Experiment No. 1

Title: Substitution Cipher – Additive, Multiplicative and Affine Cipher

Batch: B-3 **Roll No.: 16010422234** **Experiment No.: 01**

Aim: To implement substitution ciphers – Additive, Multiplicative and Affine ciphers.

Resources needed: Windows/Linux.

Theory

Pre Lab/ Prior Concepts:

Symmetric-key algorithms are a class of algorithms for cryptography that use the same cryptographic keys for both encryption of plaintext and decryption of cipher text. The keys may be identical or there may be a simple transformation to go between the two keys. The keys, in practice, represent a shared secret between two or more parties that can be used to maintain a private information link. This requirement that both parties have access to the secret key is one of the main drawbacks of symmetric key encryption, in comparison to public-key encryption. Symmetric-key encryption can use either stream ciphers or block ciphers. Transposition Cipher is block cipher. Ancient cryptographic systems are classified as: Substitution and Permutation Ciphers.

Simple Substitution Cipher
A substitution cipher replaces one symbol with another. Letters of plaintext are replaced by other letters or by numbers or symbols. In a particularly simple implementation of a simple substitution cipher, the message is encrypted by substituting the letter of the alphabet n places ahead of the current letter. For example, with n = 3, the substitution which acts as the key

plaintext: a b c d e f g h i j k l m n o p q r s t u v w x y z
ciphertext: D E F G H I J K L M N O P Q R S T U V W X Y Z A B C

The convention is plaintext will be in lowercase and the cipher text will be in uppercase. In this example, the key could be stated more succinctly as “3” since the amount of the shift is the key. Using the key of 3, we can encrypt the plaintext message: “fourscoreandsevenyearsago” by looking up each letter in the plaintext row and substituting the corresponding letter in the ciphertext row or by simply replacing each letter by the letter that is three positions ahead of it in the alphabet. In this particular example, the resulting cipher text is IRXUVFRUHDAGVHYHABHDUVDIR

To decrypt, we simply look up the ciphertext letter in the ciphertext row and replace it with the corresponding letter in the plaintext row, or simply shift each ciphertext letter backward by three. The simple substitution with a shift of three is known as Caesar's cipher because it was reputedly used with success by Julius Caesar.

Substitution ciphers are classified as monoalphabetic and polyalphabetic substitution ciphers. In monoalphabetic substitution cipher, each occurrence of character is encrypted by the same substitute character. In Polyalphabetic substitution cipher each occurrence of a character may have a different substitute due to variable Key.

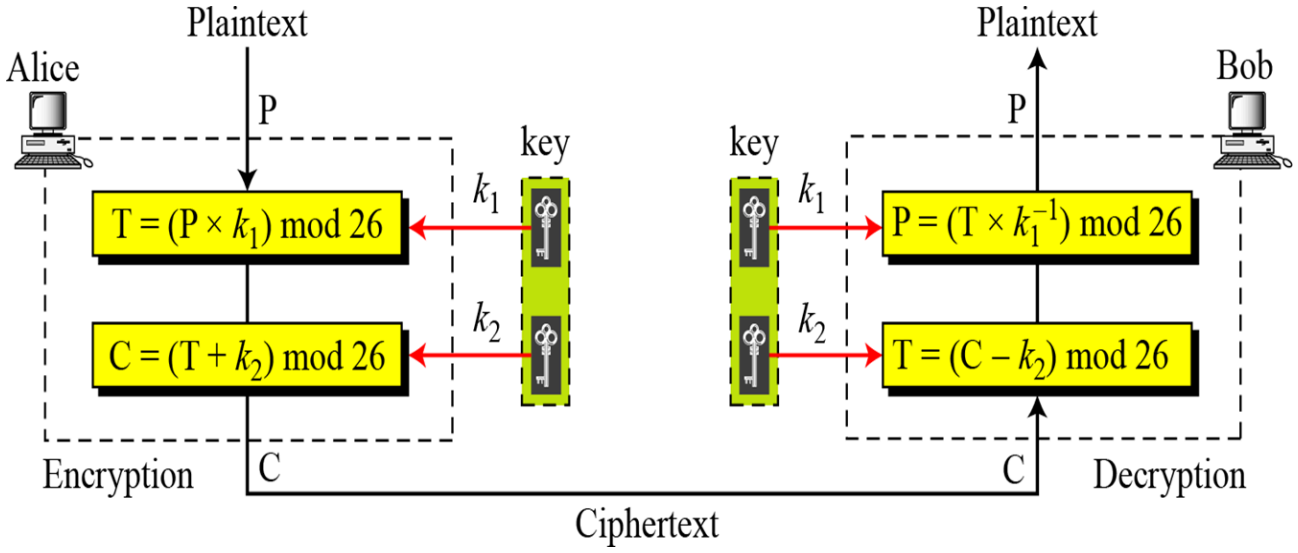
AFFINE CIPHER

The Affine cipher is a type of monoalphabetic substitution cipher which uses a combination of Additive and Multiplicative Ciphers. Each letter is enciphered with the function (ax + b)

mod 26, where b is the magnitude of the shift. The encryption function for a single letter is

C=(ax + b) mod m where 1≤a≤m, 1≤b≤m

where modulus m is the size of the alphabet and a and b are the keys of the cipher. The value a must be chosen such that a and m are coprime. The decryption function is $P = a^{-1}(c-b) \mod m$, where a^{-1} is the modular multiplicative inverse of a i.e., it satisfies the equation $a\ a^{-1} = 1 \mod m$.



