Evaluation for an Effective Homomorphic Encryption Technique for Data Security

Authors

MUHAMMAD Sanusi[1], MUSAH Abdulmumini Yakubu[2], DANIEL Okunbor[3]

Senior Lecturer[1], Research Scholar[2], Professorr[3]

[1-2]*Computer Science Department,* Uni*versity of Abuja, Nigeria.*
[3]*Department of Mathematics and Computer Science, Fayetteville State University, Fayetteville, NC 28301.*

msanusi2009@yahoo.co.uk [1], muhadafa@gmail.com [2], diokunbor@gmail.com [3],

ABSTRACT

Homomorphic encryption is the encryption scheme that supports operations on encrypted data. Homomorphic encryption can be employed in any system by using various public-key algorithms. This research paper aims to provide an evaluation for an effective available Homomorphic encryption technique to understand how Homomorphic encryption work to achieve data security when data is transferred or stored on the public environment. Homomorphic encryption is an available encryption algorithm which secures data operations on storage mediums both to process encrypted data located on remote server and to preserve privacy, homomorphic encryption is the only option know to perform operations on encrypted data because it allows the operations on the ciphertext, which can provide the same results after calculations as working directly on the raw data. In this research paper, the main focus is to provide Evaluation for an effective homomorphic encryption technique for data security based on our findings and understanding of available public-key cryptographic techniques. The case study is a proposed implementation of PySEAL, a High-Level Python-pybind11 library wrapper for the Simple Encrypted Arithmetic Library (SEAL) implemented with a Docker container. Homomorphic encryption on various operations for encryption is performed and evaluated for performance analysis. It was observed from the result and corresponding Sample T-test analysis of our method that, Homomorphic encryption Operations are much more secure, faster and guarantee data security.

KEYWORDS; *Homomorphic, Encryption, Evaluation Technique, Security Parameter.*

Date of Submission: 08-11-2019 Date of acceptance: xx-xx-xxxx

---

1.0 INTRODUCTION

Encryption is a technique whereby data, termed a message, is mathematically transformed using an encryption key to produce a ciphertext. The ciphertext can only easily be decrypted to reveal the original data if the corresponding decryption key is known. Therefore, a ciphertext can be stored openly without compromising privacy so long as the decryption key is kept secret. [1].

Homomorphic Encryption is an encryption that allows ciphertext operations to be performed directly and this concept is called "privacy Homomorphism" [2]; thus an untrusted third party can process the ciphertexts without decrypting them, [3]. The possibility of homomorphic encryption was proposed by Rivest, Adleman, and Dertouzos, (1978) and many schemes that supported either multiplication like that of, [4], ElGamal, [5], etc or Homomorphic addition scheme such as Goldwasser-Micali[6], Paillier [7] as found. The decryption of the result of ciphertext operation is equivalent to the result of the corresponding plaintext operation. Therefore, Homomorphic encryption (HE) allows arbitrary operations to be performed on ciphertexts, [8].

The major problem with HE operations is the increasing noise of ciphertext generated. When ciphertexts are randomly generated, some considerable amount of noise is produced, this noise grows as homomorphic operations proceed, and eventually affects the correctness of the decryption operation when the noise magnitude exceeds a certain threshold and to overcome this problem, [9] proposed the bootstrapping technique which solved the noise problem. However, due to its inherent complexity, bootstrapping has become a major bottleneck for the effective construction of HE schemes, [3].

This research paper evaluates and presents Homomorphic encryption techniques currently available for various types of ciphertext data operations and presents a comparative overview of their performance to show how they achieve data security in the public environment.

2.0 LITERATURE REVIEW OF RELATED WORK

The first suggested concept of Homomorphic encryption was by [10], they proposed the RSA public key encryption algorithm a multiplication homomorphism and the security of there scheme is based on integer factorization. Followed by the encryption scheme proposed by [5], the ElGamal encryption scheme is a multiplication homomorphism and the [7], Paillier encryption scheme with the addition of homomorphism property which was a provable encryption scheme with remarkable level of safety. The closer to Paillier was Dan [11]. Boneh invented a plausible scheme that encouraged unlimited additive homomorphic encryption operation with only a multiplicative operation for a function.

