Experiment No. 1

Title: Substitution Cipher

Batch:A3 **Roll No.:16010421073** **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.

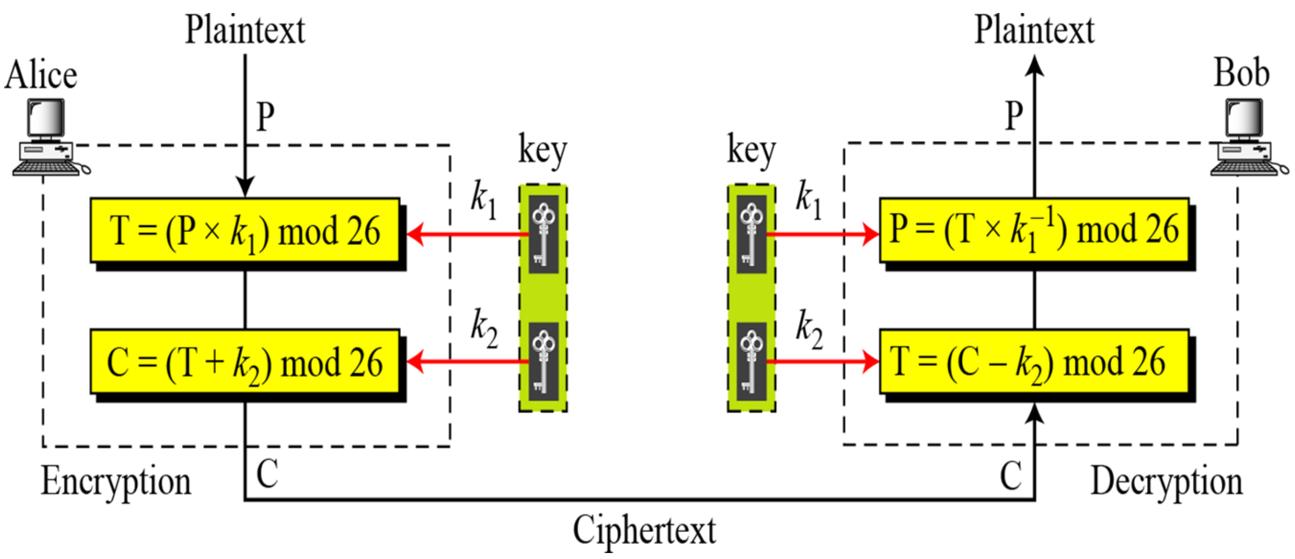
AFFINE CIPHER

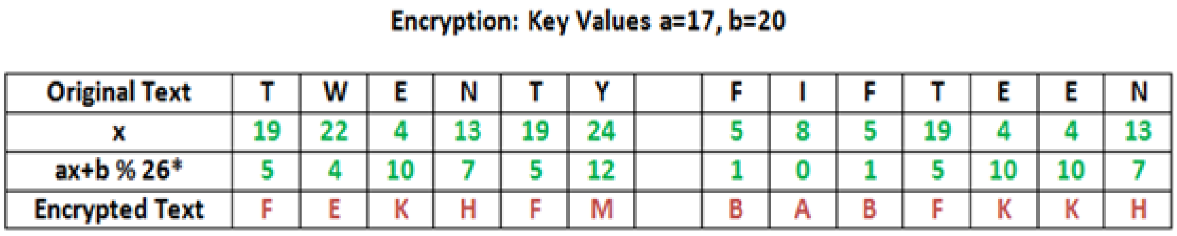
The Affine cipher is a type of monoalphabetic substitution cipher which uses a combination of Additive and Multiplicative Ciphers. Each letter is enciphered with the function (ax + b) mod

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

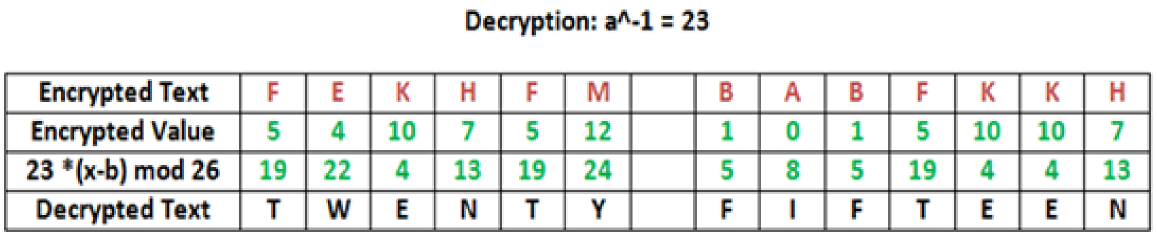
$C=(ax + b) \bmod m$ where $1 \le a \le m$, $1 \le b \le m$

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 $P = a^{-1}(c-b) \bmod m$, 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:

