Experiment 1

Date of Performance : 22/03/22 **Date of Submission:** 23/03/22

SAP Id: 60004190054 **Name : Jazib Dawre**

Div: A **Batch : A4**

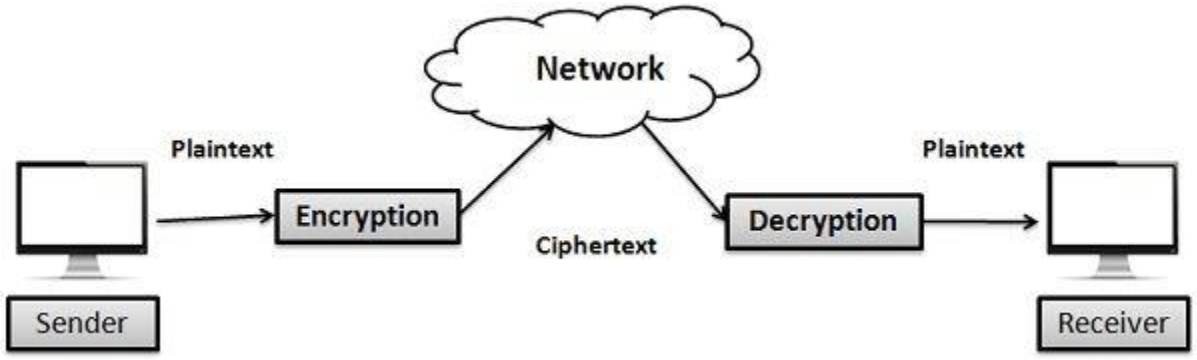
Aim of Experiment

Design and Implement Encryption and Decryption Algorithm for

a. Caesar cipher cryptographic algorithm by considering letter [A..Z] and digits [0..9]. Create two functions Encrypt() and Decrypt(). Apply Brute Force Attack to reveal secret. Create Function BruteForce(). Demonstrate the use of these functions on any paragraph.
b. Hill Cipher. Your Program Must Input Image in Gray Scale. Choose keys according to Gray Scale Intensity level. Create two functions Encrypt() and Decrypt(). Make sure to have Multiplicative Inverse Exists for one of the Key in selected Key pair of Hill Cipher. (CO1)

Theory / Algorithm / Conceptual Description

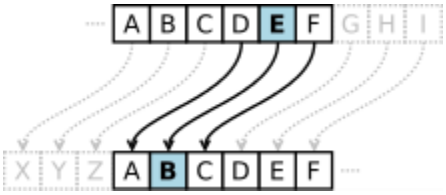
Encryption and Decryption



Encryption is the process by which a readable message is converted to an unreadable form to prevent unauthorized parties from reading it. Decryption is the process of converting an encrypted message back to its original (readable) format. The original message is called the plaintext message. The encrypted message is called the ciphertext message.

Digital encryption algorithms work by manipulating the digital content of a plaintext message mathematically, using an encryption algorithm and a digital key to produce a ciphertext version of the message. The sender and recipient can communicate securely if the sender and recipient are the only ones who know the key.

Caesar's Cipher



Caesar cipher is one of the simplest and most widely known encryption techniques. It is a type of substitution cipher in which each letter in the plaintext is replaced by a letter some fixed number of positions down the alphabet. For example, with a left shift of 3, D would be replaced by A, E would become B, and so on.

The encryption can also be represented using modular arithmetic by first transforming the letters into numbers, according to the scheme, A → 0, B → 1, ..., Z → 25.

Encryption of a letter x by a shift n can be described mathematically as

$$E_n(x) = (x + n) \mod 26.$$

Decryption is performed similarly,

$$D_n(x) = (x - n) \mod 26.$$

The replacement remains the same throughout the message, so the cipher is classed as a type of monoalphabetic substitution.

Hill Cipher

Hill cipher is a polygraphic substitution cipher based on linear algebra. Each letter is represented by a number modulo 26. Though this is not an essential feature of the cipher, this simple scheme is often used:

Letter	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
Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25

To encrypt a message, each block of n letters (considered as an n-component vector) is multiplied by an invertible n × n matrix, against modulus 26.

$$CT = K \times PT \mod 26$$

To decrypt the message, each block is multiplied by the inverse of the matrix used for encryption.

$$PT = K^{-1} \times CT \mod 26$$

The matrix used for encryption is the cipher key, and it should be chosen randomly from the set of invertible n × n matrices (modulo 26). The key matrix K should also satisfy for its determinant d = det(K) that gcd(|d|, 26) = 1.