In 2009, Gentry proposed the fully homomorphic encryption (FHE) scheme based on the ideal lattice problem, [9], and this scheme performed addition and multiplication operations of ciphertext. With more improvement to Gentry (2009), the fully homomorphic encryption technique entered the period of fast development. Dijk et al. proposed the fully homomorphic encryption scheme DGHV within the integer field, [12], and this scheme is based on the greatest common divisor (GCD) problem.

In the presentation of Brakerski et al. they proposed a fully homomorphic encryption scheme based on the LWE (learning with errors) problem, [13]. Its main idea is to address the defects of an ideal lattice-based scheme through the re-linearization technique.

So also did Stehle et al. introduced the NTRU (number theory research unit) algorithm to improve the efficiency of the initial FHE scheme D. [14]. Its security assumption is based on RLWE (ring learning with errors).

Another major contribution in technique was the study of Brakerski et al. proposed the BGV scheme in literature [15], which can support multi-bit operation, and the computation complexity is much lower than that of Gentry's initial scheme. From the initial scheme of Gentry to BGV scheme, the research on the homomorphic encryption scheme has made remarkable progress, but still far away from the actual application, [16]. Because of the remarkable discovery, Partial Homomorphic Encryption (PHE) was no longer the choice and fashion for homomorphic schemes, rather the direction for a Fully Homomorphic scheme is the order for a secure cyber world.

Many current Homomorphic encryption schemes have been proposed in literature [[13], [17], [18],[19],[20],[21],[22],[23]] and in recent years, various technologies of Homomorphic encryption technique have been broadly used for data privacy protection, such as HElib, libScarab, FHEW, and SEAL .

In contrast to the above schemes, which are either proof of concept or small-depth implementations, the authors in [24] implemented FHE for the first time to evaluate the circuit complex enough for a real-life application. In [24] implemented a variant of the BGV scheme proposed in [13], which is a leveled FHE without bootstrapping, to evaluate the AES circuit homomorphically.

Although all aforementioned implementations are published in literature, unfortunately, only a few of them are publicly available to researchers. Some of the publicly available implementations are:

2.1 HElib

HElib [25] is the most important and widely utilized. HElib implements the BGV scheme [13] with Smart-Vercauteren ciphertext packing techniques and some new optimizations. The design and implementation of HElib are documented in [25] and algorithms used in HElib are documented in [26]. HElib is designed using low-level programming, which deals with the hardware constraints and components of the computer without using the functions and commands of a programming language and hence, defined as "assembly language for HE".
It was implemented using the GPL-licensed C++ library. Since December 2014, it supports bootstrapping [27] and since March 2015, it supports multi-threading. In an important extension, the homomorphic evaluation of AES was implemented on top of HElib [28] and included in the HElib source code in [25]. Unfortunately, the

usage of HElib is not easy because of the sophistication needed for its low-level implementation and parameter selection which affects both performance and security level.

2.2 libScarab

Another notable open-source FHE implementation is libScarab [29]. In [30], they consider libScarab [29] the first open-source implementation of FHE.
The parameter selection of libScarab [29] is relatively easier than that of HElib, but it suffers from a lot of limitations [30]. For instance, it does not implement modern techniques like modulus reduction and re-linearization techniques [15] to handle the noise level or it also does not support the SIMD techniques introduced in [31]. It implements [32] and documentation is provided in [29].

2.3 Fastest Homomorphic Encryption in the West (FHEW)

Another major implementation is introduced by Ducas and Micciancio and called "Fastest Homomorphic Encryption in the West" (FHEW) [33]. It is documented in[34]. It significantly improves the time required to bootstrap the ciphertext claiming homomorphic evaluation of a NAND gate "in less than a second". A NAND gate is functionally complete. Hence, any possible boolean circuits can be built using only NAND gates. In [33], the usage of ciphertext packing and SIMD techniques provides an amortized cost. However, in FHEW such performance is achieved using only a few hundred lines of code with the use of one additional library, FFTW [35]. Later, the homomorphic computation cost of any binary gate [33] is increased by a factor of 50 by making some optimizations on the bootstrapping algorithm.
The main improvement is based on the torus representation of LWE ciphertexts. This improved the cost of bootstrapping 10 times according to the best known bootstrapping in [33]. They also further improved the noise propagation overhead algorithms using some approximations. Finally, they also reduced the size of the bootstrapping key from 1GB to 24MB by achieving the same security level.

