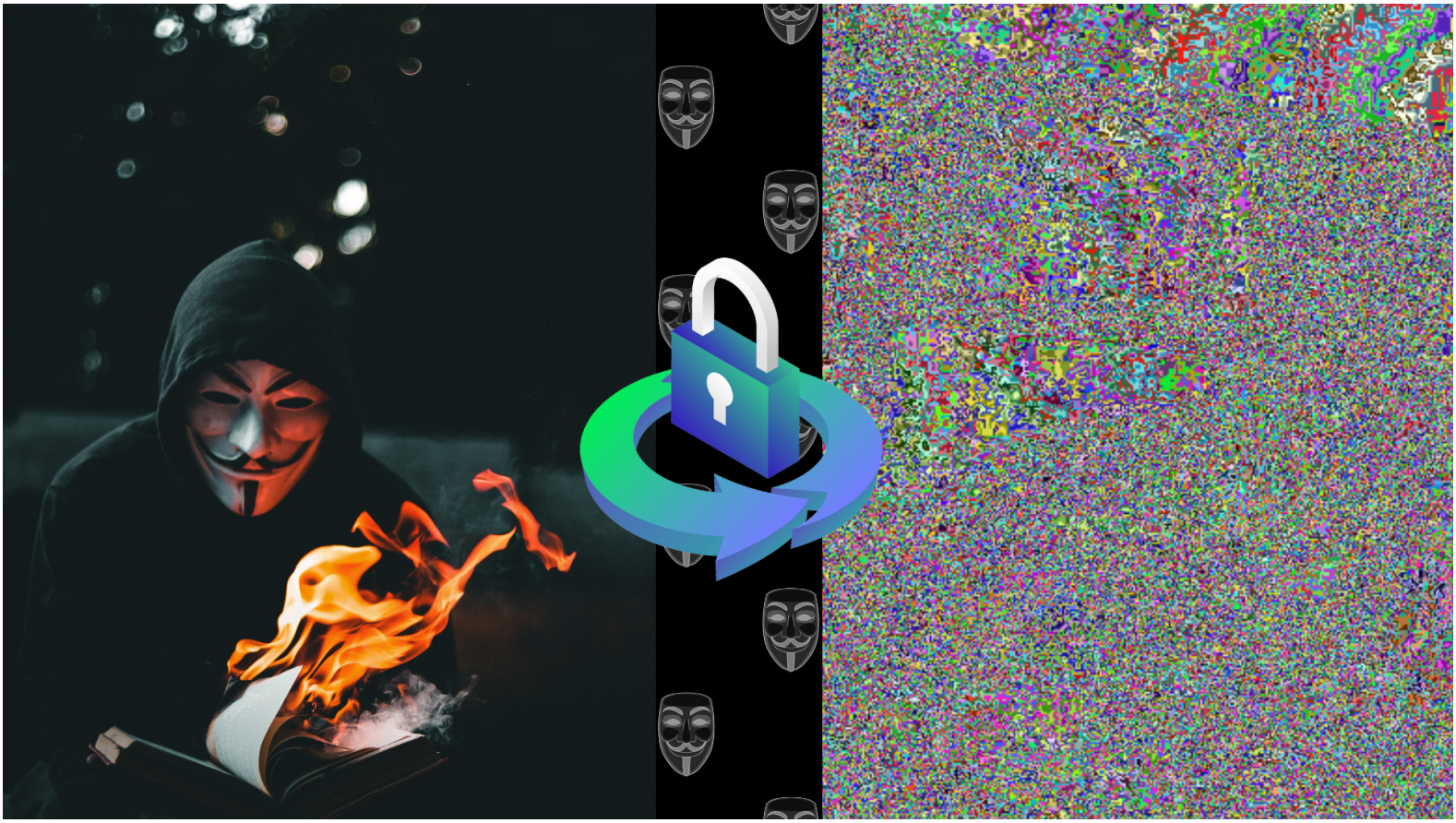


Hill Cipher Image Encryption & Decryption Algorithm

Linear Algebra Project

Project Team Members

Muhammad Mobeen (200901097)

Muhammad Awais Afzal (200901037)

Presented to: Sir Majid Khan

Abstract

Note: The following paper explores the Hill Cipher Algorithm and all of it's implementation is written on Python and hence includes Python concepts.

