International Journal of Business Data Communications and Networking
Volume 19 • Issue 1

An Accelerator to Additive Homomorphism to Handle Encrypted Data

Angelin Gladston, Anna University, Chennai, India*
S. Naveenkumar, Anna University, Chennai, India
K. Sanjeev, Anna University, Chennai, India
A. Gowthamraj, Anna University, Chennai, India

ABSTRACT

Homomorphic encryption provides a way to operate on the encrypted data so that the users can be given with the maximum feasible privacy. Homomorphic encryption is a special kind of encryption mechanism that can resolve security and privacy issues with rich text. Research gap is the performance overhead associated with this which poses a barrier to the real time implementation of this scheme. The objective of this work is to implement an algorithm to achieve increased performance and faster execution when compared with a classical cryptographical algorithm, the Paillier Cryptographical Algorithm, which is predominantly used to achieve additive homomorphism and analyse the performance gain obtained by this algorithm. The same algorithm is also integrated into an encrypted database application, CryptDB, developed by the MIT, as a replacement to the Paillier algorithm used in the application. The derived algorithms are 2600 time faster in key generation, 5 lakh times faster in encryption, and 3500 times faster in decryption, when compared with the Paillier algorithm.

KEYWORDS

Additive Homomorphism, CryptDB, Cryptography, Encrypted Data Computation, Encryption, Pailler

1. INTRODUCTION

In this modern world, people use a wide range of applications and they rely on these applications for their data security. There are numerous encryption algorithms in the world and every algorithm serves a specific purpose. But as encryption methods evolve, the art of breaking encryption algorithms also evolved. Simple encryption methods are used till late 19th centuries. The problem with those were, they can be easily broken by a human mind without any additional resources. But after late 19th century encryption algorithms are made with super protection, so that a human could not break it without any resources. But all it took was days or weeks to break them. And then the invention of computers led to public key, private key encryption methods. An advanced hybrid scheme of public key encryption is discussed by Jung et. al., (2015) and elaborated a new method called homomorphic encryption which is explained below.

DOI: 10.4018/IJBDCN.341589 *Corresponding Author

In Cryptography, encryption is a process of encoding certain sensitive information and store it in a manner that the encrypted data cannot be read without a secret key. With different security attacks being developed every day, there is a dire need for these algorithms to hold the data in an encrypted manner. Some of the common encryption algorithms known to the world are AES, DES, Blowfish, Twofish, RSA algorithms, etc. One such well used algorithm (Rajesh et. al., (2023) is Homomorphic encryption. It was proposed by Rivest, Adleman, and Dertouzos in 1978. The early stage of homomorphic encryption was really a tough process (Ryu et. al., (2023). Yang et. al., (2012) did an impressive job on finding and proving the homomorphic properties of integers, which provides a vivid view of homomorphic encryption. Almost 90 percent of banks either rent cloud storage or hire a group of technicians from an organization to help them store client's data. They will encrypt and decrypt client's data on sending and receiving. But while operating them they can't do that. This particular vulnerability leads to a situation where everything that has been done before had gone wasted.

These kinds of problems can be solved using methods like Homomorphic encryption. A new fully homomorphic encryption method is discussed by Mahmood et. al., (2018). Besides that, how multiple and multistage partial homomorphic encryption is used to develop a fully homomorphic encryption method is described and these two technologies are used in cloud applications. Homomorphic encryption allows us to perform calculations on encrypted data without decrypting it in the first place, and when decrypted the output is the same as if these operations are performed on the unencrypted data. For example, there is an algorithm called Paillier, which can perform addition and multiplication on encrypted data. Other than these operations Homomorphic addition and paillier are used in many other departments. As in Li et. al., (2021) the privacy concern in IoT is a big challenge in IoT applications and services, so this problem is encountered with Homomorphic encryption. In addition to this, paillier is used in homomorphic volume rendering as discussed by Mazza et. al., (2021).

However, the classical paillier based Homomorphic encryption does provide results for few operations, the performance of paillier algorithm is not up to mark and it consumes more cycles during runtime. Paillier encryption method's homomorphic properties are thoroughly analysed in Nassar et. al., (2015). This work theoretically stated an algorithm that can beat the traditional Paillier algorithm in addition operation. They named it Fast Additive Homomorphic Encryption, for sort FAHE. The basic idea of this paper is based on fast additive homomorphic encryption. ACD is also the same method that has been used in Eduardo et. al., (2020).

Zhang et. al., (2016) provides enough information about fully homomorphic encryption method over integers. Other than that it is also specifies how to accelerate the already exist homomorphic mechanism but only for integers. The proposed algorithm defeated Paillier in every single efficient test. But algorithms aren't of use without an application to employ them. One such application where the FAHE algorithm can be effectively applied to yield better results is, Cryptdb. Cryptdb is an MIT licensed product. This Crypt- dDB executes encrypted queries over the encrypted data stored. It provides basic database operated on the encrypted data using encrypted queries. Cryptdb already employs Paillier to perform addition and multiplication. This algorithm consumes a lot of cycles during runtime and it results in increased runtime of the encrypted queries run by cryptdb. The FAHE algorithm can be used as an effective replacement in this case, considering it has better performance over the paillier system without the loss of functionalities.

There are numerous encryption algorithms applied in real world application. Specific algorithms provide specific use cases. One such algorithm is Homomorphic Encrpytion. Homomorphic encryption allows us to perform operations on encrypted data and the results would as similar as though the operations are performed on original data.

However, the classical paillier based Homomorphic encryption does provide results for few operations, the performance of paillier algorithm is not upto mark and it consumes more cycles during runtime. This makes the paillier algorithm flawed and there is a dire need to replace the existing algorithm with a much faster one and with stronger encryption as the former. Also the paillier system uses asymmetric key system which uses different keys for the processes of encryption and decryption.

The paillier homomorphic algorithm can be used as a direct replacement in many application. One such application is "CryptDB", an MIT licensed product. This CryptdDB executes encrypted queries over the encrypted data stored. It provides basic database operated on the encrypted data using encrypted queries. This product uses paillier system as its base algorithm which has high "overhead". The algorithm consumes a lot of cycles during runtime and it results in increased runtime of the encrypted queries run by cryptdb.

Though the paillier system as numerous advantages on the field on Homomorphic encryption considering its security parameters and strengths, it also has defects in its own accord, Such as the asymmetric key system, high overhead and increased utilization of cycles. The most important drawback of paillier system is that it is not quantum-resistant meaning that they are vulnerable to the attacks by quantum super computers and Artificial Super Intelligences in future.

To solve all these disadvantages, the "Fast Additive Homomorphic Encryption" algorithm is used as an effective replacement for the paillier scheme. It uses symmetric key system and consumption of cycles are considerably lesser than that of paillier system. The most important key feature of this algorithm is that is uses the concept of "Approximate Common Divisor" problem which is seemed to be quantum-resistant and hence the Fast Additive algorithm can be concluded as quantum resistant.

With respect to the Crytdb architecture, it uses paillier system as its homomorphic layer. The new Fast additive algorithm developed can be used as an effective drop-in replacement for paillier algorithm and the overall performance of the CryptDB queries can be surged and the overhead can be brought down to great extent.