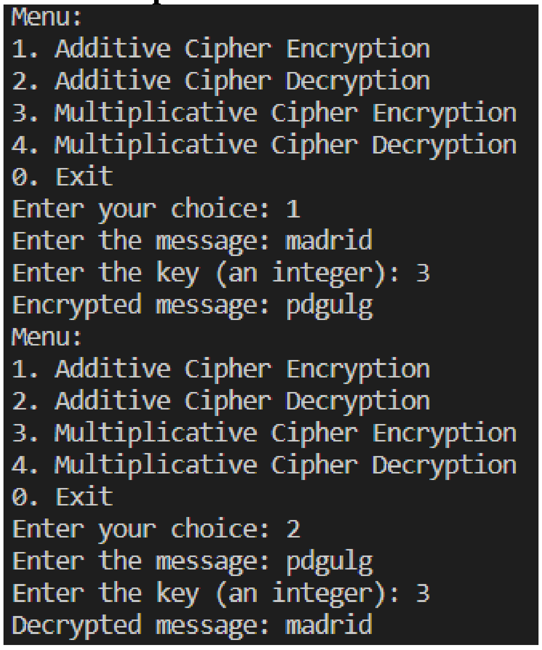
1. Additive Cipher
2. Multiplicative Cipher

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)

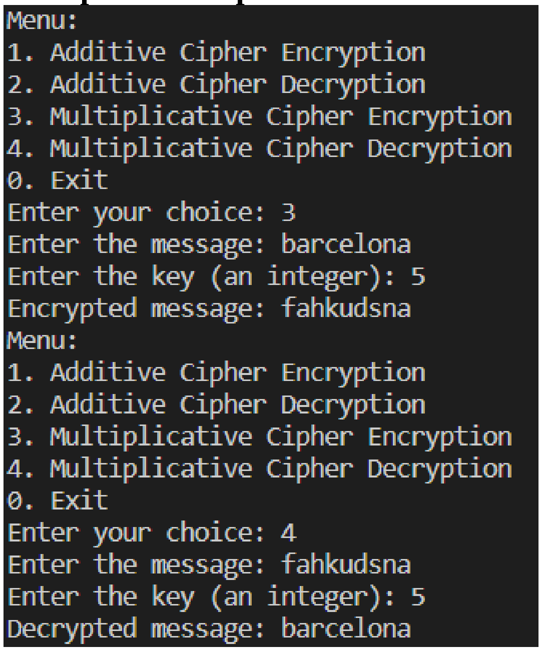
```python
def additive_encrypt(message, key):
    result = ""
    for char in message:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr((ord(char) - base + key) % 26 + base)
            result += encrypted_char
        else:
            result += char
    return result

def additive_decrypt(message, key):
    return additive_encrypt(message, -key)

def multiplicative_encrypt(message, key):
    result = ""
    for char in message:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr(((ord(char) - base) * key) % 26 + base)
            result += encrypted_char
        else:
            result += char
    return result

def multiplicative_decrypt(message, key):


    for i in range(1, 26):
        if (i * key) % 26 == 1:
            inverse_key = i
            break

    return multiplicative_encrypt(message, inverse_key)

if __name__ == "__main__":
    while True:
        print("Menu:")
        print("1. Additive Cipher Encryption")
        print("2. Additive Cipher Decryption")
        print("3. Multiplicative Cipher Encryption")
        print("4. Multiplicative Cipher Decryption")
        print("0. Exit")

        try:
            choice = int(input("Enter your choice: "))

            if choice == 1:
                message = input("Enter the message: ")
                key = int(input("Enter the key (an integer): "))
                print("Encrypted message:", additive_encrypt(message, key))
            elif choice == 2:
                message = input("Enter the message: ")
                key = int(input("Enter the key (an integer): "))
                print("Decrypted message:", additive_decrypt(message, key))
            elif choice == 3:
                message = input("Enter the message: ")
                key = int(input("Enter the key (an integer): "))
                print("Encrypted message:", multiplicative_encrypt(message,
key))
            elif choice == 4:
                message = input("Enter the message: ")
                key = int(input("Enter the key (an integer): "))
                print("Decrypted message:", multiplicative_decrypt(message,
key))
            elif choice == 0:
                print("Exiting...")
                break
            else:
                print("Invalid choice! Please enter a number from the menu.")
        except ValueError:
            print("Invalid input! Please enter a valid choice as a number.")
```

Output:
Additive cipher:



```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
0. Exit
Enter your choice: 1
Enter the message: madrid
Enter the key (an integer): 3
Encrypted message: pdgulg
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
0. Exit
Enter your choice: 2
Enter the message: pdgulg
Enter the key (an integer): 3
Decrypted message: madrid
```

Multiplicative Cipher:



```
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
0. Exit
Enter your choice: 3
Enter the message: barcelona
Enter the key (an integer): 5
Encrypted message: fahkudsna
Menu:
1. Additive Cipher Encryption
2. Additive Cipher Decryption
3. Multiplicative Cipher Encryption
4. Multiplicative Cipher Decryption
0. Exit
Enter your choice: 4
Enter the message: fahkudsna
Enter the key (an integer): 5
Decrypted message: barcelona
```

Questions:

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

Ans: **Additive Cipher (Caesar Cipher):-**

- **Lack of Key Space:**
  The Additive Cipher has a very limited key space since it only uses a single parameter - the shift value. The key space is equal to the size of the alphabet being used, which makes it susceptible to brute-force attacks.
- **Weak Security:**
  The Additive Cipher can be easily broken with modern computational power, making it unsuitable for secure communication. Brute-force attacks or simple statistical methods can quickly reveal the plaintext.
- **Lack of Encryption Strength:**
  The Additive Cipher is a monoalphabetic substitution cipher, meaning that each letter in the plaintext is always substituted with the same corresponding letter in the ciphertext. This makes it relatively weak and easy to crack.

