Indira Gandhi Delhi Technical University for Women
(Established by Govt. of Delhi vide Act 09 of 2012)
Kashmere Gate, Delhi-110006

Practical File
for
Cyber Security
(MCA-207)

Submitted By:
Shweta Rawat
06304092023
MCA 2nd year
(2023-2025)

Submitted To:
Dr. Mohona Ghosh
Assistant Professor

Experiment 1

Write a program to implement Shift Cipher taking set of English alphabets.

CODE :

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

string encrypt(const string &plaintext, int shift) {
    string ciphertext = "";
    for (char c : plaintext) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            ciphertext += (c - base + shift) % 26 + base;
        } else {
            ciphertext += c;
        }
    }
    return ciphertext;
}

string decrypt(const string &ciphertext, int shift) {
    string plaintext = "";
    for (char c : ciphertext) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            plaintext += (c - base - shift + 26) % 26 + base;
        } else {
            plaintext += c;
        }
    }
    return plaintext;
}

int main() {
    int choice, shift;
    string input, result;

    cout << "Shift Cipher Implementation\n";
    cout << "1. Encrypt a message\n";
    cout << "2. Decrypt a message\n";
    cout << "Enter your choice (1 or 2): ";
    cin >> choice;
```

```
    cout << "Enter the shift value (0-25): ";
    cin >> shift;

    if (shift < 0 || shift > 25) {
        cout << "Invalid shift value! Please enter a value between 0 and 25.\n";
        return 1;
    }

    cin.ignore();
    cout << "Enter the message: ";
    getline(cin, input);

    if (choice == 1) {
        result = encrypt(input, shift);
        cout << "Encrypted message: " << result << endl;
    } else if (choice == 2) {
        result = decrypt(input, shift);
        cout << "Decrypted message: " << result << endl;
    } else {
        cout << "Invalid choice! Please select 1 or 2.\n";
    }

    return 0;
}
```

OUTPUT :

```
Shift Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 1
Enter the shift value (0-25): 5
Enter the message: hello, how are you
Encrypted message: mjqqt, mtb fwj dtz


=== Code Execution Successful ===
```

```
Shift Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 2
Enter the shift value (0-25): 5
Enter the message: mjqqt, mtb fwj dtz
Decrypted message: hello, how are you


=== Code Execution Successful ===
```

Experiment 2

Implement the following two cipher techniques via Python code for encryption and decryption of plain text. Substitution Cipher

CODE :

```
import string

def generate_substitution_key():
    alphabet = string.ascii_lowercase
    substitution_key = input("Enter a 26-character substitution key (unique letters only): ").lower()

    if len(substitution_key) != 26 or not all(char in alphabet for char in substitution_key):
        raise ValueError("Invalid key! Ensure it's 26 unique letters.")

    return substitution_key

def encrypt_substitution_cipher(plaintext, substitution_key):
    alphabet = string.ascii_lowercase
    substitution_map = {alphabet[i]: substitution_key[i] for i in range(26)}
    ciphertext = ""

    for char in plaintext:
        if char.isalpha():
            if char.islower():
                ciphertext += substitution_map[char]
            else:
                ciphertext += substitution_map[char.lower()].upper()
        else:
            ciphertext += char

    return ciphertext

def decrypt_substitution_cipher(ciphertext, substitution_key):
    alphabet = string.ascii_lowercase
    reverse_map = {substitution_key[i]: alphabet[i] for i in range(26)}
    plaintext = ""

    for char in ciphertext:
        if char.isalpha():
            if char.islower():
                plaintext += reverse_map[char]
            else:
                plaintext += reverse_map[char.lower()].upper()
```

```
        else:
            plaintext += char

    return plaintext

def main():
    print("Substitution Cipher Implementation")
    print("1. Encrypt a message")
    print("2. Decrypt a message")
    choice = int(input("Enter your choice (1 or 2): "))

    substitution_key = generate_substitution_key()
    text = input("Enter the text: ")

    if choice == 1:
        result = encrypt_substitution_cipher(text, substitution_key)
        print("Encrypted message:", result)
    elif choice == 2:
        result = decrypt_substitution_cipher(text, substitution_key)
        print("Decrypted message:", result)
    else:
        print("Invalid choice!")