Program

A)

```python
import sys


def shift_character(c: str, bits: int) -> str:
    if ord(c) < 65:
        # Numbers
        return chr(((ord(c) - 48 + bits) % 10) + 48)
    else:
        # Uppercase Alphabets
        return chr(((ord(c) - 65 + bits) % 26) + 65)


def encrypt(plain_text: str, key: int):
    return "".join(shift_character(x, key) for x in plain_text)


def decrypt(cipher_text: str, key: int):
    return "".join(shift_character(x, -key) for x in cipher_text)


def brute_force(cipher_text: str):
    for i in range(26):
        print(f"Key: {i+1} | PT: {decrypt(cipher_text, i+1)}")


if __name__ == "__main__":
    plain_text = sys.argv[1].upper()
    n = int(sys.argv[2])

    print("\n> Caesar's Cipher\n")
    print(f"Plain Text: {plain_text}", end="\n\n")
    print(f"Key: {n}", end="\n\n")

    cipher_text = encrypt(plain_text, n)
    print(f"Cipher Text: {cipher_text}", end="\n\n")

    print("Brute Force: ")
    brute_force(cipher_text)
    print("")

    deciphered_text = decrypt(cipher_text, n)
    print(f"Deciphered Text: {deciphered_text}")
```

Output

```
> Caesar's Cipher

Plain Text: IAMJAZIB

Key: 3

Cipher Text: LDPMDCLE

Brute Force:
Key: 1 | PT: KCOLCBKD
Key: 2 | PT: JBNKBAJC
Key: 3 | PT: IAMJAZIB
Key: 4 | PT: HZLIZYHA
Key: 5 | PT: GYKHYXGZ
Key: 6 | PT: FXJGXWFY
Key: 7 | PT: EWIFWVEX
Key: 8 | PT: DVHEVUDW
Key: 9 | PT: CUGDUTCV
Key: 10 | PT: BTFCTSBU
Key: 11 | PT: ASEBSRAT
Key: 12 | PT: ZRDARQZS
Key: 13 | PT: YQCZQPYR
Key: 14 | PT: XPBYPOXQ
Key: 15 | PT: WOAXONWP
Key: 16 | PT: VNZWNMVO
Key: 17 | PT: UMYVMLUN
Key: 18 | PT: TLXULKTM
Key: 19 | PT: SKWTKJSL
Key: 20 | PT: RJVSJIRK
Key: 21 | PT: QIURIHQJ
Key: 22 | PT: PHTQHGPI
Key: 23 | PT: OGSPGFOH
Key: 24 | PT: NFROFENG
Key: 25 | PT: MEQNEDMF
Key: 26 | PT: LDPMDCLE

Deciphered Text: IAMJAZIB
```

Program

B)

```
import sys
import cv2
import numpy as np


def random_key_matrix(n):
    Mod = 256
    k = 23

    d = np.random.randint(256, size=(int(n / 2), int(n / 2)))
    identity = np.identity(int(n / 2))
    a = np.mod(-d, Mod)

    b = np.mod((k * np.mod(identity - a, Mod)), Mod)
    k = np.mod(np.power(k, 127), Mod)
    c = np.mod((identity + a), Mod)
    c = np.mod(c * k, Mod)

    A1 = np.concatenate((a, b), axis=1)
    A2 = np.concatenate((c, d), axis=1)
    A = np.concatenate((A1, A2), axis=0)

    return A, A


def get_key(size: int) -> np.ndarray:
    key, key_inv = random_key_matrix(size)
    cv2.imwrite("Key.png", key)
    return key, key_inv


def load_image(filename: str) -> np.ndarray:
    base_img = cv2.imread(filename)
    return cv2.cvtColor(base_img, cv2.COLOR_RGB2GRAY)


def encrypt(key: np.ndarray, base_img: np.ndarray) -> np.ndarray:
    encrypted_img = np.matmul(key, base_img) % 256
    cv2.imwrite("encrypted.png", encrypted_img)
    return encrypted_img
```

