Image Encryption-Decryption

Abstract:

Security and privacy in today's world are extremely necessary. Our data online passes through several sites and servers and is shared with third party owners without our consent or permission. In times like this encrypting our data can be of crucial importance. Our project focuses on encrypting the image so that it can't be misused by anyone online. Our python code will encrypt/decrypt the image file in the format of jpeg, jpg, png etc. using hill cypher algorithm. We have tried to develop a cryptography system for the grayscale and the RBG image. The following project is very fast easy to use and the image can be encrypted and decrypted within mere seconds.

Our Approach:

As mentioned above we have used hill cypher algorithm but now we will go in depth of the topic with and will explain the use of linear algebra in the project. The main important thing while working with the cryptographic system is generation of the key which has to be secretive and known by both the parties doing encryption/decryption. The following algorithm is a symmetric key algorithm which means that the key used for encryption/decryption will be same. The idea here is to convert our image into a matrix form and then multiply the image matrix with a key which we will calculate. Here the pixels are multiplied with the key matrix value and hence the original image is hidden completely.

Implementation: It takes ***m*** successive plain pixels and substitutes them with ***m*** ciphered pixels. For example, if we have three pixels P1, P2, P3, where P is our pixel matrix Hill cipher algorithm will substitute them by C1, C2, C3 where C is our encrypted matrix according to the following equations:

$$C_1=(K_{11}P_1+K_{12}P_2+K_{13}P_3)\text{modrange}$$

$$C_2=(K_{21}P_1+K_{22}P_2+K_{23}P_3)\text{modrange}$$

$$C_3=(K_{31}P_1+K_{32}P_2+K_{33}P_3)\text{modrange}$$

This can be represented by matrices as following:

$$\begin{bmatrix} C_1 \\ C_2 \\ C_3 \end{bmatrix} = \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix}$$

This can be written as **C = KP**
And for decryption the original image can obtained back by doing the following step:

$P = K^{-1} C$

One of the most important thing to notice here is that our key matrix should always be invertible for encryption process to work.

Generation of Key Matrix:
The key matrix generated must be invertible and by the following steps the key will always come out to be invertible. The generated key matrix for Hill cipher algorithm will be involutory, which means that the key matrix is its own inverse e.g. **$K^2 = I$**

Suppose there is an N x N matrix,

$$\text{Let } \mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & .. & .. & .. & .. & a_{1n} \\ a_{21} & a_{22} & .. & .. & .. & .. & a_{2n} \\ .. & .. & .. & .. & .. & .. & .. \\ a_{n1} & a_{n2} & .. & .. & .. & .. & a_{nn} \end{bmatrix} \text{be } n \times n$$

we partition them in 4 small matrix each of order N/2 x N/2

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Since **$A^2 = I$,** then we can deduce the following:

- **$A_{12}A_{21} = I - A^2_{11} = (I-A_{11})\ (I+A_{11})$** (Equation 1)
- **$A_{11} + A_{22} = 0$** **(**Equation 2**)**

So, to generate an involuntary matrix **A**:

- A_{22} can be (N/2) x (N/2) matrix, so we take it randomly
- From equation 2: $A_{11} = -A_{22}$
- **Let $A_{21} = K (I - A_{11})$** or **$(I+A_{11})$** (Equation3)

From Equation 1 and 3: $A_{21} = 1/K (I - A_{11})$ or $1/K(I+A_{11})$

Thus, from this our key matrix is obtained which is most crucial part in our project.

Failure in implementation of Advanced Techniques:

Our main goal in the project was to understand the concepts of Linear Algebra so that we can implement them in our project. The first algorithm we went with was QR decomposition using the Gram Schmidt Orthogonalization for key generation. The main problem here was our limited knowledge in writing a python code. In Gram Schmidt algorithm the problem was that code required the vector input however converting an image matrix into vector at the same point retaining the original data was not possible. Thus, that limited our chances for using Gram Schmidt algorithm. Also, we tried to use PCA for image encryption but the results with it were more for the compression and less for the encryption part hence that ruled out the use of PCA algorithm.