2.4 Simple Encrypted Arithmetic Library (SEAL)

More recently, another HE library called Simple Encrypted Arithmetic Library (SEAL) [36] is released by Microsoft. The goal of releasing this library is explained as providing a well-documented HE library that can be easily used by both crypto experts and non-experts with no crypto background like practitioners in bioinformatics [30]. The library does not have external dependencies like others and it includes automatic parameter selection and noise estimator tools, which makes it easier to use.
Finally, the security estimates of two well-known LWE-based HE libraries, HElib and SEAL, against dual lattice attacks are revised in [37]. It is shown that the parameters for some level of bits size and the security estimation are almost similar for SEAL v2.0 and HElib.

3.0 Proposed System

Traditional encryption schemes, both symmetric and asymmetric, were not designed to perform computational operations on the algebraic structure of ciphertext spaces directly instead of plaintext. Many traditional schemes are limited to partial types of computations on directly encrypted data. The restriction to one single operation is very strong, however, and instead a much more powerful fully homomorphic encryption scheme that respects both additions and multiplications is needed for many interesting applications. The first of such an encryption scheme was invented by Craig Gentry in 2009, and since then researchers have introduced some new and more efficient fully homomorphic encryption schemes. However, with the promising theoretical power of homomorphic encryption, the practical side remained underdeveloped for a long time.
The proposed system seeks to implement a Simple Encrypted Arithmetic Library using a High-level programming language (python), in other to achieve the objectives of the research.

3.1 Architecture Of The Proposed System

Upon initialization of the system, the system automatically and randomly generates nominal and image data which serves as input parameters for the Homomorphic encryption technique to perform computation on the encrypted data and generate output for analysis to be done on the results.

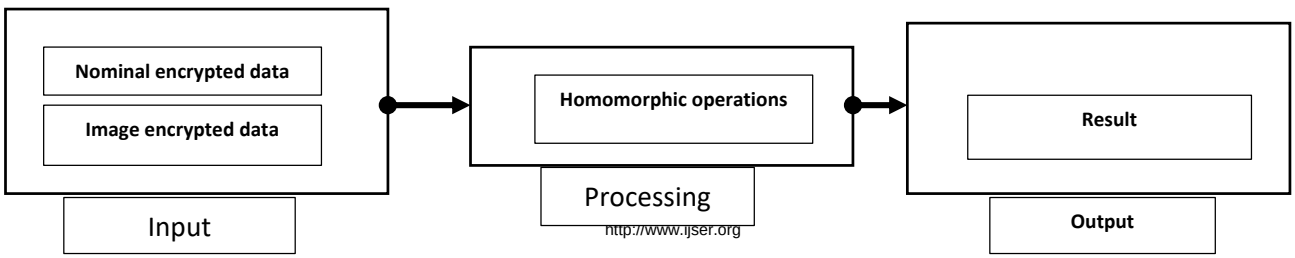

Figure 3. 1: Architecture of the proposed system

At the processing stage, a SEAL example builds and generates input data. The Python wrapper initializes the Docker from C++ to Python. The software also includes bash scripts to build the Docker container and execute a series of examples using PySEAL functions.

3.2 Sequence Flow of Proposed System

An overview description of the steps involved during encryption operation in the proposed system after instantiating the Docker for fully homomorphic encryption scheme include:

i- first step to using the library for basic encryption tasks is to instantiate a new Encryption Parameters object and set its modulus attributes.

ii- These parameters are stored in a SEAL Context object will check on the validity of the parameters.

iii- Once validated, these parameters are used to create encryption keys and Encryptor and Decryptor objects.

Once the validation of parameters for operations are set, Integer Encoder and Evaluator objects are included to encrypt input data and perform operations on the ciphertext. SEAL includes functionality to compute operations between ciphertext and plaintext, allowing for runtime improvements due to reduced encryption operational overhead.

4.0 Results

A generalized metric for any encryption algorithm is CPU and Memory utilization. In terms of technical metrics, the performance metrics for homomorphic encryption schemes and public-key encryption schemes are key size, and the time taken to perform encryption operations which is usually done by cryptanalysis.

All the timing tests we performed were done using **SEALPython** built-in timing functions. We timed how long it took to: perform key generation, encrypt a plaintext (or plaintexts), and perform a single computation on the encrypted ciphertext (or ciphertexts), and then decrypt the result. We tested addition, subtraction, multiplication, squaring a ciphertext, cubing a ciphertext, negation, and equality, which is performed automatically by **the SEALPython** library.