The rest of the paper is organized as follows: Section 2 gives an overview of the related works. Section 3 details the architecture diagram. Section 4 provides a detailed description of all the modules separately along with block diagrams. Section 5 describes the implementation of the algorithm, integration of the same into the CryptDB and the screenshots of the corresponding outputs. Section 6 presents the evaluation of the work and the performance measure comparison with the existing system.

2. RELATED WORKS

In this section, a brief review of the literature about homomorphic encryption, discussing the nuances and state of knowledge in the area, is presented. There are a lot of related works in the topic of Homomorphic encryption and related algorithms. Yang et. al., (2012) presented a very simple somewhat homomorphic encryption scheme over the integers. However, this simplicity came at the cost of a public key size in O(λ 10). Although at Crypto 2011 Coron et al. reduced the public key size to O(λ 7), it was still too large for practical applications. In this paper the public key size is further reduced to O(λ 3) by encrypting with a new form. The semantic security of our scheme is based on approximate-GCD problem of two integers. By using Gentry's techniques, we can easily convert the somewhat scheme into a practical fully homomorphic encryption scheme available in cloud computing.

Chaudhary et. al., (2019) mainly emphasizes on full homomorphic encryption and an investigation of different full homomorphic encryption schemes that uses the hardness of Ideal-lattice, integers, learning with error, elliptic curve cryptography based. Calculations can be carried out on encrypted form of data is the essence of homomorphic encryption. Homomorphic encryption has resolved the security issues for storing data on the third-party systems. Most significant category of homomorphic encryption is fully homomorphic encryption. It permits unbounded number of operations on the encrypted form of data and output by system is within the ciphertext space. This paper provides the essentials of Homomorphic encryption and its various classifications i.e. partially homomorphic encryption, somewhat homomorphic encryption and fully homomorphic encryption

Mahmood et. al., (2018) built a hybrid homomorphic encryption scheme based on the GM encryption algorithm which is additively single bit homomorphic, and RSA algorithm which is multiplicative homomorphic. The hybridization of homomorphic encryption schemes seems to be an effective way to defeat their limitations and to benefit from their resistance against the confidentiality attacks. This hybridization of homomorphic encryption algorithm lead to increase the speed 2.9 times, reduce the computation time to 66 percentage from previous one, enhanced confidentiality of the data that is stored in the cloud by enhancing security.

Also, paillier's encryption and its property to privacy-preserving computation outsourcing can be implemented on applications such as secure online voting. Nassar et. al., (2015) presented a new implementation of Paillier's cryptosystem using Python as for interface language and fast GMP C - routines for arithmetic operations. This paper also reviewed homomorphic encryption and its applications in arising cloud security issues. They presented a new and efficient implementation of Paillier's additive homomorphic encryption. Finally it reviewed on a subset of applications of Paillier's encryption in recent cloud security and privacy research. It doesn't claim in any means to be comprehensively addressing homomorphic encryption schemes or their applications. In Pallavi et. al., (2020) the study of the Paillier homomorphic secret writing theme is conducted. From the additive homomorphic property of the Paillier Cryptosystem, an approach that will facilitate in reducing the information measure throughout the transmission of the file and can create the system additional economical was studied.

Based on the public key compressing technique and the Chinese Remainder Theorem, an efficient somewhat homomorphic encryption (SWHE) scheme is proposed, whose security can be reduced to the approximate greatest common divisor problem. Then the fully homomorphic encryption scheme is obtained by using Gentry's squashing decryption circuit technique, which could resist chosen plaintext attacks. The efficiency analysis shows that the newly developed SWHE scheme's public key size presented by Zhang et. al., (2016) and cipher text size are reduced compared to DGHV scheme. And the simulation results show that the multiplication of the pro- posed SWHE scheme is much more efficient than that of DGHV scheme. HE consists of somewhat homomorphic encryption (SWHE) and fully homomorphic encryption (FHE). SWHE only evaluates low degree polynomials, while FHE allows anyone to perform arbitrary computations on the ciphertext. Because of this attractive property, FHE would allow computation services supplied by the third party without exposing the data, which is suitable to compute the encrypted data on cloud computing, e.g. cipher- text search, homomorphic machine learning and etc.

By Luan et. al., (2015) an implementation of a DGHV scheme variant using Python and the GMPY2 library has been developed. New researches in the field of homomorphic encryption schemes have made it possible to implement a variety of schemes using different techniques and programming languages This scheme was first proposed in 2010, by van Dijk et al, and later modified into two variants by Coron in 2011 and 2012, which reduced the prohibitive size of the public keys, at the cost of computational power. Besides that, this paper also presents a comparison of these implementations with the previous implementation of Coron's first variant. A fully homomorphic encryption (FHE) scheme is a cryptosystem that allows one, using only public data, to compute an arbitrary computation on the ciphertext, and get as result the encrypted computation over the plain text. In other words, having a computation f(), a message m and the encryption of said message E(m), there is a function f1 such as f1(E(m)) = E(f(m)). This characteristic is very desirable, as it allows secure processing on secret data to be executed by an unsecure third part without unencrypting the data.

Ring learning with errors (Ring-LWE) is the basis of various lattice based cryptosystems. The most critical and computationally intensive operation of Ring-LWE based cryptosystems is polynomial multiplication over rings. By Chaohui et. al., (2016) several optimization techniques to build an efficient polynomial multiplier with the number theoretic transform (NTT) were introduced. A new technique to optimize the bit-reverse operation of NTT and inverse-NTT. With additional optimizations, the polynomial multiplier reduces the required clock cycles from (8n+1.5n lg n) to (2n+1.5n lg n). By exploiting the relationship of the constant factors, the polynomial multiplier is able to reduce the number of constant factors from 4n to 2.5n, which saves about 37.5 ROM storage. In addition, they proposed a novel memory access scheme to achieve maximum utilization of the butterfly operator. With these techniques, the polynomial multiplier is capable to perform 57304/26913 polynomial multiplications per second for dimension 256/512 on a Spartan-6 FPGA.

In a model presented by Jung et. al., (2015) messages are encrypted with a PKE (Public key encryption) and computations on encrypted data are carried out using SHE or FHE after homomorphic decryption. In this model, messages are encrypted with a PKE and computations on encrypted data are carried out using SHE or FHE after homomorphic decryption. To obtain efficient homomorphic decryption, our hybrid scheme combines IND-CPA PKE without complicated message padding with SHE with a large integer message space. Furthermore, if the underlying PKE is multiplicative, the proposed scheme has the advantage that polynomials of arbitrary degree can be evaluated without bootstrapping. They constructed this scheme by concatenating the ElGamal and Goldwasser-Micali schemes over a ring. To accelerate the homomorphic evaluation of the PKE decryption, they introduced a method to reduce the degree of the exponentiation circuit at the cost of additional public keys. Using the same technique, an efficient partial solution to an open problem which is to evaluate mod q mod p arithmetic homomorphically for large p is presented. As an independent interest, we also obtain a generic method for converting from private-key SHE to public-key SHE. Unlike the method described by Rothblum, we are free to choose the SHE message space.

The above mentioned methods are partly helping to complete a process which is mentioned in the work. As the name suggests the algorithms mentioned in partially homomorphic encryption are only of use when one of the two types, i.e addition or multiplication is used. So one cannot use these kind of algorithms to perform both homomorphic operations. The next one Somewhat Homomorphic encryption uses lattice based cryptography to perform addition and multiplication on cipher texts. However, the classical paillier based homomorphic encryption does not provide results for few operations, the performance of paillier is not upto mark and it consumes more cycles during runtime. This makes the paillier algorithm flawed and there is a dire need to replace the existing algorithm with a much faster one and with stronger encryption as the former. Also the paillier system uses assymetric key system which uses different keys for the processes of encryption and decryption.