Orthogonal Approach/Gram Schmidt Approach:

As we already know the gram Schmidt is used for the orthogonalization of vectors and in the output provides with a orthogonal basis. Now diving into the implementation part, we know that as far as the description of orthogonal matrix is concern, a matrix A is called orthogonal if A preserve length of vectors, that is if. (Av, Av) = (v, v) For all vectors v in R^n . Here the orthogonal matrix is used a key matrix. The algorithm here is that the image is converted into the image matrix and the rgb values are separated intro three matrix and modulo 256 is carried out as the given matrix is in unit-8 form. The K key matrix obtained in the orthogonal hill cipher is obtained the following way:

Generation of Orthogonal key matrix:

$$\text{Let A} = \begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ m & n & o & p \end{bmatrix}$$

$$a_{11} = \begin{bmatrix} a & b \\ e & f \end{bmatrix}, a_{12} = \begin{bmatrix} c & d \\ g & h \end{bmatrix},$$

$$a_{21} = \begin{bmatrix} i & j \\ m & n \end{bmatrix}, a_{22} = \begin{bmatrix} k & l \\ o & p \end{bmatrix}$$

Therefore, $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

Which implies $A^T = \begin{bmatrix} a_{11}^T & a_{21}^T \\ a_{12}^T & a_{22}^T \end{bmatrix}$

A is orthogonal matrix which applies $A*A^T = I$

Consequently,

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} a_{11}^T & a_{21}^T \\ a_{12}^T & a_{22}^T \end{bmatrix} = I_{4x4}$$

Therefore,

$a_{11}\, a_{11}^T + a_{12}\, a_{12}^T = I$

$a_{11}\, a_{21}^T + a_{12}\, a_{22}^T = 0$

$a_{21}\, a_{11}^T + a_{22}\, a_{12}^T = 0$

$a_{21}\, a_{21}^T + a_{21}\, a_{22}^T = I$

Which results in $a_{22} = -a_{21}$

Now A = $\begin{bmatrix} a_{11} & a_{11} \\ -a_{22} & a_{22} \end{bmatrix}$

And, $A A^T = I$

$$\begin{bmatrix} 2 * a_{11} * a_{11}^T & 0 \\ 0 & 2 * a_{22} * a_{22}^T \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

Equating it,
$2\, a_{11}\, a_{11}^T = I = 2\, a_{22}\, a_{22}^T$

Thus from this we reach to conclusion that the $\mathbf{a_{11} = a_{12}}$ while the final block can be any arbitrary block which will be $\mathbf{a_{22} = -a_{21}}$

Generation of key Matrix using Gram Schmidt:

Generation of the key matrix using Gram-Schmidt orthogonalization we use the projection formula as:

$$\text{proj}_u(v) = \frac{\langle v,u \rangle}{\langle u,u \rangle} u$$

Where is the inner product of vectors v and u. Now, we take set of basis elements n, n being the dimension of pixels we want to encrypt as nxn matrix. Let this set of basis elements be {v1, v2, v3,……,v_n}. Then the set of orthonormal basis is calculated as {e1, e2, e3,……,e_n} as:

$$u_1 = v_1;\ e_1 = \frac{u1}{||u1||}$$

$$u_2 = v_2 - \text{proj}_{u1}(v_2);\ e_2 = \frac{u2}{||u2||}$$

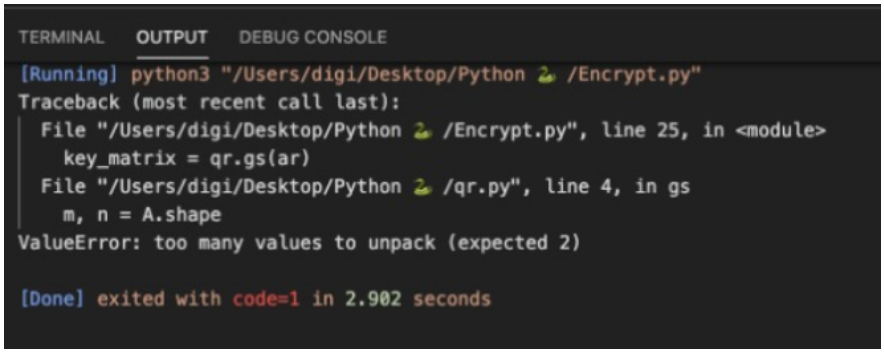
$$u_3 = v_3 - \text{proj}_{u1}(v_3) - \text{proj}_{u2}(v_3);\ e_3 = \frac{u3}{||u3||}$$

So on till,

$$u_n = v - \sum_{j=1}^{n-1} \text{proj}_{uj}(v_i);\ e_n = \frac{un}{||un||}$$

Now these orthonormal basis vectors {e1, e2, e3,……,e_n}are then placed in an n × n matrix and this matrix is then used as encryption matrix for encrypting the image information by distorting its pixels by multiplying the pixel matrix with this matrix.