if __name__ == "__main__":
    main()
```

OUTPUT :

```
Substitution Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 1
Enter a 26-character substitution key (unique letters only):
    qwertyuiopasdfghjklzxcvbnm
Enter the text: hellow
Encrypted message: itssgv

=== Code Execution Successful ===
```

```
Substitution Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 2
Enter a 26-character substitution key (unique letters only):
    qwertyuiopasdfghjklzxcvbnm
Enter the text: itssgv
Decrypted message: hellow

=== Code Execution Successful ===
```

Experiment 3

Write a program to implement (both encryption and decryption) Vigenere Cipher. Take plaintext in form of small letters from user and output capital letters as ciphertext.Also take two choices - choice 1 for encryption and choice 2 for decryption.

<u>CODE :</u>

```
def vigenere_encrypt(plaintext, key):
    ciphertext = ""
    key = key.lower()
    key_index = 0
    for char in plaintext:
        if char.islower():
            shift = ord(key[key_index]) - ord('a')
            new_char = chr((ord(char) - ord('a') + shift) % 26 + ord('A'))
            ciphertext += new_char
            key_index = (key_index + 1) % len(key)
        else:
            ciphertext += char
    return ciphertext

def vigenere_decrypt(ciphertext, key):
    plaintext = ""
    key = key.lower()
    key_index = 0
    for char in ciphertext:
        if char.isupper():
            shift = ord(key[key_index]) - ord('a')
            new_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('a'))
            plaintext += new_char
            key_index = (key_index + 1) % len(key)
        else:
            plaintext += char
    return plaintext

def main():
    print("Vigenère Cipher Implementation")
    print("1. Encrypt a message")
    print("2. Decrypt a message")
    choice = int(input("Enter your choice (1 or 2): "))

    key = input("Enter the key (letters only): ")
```

```
    if not key.isalpha():
        print("Invalid key! Please enter only letters.")
        return

    text = input("Enter the text: ")

    if choice == 1:
        result = vigenere_encrypt(text, key)
        print("Encrypted message:", result)
    elif choice == 2:
        result = vigenere_decrypt(text, key)
        print("Decrypted message:", result)
    else:
        print("Invalid choice!")

if __name__ == "__main__":
    main()
```

<u>**OUTPUT :**</u>

```
Vigenère Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 1
Enter the key (letters only): key
Enter the text: mca
Encrypted message: WGY

=== Code Execution Successful ===
```

```
Vigenère Cipher Implementation
1. Encrypt a message
2. Decrypt a message
Enter your choice (1 or 2): 2
Enter the key (letters only): key
Enter the text: WGY
Decrypted message: mca

=== Code Execution Successful ===
```

Experiment 4

Write a program to implement (both encryption and decryption) Transposition Cipher. Take plaintext in form of small letters from user and output capital letters as ciphertext.Also take two choices - choice 1 for encryption and choice 2 for decryption.

CODE :

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

// Function to encrypt using Transposition Cipher
string encryptTransposition(string plaintext, int key) {
    int len = plaintext.length();
    string ciphertext(len, ' ');

    for (int i = 0; i < len; i++) {
        ciphertext[(i * key) % len] = toupper(plaintext[i]);
    }
    return ciphertext;
}

// Function to decrypt using Transposition Cipher
string decryptTransposition(string ciphertext, int key) {
    int len = ciphertext.length();
    string plaintext(len, ' ');

    for (int i = 0; i < len; i++) {
        plaintext[i] = tolower(ciphertext[(i * key) % len]);
    }
    return plaintext;
}

int main() {
    string inputText;
    int choice, key;

    cout << "Enter your choice:\n1. Encryption\n2. Decryption\n";
    cin >> choice;
```

```
    if (choice != 1 && choice != 2) {
        cout << "Invalid choice. Please select 1 or 2.\n";
        return 1;
    }

    cout << "Enter the key (an integer): ";
    cin >> key;

    if (choice == 1) {
        cout << "Enter plaintext (in lowercase): ";
        cin >> inputText;
        string encryptedText = encryptTransposition(inputText, key);
        cout << "Ciphertext (in uppercase): " << encryptedText << endl;
    } else if (choice == 2) {
        cout << "Enter ciphertext (in uppercase): ";
        cin >> inputText;
        string decryptedText = decryptTransposition(inputText, key);
        cout << "Decrypted plaintext (in lowercase): " << decryptedText << endl;
    }

    return 0;
}
```

