Improved Paillier Homomorphic Encryption Time for a Stored Prefix Tree in a Graph Database

Jill Chandler, Aspen Olmsted  
Department of Computer Science  
Tandon School of EngineeringNew York UniversityBrooklyn, New York  
jdc9845@nyu.edu, aspeno@nyu.edu

*Abstract*—Current researchers have proposed using the Paillier homomorphic cryptosystem for securing private medical data, such as genomic data, in a cloud setting. Previous researchers were able to design a system with fast querying of encrypted SNPs in a prefix tree data structure saved in a graph database, however they experienced slow setup times for initially encrypting their data. While other homomorphic encryption schemes exist, we focus on using the C-coded Python library gmpy2 with the Paillier cryptosystem to improve the encryption time of the SNP data in a graph database. Our results show that using gmpy2 provides a dramatic reduction in encrypting SNP data saved in a prefix tree structure.

Keywords—*Paillier, homomorphic, gmpy2, graph database, cloud storage*

# Introduction

Genome wide association studies are used by researchers to investigate the relationship between genetic mutations, particularly single nucleotide polymorphisms (SNPs), and diseases [3]. Sharing this data with other researchers can help to improve healthcare [3], [7]. However, this data can be large and require high performance for computation [14]. Cloud storage is becoming a more practical and cost-effective way to store, share, and query large quantities of data [14]. However, due to privacy issues, it is difficult to securely store genomic data in the cloud [14]. Homomorphic encryption solves the issue due by allowing computations over encrypted data.

Several researchers have proposed models to securely share, store, and query SNP and disease data [6], [7], [9], [13]. Chen et. al. [6], proposed an efficient way to store SNP data in a prefix tree data structure in a graph database, Neo4j, which allowed for fast querying time of data. They, like previous researchers, used the Paillier homomorphic encryption system, however their implementation resulted in a slow encryption time (5.5 days) of their data and a large ciphertext [6], [11]. In this study we will investigate reducing the encryption time.

The GNU Multiple Precision Arithmetic Library (GMP) is a C-coded mathematical library that performs fast arbitrary precision arithmetic. Basically, this means that there is no limit to the precision except for the memory constraints of the machine [4]. The GMP library is available for C/C++, however the Neo4j database requires a driver to connect to it using a non-native language [10]. The driver for C/C++ is not well maintained [10]. Also, C/C++ is not user friendly nor practical for prototyping. A Neo4j driver is available for Python and is up to date [10]. Python also has a C-coded GMP module called gmpy2 [15].

In this paper, we implement the Paillier cryptosystem in Python in conjunction with gmpy2 to reduce the encryption time [5], [8], [15]. We will compare the encryption times and ciphertext sizes of the Paillier cryptosystem without and without gmpy2. The organization of the paper is as follows. Section II describes the related works and the limitations of current methods. In Sections III and IV, we state the hypothesis and describe the experimental method.  Section V describes our results. We conclude in Section VI and discuss future work.

# Related Works

As mentioned before, previous studies [6],[7],[9],[13] have proposed methods to securely store and query genomic data in a cloud setting.  Hasan et. al. [9] proposed using a tree index saved in a relational database while Chen et. al. [6] proposed a prefix tree with an index they referred to as a counting tree, saved in a graph database. Chen et. al.’s [6] method produced the fastest querying times however they experienced slow encryption times and a large ciphertext. All the proposed methods used the Paillier homomorphic cryptosystem [11]. Chen et. al. [6], in their future work section wanted to focus on ways to reduce the cipher text size and encryption time due to it being a performance overhead. We attempt to build on the research performed by Chen et. al. [6] by reducing the encryption time of the counting tree by utilizing gmpy2 with the Paillier cryptosystem. We also use a different dataset.

Cardoso et.al.[5], implemented the homomorphic encryption scheme DGHV in Python and gmpy2. Their results showed decreased encryption and decryption time compared to the original implementation. However, their implementation was for the DGHV. We propose to evaluate the encryption speed of the Paillier cryptosystem with gmpy2 on two genomic data sets.

Nassar et al. [8] implemented Paillier in Python with gmpy2. Their study produced promising results that showed a decrease in both encryption and decryption time compared to other Python tools such as paillier-Ivanov and paillier-Charm [8], however they did not test their implementation on a large data set and used a key size of 1024 bits. We will evaluate the encryption time on two large data sets with a key size of 3072 bits for 128-bit security according to NIST guidelines [2].

# Hypothesis

In this study, we propose implementing the Paillier cryptosystem with the gmpy2 mathematical library to reduce the encryption time of data stored in a counting tree structure as proposed by Chen et. al. [6] without increasing the ciphertext size.

# Methods

Two data sets, with 200 and 10,030 patients with 5 SNPs each, respectively were stored in a prefix tree data structure (Chen et. al.’s [6] counting tree) in the Neo4j graph database [10]. The tree was constructed according to Chen et. al. [6], however only the SNP data and SNP count were stored in each node. We used Python 3.9 for the coding language [12]. A total of 394 and 1,365 nodes were created from the two data sets, respectively, due to repeated SNPs being removed and stored as a count number. The prefix tree was constructed two times in two different Neo4j databases to evaluate the key generation speed, encryption speed, and ciphertext size of the Paillier cryptosystem with and without gmpy2. The key size used was 3072 bits. The Paillier cryptosystem, with and without gmpy2 was implemented using a modified version of the Python Paillier library [1]. Everything was run on a laptop using Ubuntu 20.04 with 12 GB RAM and an AMD Ryzen 5@ 2.38 GHz CPU.

