International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 06 Issue: 04 | Apr 2019 www.irjet.net p-ISSN: 2395-0072

Homomorphic Image Encryption

Sachin Rana[1], Om Jadhav[1] , Shivam Rajput[1], Pranjal Bhansali[1], Varshapriya Jyotinagar[2]

[1]B.Tech, Dept. of Computer Engineering, Veermata Jijabai Technological Institute, Mumbai, India.
[2]Faculty of Computer Engineering, Veermata Jijabai Technological Institute, Mumbai, India.

Abstract - *In cloud computing , user's privacy is of utmost importance when operating on user's data, which is protected using Homomorphic Encryption.[1] But it is not practical for wide usage as the types of data and services available in cloud computing are diverse. If you consider all these several data types, digital image is an important personal data for users. Many image processing services are also readily available in Cloud Computing. To safeguard user's privacy in these services we propose a scheme based on homomorphic encryption while processing the image. There are several steps in this process. First being, construction of a secret key homomorphic encryption (SKHE) for encrypting the image. Once the first step is done, it's followed by SKHE working on encrypted floating numbers, since they are used in image processing. Its then followed by converting the already present traditional encryption techniques so as to facilitate them working on encrypted pixels. The image which had undergone encryption is then processed homomorphically. Therefore services can now process the encrypted image directly (without the need for decrypting it first), and the result obtained after decryption is the same as processing the plain image itself. To explain our scheme, three common image processing instances are proposed in our paper. The experiments will show that our scheme is secure, correct, and efficient enough to be used in practical image processing applications.*

Key Words: Homomorphic encryption, Image Processing, *Paillier* cryptosystem

1.INTRODUCTION

Along with the arrival of cloud computing fever, there emerge a lot of services outsourcing applications based on cloud computing platform (such as SaaS). Users who request the service just need to upload their data to the service and wait for the result. This benefits the users greatly, but also risks breach of privacy, because the service provider (SP) can access user's plain sensitive information at will. To maintain the essential balance between the privacy issue and usability of data in cloud computing, many computable encryption technologies are proposed, one of them being homomorphic encryptions.

The central theme of how this process works can be explained as follows:

"Enc" represents the encryption process, and "Dec" represents the decryption process. For a function $f(x)$ taking the plaintexts as input, there exists a function
$f'(y)$ (where $y = Enc(x)$)
taking the cipher texts as input, such that $Dec(f'(y)) = f(x)$

When the user uploads the encrypted image, it is worked upon rather than the plain image. This input of encrypted image generates an output of encrypted result image which upon decryption gives us the required image. Its the same as working on plain image, but only we are not. And this is how the user's privacy is maintained.

Homomorphic encryption is indeed a very good way to protect privacy in service outsourcing applications, especially when handling the integer data type. But as we all very well know that the data types in cloud computing are diverse, the usage of homomorphic encryption for other types is still a challenging problem for the entire world. For example, with the popularization of photograph equipment, a large amount of digital images are generated every day. It has become one of the most popular forms of data for the users. Consequently, the online image processing services are widely used for users to edit their images. But the image may also contain privacy that the user does not want SP to see. To ensure that privacy which is of utmost importance nowadays, is maintained in this data type, we have proposed a scheme using homomorphic encryption in image processing services.

In broad terms, image processing includes all kinds of operations on the image. Handling of privacy issues is hard for all kinds of image processing techniques using one scheme. So in our paper basically, image processing mainly refers to the processes based on the concept and usage of pixels. That is, the new color of pixel is computed with the help of the old ones. The various operations on plain pixels can be seen as a function f. Operating on the old pixels and some other parameters (if necessary) as inputs, f outputs the new pixel that's required. When using the traditional image encryption techniques, one observes that operating on the encrypted pixels is meaningless since no traditional technique can process the encrypted image without decrypting it first. However, encrypting each pixel and then working on it generates pretty good results.

But unfortunately there are no such readymade encryption techniques available in the market that makes our work easy. So we'll just have to make one.

