Experiment No. 1

Title: Substitution Cipher

Batch: B2 **Roll No.:16010421119** **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.

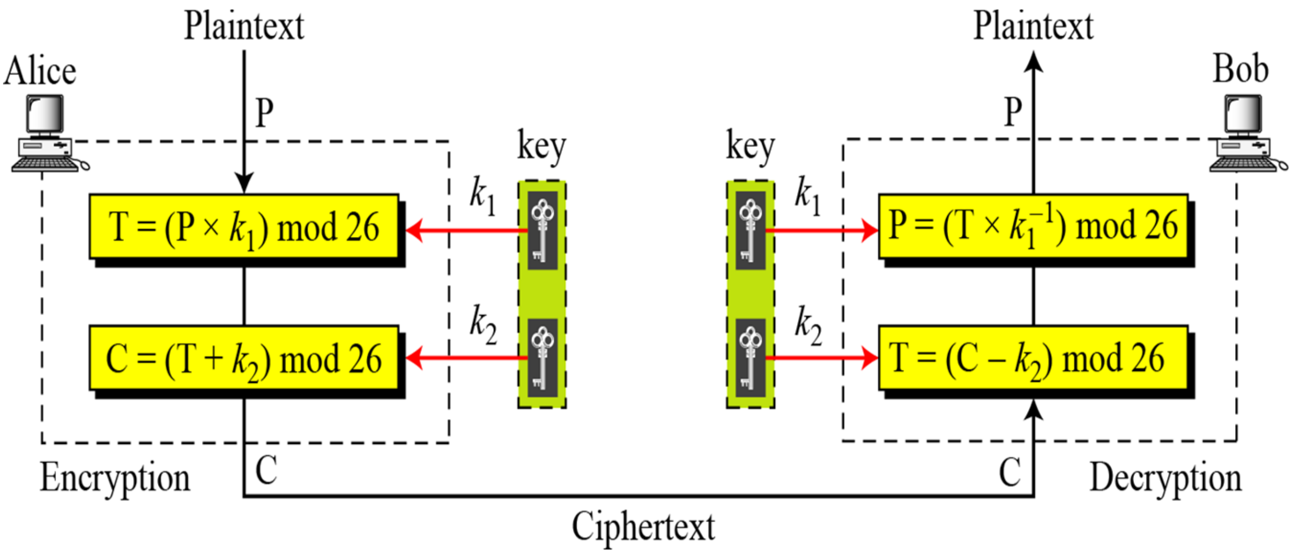
Substitution ciphers are classified as monoalphabetic and polyalphabetic substitution cipher. In monoalphabetic substitution cipher each occurrence of character is encrypted by same substitute character. In Polyalphabetic substitution cipher each occurrence of a character may have a different substitute 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 $1\le a\le m$, $1\le b\le m$

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

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)

<u>Additive cipher</u>

```
def encryption(text , key):
    # print("Encrypted!! with text" + text + " and key " + key)
    cipher = ""
    for letter in text:
        if letter.isupper():
            l = ord(letter) + key
            if l > 90:
                l = 65 + (l - 91)
            cipher += chr(l)
        else:
            l = ord(letter) + key
            if l > 122:
                l = 97 + (l - 123)
            cipher += chr(l)
    text1 = ""
    for letter in cipher:
        if letter.isupper():
            l = ord(letter) - key
            if l < 65:
                l = 91 - (65 - l)
            text1 += chr(l)
        else:
            l = ord(letter) - key
            if l < 97:
                l = 123  - (97 - l)
            text1 += chr(l)
    if text == text1:
        print("Verification complete!! encrypted text is: " , cipher)
        print(cipher)
    else:
        print("Incorrect Cipher")
        print(cipher)
        print(text)
        print(text1)

def decryption(cipher , key):
    text = ""
    for letter in cipher:
        if letter.isupper():
            l = ord(letter) - key
            if l < 65:
                l = 91 - (65 - l)
            text += chr(l)
        else:
            l = ord(letter) - key
            if l < 97:
                l = 123  - (97 - l)
            text += chr(l)
    cipher1 = ""
    for letter in text:
        if letter.isupper():
            l = ord(letter) + key
            if l > 90:
```

```python
                l = 65 + (l - 91)
            cipher1 += chr(l)
        else:
            l = ord(letter) + key
            if l > 122:
                l = 97 + (l - 123)
            cipher1 += chr(l)
    if cipher == cipher1:
        print("Verification complete!! decrypted text is: " , text)
    else:
        print("Incorrect Text")
x = True
while(x):
    print("Enter 1 for encryption \n")
    print("Enter 2 for decryption \n")
    print("Enter 3 to exit \n")
    choice = int(input("Enter your choice: "))

    if choice == 1:
        text = input("Enter the text you want to encrypt: ")
        key = int(input("Enter your key: "))
        encryption(text,key)
    elif choice == 2:
        text = input("Enter the text you want to encrypt: ")
        key = int(input("Enter your key: "))
        decryption(text,key)
    elif choice == 3:
        x = False
    else:
```

Output:

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
PS C:\Users\ISLAB\Desktop\16010421119_INS> & C:/Users/ISLAB/AppData/Local/Programs/Python/Python311/python.exe c:/Users/ISLAB/De
sktop/16010421119_INS/cipher.py
Enter 1 for encryption

Enter 2 for decryption

Enter 3 to exit

Enter your choice: 1
Enter the text you want to encrypt: Aarya
Enter your key: 4
Verification complete!! encrypted text is:  Eevce
Eevce
Aarya
Aarya
Enter 1 for encryption

