Parallel Homomorphic Encryption on a Distributed Image

CORBIN GRAHAM, Iowa State University, USA

As the use of distributed cloud-network solutions rise, more and more data will be stored in distributed, external networks. It is important to practice safe encryption of data that will be stored on external devices. However, the cost of encrypting and decrypting massive datasets is extremely high. To mitigate these effects, it is in a user's best interest to decrypt as infrequently as possible. To effectively secure data, and limit the need for decryption, we will be implementing homomorphic encryption on an image. To maximize efficiency, and prepare the data for distribution, we will also be implementing a chunking algorithm to demonstrate benefits in speed, and security with the ability to operate on encrypted, distributed data.

ACM Reference Format:
Corbin Graham. 2023. Parallel Homomorphic Encryption on a Distributed Image. 1, 1, Article MATH 424 High Performance Computing (December 2023), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Homomorphic encryption is a technique for encrypted data in a way that it may still be operated on while encrypted. In the case of images, you can still perform image operations, like adding filters, combining images, or other modifying techniques without the need to decrypt the image. Traditionally, images stored on cloud-server networks would need to be downloaded to be modified, then re-uploaded to the cloud -server network. Now, add on encryption before uploading, and decryption after downloading. This becomes incredibly expensive when working with massive sets of images, crippling reach, and capability. Homomorphic encryption offers a solution, such that it is not only asymmetric, meaning that more data can be encrypted with the same public key from anywhere, but that it allows the public key holder to modify the data while it is encrypted. There are far superior techniques developed for Full Homomorphic Encryption (FHE) than I can develop in the span of this project, and I am not trying to re-invent the wheel. I will be using partial homomorphic encryption with Paillier's algorithm.

Traditionally, encryption of an image is quite slow. Assuming a full-color image is an $M \times N \times 3$ matrix of 8-bit integers, where the last layer is the RGB value, the complexity of encrypting the image would be $O(M \times N \times 3)$. When working with massive datasets for High Performance Computing (HPC), having to constantly encrypt and decrypt images for computation would be quite slow. We can first accelerate this by the implementation of homomorphic encryption, to limit the amount of decryption required to perform operations on an image. Then, by further breaking up an image into chunks, based on the number of processors in the HPC Cluster, we can perform encryption, decryption, and operations on the image at a much smaller scale. And we can securely store an image in a cloud-server network, as these chunks. By implementing a novel chunking algorithm for High Performance Computers, one that has multiple processors, we can split the image into smaller 'chunks' to perform the encryption, and decryption on. These chunks can also be further distributed on a cloud-server network, then recollected, as chunks, for their decryption. The chunking algorithm should also separate them in a way that they may be able to be operated on, using the homomorphic encryption, without the need for recollection.

Author's address: Corbin Graham, Iowa State University, Ames, Iowa, USA, cgraham1@iastate.edu.

XXXX-XXXX/2023/12-ARTMATH 424 High Performance Computing $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1.1 Problem Being Solved

As described, image encryption, and decryption are expensive. The goal of this project is to limit the expense from every angle. There are two major facets to this solution: Encryption/Decryption, and operations on an encrypted image. By implementing this solution, we should limit the time for encryption/decryption with chunking, limit the number of decryptions, and limit the complexity of each operation with the chunks.

2 RELATED WORK

Encryption is a broad term for morphing some data in a reproducible way such that any viewer would no longer understand it until it has been decrypted. Modern encryption employs several mathematical techniques to complicate the encryption making it more difficult for a bad actor to decrypt and understand the data, all while remaining decryptable to the good actor without any data loss. Homomorphic encryption is the mathematical technique of encrypting data such that any operation performed on the data while encrypted will have the same effect on its decrypted counterpart.

2.1 Paillier's Algorithm

In 1999, Pascal Paillier proposed an algorithm for homomorphic encryption that would allow for an unlimited number of addition operations to be performed on the encrypted data. Like traditional asymmetric encryption models, this employs dual-prime key generation. This system proposed using a probabilistic model for computing random values to be included in the ciphertext generation. The decryption worked to accurately predict the distribution of the random variable on itself with a prime number of equal parity to the encrypting prime. Using the greatest common denominator of two primes during generation with the distribution function of such primes and their multiple, should always return 1.

2.2 Efficient Homomorphic Medical Image Encryption

In August 2017, a team of researchers in India at the School of Computing at SASTRA University, introduced some key accelerations to Paillier's Algorithm. These contributions would allow for greater speeds of encrypting 2+-dimension matrices for image encryption. Their research was proposed for medical data usage, but can be applied to any image data, or 2+-dimensional matrices. Their proposal including using a small, positive, recurring integer instead of n (from key generation) and using the inverse of this integer in the decryption step. While it is a great approach for generating smaller image-set sizes, while still performing full encryption, it is far less secure than using the prime multiplicative of other encryption schemes.