Fully homomorphic encryption inherits multiple properties from the second method but instead of using lattice based cryptography, this method uses approximate common divisor problem (ACD) which is a NP-Hard problem and quantum resisting. It fully discussed on ACD and fully homomorphic encryption but the difference is it is an Accelerator to the homomorphic addition and uses the ACD method to prove it.

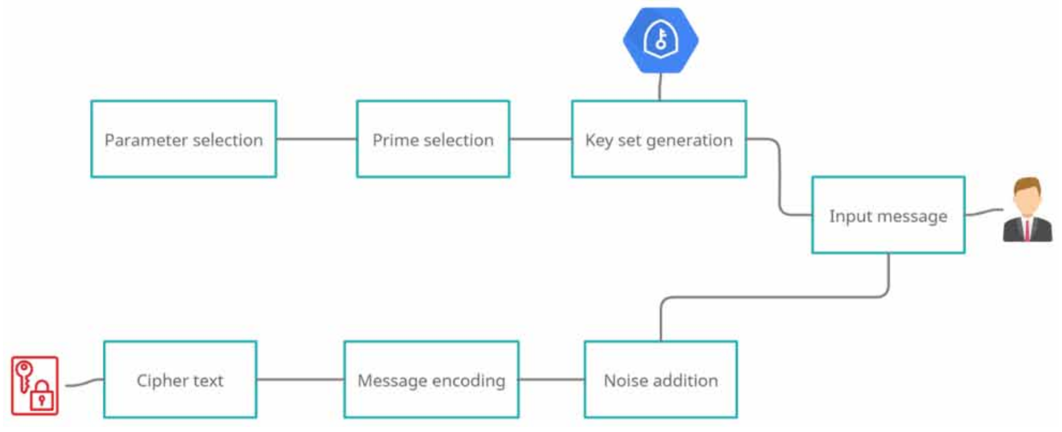
In addition to that the future work mentioned in the paper is also implemented. Cryptdb executes encrypted queries over the encrypted data stored. It provide basic database operated on encrypted data using. It uses Paillier system as its base algorithm which has high “overhead”. The algorithm consumes a lot of cycles during run-time and it results in increased runtime of queries run by cryptdb. The contribution of this thesis is replacing the paillier encryption method with the new fully additive homomorphic encryption in Cryptdb and observing the changes in performance of Cryptdb.

3. EXPERIMENTAL DESIGN

This section presents the detailed block diagram of the entire project. The block diagrams are distinguished into three parts. First, the implementation of the algorithm is presented in a neat manner. Next, how the CryptDB is designed and how it is working is displayed as a separate block diagram. The final diagram describes how the derived algorithm is integrated into the existing CryptDB application as a replacement to the Paillier module to achieve increased performance gain.

Figure 1 displays the working of the algorithm in a detailed manner. The algorithm works by beginning with the initialization of necessary and needed parameters such as lambda, rho, eeta, gamma and p. With those values, encryption and decryption key set are generated as a part of the entire key set of the algorithm. The input plain text message is encoded and then encrypted using the calculated key values. Cipher texts can be added as plain texts for the number of times as defined in

Figure 1. Block diagram



the parameter initialization as similar to the normal addition of values. Added encrypted cipher text can be decrypted with the decryption key set to obtain the result as same as the plain text addition.

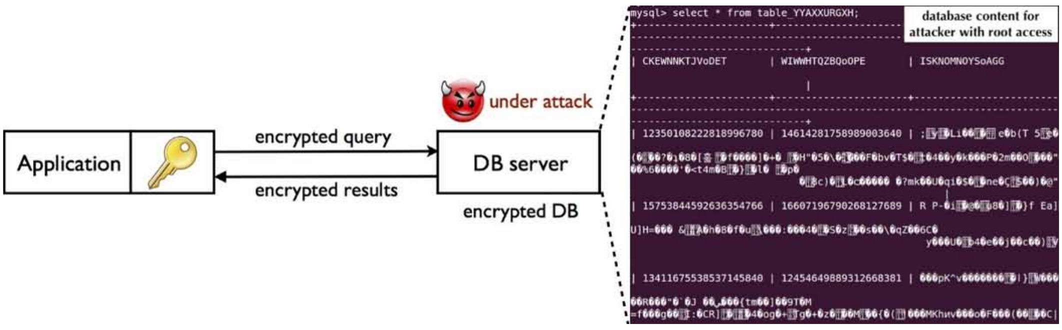
Figure 2 shows the block diagram of the CryptDB application. The application is designed in such a manner that the front end will be issuing the DML and DDL queries to the database. Before reaching the database server, an intermediate server is set up which will be doing all the necessary steps to work with the encrypted database. All the front end queries will be sent in the encrypted format to the database. With the help of the Onion layer of encryptions and SQL-aware encryption strategies, CryptDB server will encrypt, decrypt and modify the SQL queries to achieve the operations on the encrypted values. The database server will be having only the encrypted values of the user data.

The derived algorithm is applied for the replacement of the Paillier module in the CryptDB application as shown in Figure 3.

Figure 3 shows the integration of the derived algorithm into the CryptDB application. Both the header and cc files of the Paillier module are completely changed with the derived algorithm. Along with those files, some of the other modules of the application are also changed appropriately to ensure the perfect working of the application with the increased performance gain and faster execution of the queries.

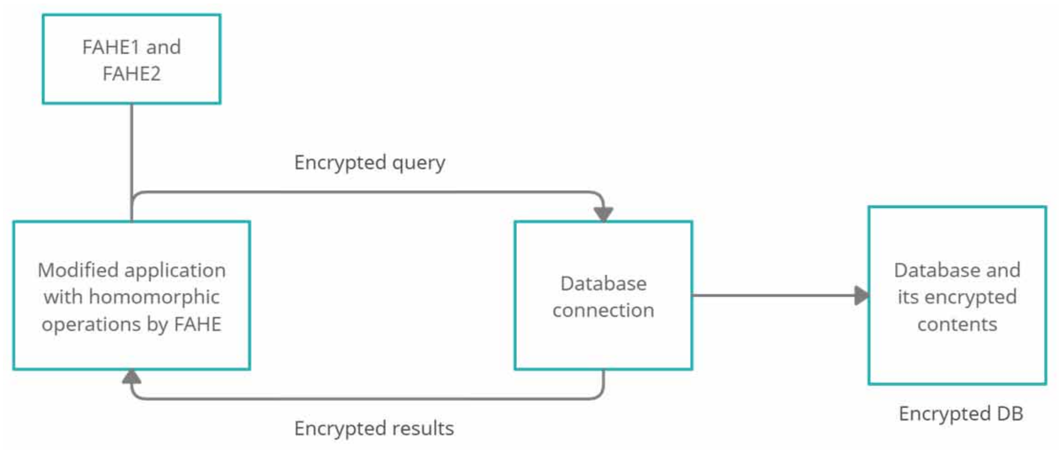
The derived algorithm has various parameters for achieving different amount of security levels. These parameters influences the strength of the output cipher text. They can be tuned to achieve the algorithm. There are two algorithms with a slight variation between them. The second one is even