Secure Image transmission on internet has become an important requirement these days. For secure transmission we use the cryptography which plays a important role in the security for communication among various channels. By the techniques involved in cryptography we can transform readable form of message (plaintext) into an unreadable form (cipher text) and vice-versa. In these days cryptography is a part of network security which can be used for encryption of text, audio, video, graphics and other multimedia files. Here we are talking about the image encryption which provides an original image into the encrypted image.

Hill Cipher

The Hill cipher algorithm is one of the symmetric key algorithms that have several advantages in data encryption. But, the inverse of the key matrix used for encrypting the plaintext does not always exist. Then if the key matrix is not invertible, then encrypted text cannot be decrypted. In the Involutory matrix generation method the key matrix used for the encryption is itself invertible. So, at the time of decryption we need not to find the inverse of the key matrix. The objective of this paper is to encrypt an image using a technique different from the conventional Hill Cipher. In this paper we have implemented a novel and advanced method to encrypt and decrypt the images.

Hill (AdvHill) encryption technique has been proposed which uses an involutory key matrix. The scheme is a fast encryption scheme which overcomes problems of encrypting the images with homogeneous background. A comparative study of the proposed encryption scheme and the existing scheme is made. The output encrypted images reveal that the proposed technique is quite reliable and robust.

Goals

1. To encrypt images of variable formats (png, jpg, bitmap, etc) and sizes.
2. Encryption should be done with valid inversible keys.

Problem

Since we have implemented Hill Cipher on Python so we ran into multiple problems and following points out the major ones.

1. A color image has mostly 3 channels of information (R,G,B) that we needed to encrypt. In this case we cannot use a normal 2x2 Hill Cipher Algorithm, so we have to implement 3x3 Hill Cipher Algorithm which is much more complex than the standard one.
2. Image conversions to and from Numpy Arrays.
3. Convertible ASCII key and a system to validate if the key is inversible or not.
4. Writing custom code for taking inverse of key and adjoint in decryption process.

Methodology

Following discuses the process of developing code for encryption and decryption using Hill Cipher Algorithm.

Encryption

Encryption process is was easy to implement as we have to simply implement the following formula:-

C = Ek (P) mod 256

C = Ciphered Pixel (Each C represents a single pixel as it contains the complete information in python list form i.e [R, G, B].

Ek = Encryption Key (Input from user and converted to ASCII values)

P = Unencrypted Pixel Coloumn Vector

mod 256 = As the range of RGB values is 0-255, hence mod 256

Following code snippet shows encryption code.

```
def encrypt(path): #Takes an image and returns an encrypted image

    col_vecls = img_to_colvecls(path)

    encrypted_vecls = []

    for i in col_vecls:

        encrypted_vecls.append((i @ Key)%moder)

    encrypted_img = colvecls_to_img(encrypted_vecls, path)

    encrypted_img.save("encrypted.png")
```

Decryption

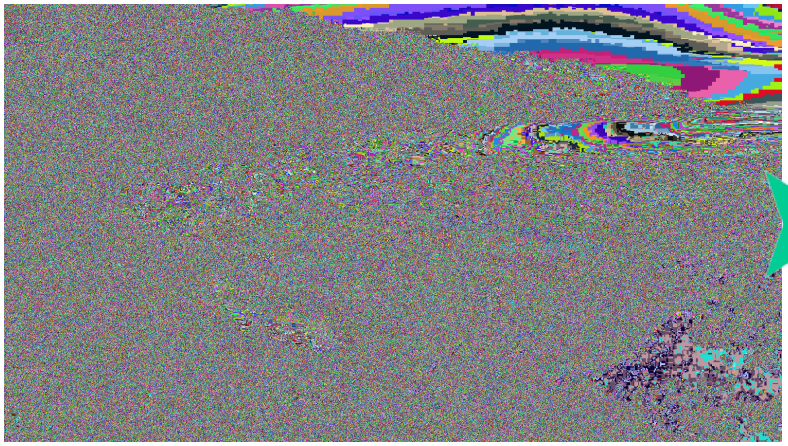
Decryption process was more complex to develop and certainly very intuitive too. We first needed to develop an efficient way to generate key inverses and validate them if they are inversible or not. Now let's dive in the intuition behind it.

There was no readily available off the shelf functions from Numpy or any python math libraries which can calculate inverse of key successfully. In theory they could but they were returning float values which is very un idealistic for our case as float values create ambiguities which can make our Hill Cipher Algorithm highly unaccurate. So we had to develop our own function to formulate the key inverses and also checks if it even exists or not. The other custom algorithm we had to develop was to take adjoint of the key.