The timing functions keep track of the overall time spent in a particular function as well as the number of times that function was entered. From those two numbers, they output the averaged time spend in each function and a sample T-test analysis was carried out to compare the time taken to perform all operation. Results of operation are presented in Table 4.1- 4.4 below:

4.1 Output Result Table A

size of data (bit size)	batch (seconds)	unbatch (seconds)	encrypt (seconds)	decrypt (seconds)	add (seconds)	multiply (seconds)
H_200bit	0.000821	0.071707	0.267089	0.220592	0.000168	0.281865
T_200bit	0.049729	0.1770489	0.2274494	0.387216	0.049685	0.317117
H_400bit	0.000935	0.13148	0.157782	0.106762	0.000183	0.216166
T_400bit	0.048104	0.1836508	0.3763411	0.134128	0.049935	0.3773777
H_600bit	0.001279	0.113834	0.220208	0.205097	0.000175	0.35195
T_600bit	0.043434	0.1130827	0.2137758	0.178049	0.050247	0.3515731
H_800bit	0.000812	0.06317	0.101607	0.09658	0.000183	0.246098
T_800bit	0.062839	0.126332	0.3007167	0.241694	0.049973	0.3334784

Table 4. 1: Output Result Table A

Output Result Table A as presented above shows our results for four different bit size which include 200,400,600, and 800 bit size of data generated batch, unbatch, encryption, decryption, add and multiply operations time(sec) for Homomorphic and Traditional process with prefix ‘H’ and ‘T’ to denote Homomorphic and Traditional operations respectively.

4.2 Output Result Table B

size of data (bit size)	multiply plain (seconds)	square (seconds)	relinearize (seconds)	rotate rows one step (seconds)	rotate rows random (seconds)	rotate columns (seconds)
H_200bit	0.339169	0.321794	0.168615	0.246875	0.355854	0.236465
T_200bit	0.2677757	0.3366061	0.2531159	0.244939	0.5622537	0.3273023
H_400bit	0.143382	0.14186	0.214965	0.216388	0.232031	0.153514
T_400bit	0.1303494	0.4503302	0.1917181	0.430935	0.4088163	0.2970336
H_600bit	0.205517	0.187237	0.210848	0.335486	0.343946	0.316247
T_600bit	0.2670378	0.3810691	0.2315484	0.3766064	0.4830965	0.1790967
H_800bit	0.144389	0.260109	0.246011	0.323896	0.230739	0.199149
T_800bit	0.2831611	0.2493968	0.1918303	0.4205362	0.4270894	0.2573443

Table 4. 2: Output Result Table B

Output Result Table B as presented above shows our results for four different bit size which include 200,400,600, and 800 bit size of data generated for plain text multiplication, square, relinearization, rows rotation one step, rows rotation randomly and columns rotation operations time(sec) for Homomorphic and Traditional process with prefix 'H' and 'T' to denote Homomorphic and Traditional operations respectively.

4.3 Output Result Table C

size of data (bit size)	batch (seconds)	unbatch (seconds)	encrypt (seconds)	decrypt (seconds)	add (seconds)	multiply (seconds)
H_1000bit	0.000901	0.170807	0.192162	0.151109	0.000163	0.29056
T_1000bit	0.049537	0.185841	0.1726246	0.338708	0.050177	0.2503214
H_1200bit	0.035221	0.081619	0.19332	0.242377	0.000154	0.217603
T_1200bit	0.061728	0.1922932	0.4819045	0.227831	0.050163	0.315538
H_1400bit	0.000736	0.134179	0.189493	0.293423	0.000151	0.2882
T_1400bit	0.04744	0.1941488	0.2376666	0.283295	0.050086	0.2721969
H_1600bit	0.000822	0.155803	0.4039	0.07864	0.00021	0.40764
T_1600bit	0.066834	0.1615712	0.2679297	0.308732	0.050424	0.3610805

Table 4. 3: Output Result Table C

Output Result Table C as presented above shows our results for four different bit size which include 1000,1200,1400, and 1600 bit size of data generated for batch, unbatch, encryption, decryption, add and multiply operations time(sec) for Homomorphic and Traditional process with prefix 'H' and 'T' to denote Homomorphic and Traditional operations respectively.