Figure 2. Block diagram - CryptDB



```
mysql> select * from table_YYAXXURGXH;
+-------------------------+-------------------------
-------------------------------------------------
--------------------------+
| CKEWNNKTJVoDET       | WIWWHTQZBQoOPE       | ISKNOMNOYSoAGG

                          |
+-------------------------+-------------------------+------------------------
-----------------------------------------------------------------------
--------------------------+
| 12350108222818996780 | 14614281758989003640 | ;[illegible]
[illegible]
[illegible]
| 15753844592636354766 | 16607196790268127689 | R P-[illegible]
U]H=[illegible]
[illegible]

| 13411675538537145840 | 12454649889312668381 | [illegible]
[illegible]
[illegible]
```

Figure 3. Block diagram - CryptDB with FAHE


FAHE1 and FAHE2
Encrypted query
Modified application with homomorphic operations by FAHE
Database connection
Database and its encrypted contents
Encrypted results
Encrypted DB

faster than the first one and also produces cipher texts in smaller sizes than the former algorithm. Some of the modules of the algorithm are described in detail in the forthcoming sections.

3.1 Key Generation

Initializing the necessary parameters before the key generation module, the key generation module will compute certain values needed with the initialized parameters. This will produce a cryptographically unique and random key to encrypt the plain text to cipher text. The key is a set of values of some which will be used for encryption and some will be used for the purpose of decryption. Unlike Paillier cryptosystem, it is a symmetric algorithm wherein a single key set is used for both encryption and decryption operations as shown in Figure 4.

Computing ρ, η and γ from the given parameters and feed them to a formula along with a prime and the formula generates keys for both encryption and decryption.

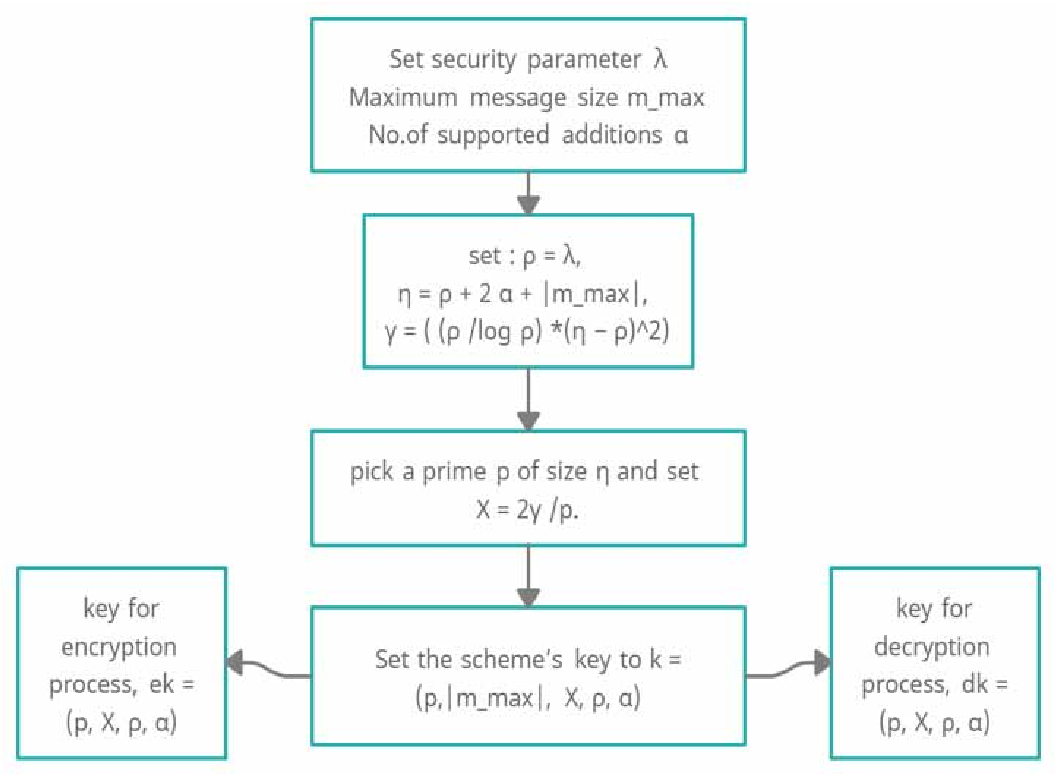
3.1.1 Pseudo Code

```
1.  $\lambda$ = security parameter
2.  m_max = maximum message size
3.  $\alpha$ = total number of supported additions
4.  Compute $\rho = \lambda$
5.  Compute $\eta = \rho + 2\alpha +$ m_max
6.  Compute $\gamma = ((\rho / \log(\rho)) * (\eta - \rho)^2)$
7.  Pick a prime p of size $\eta$
8.  Set X = $2^\gamma/p$
9.  Set the scheme's key to k = (p,|$m\ max$|, X, $\rho$, $\alpha$)
10. Encryption key ek = (p, X, $\rho$, $\alpha$)
11. Decryption key dk = (p,|$m\ max$|, $\rho$, $\alpha$)
```

3.2 Encryption

Encryption will use certain values from the generated key set to produce the cipher text. The plain text message input fed to the system is encoded to another format. The encoded message will then further processed to output the cipher text from the given plain text message. The cipher text will be a

Figure 4. Block diagram: Key generation


Set security parameter λ
Maximum message size m_max
No.of supported additions α
set : ρ = λ,
η = ρ + 2 α + |m_max|,
γ = ((ρ /log ρ) *(η − ρ)^2)
pick a prime p of size η and set
X = 2γ /p.
Set the scheme's key to k = (p,|m_max|, X, ρ, α)
key for encryption process, ek = (p, X, ρ, α)
key for decryption process, dk = (p, X, ρ, α)

cryptographically random one for the input plain text. That is, the cipher text will be different in every time for the same plain text and the key set. Thus making it harder for various kinds of crypt analysis.

Plain text is fed into a formula with some noise and the output of this block will be encrypted text as shown in Figure 5.

3.2.1 Pseudo Code

```
1. Get plain text m
2. Compute M = (m << ($\rho + \alpha$)) + noise
3. Compute n = p.q
4. Encrypted message c = M + n
```

3.3 Decryption

The cipher text is fed to the decryption module to obtain the original plain text. For the decryption process, some other values from the key set is utilized. The algorithm is based on the Approximate Common Divisor (ACD) problem. This is by the fact that the Approximate Common Divisor problem is a hard problem and also post-quantum i.e., the algorithm is resistant to attacks even with quantum computers. To make the algorithm to exhibit quantum-resistant, certain parameters have to be set to meet the necessary conditions.

Cipher text is given as input to this block and processed using a formula and output will be a plain text as shown in Figure 6.