After solving these problems it was easy to implement decryption as it was just like assembling a toy. Following is the code snippet that rerpresents decryption function. Note it does not include the dependencies.

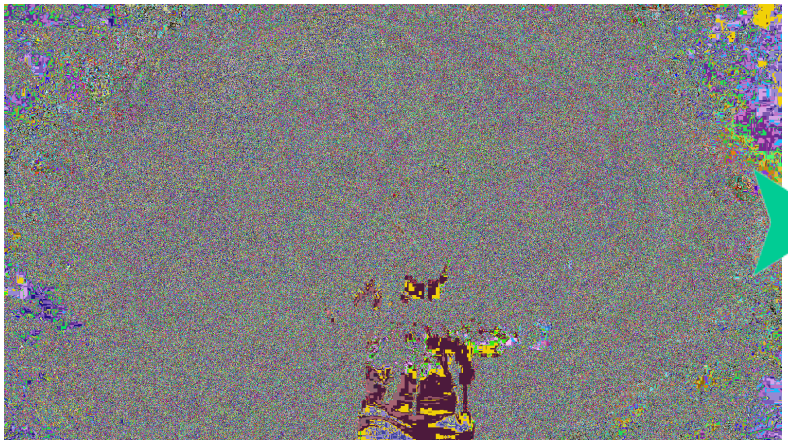
```
def decrypt(path):

    col_vecls = img_to_colvecls(path)

    key_inv = Key_Inverse()

    decrypted_vecls = []

    for colvec in col_vecls:

        decrypted_vecls.append((colvec @ key_inv)%moder)

    decrypted_img = colvecls_to_img(decrypted_vecls, path)

    decrypted_img.save("decrypted.png")
```

Showcase



Encrypted_Paradice.png

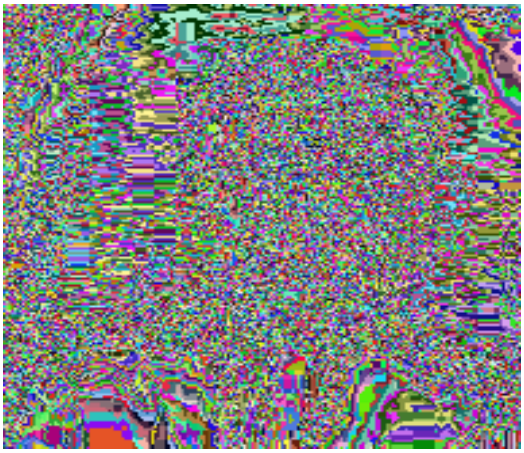
Decrypted_Paradise.png



Encrypted_Paradice.png

Decrypted_Paradise.png

Sir_Majid.jpg



Encrypted_Sir_Majid.png

Project Files

Full project is available as opensource on Github, along with the Image encryption algorithms we have developed two more algorithms for conversion of “ASCII” and “Plain Text” messages respectively. Following are the links:-

1. Hill Cipher For Images: github.com/muhammad-mobeen/Hill-Cipher-ImageX
2. Hill Cipher For ASCII: github.com/muhammad-mobeen/Hill-Cipher-ASCII
3. Hill Cipher For Plain Text: github.com/muhammad-mobeen/Hill-Cipher-Plain

Note: I will be keeping these projects updated as I am planing to add some more features to it, so do visit my Github :)