OUTPUT :

```
Enter your choice:
1. Encryption
2. Decryption
1
Enter the key (an integer): 3
Enter plaintext (in lowercase): igdtuw
Ciphertext (in uppercase): U  W
```

```
Enter your choice:
1. Encryption
2. Decryption
2
Enter the key (an integer): 5
Enter ciphertext (in uppercase): IGDTUW
Decrypted plaintext (in lowercase): iwutdg
```

Experiment 5

Using RSA algorithm, implement digital signatures

1. Display public and private key pair of two participants Alice and Bob
2. Based on the public private key pair, perform
    a. Sign-then-encrypt : First encrypt the message using private key of Alice followed by encryption using public key of Bob
    b. Encrypt-then-sign: First encrypt the message using public key of Bob followed by encryption using private key of Alice
3. Perform decryption for both cases.

CODE :

```
#include <iostream>
#include <cmath>
#include <string>
#include <tuple>

using namespace std;

// Function to calculate greatest common divisor (GCD)
int gcd(int a, int b) {
    while (b != 0) {
        int temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

// Function to calculate modular exponentiation (base^exp % mod)
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1) {
            result = (result * base) % mod;
        }
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
```

```
}

// RSA Key Generation
tuple<int, int, int> generateRSAKeys() {
    int p = 61, q = 53; // Two prime numbers (example values)
    int n = p * q;      // Public modulus
    int phi = (p - 1) * (q - 1);

    // Find public key `e` such that 1 < e < phi and gcd(e, phi) = 1
    int e = 3;
    while (gcd(e, phi) != 1) {
        e++;
    }

    // Find private key `d` such that (d * e) % phi = 1
    int d = 1;
    while ((d * e) % phi != 1) {
        d++;
    }

    return make_tuple(n, e, d); // Return n (modulus), e (public key), and d (private key)
}

// Sign the message using the sender's private key
long long signMessage(int message, int privateKey, int modulus) {
    return modExp(message, privateKey, modulus);
}

// Encrypt the message using the receiver's public key
long long encryptMessage(int message, int publicKey, int modulus) {
    return modExp(message, publicKey, modulus);
}

// Decrypt the message using the receiver's private key
long long decryptMessage(long long encryptedMessage, int privateKey, int modulus) {
    return modExp(encryptedMessage, privateKey, modulus);
}

// Workflow: Sign-then-encrypt and Encrypt-then-sign
int main() {
    // Generate keys for Alice and Bob
    auto [nAlice, eAlice, dAlice] = generateRSAKeys();
    auto [nBob, eBob, dBob] = generateRSAKeys();
```

```
    cout << "Alice's Public Key (e, n): (" << eAlice << ", " << nAlice << ")\n";
    cout << "Alice's Private Key (d, n): (" << dAlice << ", " << nAlice << ")\n";
    cout << "Bob's Public Key (e, n): (" << eBob << ", " << nBob << ")\n";
    cout << "Bob's Private Key (d, n): (" << dBob << ", " << nBob << ")\n\n";

    // Message to be sent (numeric value for simplicity)
    int message = 89;
    cout << "Original Message: " << message << "\n\n";

    // Sign-then-encrypt
    cout << "Sign-then-encrypt:\n";
    long long signedMessage = signMessage(message, dAlice, nAlice);
    long long encryptedMessageSTE = encryptMessage(signedMessage, eBob, nBob);
    cout << "Signed Message: " << signedMessage << "\n";
    cout << "Encrypted Message (Sign-then-encrypt): " << encryptedMessageSTE << "\n";

    // Decryption for Sign-then-encrypt
    long long decryptedSTE = decryptMessage(encryptedMessageSTE, dBob, nBob);
    long long verifiedSTE = decryptMessage(decryptedSTE, eAlice, nAlice);
    cout << "Decrypted Message (Sign-then-encrypt): " << verifiedSTE << "\n\n";

    // Encrypt-then-sign
    cout << "Encrypt-then-sign:\n";
    long long encryptedMessage = encryptMessage(message, eBob, nBob);
    long long signedMessageETS = signMessage(encryptedMessage, dAlice, nAlice);
    cout << "Encrypted Message: " << encryptedMessage << "\n";
    cout << "Signed Message (Encrypt-then-sign): " << signedMessageETS << "\n";

    // Decryption for Encrypt-then-sign
    long long verifiedETS = decryptMessage(signedMessageETS, eAlice, nAlice);
    long long decryptedETS = decryptMessage(verifiedETS, dBob, nBob);
    cout << "Decrypted Message (Encrypt-then-sign): " << decryptedETS << "\n";

    return 0;
}
```