Encryption: Key Values a=17, b=20

Original Text	T	W	E	N	T	Y		F	I	F	T	E	E	N
x	19	22	4	13	19	24		5	8	5	19	4	4	13
ax+b % 26*	5	4	10	7	5	12		1	0	1	5	10	10	7
Encrypted Text	F	E	K	H	F	M		B	A	B	F	K	K	H

Decryption: a^-1 = 23

Encrypted Text	F	E	K	H	F	M		B	A	B	F	K	K	H
Encrypted Value	5	4	10	7	5	12		1	0	1	5	10	10	7
23 *(x-b) mod 26	19	22	4	13	19	24		5	8	5	19	4	4	13
Decrypted Text	T	W	E	N	T	Y		F	I	F	T	E	E	N

Activity:
Implement the following substitution ciphers:

1. Additive Cipher
2. Multiplicative Cipher
3. Affine Cipher

Implementation:
Implement a menu driven program. It should have an encryption function and decryption function for each cipher. Function should take a message and a key as input from the user and display the expected output.

Results: (Program with output as per the format)

Program:

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

string additiveCipherEncrypt(string message, int key) {
    string encryptedMessage = "";
    for (char& c : message) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char shifted = (c - base + key) % 26 + base;
            encryptedMessage += shifted;
        } else {
            encryptedMessage += c;
        }
    }
    return encryptedMessage;
}

string additiveCipherDecrypt(string encryptedMessage, int key) {
    string decryptedMessage = "";
    for (char& c : encryptedMessage) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char shifted = (c - base - key + 26) % 26 + base;
            decryptedMessage += shifted;
        } else {
            decryptedMessage += c;
        }
    }
    return decryptedMessage;
}
```

```
string multiplicativeCipherEncrypt(string message, int key) {
    string encryptedMessage = "";
    for (char& c : message) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char shifted = (c - base) * key % 26 + base;
            encryptedMessage += shifted;
        } else {
            encryptedMessage += c;
        }
    }
    return encryptedMessage;
}

int findMultiplicativeInverse(int key, int mod) {
    for (int i = 1; i < mod; ++i) {
        if ((key * i) % mod == 1) {
            return i;
        }
    }
    return -1;
}

string multiplicativeCipherDecrypt(string encryptedMessage, int key)
{
    int inverse = findMultiplicativeInverse(key, 26);
    if (inverse == -1) {
        return "Error: No multiplicative inverse found.";
    }
    string decryptedMessage = "";
    for (char& c : encryptedMessage) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char shifted = (c - base) * inverse % 26 + base;
            decryptedMessage += shifted;
        } else {
            decryptedMessage += c;
        }
    }
    return decryptedMessage;
}

string affineCipherEncrypt(string message, int key_m, int key_a) {
```

```
    string encryptedMessage = "";
    for (char& c : message) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
            char shifted = ((c - base) * key_m + key_a) % 26 + base;
            encryptedMessage += shifted;
        } else {
            encryptedMessage += c;
        }
    }
    return encryptedMessage;
}

string affineCipherDecrypt(string encryptedMessage, int key_m, int
key_a) {
    int inverse = findMultiplicativeInverse(key_m, 26);
    if (inverse == -1) {
        return "Error: No multiplicative inverse found.";
    }
    string decryptedMessage = "";
    for (char& c : encryptedMessage) {
        if (isalpha(c)) {
            char base = isupper(c) ? 'A' : 'a';
             char shifted = (inverse * ((c - base) - key_a + 26)) %
26 + base;
            decryptedMessage += shifted;
        } else {
            decryptedMessage += c;
        }
    }
    return decryptedMessage;
}

int main() {
    int choice;
    string message;
    int key, key_m, key_a;

    do {
        cout << "Menu:" << endl;
        cout << "1. Additive Cipher Encryption" << endl;
        cout << "2. Additive Cipher Decryption" << endl;
        cout << "3. Multiplicative Cipher Encryption" << endl;
        cout << "4. Multiplicative Cipher Decryption" << endl;
```

```
        cout << "5. Affine Cipher Encryption" << endl;
        cout << "6. Affine Cipher Decryption" << endl;
        cout << "7. Exit" << endl;
        cout << "Enter your choice: ";
        cin >> choice;

        switch (choice) {
            case 1:
                cout << "Enter message to encrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter key (integer): ";
                cin >> key;
                                cout << "Encrypted message: " <<
additiveCipherEncrypt(message, key) << endl;
                break;
            case 2:
                cout << "Enter message to decrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter key (integer): ";
                cin >> key;
                                cout << "Decrypted message: " <<
additiveCipherDecrypt(message, key) << endl;
                break;
            case 3:
                cout << "Enter message to encrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter key (integer): ";
                cin >> key;
                                cout << "Encrypted message: " <<
multiplicativeCipherEncrypt(message, key) << endl;
                break;
            case 4:
                cout << "Enter message to decrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter key (integer): ";
                cin >> key;
                                cout << "Decrypted message: " <<
multiplicativeCipherDecrypt(message, key) << endl;
                break;
            case 5:
```