4.4 Output Result Table D

size of data (bit size)	multiply plain (seconds)	square (seconds)	relinearize (seconds)	rotate rows one step (seconds)	rotate rows random (seconds)	rotate columns (seconds)
H_1000bit	0.212816	0.300811	0.234913	0.310528	0.363118	0.211072
T_1000bit	0.251935	0.3723326	0.3485172	0.3668658	0.4893032	0.2427816
H_1200bit	0.262981	0.371831	0.192135	0.288682	0.657771	0.166061
T_1200bit	0.1588317	0.3300725	0.2304527	0.3961299	0.2544516	0.2010481
H_1400bit	0.114312	0.313688	0.381029	0.206594	0.300819	0.103948
T_1400bit	0.1042637	0.4281549	0.3427051	0.3177802	0.5923078	0.2772842
H_1600bit	0.287278	0.254752	0.207039	0.137029	0.320866	0.198445
T_1600bit	0.226739	0.2599971	0.1847683	0.3049691	0.4182713	0.2280953

Table 4. 4: Output Result Table D

Output Result Table B as presented above shows our results for four different bit size which include 1000,1200,1400, and 1600 bit size of data generated for plain text multiplication, square, relinearization, rows rotation one step, rows rotation randomly and columns rotation operations time(sec) for Homomorphic and Traditional process with prefix 'H' and 'T' to denote Homomorphic and Traditional operations respectively.

5.0 Results Analysis

A clustered column bar graph will be used to graphically compare the results output for the presented result in the previous section.

Chart below provides an analysis of Output Result Table A-D:

5.1 Result Analysis of Table A

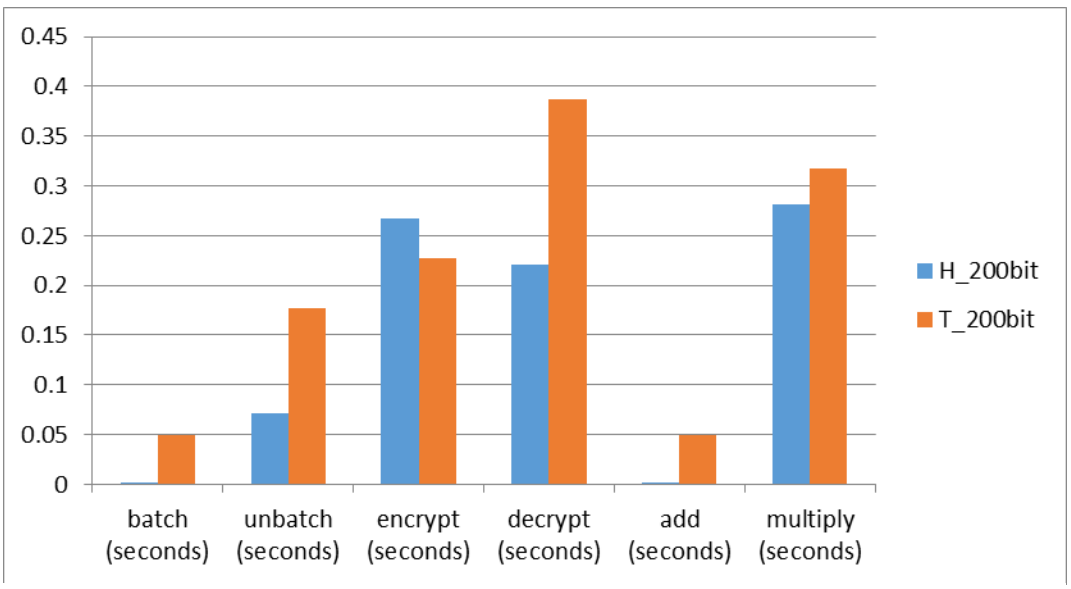

Figure 5. 1: Comparing Time of Operations For Table 4.1

From the analysis chart presented in figure 5.1, Homomorphic encryption is faster in all operations except for encryption operation at a data size of 200bit.