3 APPROACH

The primary objective of this research is to propose a new approach to working with image data in a cloud environment. This approach is intended for massive datasets with many images in them with expected operation on a parallel, HPC system. To implement this approach, we will be using Paillier's algorithm for encrypting the data with minor changes to improve performance. For chunking, the approach will be a novel chunking algorithm that segments data based on the number of processors on the HPC system.

4 METHOD

To conduct an experiment proving the practical application of homomorphic encryption on an image in parallel using chunking, we will design a program that first implements a chunking algorithm to separate the image, then a method of encrypting that image using Paillier's algorithm.

4.1 Apparatus and Instrumentation

To simplify the programming of this project, we will be using a combination of Python and C. Python will be used to gather images, convert images to an 8-bit $M \times N \times 3$ matrix format, and transmitted to the C program. The C program will handle the chunking of the matrices, encryption, modification (operation), and decryption. For running each of these methods in parallel, we will be using OpenMPI's C library and compiler. Python will be used finally for printing tables and charts to demonstrate the impact of parallelization. In Python, we will be using the matplotlib, scikit-image, and pandas libraries.

4.2 Materials

The images were gathered from Python's scikit-image library. These are stock images commonly used in image-related processing projects. The prime numbers were pre-generated to simplify the process, and ensure consistency.

4.3 Design

The design is fairly basic for testing purposes but could be upscaled to support larger systems quite easily. The image is chunked, encrypted, operated upon, then decrypted.

An assumption before the design is to skip public key generation for efficiency purposes and use a known public key pair: $p = 149; q = 197$. Both are in the 8-bit range, resulting in a 16-bit product that only doubles the size. One current limitation is the size of the image on encryption as the bit-rate increases. This can also hinder performance.

4.3.1 Key Generation. With p, q known, n is the product of p and q. $\lambda = \texttt{lcm}((p-1),(q-1))$. g is a random 16-bit integer (for 16-bit encryption), where $g = 56602$.

4.3.2 Chunking. The image is chunked into smaller parts by the number of processors available on the HPC system.

Let K be the number of processors on the High Performance Computer. Assume the image is of $M \times N \times 3$ size.

$$\texttt{chunk(Image, K)} \rightarrow I_0..I_K$$

$$D = M//K$$

$$\forall i, j \in D \ \texttt{return} \ I_D \leftarrow \texttt{Image}[i:j]$$

4.3.3 Encryption. The image is encrypted using the smaller chunks from the chunking method.

Let $1 < r < n$ where $r \in \mathbb{Z}_n^+$.

$$\texttt{encrypt(I, n, g)} \rightarrow C_I$$

$$m \leftarrow \texttt{flatten}(I)$$

$$C = (g^m \mod n^2 * r^n \mod n^2) \mod n^2$$

4.3.4 Addition Operation. The addition operation is the only operation that the Paillier algorithm allows on encrypted data. There are Fully Homomoprhic Encryption (FHE) techniques to allow for more operations, but this remains simple and effective as a means of proving the use.

$$\texttt{addition(C(I), v, n, g)} \rightarrow A(C_I)$$

$$A(C_I) \leftarrow C_I \times \texttt{encrypt}(v) \mod n^2$$

4.3.5 Decryption. Decryption is applying the private key to the cryptographic scheme. The image is then restored to its original form from each chunk.

$$\texttt{decrypt}(A(C_{I..I_K})), \lambda, u) \rightarrow A(C_I)$$

$$\texttt{Image} \leftarrow (((c^{\lambda} \mod n^2) - 1)/n) \times u) \mod n) \forall c \in C_{I..I_K}$$

4.4 Procedure

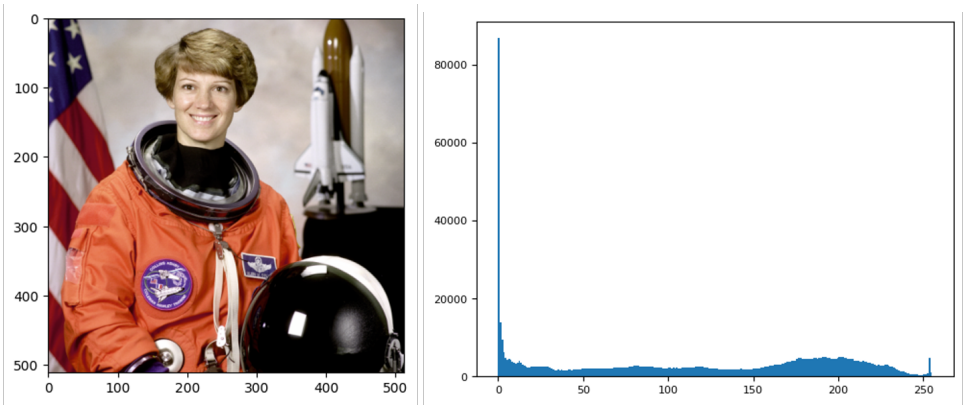
The procedure is getting images from python, sending them to the C program, chunking, encryption, performing operation, then decrypting and returning the result to python for analysis.