The main error we faced while implementing this was that we wrote two algorithm the first one was just for small matrices and couldn't solve a certainly large image matrix.



While for the second time we worked our way through large input values but the algorithm was vector specific and couldn't process the image matrix and only took vectors as input. Hence, we failed in the implementation of the following algorithm:

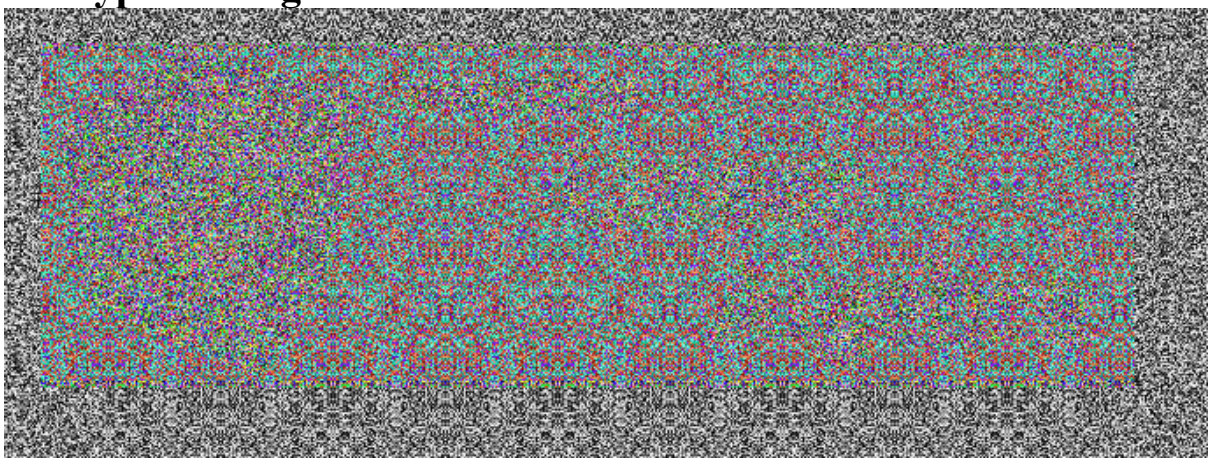
```
[Running] python3 "/Users/digi/Desktop/Python  /encrypt.py"
Traceback (most recent call last):
  File "/Users/digi/Desktop/Python  /encrypt.py", line 30, in <module>
    image_matrix = gs.gram_schmidt(image_matrix)
  File "/Users/digi/Desktop/Python  /gs.py", line 89, in gram_schmidt
    raise TypeError("Argument array must all be of type 'vector.vector'")
TypeError: Argument array must all be of type 'vector.vector'

[Done] exited with code=1 in 2.632 seconds
```

Results:

Input Image:

Encrypted Image:



Decrypted Image:

Conclusion:

As mentioned above our code works and provides us with these two images. Here two things are to be noted first as we are adding weight to individual pixel the size of encrypted image increases which is expected and the image is completely hidden it is not

at all visible. Secondly when we decrypt the image size of the image is almost same as the original image.

Future Works:

The first thing to do will be try implementing with the PCA work around the compression and the encryption part. Also, we would focus on making the quality of the image better as PCA has loads of applications in the field of Computer Science. Also, for the Gram Schmidt once we have proper knowledge would like to work around it and make it run without any errors. Another focus would be implementing a graphic user interface for user so that it becomes easy for the non-technical persons to use it.

References:

Madhusudhan Reddy, K., Itagi, A., Dabas, S., & Kamala Prakash, B. (2018). Image Encryption Using Orthogonal Hill Cipher Algorithm. *International Journal of Engineering & Technology, 7*(4.10), 59-63. doi:

Aashish R Follow. (2014, July 15). Image encryption and decryption. Retrieved November 4, 2020, from https://www.slideshare.net/ramamurthyaashish/image-encryption-and-decryption

Mofarreh-Bonab, Mohammad. (2015). Image Encryption by PCA. Communications on Applied Electronics. 3. 28-30. 10.5120/cae2015651893

Failla, Pierluigi & Barni, Mauro. (2011). Gram-Schmidt Orthogonalization on Encrypted Vectors. 10.1007/978-88-470-1818-1_7.