# Results

Tables I and II summarize the results of the experiment. Implementation of the Paillier cryptosystem without gmpy2 is labeled as P-WO and with gmpy2 is labeled P-W.

TABLE I 200 Patients with 5 SNPs Each; 394 Nodes Created

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Paillier Implementation Comparisons for 394 Nodes** | | |
| ***Keys (ms)*** | ***Encryption (ms)*** | ***Database (MB)*** |
| P-WO | 1455.06 | 106849.13 | 2.06 |
| P-W | 19.00 | 19739.32 | 2.06 |

TABLE 2 10,030 Patients with 5 SNPs Each; 1,365 Nodes Created

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Paillier Implementation Comparisons for 1,365 Nodes** | | |
| ***Keys (ms)*** | ***Encryption (ms)*** | ***Database (MB)*** |
| P-WO | 241.86 | 254214.44 | 4.87 |
| P-W | 113.83 | 74436.96 | 4.87 |

1. Comparison of the total key generation and encryption time for each Paillier Implementation

The results in Table I, Table II, and Fig. 1 show a vast improvement in the key generation time and encryption time of the prefix tree after implementing gmpy2 without increasing the ciphertext size. Figure 1 shows a graphical comparison of the total time (key generation time and encryption time) in milliseconds between the two implementations. The implementation with gmpy2 resulted in an average of an almost five-and-a-half-fold reduction in time to generate the key pair and encrypt the SNP and count data in the prefix tree compared to the implementation without gmpy2.

# Conclusion and Future Work

Our research successfully demonstrates that the Paillier cryptosystem implemented in Python3 with gmpy2 significantly reduces the encryption time of data stored in a prefix tree without increasing the size of the ciphertext. Our future work will focus on ways to reduce the ciphertext size.

##### References

1. CSIRO's Data61. (2018) Python Paillier Library. [Online]. Available: https://github.com/data61/python-paillier
2. D. Giry, “NIST report on cryptographic key length and cryptoperiod (2020),” *Keylength*, 24-May-2020. [Online]. Available: https://www.keylength.com/en/4/. [Accessed: 07-Dec-2021].
3. “Genome-wide association studies fact sheet,” *Genome.gov*, 17-Aug-2020. [Online]. Available: https://www.genome.gov/about-genomics/fact-sheets/Genome-Wide-Association-Studies-Fact-Sheet. [Accessed: 29-Nov-2021].
4. GMP: GNU Multiple Precision Arithmetic Library. [Online]. Available: https://gmplib.org/
5. L. Cardoso dos Santos, G. Rodrigues Bilar and F. Dacêncio Pereira, "Implementation of the fully homomorphic encryption scheme over integers with shorter keys," *2015 7th International Conference on New Technologies, Mobility and Security (NTMS)*, 2015, pp. 1-5, doi: 10.1109/NTMS.2015.7266495.
6. L. Chen, M. M. Aziz, N. Mohammed and X. Jiang, "Secure large-scale genome data storage and query," *Computer Methods and Programs in Biomedicine*, vol. 165, pp. 129-137, Oct. 2018, doi.org/10.1016/j.cmpb.2018.08.007.
7. M. Kantarcioglu, W. Jiang, Y. Liu and B. Malin, "A Cryptographic Approach to Securely Share and Query Genomic Sequences," in *IEEE Transactions on Information Technology in Biomedicine*, vol. 12, no. 5, pp. 606-617, Sept. 2008, doi: 10.1109/TITB.2007.908465.
8. M. Nassar, A. Erradi and Q. M. Malluhi, "Paillier's encryption: Implementation and cloud applications," *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, 2015, pp. 1-5, doi: 10.1109/ARCSE.2015.7338149.
9. M. Z. Hasan, M. S. Mahdi, M. N. Sadat, and N. Mohammed, “Secure count query on encrypted genomic data,” *Journal of Biomedical Informatics*, vol. 81, pp. 41–52,  May 2018, doi: 10.1016/j.jbi.2018.03.003
10. Neo4j (Version 4.3.5). [Online]. Available: https://neo4j.com/
11. P. Paillier, “Public-key cryptosystems based on composite degree residuosity classes,” Advances in Cryptology — EUROCRYPT ’99, pp. 223–238, 1999, doi: 10.1007/3-540-48910-X\_16
12. Python (Version 3.9). [Online]. Available: https://www.python.org/downloads/
13. R. Ghasemi, M. M. Al Aziz, N. Mohammed, M. H. Dehkordi and X. Jiang, "Private and Efficient Query Processing on Outsourced Genomic Databases," *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 5, pp. 1466-1472, Sept. 2017, doi: 10.1109/JBHI.2016.2625299.
14. V. Cogo and A. Bessani, "Enabling the Efficient, Dependable Cloud-Based Storage of Human Genomes," *2019 38th International Symposium on Reliable Distributed Systems Workshops (SRDSW)*, 2019, pp. 19-24, doi: 10.1109/SRDSW49218.2019.00011.
15. Van Horsen, Case. (2016) *GMPY2* (Version 2.0.8). [Online]. Available: https://pypi.org/project/gmpy2/