5.2 Result Analysis of Table B

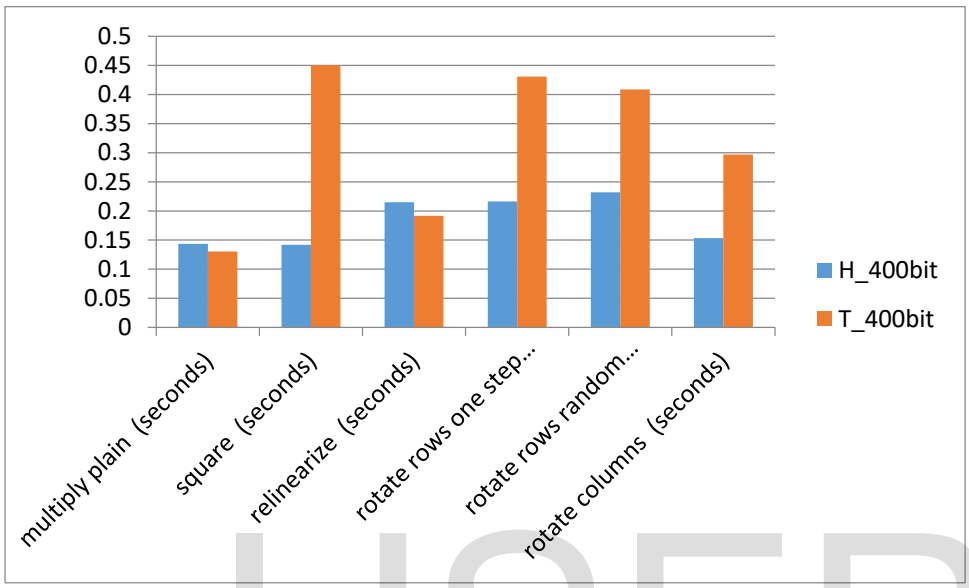

Figure 5. 2: Result Analysis of Table B

From the analysis chart presented in figure 5.2, Homomorphic encryption time is about the same for multiplication and Relinearize operations and faster in other operations at a data size of 400bit.

5.3 Result Analysis of Table C

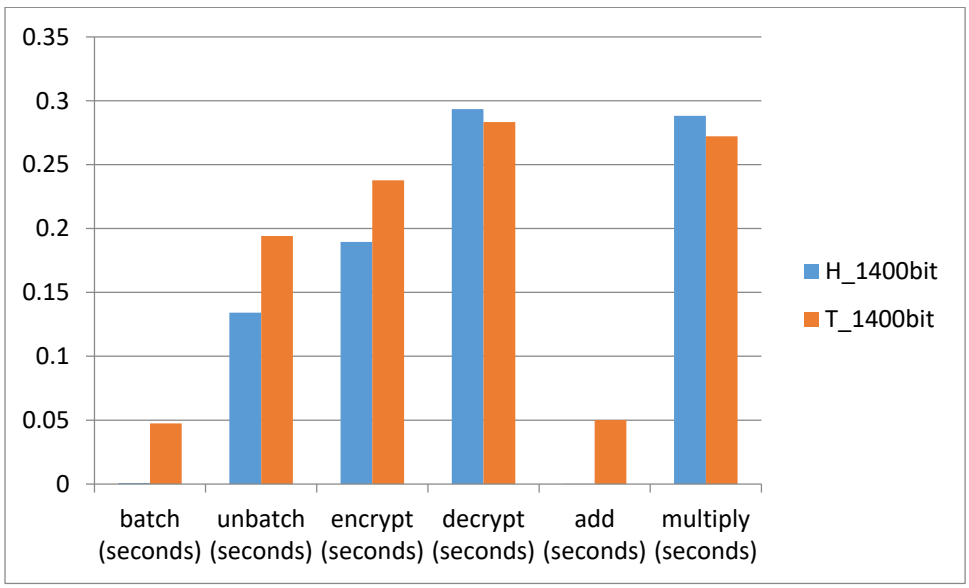

Figure 5. 3: Result Analysis of Table C

From the analysis chart presented in figure 5.3, at a much bigger data size of about 1400bit Homomorphic encryption time shows almost similar operational time except for batch and addition operations.