```
                cout << "Enter message to encrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter multiplier (integer): ";
                cin >> key_m;
                cout << "Enter additive (integer): ";
                cin >> key_a;
                                cout << "Encrypted message: " <<
affineCipherEncrypt(message, key_m, key_a) << endl;
                break;
            case 6:
                cout << "Enter message to decrypt: ";
                cin.ignore();
                getline(cin, message);
                cout << "Enter multiplier (integer): ";
                cin >> key_m;
                cout << "Enter additive (integer): ";
                cin >> key_a;
                                cout << "Decrypted message: " <<
affineCipherDecrypt(message, key_m, key_a) << endl;
                break;
            case 7:
                cout << "Exiting the program." << endl;
                break;
            default:
                cout << "Invalid choice! Please try again." << endl;
                break;
        }
    } while (choice != 7);

    return 0;
}
```

Output:

```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 1
Enter message to encrypt: HELLO
Enter key (integer): 3
Encrypted message: KHOOR
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 2
Enter message to decrypt: KHOOR
Enter key (integer): 3
Decrypted message: HELLO
```

```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 3
Enter message to encrypt: HELLO
Enter key (integer): 3
Encrypted message: VMHHQ
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 4
Enter message to decrypt: VMHHQ
Enter key (integer): 3
Decrypted message: HELLO
```

```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 5
Enter message to encrypt: HELLO
Enter multiplier (integer): 3
Enter additive (integer): 2
Encrypted message: XOJJS
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 6
Enter message to decrypt: XOJJS
Enter multiplier (integer): 3
Enter additive (integer): 2
Decrypted message: HELLO
```

```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
5. Affine Cipher Encryption
6. Affine Cipher Decryption
7. Exit
Enter your choice: 7
Exiting the program.
PS C:\Users\chand\INS\EXP1>
```

Questions:

1) **Write down the flaws of Additive cipher and Multiplicative Cipher:**

Ans: Additive Cipher (Caesar Cipher)
Weak against brute-force attacks: Since there are only 25 possible keys in a standard Caesar cipher (26 if you include the key that does nothing), it is vulnerable to brute-force attacks where an attacker tries all possible keys.
Frequency analysis vulnerability: Similar to other substitution ciphers, the additive cipher is susceptible to frequency analysis. A determined attacker can analyze the frequency of letters in the ciphertext to deduce the plaintext.

Multiplicative Cipher
Limited key space: The key space is restricted by the modulus
m (usually 26 in the case of the English alphabet), which limits the number of possible keys and makes it susceptible to brute-force attacks.
Weak against chosen plaintext attacks: If the attacker can choose plaintexts and observe the corresponding ciphertexts, they might be able to deduce the key using known plaintext attacks.

2) **Compare Affine cipher and Vigenere Cipher.**

Ans: Affine Cipher
Key structure: Uses two keys: one for multiplication (a) and another for addition (b).
Strengths:
The combination of multiplication and addition makes it stronger than simple substitution ciphers like the Caesar cipher.
It increases the key space compared to monoalphabetic ciphers.
Weaknesses:
Vulnerable to frequency analysis attacks, especially if the modulus (m) is known.

If the keys (a and m) are not chosen properly (e.g., if gcd(a, m) ≠ 1), it can be easier to break.

Vigenere Cipher
Key structure: Uses a keyword that is repeated to encrypt the message.
Strengths:
It's polyalphabetic, meaning the encryption of each letter depends on a different part of the key, which complicates frequency analysis.
More resistant to simple frequency analysis compared to monoalphabetic ciphers.
Weaknesses:
Vulnerable to Kasiski examination and Friedman tests if the key length can be guessed.
If the key is shorter than the plaintext, it can be vulnerable to statistical methods to guess the key length and then break the cipher.

Comparison:
Security: Affine cipher is generally stronger than Vigenere cipher against basic attacks like frequency analysis due to its use of two mathematical operations.
Complexity: Vigenere cipher is more complex to understand and implement correctly compared to the affine cipher.
Key Management: Affine cipher requires careful selection of keys (especially the multiplier) to avoid weak configurations, whereas Vigenere cipher requires a sufficiently long and random key to be secure.

Outcomes: Illustrate different cryptographic algorithms for security.

Conclusion:
In this experiment, we implemented three classic substitution ciphers: Additive Cipher (Caesar Cipher), Multiplicative Cipher, and Affine Cipher. These ciphers are fundamental in cryptography and illustrate different approaches to encrypting messages using simple mathematical transformations.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References: Books/ Journals/ Websites:

1. Behrouz A. Forouzan, “Cryptography and Network Security”, Tata McGraw Hill
2. William Stalling, “Cryptography and Network Security”, Prentice Hall