Experiment No. 1

Title: Substitution Cipher

Batch: A3 **Roll No.: 16010421059** **Experiment No.:1**

Aim: To implement substitution ciphers – Affine and Vigenere cipher.

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 the Caesar’s cipher because it was reputedly used with success by Julius Caesar.

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

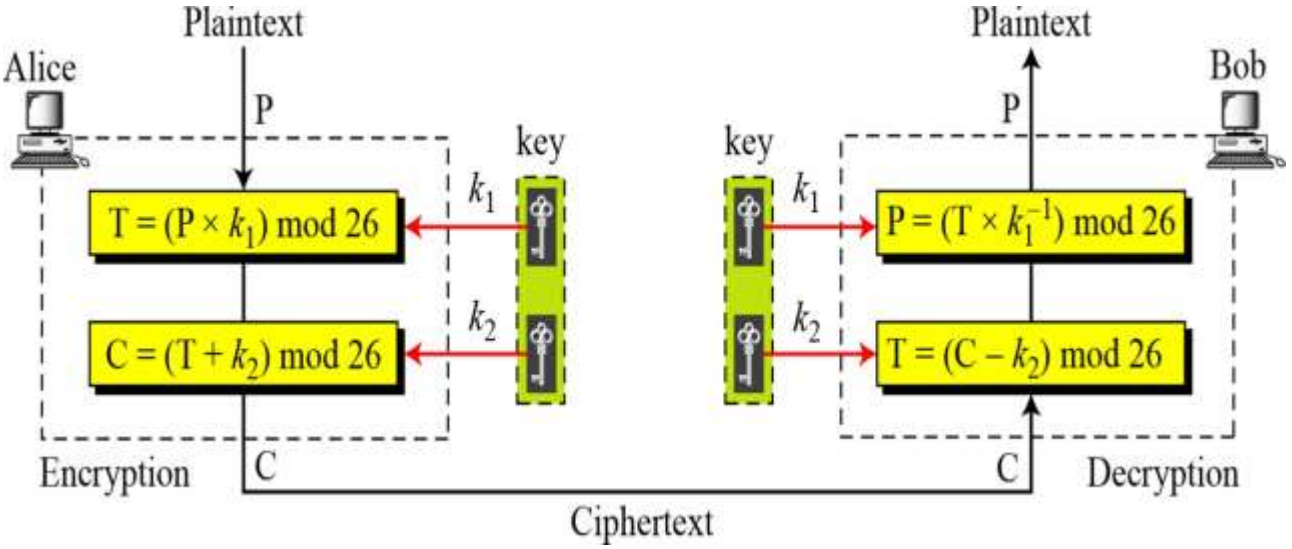
AFFINE CIPHER

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

$C=(ax + b) \bmod m$ where $1 \leq a \leq m$, $1 \leq b \leq m$

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 $P = a^{-1}(c-b) \bmod m$, where a^{-1} is the modular multiplicative inverse of a i.e., it satisfies the equation $a\, a^{-1} = 1 \bmod m$.



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

Implementation:

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

Results: (Program with output as per the format)

```
#include <stdio.h>
#include <string.h>


void addEncrypt(char text[], int key) {
    int i;
    for (i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 'A' && text[i] <= 'Z')
            text[i] = ((text[i] - 'A') + key) % 26 + 'A';
        else if (text[i] >= 'a' && text[i] <= 'z')
            text[i] = ((text[i] - 'a') + key) % 26 + 'a';
    }
}


void addDecrypt(char text[], int key) {
    int i;
    for (i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 'A' && text[i] <= 'Z')
            text[i] = ((text[i] - 'A') - key + 26) % 26 + 'A';
        else if (text[i] >= 'a' && text[i] <= 'z')
            text[i] = ((text[i] - 'a') - key + 26) % 26 + 'a';
    }
}


void multiEncrypt(char text[], int key) {
    int i;
    for (i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 'A' && text[i] <= 'Z')
            text[i] = ((text[i] - 'A') * key) % 26 + 'A';
        else if (text[i] >= 'a' && text[i] <= 'z')
            text[i] = ((text[i] - 'a') * key) % 26 + 'a';
    }
}

void multiDecrypt(char text[], int key) {
    int i, inverse = 0;
    for (i = 0; i < 26; i++) {
        if ((key * i) % 26 == 1) {
            inverse = i;
            break;
        }
    }
    if (inverse == 0) {
        printf("Error: Invalid key for decryption.");
        return;
```

```
    }

    for (i = 0; text[i] != '\0'; i++) {
        if (text[i] >= 'A' && text[i] <= 'Z')
            text[i] = ((text[i] - 'A') * inverse) % 26 + 'A';
        else if (text[i] >= 'a' && text[i] <= 'z')
            text[i] = ((text[i] - 'a') * inverse) % 26 + 'a';
    }
}

int main() {
    char text[100];
    int key, choice;

    printf("Enter text to encrypt/decrypt: ");
    fgets(text, sizeof(text), stdin);

while(1){
    printf("1. Additive Encryption\n");
    printf("2. Additive Decryption\n");
    printf("3. Multiplicative Encryption\n");
    printf("4. Multiplicative Decryption\n");
    printf("5. Exit\n");
    printf("Choose an option: ");
    scanf("%d", &choice);

    printf("\nEnter the encryption/decryption key: ");
    scanf("%d", &key);

    switch (choice) {
        case 1:
            addEncrypt(text, key);
            printf("Encrypted text: %s\n", text);
            break;
        case 2:
            addDecrypt(text, key);
            printf("Decrypted text: %s\n", text);
            break;
        case 3:
            multiEncrypt(text, key);
            printf("Encrypted text: %s\n", text);
            break;
        case 4:
            multiDecrypt(text, key);
            printf("Decrypted text: %s\n", text);
            break;
        case 5:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
}
    return 0;


}
```