All the homomorphic techniques known as of today work on textual data. Image encryption using homomorphic encryption is a long way yet. Unpadded-RSA and ElGamal cryptosystems both give satisfactory results only when f consists of pure multiplication operations [10]. But both addition and multiplication operations are necessary in image processing. Neither do Goldwasser and Micali nor Paillier cryptosystems fit image encryption, since f consists of only pure addition operations. Gentry's fully homomorphic encryption too is too complex to apply in image processing, because it encrypts one bit at a time, with the runtime more than 30 seconds even in "toy" security level.

The traditional Homomorphic Encryption techniques are developed for integers, but floating numbers are involved in operations in image processing which is all together a different daemon in itself, these cryptosystems hence cannot be used directly either[11]. Some changes need to be made to them before applying them on floating numbers. Besides are all public key encryptions which need a large key size to ensure the security, so it's a waste of time. Thus, the sizes of cipher texts are too large to store or transmit online.

The above analysis reflects the strong need for an efficient homomorphic encryption technique, which can support both addition and multiplication operations on floating numbers, since images work on floating numbers. Then it can be used for image encryption and thereby processing the encrypted image directly without decrypting it first. Upon research it seems that no existing work has ever proposed any solution for this aforementioned problem.

Our contributions to solve these problems stated above are as follows:

(1)We make changes to the Paillier homomorphic encryption to propose an efficient secret key homomorphic encryption which can support addition and multiplication operations on floating numbers.

(2)We use the above stated method in image encryption which gives the instances of processing encrypted image.

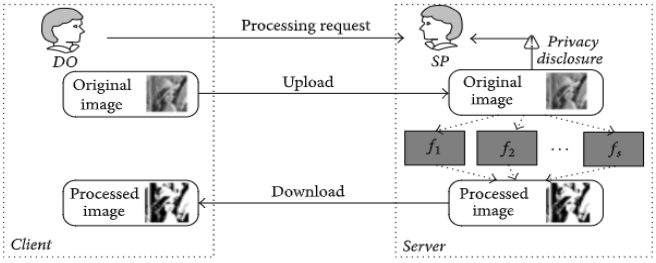
(3)We propose an encrypted image processing model based on homomorphic encryption.

2. Homomorphic Encryption Implementation in Image Processing:

2.1 Background Study:

2.1.1Encrypted Image Processing Model:

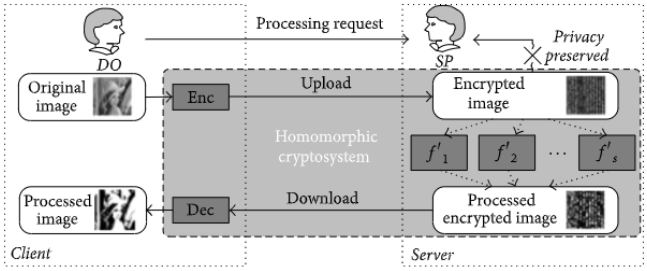
First of all, a model is constructed for processing an image which is in encrypted format with the central idea being homomorphic encryption; see Figure A.



(a) Image processing service model

Figure A (a) is a common online image processing service model. The service has a set of image processing functions as . When image data owner uploads the image for a specific kind of process request, cloud service provider (CSP) will choose the corresponding function to process the image. As CSP can access the original image, privacy is breached.

Secondly, see Figure A (b),



(b) Encrypted image processing service model

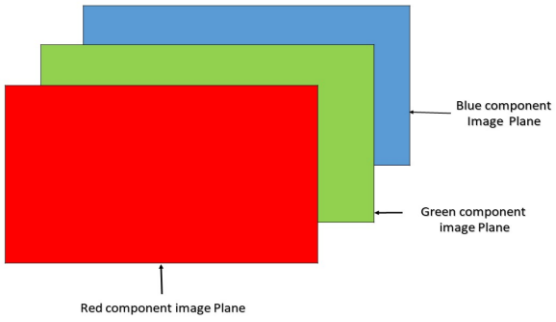
Data owner first has to encrypt the image using a homomorphic encryption prior to uploading it on the cloud . Thanks to the owners encryption efforts, CSP can operate on the encrypted image using a corresponding function and return to the Data Owner the encrypted result image. After decryption of the encrypted result image, Data Owner can get the correct processed image. It's same as the CSP working on a plain image, but only he's not.