Source Code

```
import numpy as np
import cv2
from PIL import Image
import matplotlib.image as img
from os import system as bas

moder = int(256)

#Image to matrix converter
def img_to_colvecls(path):
    img = cv2.imread(path, cv2.IMREAD_UNCHANGED)
    img = cv2.cvtColor(img, cv2.COLOR_BGRA2RGBA)
    rows = img.shape[0]
    cols = img.shape[1]
    col_vecls = []
    for i in range(0,rows):
        for j in range(0,cols):
            col_vecls.append(np.array([img[i][j][0], img[i][j][1], img[i][j][2]]))
    return col_vecls
```

```
def colvecls_to_img(colvecls, path):
    img = cv2.imread(path, cv2.IMREAD_UNCHANGED)
    img = cv2.cvtColor(img, cv2.COLOR_BGRA2RGBA)
    rows = img.shape[0]
    cols = img.shape[1]
    k = 0
    for i in range(0,rows):
        for j in range(0,cols):
            img[i][j][0] = colvecls[k][0]
            img[i][j][1] = colvecls[k][1]
            img[i][j][2] = colvecls[k][2]
            # img[i][j][3] = 255
            k+=1
    return Image.fromarray(img,'RGBA')

def conv_ascii(elm):
    if type(elm) == int:
        return chr(elm+129)
    elif type(elm) == str:
        return ord(elm)-129
    else:
        print("Wierd data type found: ", type(elm))
        bas("pause")

def Key_Generator():
    key_str = "Xenoblast"
    return np.array([conv_ascii(s) for s in key_str]).reshape(3,3)

Key = Key_Generator()

def does_inverse_exist():
    det_key = int(np.linalg.det(Key))%moder
    try:
        inv_mod = pow(det_key, -1, moder)
    except:
        return False
    return True
```

```
def verify_key_inv(key_inv): #Verify if the result is identity matrix. Then good to go!
    if np.array_equal((Key@key_inv)%moder,np.identity(3)):
        return True
    else:
        return False

def adjointer(matrix):
    mtrx = matrix.ravel()  #ravel() converts 2d array to 1d
    A= +((mtrx[4]*mtrx[8])-(mtrx[5]*mtrx[7]))
    B= -((mtrx[3]*mtrx[8])-(mtrx[5]*mtrx[6]))
    C= +((mtrx[3]*mtrx[7])-(mtrx[6]*mtrx[4]))
    D= -((mtrx[1]*mtrx[8])-(mtrx[2]*mtrx[7]))
    E= +((mtrx[0]*mtrx[8])-(mtrx[2]*mtrx[6]))
    F= -((mtrx[0]*mtrx[7])-(mtrx[1]*mtrx[6]))
    G= +((mtrx[1]*mtrx[5])-(mtrx[2]*mtrx[4]))
    H= -((mtrx[0]*mtrx[5])-(mtrx[2]*mtrx[3]))
    I= +((mtrx[0]*mtrx[4])-(mtrx[1]*mtrx[3]))
    cofactor = np.array([[A, B, C],
                         [D, E, F],
                         [G, H, I]])
    adjnt = cofactor.T
    return adjnt #convert back to 2d array + transpose

def Key_Inverse():
    det_key = int(np.linalg.det(Key))%moder
    det_inv_mod = pow(det_key, -1, moder)
    adj_key = adjointer(Key)%moder
    key_inv = (det_inv_mod*adj_key)%moder
    return key_inv

def is_key_valid():
    mod_inv = does_inverse_exist()
    key_inv = False #Default value
    if mod_inv:
        key_inv = verify_key_inv(Key_Inverse())
        if key_inv:
            print("Key Verified Successfully!")
```

```
            return True
    print("Error: Key failed to get verified :(")
    print("--> Inverse_Mod_Exists: ",mod_inv)
    print("--> Key_Inverse_Exists: ",key_inv)
    print("\nYour Key is not compatible. Please change the key!\n")
    return False


def encrypt(path): #Takes an image and returns an encrypted image
    col_vecls = img_to_colvecls(path)
    encrypted_vecls = []
    for i in col_vecls:
        encrypted_vecls.append((i @ Key)%moder)
    encrypted_img = colvecls_to_img(encrypted_vecls, path)
    encrypted_img.save("encrypted.png")


def decrypt(path):
    col_vecls = img_to_colvecls(path)
    key_inv = Key_Inverse()
    decrypted_vecls = []
    for colvec in col_vecls:
        decrypted_vecls.append((colvec @ key_inv)%moder)
    decrypted_img = colvecls_to_img(decrypted_vecls, path)
    decrypted_img.save("decrypted.png")


if __name__ == "__main__":
    if not is_key_valid():
        pass
    else:
        key_str = ""
        for i in Key.ravel():
            key_str += conv_ascii(int(i))
        print("Key:",key_str)
        encrypt(str(input("Enter Image path: ")))
        print("Encryption Done!")
        decrypt("encrypted.png")
```

References

1) https://www.researchgate.net/publication/229012891_Image_Encryption_Using_Advanced_Hill_Cipher_Algorithm.
2) A Secure Image Encryption Algorithm Based on Hill Cipher SystemBuletin Teknik Elektro dan Informatika (Bulletin of Electrical Engineering and Informatics) Vol.1, No.1, March 2012, pp. 51~60 ISSN: 2089-3191
3) “IMAGE ENCRYPTION USING THE STANDARD HILL CIPHER”:https://www.ijarcs.info/index.php/Ijarcs/article/view/149