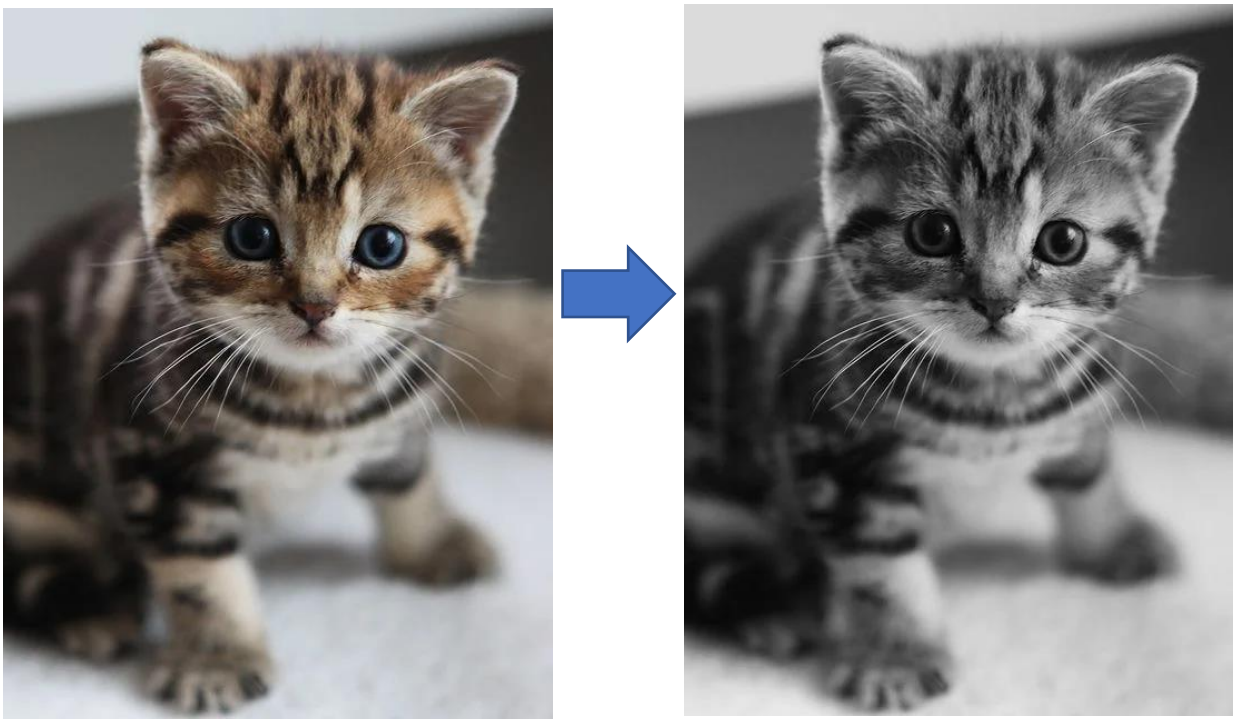
```python
def decrypt(key_inv: np.ndarray, encrypted_img: np.ndarray) -> np.ndarray:
    decrypted_img = np.matmul(key_inv, encrypted_img) % 256
    cv2.imwrite("decrypted.png", decrypted_img)
    return decrypted_img


if __name__ == "__main__":
    original = load_image(sys.argv[1])
    key, key_inv = get_key(original.shape[0])

    encrypted_img = encrypt(key, original)
    decrypted_img = decrypt(key_inv, encrypted_img)
```