The result is the same as Figure A(a). But as the image in server is always stored in encrypted form, the privacy is maintained.

2.1.2 RGB image representation

Any given RGB image can be seen as a collection of three different images(a red scale image, a green scale image and a blue scale image) placed on top of each other like a stack, and when these set of images are fed into the red, green and blue inputs of a color monitor, it produces a color image on the screen[2].

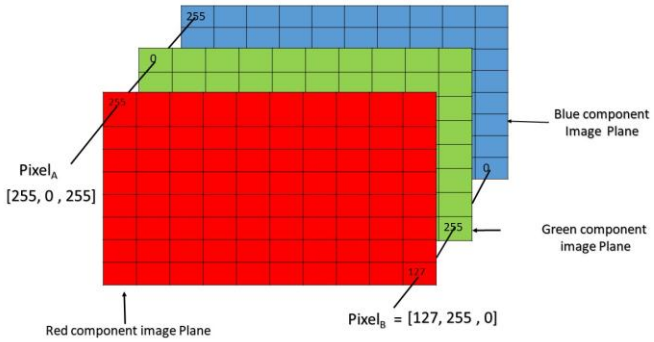
An RGB image is often referred to as a "true color image" only because of the precision with which a real-life image can be replicated.



[RGB image is a three layered image]

If you study MATLAB, any RGB image is basically represented as a M*N*3 array of color pixels, where each color pixel is represented with three values which correspond to red, blue and green color component of RGB image at a specified spatial location at a particular point of time. So, the color of any given pixel is calculated by the combination of the red, green, and blue intensities stored in each color plane at the pixel's location [3].

Here each color plane is an array of dimensions M*N which store the values of the particular color in them.



Pixel of an RGB image are formed from the corresponding pixel of the three component images

As can be observed in the image above, Pixel (A) has a value of (255, 0, 255) which is calculated by the combination of intensities stored in the red color plane, the green color plane and the blue color plane respectively and is stored in an array format.

Likewise, pixel (B) has value of (127, 255, 0) and is calculated in a manner similar to as used for the calculation for pixel (A).

Color planes of RGB image:

Consider an RGB image array 'Img' then, Img(:, :, 1) represents the Red color plane of the RGB image selected Img(:, :, 2) represents the Green color plane of the RGB image selected Img(:, :, 3) represents the Blue color plane of the RGB image under study.

RGB image array range:

In MATLAB term, any RGB image array can belong to any of the 3 classes. They are 'double', 'uint8', or 'uint16' data type classes. The data type class of color component determines the range of values it can accept [7].

For example, if an RGB image is of class 'double' then each color component is a value between 0 and 1 only. No other values beyond this range are acceptable.

Likewise, if an RGB image belongs to the class 'uint8', the range of values that each color component can have is restricted to [0 – 255] . [0 – 65535] is the range of values acceptable if the RGB image is of class 'uint16'.

Bit depth:

The number of bits that are used to store a pixel value of the component image under study determines the bit depth of any given RGB image.

For example, if each color component image is an 10-bit image, the RGB image will be said to have a bit depth of 30.

Possible number of colors in RGB image:

Let an RGB image belong to class 'uint16', i.e. the range of values a color component plane can have is [0 – 65535] (a total of 65536 shades of that color). So, each individual color plane of any given RGB image under study is capable of showing 65536 shade of that color. So the total number of combination of color that can be represented in any given RGB image is 65536 X 65536 X 65536 = 2.814 E14. That's a lot of combinations.

