**Image Cryptography Based on Rubix's Cube Principle**

Implementation of image encryption and decryption using Rubix's Cube Principle. This algorithm is based on the paper ["A Secure Image Encryption Algorithm Based on Rubik's Cube Principle"](https://www.hindawi.com/journals/jece/2012/173931/) by Khaled Loukhaoukha, Jean-Yves Chouinard and Abdellah Berdai.

**Algorithm Overview**

Given an input image having the three R,G,B matrices of size M X N Hyperparameters include α - used for vector creation ITER\_MAX - maximum number of times to carry out operations

**A. Encyption**

1. Create two vectors Kr and Kc with |Kr|=M & |Kc|=N. The values of these vectors are randomly picked from 0 to

2α -1

2. Repeat below steps ITER\_MAX number of times

i. **Rolling Rows:**

◦ The sum of all pixel values of every row of the image RGB matrices are calculated one by one.

◦ If the sum of a given row rowNumber is even, Roll the row to the right Kr[rowNumber] times Otherwise roll to the left Kr[rowNumber] times.

ii. **Rolling Columns:**

◦ The sum of all pixel values of every column of the image RGB matrices are calculated one by one.

◦ If the sum of a given row columnNumber is even, roll the column up Kc[columnNumber] times. Otherwise roll the column down Kc[columnNumber] times.

iii. **XORing Pixels:**

◦ For every pixel(i,j), XOR the pixel with the below two values

- Value #1 - `Kc[columnNumber]` if `i` is odd else 180 rotated bit version of `Kc[columnNumber]`

- Value #2 - `Kr[rowNumber]` if `j` is even else 180 rotated bit version of `Kr[rowNumber]`

**B. Decryption**

Given an encrypted image, vectors Kr and Kc & ITER\_MAX , decryption can be done by following the reverse procedure - XORing pixels → Rolling Columns → Rolling Rows ITER\_MAX number of times

**Prerequisites**

* Python3 ( <https://www.python.org/downloads/> )
* Python3 package dependencies - Run pip3 install -r requirements.txt

**Running**

1. Using the crypto\_client.py script supplying neccessary parameters

$ python3 crypto\_client.py -h

usage: crypto\_client.py [-h] [--type TYPE] [--image IMAGE]

[--alpha ALPHA] [--iter\_max ITER\_MAX]

[--key KEY] [--output\_image OUTPUT\_IMAGE]

2. Using rubikencryptor python package

from rubikencryptor.rubikencryptor import RubikCubeCrypto from PIL import Image

# Encrypt image

input\_image = Image.open('image1.png')

encryptor = RubikCubeCrypto(input\_image)

encrypted\_image = encryptor.encrypt(alpha=8, iter\_max=10, key\_filename='key.txt') encrypted\_image.save('encrypted\_image.png')

# Decrypt image

decryptor = RubikCubeCrypto(encrypted\_image)

decrypted\_image = decryptor.decrypt(key\_filename='key.txt') decrypted\_image.save('decrypted\_image.png')

**Example -**

Original Image



Run Encryption on the Original Image

$ python3 crypto\_client.py --type encrypt

--image example/original.png

--output\_image example/encrypted.png

--key example/encoded\_key.txt

--alpha 8 --iter\_max 10

encrypted image is stored at example/encrypted.png & key is stored at example/encoded\_key.txt

Encrypted Image



Run Decryption on the Encryped Image using the Key

$ python3 crypto\_client.py --type decrypt

--image example/encrypted.png

--output\_image example/decrypted.png

--key example/encoded\_key.txt

decrypted image is stored at example/decrypted.png

Decrypted Image –