5.4 Result Analysis of Table D

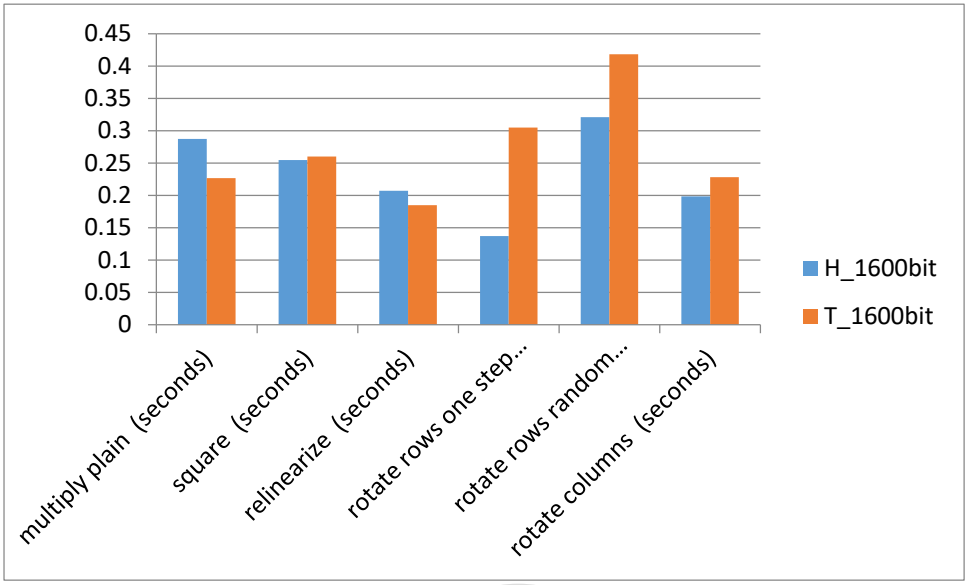

Figure 5. 4: Result Analysis Of Table D

From the analysis chart presented in figure 5.4, at a much bigger data size of about 1400bit Traditional encryption operational time shows similar operational time for encryption.

6.0 CONCLUSION

In this research paper, we have provided an evaluation of the time taken for various Homomorphic encryption operations and proposed a method for demonstrating how Homomorphic encryption works. We discovered the existence of various techniques to achieving Homomorphic computations, we also found that each Homomorphic encryption technique has an underlying algorithm and every algorithm has its benefits according to different parameters and these parameters are as a result of effective encryption techniques (ideal lattice, lattice base technique, learning with error (LWE), ring learning with error (RLWE)). From the work completed in this research, it is observed from the result and corresponding Sample T-test analysis of our method that, Homomorphic encryption Operations are much more secure, faster and guarantee data security.

REFERENCE

1. Aslett, L.J.M., P.M. Esperanca, and C.C. Holmes, *A review of homomorphic encryption and software tools for encrypted statistical machine learning.* 2015.
2. Lee, H., J. Alves-Foss, and S. Harrison. *The use of encrypted functions for mobile agent security*. in the *37th Annual Hawaii International Conference on System Sciences, 2004. Proceedings of the*. 2004. IEEE.
3. Wang, X., T. Luo, and J. Li, *A More Efficient Fully Homomorphic Encryption Scheme Based on GSW and DM Schemes.* Security and Communication Networks, 2018. **Volume 2018**: p. 14
4. Rivest, R.L., A. Shamir, and L. Adleman, *A method for obtaining digital signatures and public-key cryptosystems.* Communications of the ACM, 1978. **21**(2): p. 120-126.