Figure 5. Block diagram: Encryption

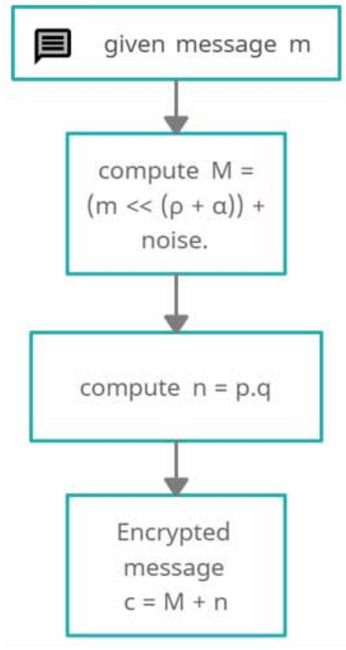
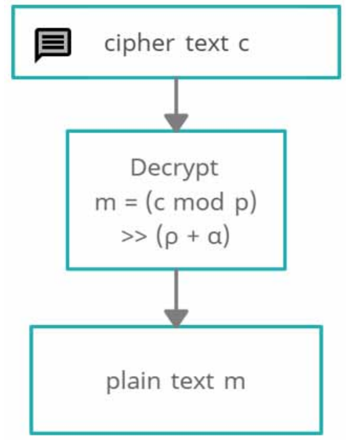

given message m
compute M = (m << (ρ + α)) + noise.
compute n = p.q
Encrypted message c = M + n

Figure 6. Block diagram: Decryption



3.3.1 Pseudo Code

```
1. Get cipher text c
2. plain text m = (c mod p) >> $(\rho + \alpha)$
```

3.4 Homomorphic Addition

Homomorphism in security means that operating on encrypted values and then producing the same results as when done on plain text on decryption. Homomorphic addition is adding two cipher texts and achieving the result of addition of them in plain text after decryption. In the derived algorithm, the cipher texts can be added as it is and can be decrypted to get the desired output. The number of additions is limited by the value of a parameter. Yet the parameter can be modified to achieve practically unlimited number of additions using the derived algorithm.

The algorithm is expected to produce better performance for all the operations including key generation, encryption, decryption and homomorphic addition when compared with Paillier cryptosystem. It takes as input, two cipher texts are fed into this unit and gives as output the addition of those two cipher texts as shown in Figure 7.

3.4.1 Pseudo Code

1. Get cipher text c1
2. Get cipher text c2
3. Perform addition using c1 and c2
4. Store the value in a variable c add

3.5 Prime Generation

This module gets the length as an input. The length is usually in terms of number of bits that the prime number needs to be generated. A number is generated in random between $2^{bits-1} + 1$ and 2^{bits}. This number is checked whether prime or not. The function is executed until a prime number is found. When a prime number is obtained, the same will be returned to the function call. The primality test is done using Rabin Miller Primality test which is an unconditional probabilistic algorithm.

It takes as input, length of the prime number to be generated in bits and gives prime number of given bits in length as shown in Figure 8.

3.5.1 Pseudo Code - Rabin-Miller

```
Input: n > 3, an odd integer to be tested for primality;
Input: k, a parameter that determines the accuracy of the test
Output: composite if n is composite, otherwise probably prime
write n - 1 as $2^s$·d with d odd by factoring powers of 2 from n - 1
LOOP: repeat k times: pick a randomly in the range [2, n - 2] x ←
$a^d$
mod n if x = 1 or x = n - 1 then do next LOOP for r = 1 .. s - 1 x
← $x^2$
mod n if x = 1 then return composite if x = n - 1 then do next
LOOP return  composite return probably prime
```

Figure 7. Block diagram: Homomorphic addition

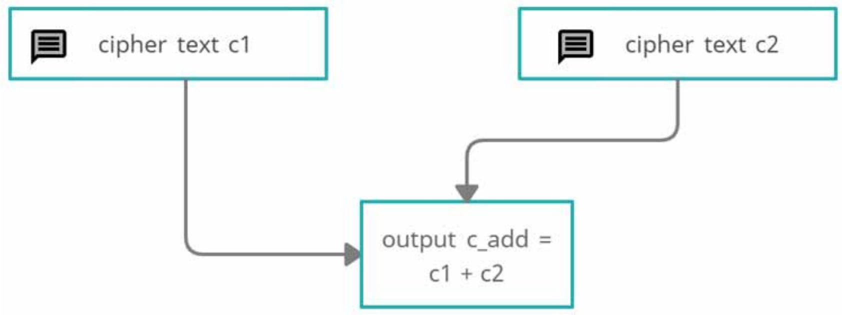
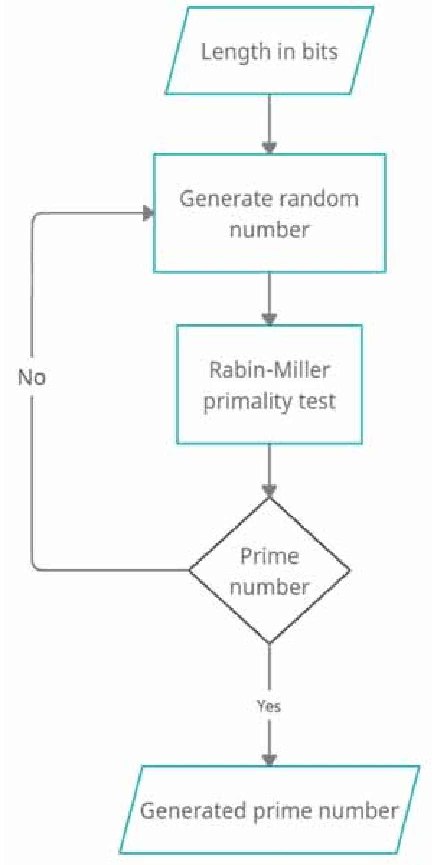


Figure 8. Block diagram: Prime generation



3.6 CryptDB: Setting Up

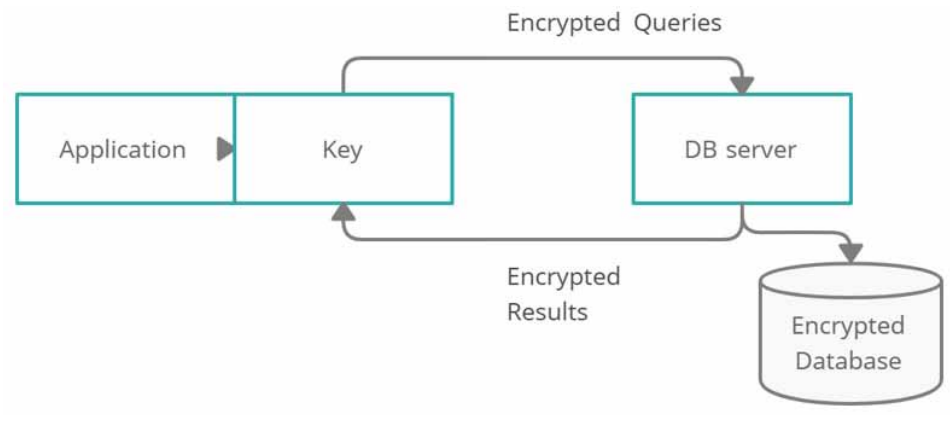
CryptDB is an open source experimental modified MySQL database application which operates over completely encrypted values. It works on the basis of SQL Aware Encryption schemes which encrypts data items depends upon the provided SQL queries. The various encryption layers enables the SQL operations over the encrypted values. Sensitive information of the users stored in a third-party cloud service providers poses a huge threat to the privacy of the user data. Thus, CryptDB provides a way to overcome the above described security concern.

3.6.1 Implementation Overview

To overcome the above discussed security threat, CryptDB stores all the user information in encrypted format. For every database query, queries will be sent to the server in encrypted format. These encrypted queries are executed on the encrypted database values and the result is also sent in an encrypted manner. The decryption will take place only at the front end system interface of the authenticated user.