Enter 2 for decryption

Enter 3 to exit

Enter your choice: 
```

Multiplicative cipher

```
def multiplicative_cipher_encrypt(plaintext, key):
    ciphertext = ""
    for char in plaintext:
        if char.isalpha():
            is_uppercase = char.isupper()
            char_idx = ord(char.upper()) - ord('A')
            encrypted_idx = (char_idx * key) % 26
            encrypted_char = chr(encrypted_idx + ord('A'))
            ciphertext += encrypted_char if is_uppercase else
encrypted_char.lower()
        else:
            ciphertext += char

    return ciphertext


def multiplicative_cipher_decrypt(ciphertext, key):
    def mod_inverse(a, m):
        for x in range(1, m):
            if (a * x) % m == 1:
                return x
        return None

    inverse_key = mod_inverse(key, 26)
    if inverse_key is None:
        raise ValueError(
            "Key does not have a modular inverse. It must be coprime with
26.")

    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            is_uppercase = char.isupper()
            char_idx = ord(char.upper()) - ord('A')
            decrypted_idx = (char_idx * inverse_key) % 26
            decrypted_char = chr(decrypted_idx + ord('A'))
            decrypted_text += decrypted_char if is_uppercase else
decrypted_char.lower()
        else:
            decrypted_text += char
    return decrypted_text
# Example usage
plaintext = "HELLO WORLD"
key = 7
encrypted_text = multiplicative_cipher_encrypt(plaintext, key)
print("Encrypted:", encrypted_text)
decrypted_text = multiplicative_cipher_decrypt(encrypted_text, key)
print("Decrypted:", decrypted_text)
```

}

Output :

```
KQHOQ
PS D:\Github\SEM-5\Information and Network Security (Lab)> & C:/Users/Aarya/python.exe "d:/Github/SEM-5/Information and Network Security (Lab)/EXP 1/multiplicative.py"
Encrypted: XCZZU YUPZV
Decrypted: HELLO WORLD
PS D:\Github\SEM-5\Information and Network Security (Lab)>
```

Questions:

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

Affine Cipher:

- The affine cipher is **slightly more complicated than the Caesar cipher**. • It isa type of simple substitution cipher that is **very easy to crack**. • Affine ciphers can be cracked if any 2 characters are known. As hand ciphers,
  affine ciphers are too complex to be practically applied without the aid of anexplicit lookup table.
- While one could construct an encryption table using an affine map, doing so wouldnot seem to offer any particular advantage over other methods actually used in practice.

Vigenere Cipher

- It is a type of simple substitution cipher that is **very easy to crack**.
- Vigenere ciphers can be cracked if any 2 characters are known.
- Vigenere ciphers are too complex to be practically applied without the aid of anexplicit lookup table
- The Vignere cipher is **slightly more complicated than the Caesar cipher**.

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

ANS2:-

```python
def affine_encrypt(text, key):
    a, b = key
    return ''.join([chr(((a * (ord(t) - ord('A')) + b) % 26) + ord('A')) for t
in text.upper().replace(' ', '')])


def affine_decrypt(cipher, key):
    a, b = key
    a_inv = pow(a, -1, 26)
    return ''.join([chr(((a_inv * (ord(c) - ord('A') - b)) % 26) + ord('A'))
for c in cipher])


def vigenere_encrypt(text, key):
    key = key.upper()
    return ''.join([chr(((ord(text[i]) + ord(key[i % len(key)]) - 2 *
ord('A')) % 26) + ord('A')) for i in range(len(text))])


def vigenere_decrypt(cipher, key):
```

```
    key = key.upper()
    return ''.join([chr(((ord(cipher[i]) - ord(key[i % len(key)]) + 26) % 26)
+ ord('A')) for i in range(len(cipher))])


text = input("Please enter text: ")
a = int(input("Please enter first key for Affine cipher: "))
b = int(input("Please enter second key for Affine cipher: "))
affine_key = (a, b)
vigenere_key = input("Please enter key for Vigenere cipher: ")
affine_cipher = affine_encrypt(text, affine_key)
vigenere_cipher = vigenere_encrypt(text, vigenere_key)
print(affine_cipher)
print(vigenere_cipher)
affine_plain = affine_decrypt(affine_cipher, affine_key)
vigenere_plain = vigenere_decrypt(vigenere_cipher, vigenere_key)
```

Output:

```
LRIPR
Traceback (most recent call last):
  File "d:\Github\SEM-5\Information and Network Security (Lab)\EXP 1\vignere.py", line 31, in <module>
    affine_plain = affine_decrypt(affine_cipher, affine_key)
  File "d:\Github\SEM-5\Information and Network Security (Lab)\EXP 1\vignere.py", line 8, in affine_decrypt
    a_inv = pow(a, -1, 26)
ValueError: base is not invertible for the given modulus
PS D:\Github\SEM-5\Information and Network Security (Lab)> & C:/Users/Aarya/python.exe "d:/Github/SEM-5/Information and Network Security (Lab)/EXP 1/vignere.py"
Please enter text: Aarya
Please enter first key for Affine cipher: 1
Please enter second key for Affine cipher: 2
Please enter key for Vigenere cipher: 1
CCTAC
KQHOQ
PS D:\Github\SEM-5\Information and Network Security (Lab)>
```

Outcomes:

CO 1 Describe the basics of Information Security

Conclusion:

In This experiment additive substitution cipher and multiplicative cipher was learnt and executed in python programming language . Also flaws and advantages of both and how Affine cipher and Vignere cipher are better than additive and multiplicative cipher was learnt .

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