OUTPUT:

```
/tmp/dLOoldrFuN.o
Enter text to encrypt/decrypt: hello
1. Additive Encryption
2. Additive Decryption
3. Multiplicative Encryption
4. Multiplicative Decryption
5. Exit
Choose an option: 3
Enter the encryption/decryption key: 3
Encrypted text: vmhhq

1. Additive Encryption
2. Additive Decryption
3. Multiplicative Encryption
4. Multiplicative Decryption
5. Exit
Choose an option: 4
Enter the encryption/decryption key: 3
Decrypted text: hello

1. Additive Encryption
2. Additive Decryption
3. Multiplicative Encryption
4. Multiplicative Decryption
5. Exit
Choose an option: 3
Enter the encryption/decryption key: 4
```

Questions:

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

Ans:

Additive Cipher (Caesar Cipher) flaws:

- **Limited key space:** The Additive Cipher has a small key space since there are only 26 possible keys (for the English alphabet). This makes it vulnerable to brute-force attacks.
- **Vulnerable to frequency analysis:** The encrypted text often maintains the same letter frequency as the original text, making it susceptible to frequency analysis attacks.
- **Easily breakable with known-plaintext attacks:** If an attacker knows or can guess any part of the plaintext and its corresponding ciphertext, they can easily determine the key and decrypt the entire message.

Multiplicative Cipher flaws:

- **Limited key space:** The Multiplicative Cipher also has a limited key space since there are only 12 possible keys for the English alphabet (if the key and 26

are coprime). This makes it susceptible to brute-force attacks.

- **Vulnerable to known-plaintext attacks:** Similar to the Additive Cipher, if an attacker has access to known plaintext-ciphertext pairs, they can find the key and decrypt the message.
- **Double letters and key space:** If two letters in the plaintext are the same and have the same position in the alphabet (e.g., "e" in "see"), they will have the same ciphertext letter, potentially revealing patterns and reducing the effective key space.

2) Implement/Write down the code of Affine cipher and Vigenere Cipher:
Ans:

Code for Affine Cipher:

```
#include <stdio.h>
#include <ctype.h>

int mod_inverse(int a, int m) {
    for (int x = 0; x < m; x++) { // Start the loop from x = 0
        if ((a * x) % m == 1) {
            return x;
        }
    }
    return -1;
}

void affine_encrypt(char plaintext[], char encrypted_text[], int a, int b) {
    int alphabet_size = 26;
    int i = 0;
    char encrypted_char;

    while (plaintext[i]) {
        if (isalpha(plaintext[i])) {
            if (isupper(plaintext[i])) {
                encrypted_char = ((a * (plaintext[i] - 'A') + b) % alphabet_size) + 'A';
            } else {
                encrypted_char = ((a * (plaintext[i] - 'a') + b) % alphabet_size) + 'a';
            }
            encrypted_text[i] = encrypted_char;
        } else {
            encrypted_text[i] = plaintext[i];
        }
        i++;
    }
    encrypted_text[i] = '\0';
}

void affine_decrypt(char encrypted_text[], char decrypted_text[], int a, int b) {
    int alphabet_size = 26;
    int i = 0;
    int inverse = mod_inverse(a, alphabet_size); // Calculate the modular inverse
    if (inverse == -1) {
        printf("Modular inverse of 'a' does not exist. Cannot decrypt.\n");
        return;
    }

    char  decrypted_char;

    while (encrypted_text[i]) {
        if (isalpha(encrypted_text[i])) {
            if (isupper(encrypted_text[i])) {
                decrypted_char = ((inverse * (encrypted_text[i] - 'A' - b + alphabet_size))
% alphabet_size) + 'A';
            } else {
```

```
            decrypted_char = ((inverse * (encrypted_text[i] - 'a' - b + alphabet_size))
% alphabet_size) + 'a';
            }
            decrypted_text[i] = decrypted_char;
        } else {
            decrypted_text[i] = encrypted_text[i];
        }
        i++;
    }
    decrypted_text[i] = '\0';
}

int main() {
    char plaintext[100];
    char encrypted_text[100];
    char decrypted_text[100];
    int a, b;

    printf("Enter the word to be encrypted: ");
    fgets(plaintext, sizeof(plaintext), stdin);

    printf("Enter the value of 'a': ");
    scanf("%d", &a);

    printf("Enter the value of 'b': ");
    scanf("%d", &b);

    affine_encrypt(plaintext, encrypted_text, a, b);
    printf("Encrypted text: %s\n", encrypted_text);

    affine_decrypt(encrypted_text, decrypted_text, a, b);
    printf("Decrypted text: %s\n", decrypted_text);

    return 0;
}
```

Output of Affine Cipher:

```
Output
/tmp/vS8c08hwWL.o
Enter the word to be encrypted: nidhi
Enter the value of 'a': 9
Enter the value of 'b': 20
Encrypted text: hovfo

Decrypted text: nidhi
```

Code for Vigenere Cipher:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

void remove_newline(char str[]) {
   str[strcspn(str, "\n")] = '\0';
}

void vigenere_encrypt(char plaintext[], char keyword[], char encrypted_text[]) {
   int i, j;
   int keyword_length = strlen(keyword);
   int alphabet_size = 26;

   for (i = 0, j = 0; plaintext[i] != '\0'; i++) {
      if (isalpha(plaintext[i])) {
         int key = tolower(keyword[j % keyword_length]) - 'a';
         char base = isupper(plaintext[i]) ? 'A' : 'a';
         char encrypted_char = (plaintext[i] - base + key) % alphabet_size + base;
         encrypted_text[i] = encrypted_char;
         j++;
      } else {
         encrypted_text[i] = plaintext[i];
      }
   }
   encrypted_text[i] = '\0';
}

void vigenere_decrypt(char encrypted_text[], char keyword[], char
decrypted_text[]) {
   int i, j;
   int keyword_length = strlen(keyword);
   int alphabet_size = 26;

   for (i = 0, j = 0; encrypted_text[i] != '\0'; i++) {
      if (isalpha(encrypted_text[i])) {
         int key = tolower(keyword[j % keyword_length]) - 'a';
         char base = isupper(encrypted_text[i]) ? 'A' : 'a';
         char decrypted_char = (encrypted_text[i] - base - key + alphabet_size) %
alphabet_size + base;
         decrypted_text[i] = decrypted_char;
         j++;
      } else {
         decrypted_text[i] = encrypted_text[i];
      }
   }
   decrypted_text[i] = '\0';
}

int main() {
   char plaintext[100];
   char encrypted_text[100];
   char decrypted_text[100];
   char keyword[100];
```

```
    printf("Enter the word to be encrypted: ");
    fgets(plaintext, sizeof(plaintext), stdin);
    remove_newline(plaintext);

    printf("Enter the keyword: ");
    fgets(keyword, sizeof(keyword), stdin);
    remove_newline(keyword);

    vigenere_encrypt(plaintext, keyword, encrypted_text);
    printf("Encrypted text: %s\n", encrypted_text);

    vigenere_decrypt(encrypted_text, keyword, decrypted_text);
    printf("Decrypted text: %s\n", decrypted_text);

    return 0;
}
```

Output for Vignere Cipher:

```
Output

/tmp/vS8c08hwWL.o
Enter the word to be encrypted: nidhi
Enter the keyword: key
Encrypted text: xmbrm
Decrypted text: nidhi
```

Outcomes: CO 1 Describe the basics of Information Security

Conclusion: Through this experiment, I understood the concepts of substitution ciphers – Affine and Vigenere cipher and also learned how to implement them.

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