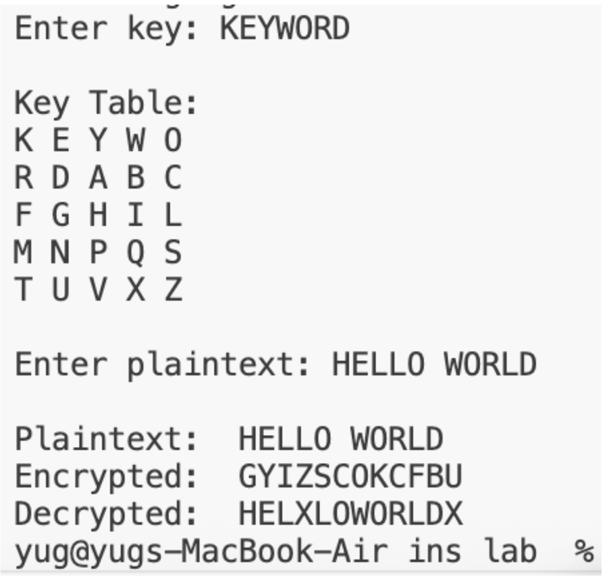
- Historically used for military communication.
- Good for **educational purposes** to demonstrate digraph encryption.

Source code:

```cpp
#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  %
```