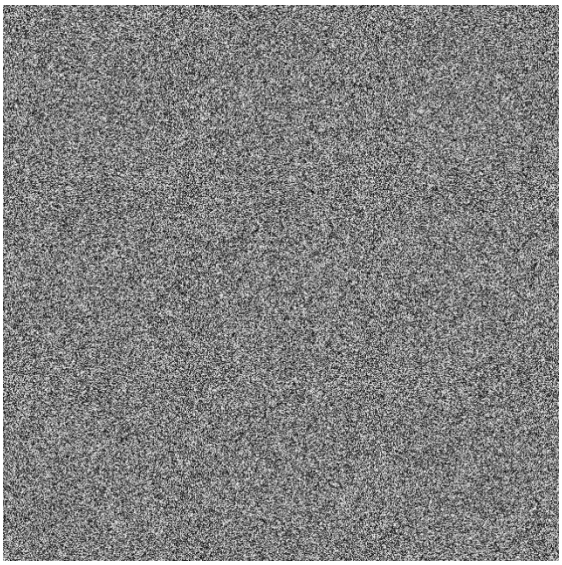
Output

1. RGB to Grayscale



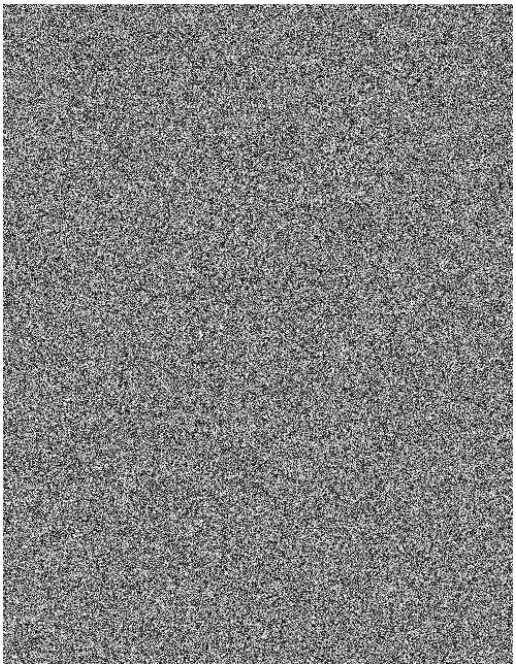
Original Image Grayscale Image

2. Encryption



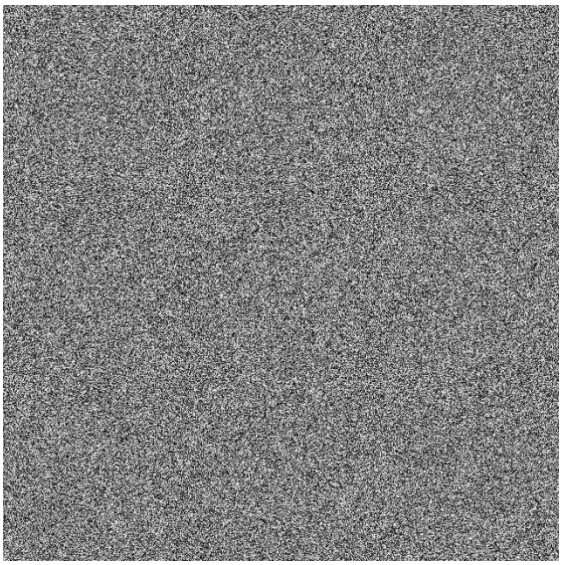
Key Matrix

Grayscale Image

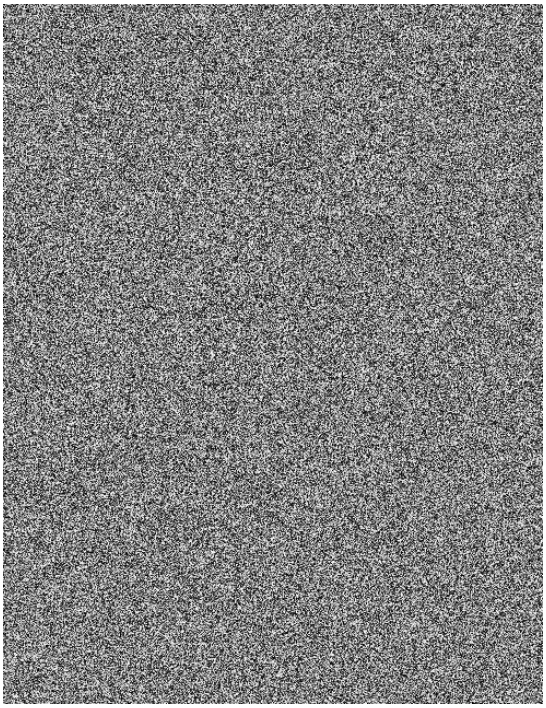


Encrypted Image

3. Decryption



Inverse Key Matrix



Encrypted Image

Decrypted Image

Conclusion

Encryption methods allow for obfuscation of data when it is vulnerable while being transported over various channels. Key based encryption algorithms are quite prominent which rely on a pre-shared key between the sender and the recipient. The sender encrypts the data using the key and the receiver decrypts it with the key. Attackers without the key should not be able to decrypt and understand the data. Caesar's cipher is a simple form of key based cipher which moves the characters in the message by a key value in a circular fashion. The receiver with the correct key then moves the characters back to decrypt the message. The hill cipher is a more complex and performs matrix multiplication and algebraic operations on the plain text to encrypt the data. Thus, it is more resilient to brute force attacks but is still vulnerable to known-plain-text attacks, i.e., the key can be inferred if the plain text and cipher text are both available.