CryptDB utilizes various security algorithms to encrypt the data into various layers. To name some of the algorithms, AES, DES, CBC, HMAC and Paillier. Paillier is used to achieve the HOM (Homomorphic Encryption) layer which is used to perform operations on constants such as adding numerals. Paillier algorithm possess the property of additive homomorphism which will sum numbers in their encrypted format and the decryption yields the summed up value as shown in Figure 9.

Figure 9. Block diagram: CryptDB



3.6.2 Application and Algorithm

Plain text queries fed to the application is converted into encrypted queries and then provided to the database server. Constant values are encrypted in the HOM layer and then stored in the database server. When a query involves operations on the constant values, FAHE1 and FAHE2 are employed instead of Paillier in the CryptDB application. This will yield query results in an efficient manner.

It takes as input, plain text queries are given to the database server which then encrypts and sends the encrypted queries to the database and gives encrypted results which are sent to the database which decrypts and gives plain text results.

3.7 Integrating CryptDB With FAHE1 and FAHE2

In this application, homomorphic operations such as addition and multiplication are done with the help of Paillier cryptographical system. These modules are replaced with the derived algorithms FAHE1 & FAHE2 (Fast Additive Homomorphic Algorithm). This helps to reduce the execution time of the encrypted SQL queries and thus provides higher amount of performance gain. The modified application is tested with some sample queries and the performance analysis is bench marked against the standard algorithm for comparison.

It takes as input, plain text queries which involves operations on constants are encrypted and then processed with the HOM layer which comprises the FAHE1 and FAHE2 and gives encrypted results of the homomorphic additive process which on decryption yields the desired results.

4. EXPERIMENTAL RESULTS

This section provides a detailed description of the implementation of the entire project. First part of this chapter describes how the algorithm is implemented and their corresponding outputs are also shown. The outputs of both FAHE1 and FAHE2 are displayed individually. The next section shows how to set up the CryptDB application to create and run the encrypted databases. The further sections of the chapter explains how to integrate FAHE1 and FAHE2 into the CryptDB as a replacement to the Paillier algorithm and thus improve the performance of the encrypted databases. Some of the changes needed in the other modules of the application are also covered in the below sections.

4.1 FAHE1 and FAHE2 Algorithm Implementation

The proposed algorithm is implemented at first as an initial progress towards the project. All the modules in the algorithm such as key generation, encryption, decryption and addition are implemented as individual functions so that the implementation can be extended to a greater amount due to the high amount of cohesion and less coupling between the functionalities. The implemented algorithm is then integrated into the CryptDB to achieve improved performance gain.

The function calls to the functionalities such as key generation, prompt for the user input and encryption. All the implementation details are abstracted inside the individual functions. This improves the readability of the code to a higher amount. Also any future changes to the functions can be employed easily with respect to the necessary modules alone. The implementation details of the functionalities are described in the following sub sections.

4.1.1 FAHE1

To begin the implementation of the algorithm, necessary parameters are first initialized and further needed values are derived from the former parameters. From the Key set, encryption and decryption is performed using the appropriate formulas with the help of the encryption and decryption key sets respectively.

4.1.2 FAHE2

FAHE2 is also implemented in a similar fashion with a small change in the encryption and decryption processes since this involves the application of an additional noise to the algorithm.

4.1.3 Prime Generation

For generating a prime number of given size, a random number is generated and the number is tested whether prime or not. The function will be executed until a prime number is found. For the purpose of testing for prime, RabinMiller primality test is used. This is an unconditional probabilistic algorithm which determines whether a number given is probable to be prime. The random number is generated with the help of the urandom class by initializing with a seed value. Once the generated value possess the property of primality, the same will be returned in a ZZ class format.

4.2 Setting Up CryptDB

CryptDB is an encrypted database where all the user data will be stored in an encrypted format and the SQL queries will also be executed over the encrypted values. The application is set up in an Ubuntu 12.04 LTS system with a RAM of size 2 GB and a disk space of 20 GB. Before building the application, necessary packages such as Bazaar, Bison, Gtkdoc, Autoconf, Automake, Libtool, Flex, Gcc-4.6, G++-4.6, Cmake and G++ have been installed into the system with the help of Ubuntu libraries. Then the application is built successfully following the required procedures. Once the output files are generated from the build, MySQL Server has been configured with a proxy using the command shown below. Thus, all the subsequent MySQL Connections to the same back end address will be interrupted by the CryptDB server and operated as an encrypted database. The necessary outputs are shown in the below figures.

4.3 FAHE Implementation in CryptDB

Among the various encryption layers in the application, HOM layer of encryption is achieved with the help of the Paillier cryptographical algorithm which possess the property of additive homomorphism. This HOM layer is used when the SQL queries involves operations on the numbers. Thus, under the folder “crypto”, Paillier algorithm is implemented in files “paillier.hh” and “paillier.cc”. These contains function declarations in the header file and the function definitions in the cpp file of the functionalities key generation, encryption, decryption and addition. These function definitions are modified with the FAHE implementation code appropriately. The function definitions in the header are also changed with the necessary arguments to the function. Thus FAHE1 algorithm is implemented into the CryptDB application replacing the Paillier Cryptographical algorithm.

4.4 Changing the Makefrag

For generating random numbers, a linux package called “gmp” has to be implemented. Some of the methods in this package are not linked during compile time implicitly. So they should be linked dynamically in run time. Thus the Makefrag file is modified with a linker flag “-lgmpxx”.

4.5 Changing the User Defined Functions Handlers

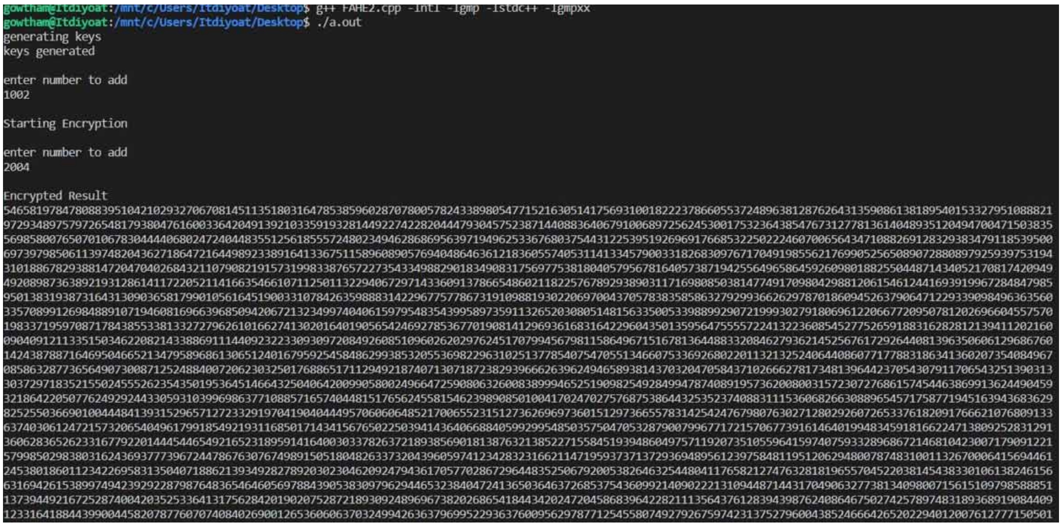
CryptDB utilizes the feature of User Defined Functions (UDF) to create the encrypted database application and also to process the encrypted SQL queries. These UDFs for the respective SQL operations are declared in the “cryptohandlers.cc” file which will make calls to the UDFs for the appropriate encryption algorithms including Paillier. Paillier is used for the purpose of addition of numbers. This will make calls to the “paillier.cc” file which contains code for the implementation of the FAHE. The encrypted addition by the method of FAHE is performed in the User De- fined Function defined in the file “edb.cc” which is located under the folder “udf”. Thus the Paillier algorithm used for the purpose of summation is completely replaced by the FAHE algorithm in the CryptDB application. The application is rebuilt and executed to utilize higher performance gain with the SQL queries involving summation. The respective changes in Figures 10, 11, 12 and 13 show the output of the operations such as insertion, updation and selection of rows in the CryptDB application with FAHE1 and FAHE2 as a replacement to the Paillier algorithm.