OUTPUT :

```
Alice's Public Key (e, n): (7, 3233)
Alice's Private Key (d, n): (1783, 3233)
Bob's Public Key (e, n): (7, 3233)
Bob's Private Key (d, n): (1783, 3233)

Original Message: 89

Sign-then-encrypt:
Signed Message: 236
Encrypted Message (Sign-then-encrypt): 89
Decrypted Message (Sign-then-encrypt): 89

Encrypt-then-sign:
Encrypted Message: 206
Signed Message (Encrypt-then-sign): 89
Decrypted Message (Encrypt-then-sign): 89
```

Experiment 6

Read cuckoo hashing from “geekeforgeeks” and implement cuckoo hashing using hash functions given on https://www.geeksforgeeks.org/cuckoo-hashing/

CODE :

```
#include <iostream>
#include <vector>
#include <cmath>
#include <cstring>

using namespace std;

const int MAX_REHASHES = 10; // Maximum rehash attempts
const int TABLE_SIZE = 11;   // Size of the hash tables

class CuckooHashing {
private:
    vector<int> table1, table2; // Two hash tables
    int size;                   // Current number of elements

    // Hash functions
    int h1(int key) {
        return key % TABLE_SIZE;
    }

    int h2(int key) {
        return (key / TABLE_SIZE) % TABLE_SIZE;
    }

    // Rehash the entire table when a cycle occurs
    void rehash() {
        cout << "Rehashing..." << endl;
        vector<int> oldTable1 = table1;
        vector<int> oldTable2 = table2;
        table1.assign(TABLE_SIZE, -1);
        table2.assign(TABLE_SIZE, -1);
        size = 0;

        // Reinsert elements into new tables
        for (int key : oldTable1) {
            if (key != -1)
                insert(key);
```

```
        }
        for (int key : oldTable2) {
            if (key != -1)
                insert(key);
        }
    }

public:
    // Constructor
    CuckooHashing() : size(0) {
        table1.assign(TABLE_SIZE, -1);
        table2.assign(TABLE_SIZE, -1);
    }

    // Insert a key into the hash table
    void insert(int key) {
        if (lookup(key)) {
            cout << "Key " << key << " already exists." << endl;
            return;
        }

        int cycleCheck = 0;
        int currentKey = key;
        for (int attempts = 0; attempts < MAX_REHASHES; ++attempts) {
            int pos1 = h1(currentKey);

            // Try inserting into table1
            if (table1[pos1] == -1) {
                table1[pos1] = currentKey;
                ++size;
                return;
            }

            // Displace the key from table1
            swap(currentKey, table1[pos1]);

            int pos2 = h2(currentKey);

            // Try inserting into table2
            if (table2[pos2] == -1) {
                table2[pos2] = currentKey;
                ++size;
                return;
            }
```

```
        // Displace the key from table2
        swap(currentKey, table2[pos2]);

        if (++cycleCheck > size) {
            rehash();
            insert(key);
            return;
        }
    }

    // If insertion fails after MAX_REHASHES, rehash