5 RESULTS

The results show that it is possible to encrypt, decrypt, and perform operations on the chunked images. There is also clear acceleration of these times by performing these operations in parallel.

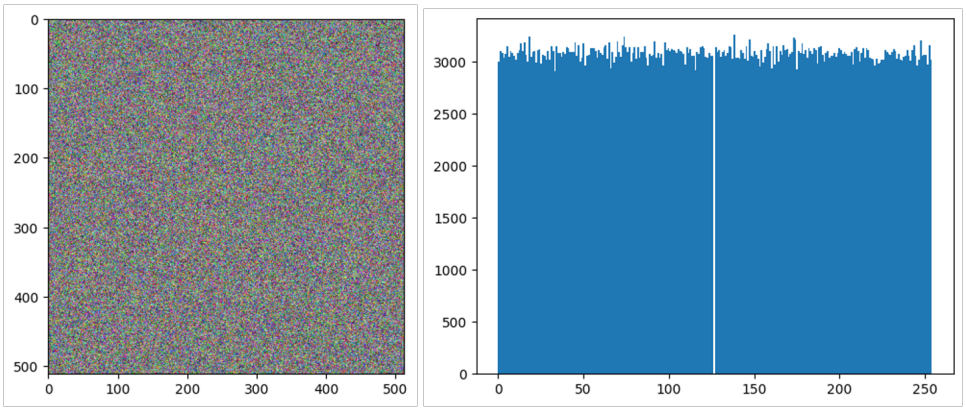
5.1 Original Image

The example image is from the scikit-image dataset. The image has a shape of 502×3. Besides is the histogram of the color distribution in the third layer.



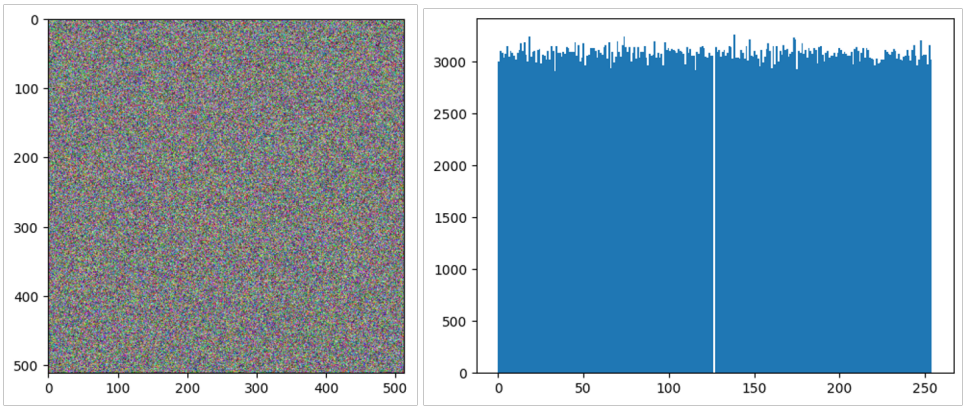
5.2 Encrypted Image

The image is encrypted in chunks, and restitched together, with a reshape to conver the now 16-bit values to an 8-bit displayable size. Besides is the histogram of the color distribution in the third layer.



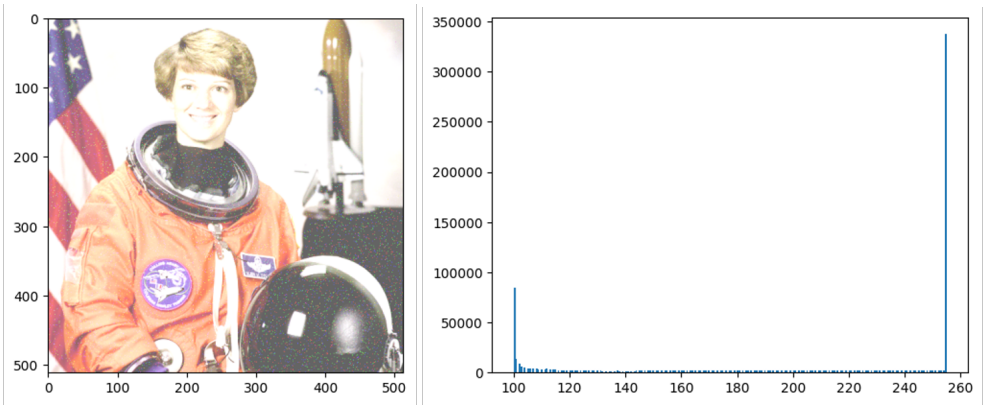
5.3 Modified Encrypted Image

Here is the encrypted image that has been modified with a 40% brightness increase. Besides is the histogram of the color distribution in the third layer.