5. RESULTS ANALYSIS

In this section, the performances of FAHE1 and FAHE2 are presented and are compared with the performance of the Paillier cryptosystem. The choice of Paillier is motivated by the fact that it is among the most efficient algorithms supporting homomorphic encryption and thus, provides the same functionality as FAHE1 and FAHE2 in a pre-quantum setting. The benchmarked operations were:

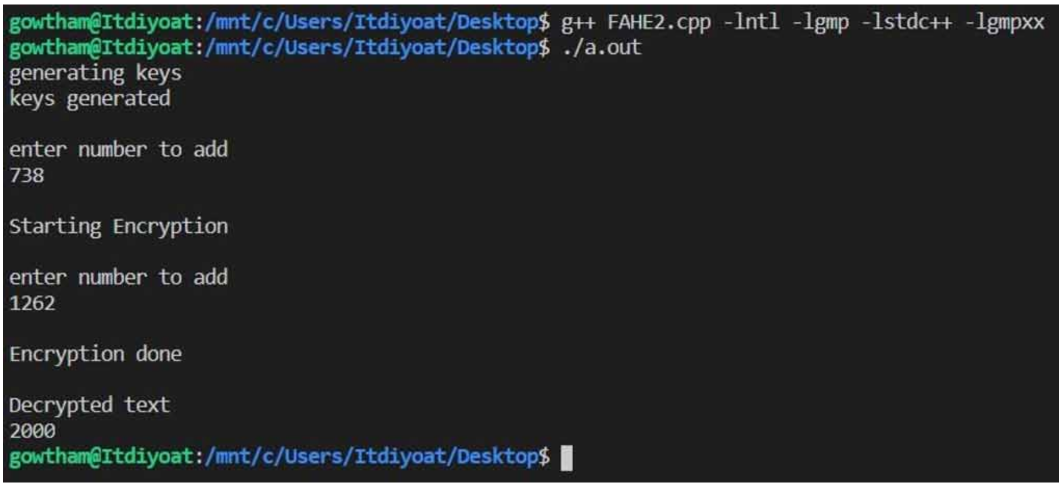
1. Key Generation
2. Encryption
3. Decryption

Figure 10. FAHE2 - Encryption - Output



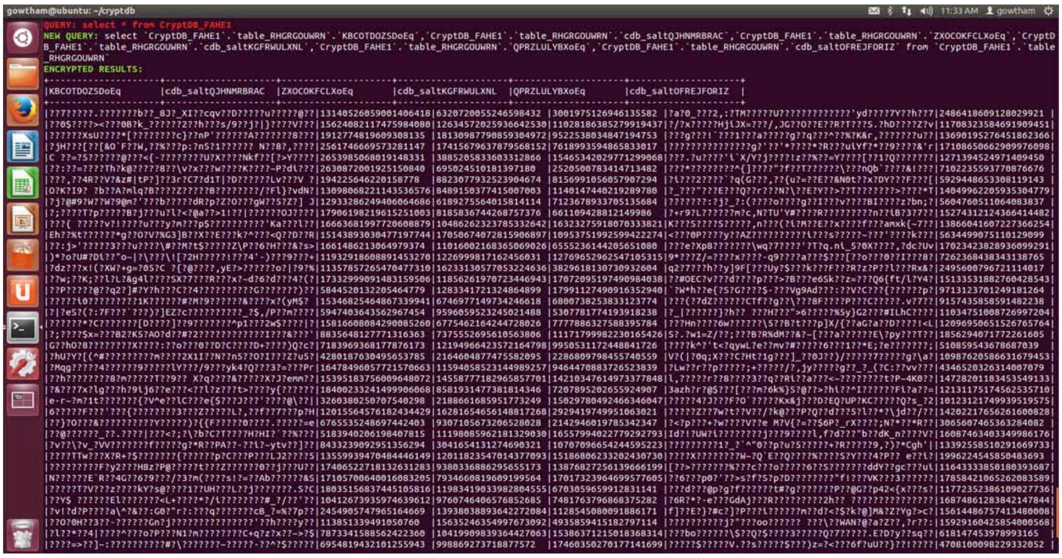
```
gowtham@Itdiyoat:/mnt/c/Users/Itdiyoat/Desktop$ g++ FAHE2.cpp -lntl -lgmp -lstdc++ -lgmpxx
gowtham@Itdiyoat:/mnt/c/Users/Itdiyoat/Desktop$ ./a.out
generating keys
keys generated

enter number to add
1002

Starting Encryption

enter number to add
2004

Encrypted Result
546581978478088395104210293270670814511351803164785385960287078005782433898054771521630514175693100182223786605537248963812876264313590861381895401533279510888 21
972934897579726548179380476160033642049139210335919328144922742282044479304575238714408836406791006897256245300175323643854767312778136140489351204947004715 03835
5698580076507010678304444068024724044835512561855572480234946286869563971949625336768037544312253951926969176685322502224607006564347108826912832938347911 8539500
697397985061139748204362718647216449892338916413367511589608905769404864636121836055740531141334579003318268309767170491985562176990525650890728808979259397 53194
310188678293881472047040268432110790821915731998338765722735433498829018349083175697753818040579567816405738719425564965864592609801882550448714340521708174 20949
4920898736389219312861411722052114166354661071125011322940672971433609137866548602118225767892938903117169808503814774917098042988120615461244169391996728484 7985
950138319387316431309036581799010561645190033107842635988831422967757786731910988193022069700437057838358586327929936626297870186094526379064712293390984963 63560
3357089912698488910719460816966396850942067213234997404061597954835439958973591132652030805148156335005339889929072199930279180696122066772095078120269660455 7570
1983371959708717843855338133272796261016627413020164019056542469278536770190814129693616831642296043501359564755557224132236085452775265918831628281213941120 2160
09040912113351503462208214338869111440923223309309720849260851096026202976245170799456798115864967151678136448833208462793621452567617292644081396350606129686760
14243878871646950466521347958968613065124016795925458486299385320553698229631025137785407547055134660753369268022011321325240644086077177883186341360207354084967
08586328773656490730087125248840072062303250176886517112949218740713071872382939666263962494658938143703204705843710266627817348139644237054307911706543251390313
303729718352155024555262354350195364514664325040642009905800249664725908063260083899946525190982549284994787408919573620080031572307276861574544638699136244 90459
3218642205077624929244330593103996986377108857165740448151765624558154623989085010041702470275768753864432535237408831115360682663088965457175877194516394368 3629
82525503669010044484139315296571272332919704190404449570606064852170065523151273626969736015129736655783142542476798076302712802926072653376182091766621076809133
63740306124721573206540496179918549219311685017143415676502250394143640668840599299548503575047053287900799677172157067739161464019948345918166224713809252831291
36062836526233167792201444544654921652318959141640030337826372189385690181387632138522715584519394860497571192073510559641597407593328968672146810423007179091221
57998502983803162436937773967244786763076749891505180482633732043960597412342832316621147195937371372936948956123975848119512062948007874831001132670006415694461
24538018601123422695831350407188621393492827892030230462092479436170577028672964483525067920053826463254480411765821274763281819655704522038145438330106138246156
63169426153899749423929228798764836546460569788439053830979629446532384047241365036463726853754360992140902221310944871443170490632773813409800715615109798588851
13739449216725287400420352533641317562842019020752872189309248969673820268654184434202472045868396422821113564376128394398762408646750274257897483189368919084409
12331641884439900445820787760707408402690012653606063703249942636379699522936376009562978771254558074927926759742313752796004385246664265202294012007612777150501
```