5. ElGamal, T., *A public-key cryptosystem and a signature scheme based on discrete logarithms.* IEEE transactions on information theory, 1985. **31**(4): p. 469-472.
6. Goldwasser, S. and S. Micali, *Probabilistic encryption.* Journal of computer and system sciences, 1984. **28**(2): p. 270-299.
7. Paillier, P. *Public-key cryptosystems based on composite degree residuosity classes*. in *International Conference on the Theory and Applications of Cryptographic Techniques*. 1999. Springer.
8. Coron, J.-S., T. Lepoint, and M. Tibouchi. *Practical multilinear maps over the integers*. in *Annual Cryptology Conference*. 2013. Springer.
9. Gentry, C. and D. Boneh, *A fully homomorphic encryption scheme*, in *Computer Science*. 2009, Stanford University Stanford: Stanford University Stanford.
10. Rivest, R.L., L. Adleman, and M.L. Dertouzos, *On data banks and privacy homomorphisms.* Foundations of secure computation, 1978. **4**(11): p. 169-180.
11. Boneh, D., E.-J. Goh, and K. Nissim. *Evaluating 2-DNF formulas on ciphertexts*. in *Theory of Cryptography Conference*. 2005. Springer.
12. Van Dijk, M., et al. *Fully homomorphic encryption over the integers*. in *the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2010. Springer.
13. Brakerski, Z. and V. Vaikuntanathan. *Fully homomorphic encryption from ring-LWE and security for key-dependent messages*. in *Annual cryptology conference*. 2011. Springer.
14. Stehlé, D. and R. Steinfeld. *Making NTRU as secure as worst-case problems over ideal lattices*. in the *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2011. Springer.
15. Brakerski, Z., C. Gentry, and V. Vaikuntanathan, *(Leveled) fully homomorphic encryption without bootstrapping.* ACM Transactions on Computation Theory (TOCT), 2014. **6**(3): p. 13.
16. Min, Z., et al., *A privacy protection-oriented parallel fully homomorphic encryption algorithm in cyber-physical systems.* EURASIP Journal on Wireless Communications and Networking, 2019. **2019**(1): p. 15.
17. López-Alt, A., E. Tromer, and V. Vaikuntanathan. *On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption*. in *Proceedings of the forty-fourth annual ACM symposium on Theory of computing*. 2012. ACM.
18. Gentry, C., A. Sahai, and B. Waters. *Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based*. in *Annual Cryptology Conference*. 2013. Springer.
19. Cheon, J.H., et al. *Batch fully homomorphic encryption over the integers*. in *the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2013. Springer.
20. Gaithuru, J.N., and M. Bakhtiari. *Insight into the operation of NTRU and a comparative study of NTRU, RSA and ECC public-key cryptosystems*. on *2014 8th. Malaysian Software Engineering Conference (MySEC)*. 2014. IEEE.
21. Chen, H., Y. Hu, and Z. Lian, *Double batch for RLWE-based leveled fully homomorphic encryption.* Chinese Journal of Electronics, 2015. **24**(3): p. 661-666.
22. Cheon, J.H., et al., *CRT-based fully homomorphic encryption over the integers.* Information Sciences, 2015. **310**: p. 149-162.
23. Garg, S., et al. *Attribute-based encryption for circuits from multilinear maps*. in *the Annual Cryptology Conference*. 2013. Springer.
24. Brakerski, Z., C. Gentry, and V. Vaikuntanathan, *(Leveled) fully homomorphic encryption without bootstrapping.* 2012.
25. Halevi, S. and V. Shoup, *Design and implementation of a homomorphic-encryption library.* IBM Research (Manuscript), 2013. **6**: p. 12-15.

26. Halevi, S. and V. Shoup. *Algorithms in HElib-An Implementation of homomorphic encryption*. in *International Cryptology Conference*. 2014. Springer.
27. Halevi, S. and V. Shoup. *Bootstrapping for helib*. in *Annual International conference on the theory and applications of cryptographic techniques*. 2015. Springer.
28. Gentry, C., S. Halevi, and N.P. Smart. *Better bootstrapping in fully homomorphic encryption*. in *International Workshop on Public Key Cryptography*. 2012. Springer.
29. Perl, H., M. Brenner, and M. Smith. *Poster: an implementation of the fully homomorphic smart-vercauteren crypto-system*. in *ACM Conference on Computer and Communications Security*. 2011.
30. Acar, A., et al., *A survey on homomorphic encryption schemes: Theory and implementation.* ACM Computing Surveys (CSUR), 2018. **51**(4): p. 79.
31. Smart, N.P. and F. Vercauteren, *Fully homomorphic SIMD operations.* Designs, codes, and cryptography, 2014. **71**(1): p. 57-81.
32. Smart, N.P. and F. Vercauteren. *Fully homomorphic encryption with relatively small key and ciphertext sizes*. in *International Workshop on Public Key Cryptography*. 2010. Springer.
33. Ducas, L. and D. Micciancio. *FHEW: bootstrapping homomorphic encryption in less than a second*. in *the Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2015. Springer.
34. Fhew, *GitHub page: https://github.com/lducas/FHEW*. 2018: github.com.
35. Frigo, M. and S.G. Johnson, *The design and implementation of FFTW3.* Proceedings of the IEEE, 2005. **93**(2): p. 216-231.
36. Chen, H., K. Laine, and R. Player. *Simple encrypted arithmetic library-SEAL v2. 1*. in *International Conference on Financial Cryptography and Data Security*. 2017. Springer.
37. Albrecht, M.R. *On dual lattice attacks against small-secret LWE and parameter choices in HElib and SEAL*. in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. 2017. Springer.

\