5.4 Modified Decrypted Image

The modified image is decrypted in parallel and there is a clear 40% brightness increase. Besides is the histogram of the color distribution in the third layer. It is clear that the values are now all much more polar than they were in the original image.



5.5 Acceleration in Parallel

To measure the acceleration by running the methods in parallel, we will be using time and gain. Result will be the amount of time gained.

Number of Processors	Time to Completion (s)	Result
1	6.7641000000000e-02	+0.0000000000000e-00
2	3.2420000000000e-02	-3.5221000000000e-02
4	2.3705000000000e-02	-8.7150000000000e-03
8	8.1145780713996e-03	+1.5590421930000e-02
16	4.3416435946710e-03	-3.7729344770000e-03
32	2.9594886291306e-03	-1.3821549660000e-03

The above table demonstrates that running the algorithm in parallel has significant speed advantages, but since the image example is only 502×502, it becomes far slower towards the end when it is forced to break the image into smaller and smaller chunks.

5.6 Error Methods

Root Mean Squared Error is used to detect the error rate in the decrypted image when compared to the original.

$$\textbf{RMSE} = \sqrt{\frac{\Sigma(y_i - \hat{y}_i)^2}{N - P}}$$

We can detect the error from comparing a non-modified decrypted image to the original, and for our example image of the astronaut, we see it has a 0.125% error, showing that even without modification there are still some losses, but that can be improved with better encryption techniques and is unaffected when we compare against the non-parallel implementations of the algorithm.

6 DISCUSSION AND LIMITATIONS

From testing, it was clear that there were a few limitations that need to be addressed before a practical model of this project could ever be deployed. The first, and possibly most important limitation, is that the image can

only be chunked so small before the tiny size of the chunk increase the time complexity required to perform the operation.

The next, and also very important factor, is the amount of noise generated by the encryption and decryption. Even comparing the non-modified image, encrypted and decrypted, there was 0.125% noise in the generated image. This is insignificant to the human eye, but for many computational tasks, it is important the image remain unchanged.

7 CONCLUSION AND FUTURE

Testing has proven the feasibility of this idea. The image can be encrypted, modified, and decrypted, all as chunks in parallel with insignificant effects on the quality of the image. This proves the original question of can it be done. Now, we must ask, should it be done? And to what extent?

Going forward, this could be used as a method for encrypting valuable data, but still allowing operations on it while it is in an encrypted state. This, coupled with cloud computing, can give users the ability to view and modify images that are stored in the cloud and perform high-level operations on those images without very having to download them. This would make it so we need less local power on our devices and can offset this power with cloud-based networks.

Implementing this solution is most valuable on a large infrastructure, like a cloud network, but is less valuable when performed locally. This is because when operating on one image, the effects are minimal, however, when you have a massive dataset of images, this empowers far more computation without having to sacrifice security.

8 DECLARATION OF CONFLICTS OF INTEREST

Research was conducted with an elementary understanding of High Performance Computing and encryption algorithms. Outcomes may change dependant of understanding and expertise.

9 ACKNOWLEDGEMENTS

This project and research was conducted for the final project in MATH 424: High Performance Computing at Iowa State University in Fall 2023 for Dr. Songting Luo.

10 REFERENCES

Public-Key Cryptosystems Based on Composite Degree Residuosity Classes
https://www.cs.tau.ac.il/~fiat/crypt07/papers/Pai99pai.pdf

Somewhat Practical Fully Homomorphic Encryption
https://eprint.iacr.org/2012/144.pdf

Carmichael function
https://en.wikipedia.org/wiki/Carmichael_function

Modular multiplicative inverse
https://en.wikipedia.org/wiki/Modular_multiplicative_inverse

RSA (cryptosystem)
https://en.wikipedia.org/wiki/RSA_(cryptosystem)

Homomorphic encryption
https://en.wikipedia.org/wiki/Homomorphic_encryption

Paillier cryptosystem
https://en.wikipedia.org/wiki/Paillier_cryptosystem

An Efficient Homomorphic Medical Image Encryption Algorithm For Cloud Storage Security
https://www.sciencedirect.com/science/article/pii/S1877050917319798

Distributed Image Encryption Based On a Homomorphic Cryptographic Approach
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8993025

An Efficient Secret Key Homomorphic Encryption Used in Image Processing Service
https://www.hindawi.com/journals/scn/2017/7695751/

Parallelizing fully homomorphic encryption for a cloud environment
https://www.elsevier.es/en-revista-journal-applied-research-technology-jart-81-articulo-parallelizing-fully-homomorphic-encryption-for-S166564231500005X