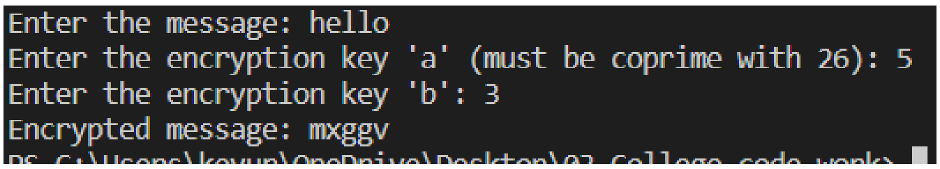
Multiplicative Cipher:

- **Limited Key Space:**
  The Multiplicative Cipher also suffers from a limited key space. The only parameter used is the multiplier, which must be coprime with the size of the alphabet. The small key space makes it vulnerable to brute-force attacks.
- **Vulnerable to Known Plaintext Attacks:**
  If an attacker obtains knowledge of the plaintext and the corresponding ciphertext, they can deduce the multiplier used in the cipher by performing a simple mathematical operation. This weakness is known as a known-plaintext attack.
- **Noisy Encryption:**
  The Multiplicative Cipher can produce ciphertexts with large numbers, making the output potentially noisy and more challenging to transmit or handle in certain systems.

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

Affine cipher:

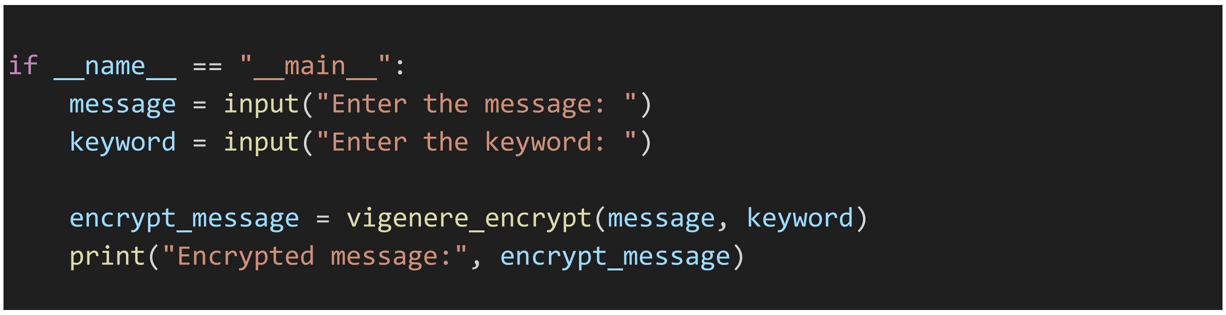
```
def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return None

def affine_encrypt(message, a, b):
    result = ""
    m = 26

    for char in message:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            encrypted_char = chr(((ord(char) - base) * a + b) % m + base)
            result += encrypted_char
        else:
            result += char

    return result

if __name__ == "__main__":
    message = input("Enter the message: ")
    a = int(input("Enter the encryption key 'a' (must be coprime with 26): "))
    b = int(input("Enter the encryption key 'b': "))

    encrypted_message = affine_encrypt(message, a, b)
    print("Encrypted message:", encrypted_message)
```



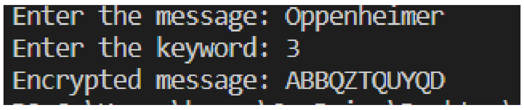
```
Enter the message: hello
Enter the encryption key 'a' (must be coprime with 26): 5
Enter the encryption key 'b': 3
Encrypted message: mxggv
PS C:\Users\kavun\OneDrive\Desktop\02 College code work>
```

Vignere Cipher:

```
def vigenere_encrypt(message, keyword):
    message = message.upper()
    keyword = keyword.upper()
    encrypt_message = ""

    keyword_index = 0
    for char in message:
        if char.isalpha():
            shift = ord(keyword[keyword_index]) - ord('A')
            encrypted_char = chr((ord(char) - ord('A') + shift) % 26 +
ord('A'))
            encrypt_message += encrypted_char
            keyword_index = (keyword_index + 1) % len(keyword)
        else:
            encrypt_message += char

    return encrypt_message
```



```python
if __name__ == "__main__":
    message = input("Enter the message: ")
    keyword = input("Enter the keyword: ")

    encrypt_message = vigenere_encrypt(message, keyword)
    print("Encrypted message:", encrypt_message)
```



```
Enter the message: Oppenheimer
Enter the keyword: 3
Encrypted message: ABBQZTQUYQD
```

Outcomes:
CO1 : Describe the basics of Information Security.

Conclusion:
Thus we successfully implemented menu driven program for additive cipher and multiplicative cipher.

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