2.2 Design:

2.2.1 Paillier cryptosystem Algorithm:

The working of the scheme is as follows:

Key generation:

1. Choose two very large prime numbers p and q at random, and independently of each other, such that g.c.d of (pq, (p-1)(q-1)) = 1. This property is assured if both primes are of equal length.
2. Compute n (by using the formula n=pq) and λ= l.c.m (p-1, q-1) which means Least Common Multiple.
3. Select a integer randomly as g where $g \in Z^*n^2$
4. Ensure that the n calculated above divides the order of g by checking the existence of the following modular multiplicative inverse: $\mu = (L(g^{\lambda} \bmod n^2))^{-1} \bmod n$, where function L is defined as L(x) = (x-1)/n .

- The public (encryption) key used here is (n,g)
- The private (decryption) key used here is (λ, μ)

If using p,q of equivalent length, a simpler variant of the above key generation steps would be to set g=n+1 and λ = Ø(n), μ= Ø(n)^-1 mod n , where Ø(n) = (p-1)(q-1) .

Encryption

1. Let m be a message to be encrypted where 0<=m<n.
2. Select at random an integer r where 0<r<n and r∈ Z^*n^2 (i.e., ensure g.c.d (r, n)=1)
3. Compute cipher text as: $c= g^m r^n \bmod n^2$

Decryption

1. Let c be the cipher text to decrypt, where $c \in Z^*n^2$

2. Compute the plaintext message as: $m = L(c^{\lambda} \bmod n^2) . \mu \bmod n$

2.2.2 Homomorphic properties:

One of the main feature of the Paillier cryptosystem is that it supports homomorphic properties. Moreover it undergoes non-deterministic encryption. The encryption function used in this system follows additive homomorphic scheme. The two properties can be described as follows:

- **Homomorphic addition of two numbers**

If we take the product of two ciphertexts, it will decrypt to the sum of their corresponding original numbers[8],

$$Dec\ (Enc\ (A, r1).\ Enc\ (B, r2)\ mod\ n^2) = (A+B)\ mod\ n,$$

where A and B are real numbers

.

The product of a ciphertext with another number with the power g on decryption will result in the sum of the corresponding numbers,

$$Dec\ (Enc\ (A, r1).\ g^B\ mod\ n^2) = (A+B)\ mod\ n.$$

- **Homomorphic multiplication of numbers**

If we take an encrypted number raised to the power of another number, it will decrypt to the product of the two numbers,

$$Dec\ (Enc\ (A, r1)^B\ mod\ n^2) = A.B\ mod\ n.$$
$$Dec\ (Enc\ (B, r2)^A\ mod\ n^2) = A.B\ mod\ n.$$

If we consider a general approach, we can say that any encrypted number which is raised to a constant *x* will decrypt to the product of the number and the constant,

$$Dec\ (Enc\ (A, r1)^x\ mod\ n^2) = x.\ A\ mod\ n.$$

However, when we use Paillier encryptions where two numbers are provided, there is no known way to exactly compute an encryption of the product of these numbers unless you know the private key which is only known to the user himself.

2.2.3 Technologies Used:

1)Python Imaging Library (abbreviated as **PIL**) (Pillow according to newer versions) is a free library for the Python programming language that strongly supports for manipulating, saving or opening many different image file formats. It's a must use library for any image related processing [4].

PIL or Pillow offers several standard image manipulation procedures. These include the following important ones:

1. Masking and transparency handling,
2. Per-pixel manipulations,
3. Image enhancing, such as sharpening, adjusting brightness, contrast or color,
4. Image filtering, such as blurring, contouring, smoothing, or edge finding,
5. Adding text to images and much more.

Some of the file formats that are supported are PPM, PNG, JPEG, GIF, TIFF, and BMP. It is also possible to create new file decoders to expand the library of file formats accessible.