    rehash();
    insert(key);
}

// Delete a key from the hash table
void remove(int key) {
    int pos1 = h1(key);
    if (table1[pos1] == key) {
        table1[pos1] = -1;
        --size;
        cout << "Key " << key << " deleted from table1." << endl;
        return;
    }

    int pos2 = h2(key);
    if (table2[pos2] == key) {
        table2[pos2] = -1;
        --size;
        cout << "Key " << key << " deleted from table2." << endl;
        return;
    }

    cout << "Key " << key << " not found." << endl;
}

// Lookup a key in the hash table
bool lookup(int key) {
    int pos1 = h1(key);
    if (table1[pos1] == key)
        return true;

    int pos2 = h2(key);
```

```
        if (table2[pos2] == key)
            return true;

        return false;
    }

    // Display the hash tables
    void display() {
        cout << "Table 1: ";
        for (int i = 0; i < TABLE_SIZE; ++i) {
            if (table1[i] != -1)
                cout << table1[i] << " ";
            else
                cout << "- ";
        }
        cout << endl;

        cout << "Table 2: ";
        for (int i = 0; i < TABLE_SIZE; ++i) {
            if (table2[i] != -1)
                cout << table2[i] << " ";
            else
                cout << "- ";
        }
        cout << endl;
    }
};

// Driver code
int main() {
    CuckooHashing hashTable;

    vector<int> keys = {20, 50, 53, 75, 100, 67, 105, 3, 36, 39};

    for (int key : keys) {
        cout << "Inserting " << key << "..." << endl;
        hashTable.insert(key);
        hashTable.display();
    }

    cout << "\nLooking up 53: " << (hashTable.lookup(53) ? "Found" : "Not Found") << endl;
    cout << "Looking up 200: " << (hashTable.lookup(200) ? "Found" : "Not Found") << endl;

    cout << "\nDeleting 53..." << endl;
```

```
    hashTable.remove(53);
    hashTable.display();

    return 0;
}
```

OUTPUT :

```
Inserting 20...
Table 1: - - - - - - - - - 20 -
Table 2: - - - - - - - - - - -
Inserting 50...
Table 1: - - - - - - 50 - - 20 -
Table 2: - - - - - - - - - - -
Inserting 53...
Table 1: - - - - - - 50 - - 53 -
Table 2: - 20 - - - - - - - - -
Inserting 75...
Table 1: - - - - - - 50 - - 75 -
Table 2: - 20 - - 53 - - - - - -
Inserting 100...
Table 1: - 100 - - - - 50 - - 75 -
Table 2: - 20 - - 53 - - - - - -
Inserting 67...
Table 1: - 67 - - - - 50 - - 75 -
Table 2: - 20 - - 53 - - - - 100 -
Inserting 105...
Table 1: - 67 - - - - 105 - - 53 -
Table 2: - 20 - - 50 - 75 - - 100 -
Inserting 3...
Table 1: - 67 - 3 - - 105 - - 53 -
Table 2: - 20 - - 50 - 75 - - 100 -
Inserting 36...
Table 1: - 67 - 36 - - 105 - - 53 -
Table 2: 3 20 - - 50 - 75 - - 100 -
Inserting 39...
Table 1: - 100 - 36 - - 50 - - 75 -
Table 2: 3 20 - 39 53 - 67 - - 105 -
```

```
Looking up 53: Found
Looking up 200: Not Found

Deleting 53...
Key 53 deleted from table2.
Table 1: - 100 - 36 - - 50 - - 75 -
Table 2: 3 20 - 39 - - 67 - - 105 -
```