2) NumPy is a generic array-processing package. It provides us with a very much needed high-performance multidimensional array object, and tools for being able to work on these arrays [5].
It is the fundamental package for scientific computation with Python. It's a must use library when any type of scientific computations are involved. It contains various features including these important ones:

1. Sophisticated (broadcasting) functions
2. Useful linear algebra, Fourier transform, and random number capabilities
3. A powerful N-dimensional array object
4. Tools for integrating C/C++ and Fortran code[9]

Other than these important scientific uses, NumPy can also be used as an efficient container with multi-dimensional feature to store generic data. Besides this, Numpy also defines a set of arbitrary data-types which can be used to perform speedy integration with wide variety of databases.

3) SciPy is a free and open-source Python library used for scientific computation and technical computing. It's also a must use library for scientific computation and it builds on NumPy library[6].
SciPy contains modules for special functions integration , optimization, linear algebra, interpolation, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.
SciPy is a part of the NumPy stack which includes tools like Pandas, Matplotlib and SymPy, Pand an expanding set of scientific computing libraries. This NumPy stack has similar users to other applications such as Scilab,MATLAB, and GNU Octave.

3. Implementation:

The project aims at deploying our code on two nodes, one acting as client and the other one as sever machine. The client is responsible for generating keys, encrypting the image and sending the (key , image) pair to the server. Along with that the client has the choice to select a frame to be applied on the processed image. The server in turns perform homomorphic operation and provides the final image with applied frame.

JPG to RGB Conversion

The first step is the conversion of the image into its RGB pairs. The image is transformed into a 3-D array represented by a 2-D matrix where row and column number specifies the pixel coordinates in the image. In each element of this matrix we have three pairs of RGB values which range from 0-255. Hence we have transformed a complex image into simple numbers which can be operated easily by the proposed model.

Initially we had 2048*2048 resolution image. It contains 41,94,304 pixels. We converted it into RGB format which includes RGB values for each and every pixels. The data is saved in RGB.txt file with 4194304 rows.

Initial Image

Private and Public Key Generation

In order to encrypt the image we are using Pailler's Algorithm. The image is encrypted using private keys generated as a result of the proposed cryptosystem. The public key is sent to the server node to perform the subsequent operations.

In this system values for 'p' and 'q' will be generated randomly. Depending on the values selected and algorithm proposed, we calculate values for 'n',' λ' and 'μ' respectively.

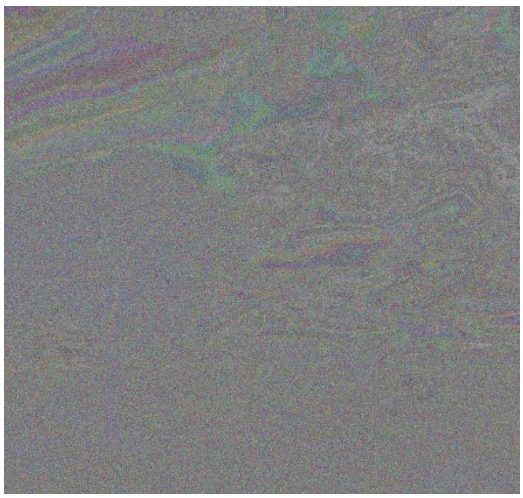
However in our model , we have to select two prime numbers namely p,q such that their product is greater than 255. This is done in order to conserve the 0-255 greyscale levels of the RGB pixels.

The first possible pair is 17,19 , such that calculated n is 323. Later on in order to quantize the values between 255-323 we can just take a mod 256 of the final decrypted values. In this way the decrypted pixel value 256 will represent 0 and so on.

Also 'g' is a random variable that can lie anywhere from 0 to n^2. However in our implementation we have restricted the range from 0 to n/2. This is done in order to save computational time utilized for calculating $g^{m.}$

RGB Encryption

After keys are generated, the first work is done at client side. The image has to be encrypted on client side before sending it to a third party(server node here) to maintain user privacy. With the help of Public Key(n,g), we encrypted our RGB.txt values and form the image from ciphertext values which looks like this:



Encrypted Image

Frame Detection and Encryption

The client has made his choice of frame that needs to be applied over image. Here we have considered that he wants to apply the following frame. The frame is encrypted at server side. In order to detect the frame, we used the condition of frame pixel detection. The pixel is encrypted only if it has some color values other than pure white. This way we can separate the frame pixels from unwanted pixels and hence reduce some computational time.

Original frame

Operation on Encrypted Image

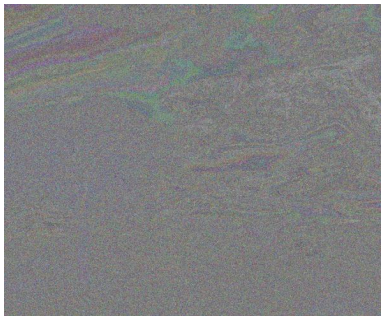
With the help of homomorphic encryption property mentioned earlier, our image will get filtered in an encrypted format i.e. frame will get applied.

This can be done in various ways. Firstly we can just replace the pixels in matrix by the frame pixels. This works fine for any image , however if we want to use complex filters like changing the grayscale level of an image, replacement policy is not an option.

The other approach is to get acknowledgement from client after every additive homomorphic operation for each and every pixel. However this design is not an optimal solution to the problem. The time and space complexity will increase exponentially with higher resolution images.

Paillier cryptosystem provides an easy solution to this problem. The solution is explained here in short and kept as the future scope of this project. The image can also be represented by CMYK encoding. Hence reducing the value range from 0-255 to 0-63. Then color addition by 1 unit can be done on each pixel until desired frame value is achieved by using the homomorphic addition property as stated in the paper. Using this way, iterations at each pixel are greatly reduced and become more feasible as an approach. However a more optimal solution can still be out there and needs to be introduced.

The image with applied frame will look as following



Decrypting the Final Image

The encrypted image is sent to user as either .JPG or .txt format. With the help of private key (λ,μ), client can easily decrypt the input received and develop the final filtered image as shown in the figure below:

Final Image

4. Conclusion:

In the proposed model we have successfully studied about image processing and were able to implement homomorphic encryption techniques in order to provide a secure and consistent means of data security.

The pre-existing Image Encryption techniques aimed at peer to peer sharing, and one has to take the risk of trusting the other entity usually third party apps. We find that most schemes aim at achieving a tradeoff between time complexity and efficiency. However using homomorphic technique the security is improved and the total control lies within the user. It provides total peer to peer security.

The domain is still new and developing and a lot needs to be introduced before it can be practically introduced in the society. The problem lies within extracting features from the encrypted images which plays a vital role in Machine learning using image processing and can be considered as a part of future scope.

Furthermore, we proposed a few open issues in some categories that need future research. We hope our study will help to shape future research directions in this promising area of security and privacy preservation of the data shared.

5. References:

1. An Efficient Secret Key Homomorphic Encryption Used in Image Processing Service by Pan Yang, Xiaolin Gui, Jian An and Feng Tian.
2. Face Detection and Face Recognition in Python Programming Language by Primoˇz Podrˇzaj and Boris Kuster
3. https://www.geeksforgeeks.org/matlab-rgb-image-representation/ (online)
4. https://en.wikipedia.org/wiki/Python_Imaging_Library (online)
5. https://www.geeksforgeeks.org/numpy-in-python-set-1-introduction/ (online)
6. https://en.wikipedia.org/wiki/SciPy (online)
7. https://www.ece.ucsb.edu/Faculty/Manjunath/courses/ece178W03/matlabip.htm (online)
8. http://npm.taobao.org/package/paillier-js (online)
9. https://indianpythonista.wordpress.com/2017/01/31/introduction-to-numpy/ (online)
10. Homomorphic Encryption: Theory & Application by Jaydip Sen
11. A Comprehensive Study of Fully Homomorphic Encryption Schemes by Majedah Alkharji, Hang Liu, Mayyada Al Hammoshi