Figure 11. FAHE2 - Decryption - Output



```
gowtham@Itdiyoat:/mnt/c/Users/Itdiyoat/Desktop$ g++ FAHE2.cpp -lntl -lgmp -lstdc++ -lgmpxx
gowtham@Itdiyoat:/mnt/c/Users/Itdiyoat/Desktop$ ./a.out
generating keys
keys generated

enter number to add
738

Starting Encryption

enter number to add
1262

Encryption done

Decrypted text
2000
gowtham@Itdiyoat:/mnt/c/Users/Itdiyoat/Desktop$
```

Figure 12. CryptDB - Update and Select - Output - With FAHE1

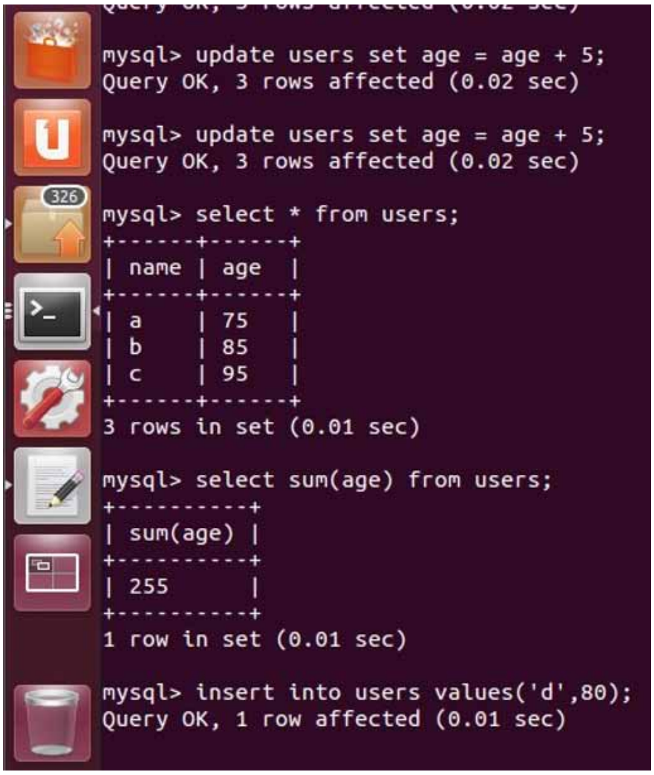


```
gowtham@ubuntu: ~/cryptdb
QUERY: select * from CryptDB_FAHE1
NEW QUERY: select `CryptDB_FAHE1`.`table_RHGRGOUWRN`.`KBCOTDOZSDoEq`,`CryptDB_FAHE1`.`table_RHGRGOUWRN`.`cdb_saltQJHNMRBRAC`,`CryptDB_FAHE1`.`table_RHGRGOUWRN`.`ZXOCOKFCLXoEq`,`CryptD
B_FAHE1`.`table_RHGRGOUWRN`.`cdb_saltKGFRWULXNL`,`CryptDB_FAHE1`.`table_RHGRGOUWRN`.`QPRZLULYBXoEq`,`CryptDB_FAHE1`.`table_RHGRGOUWRN`.`cdb_saltOFREJFORIZ` from `CryptDB_FAHE1`.`table
_RHGRGOUWRN`
ENCRYPTED RESULTS:
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|KBCOTDOZSDoEq       |cdb_saltQJHNMRBRAC  |ZXOCOKFCLXoEq       |cdb_saltKGFRWULXNL  |QPRZLULYBXoEq       |cdb_saltOFREJFORIZ  |
+--------------------+--------------------+--------------------+--------------------+--------------------+--------------------+
|???????.???????b??_8J?_XI??cqv??D?????u?????@??|13140526059001406418|6320720055246598432 |3001975126946135582 |?a?0_???2,:?TM?????U?????????????'yd?????Y??h??7|2486418609128029921 |
|??0$????><???0B?k_??????2??h???s/9??j?|}????V???|15624082117475984080|12634572025936642530|11028186385279919437|?/?x?????HjlJX=???/,JG??O??E??R?T???S.?hD????Z?v|11708323584691909451|
|??????XsU????*[????????c}??nP'??????A???????8???|1912774819609308135 |18130987790859304972|9522538034847194753 |???g????!`???1???a?????g??q???^??%?K&r,???????u??|13690195276451862366|
|?jH???[??[&O`F??W,??%???p:?nS?1??????? N??8?,????|2561746669573281147 |17415679637879568152|7618993594865833017 |???????????????g?'??'*????*?R???uiYf?*??9????&'r|17108650662909976098|
|C ??=?5??????@???<{-????????U?X????Nkf??[?>Y????|2653985068019148331 |3885205833603312866 |15465342029771299068|???.?u????'i`X/Y?j?????!z??%??=Y????[??1?Q???????|1271394524971409450 |
|??:??=????Th?k@?????8??\v?x???w????K????-P?di???|2630872001925150840 |695024510181397180  |2520500783414713482 |???*????????^{]????"?f??T??????\???nQb`????&!???|7102235593770876676 |
|???,??4R??V?&z#|tP?]??3r?C??d1T|?D??????Lv???V ?|194225646220158778  |8823077932523904674 |8156991056057907294 |?l???2????`?q(G???,??{u?=??E??&N0t??x?DY???F???[|5929448653308119143 |
|O?K?I9? ?b??A?mlq?B????Z?????B????????/?Fl}?vdN?|13098068221143536576|8489150377415007003 |1140147440219289780 |?_???"???E???Q??r???N?\??E?Y??>??T???????>????*T|14049962205935304779|
|?j?@#9?W??W?9@m?'???b?????dR?p?Z?O???gW??5?Z?] J|12933286249406064686|6189275564015814114 |[illegible]
|?;????T?p?????B?j???u?l?<?@a??>1!??|??????OJ????|17906198219615251003|8185836744268757376 |[illegible]
|???{ ?????v?!????u???y?m???p$??????????'Ka???l??|16663681997720608879|10486262323785332642|[illegible]
|Eh??%t??????*g??O?V?%G3]B??X??E???k?^???<Q??D??R|15143893030477197744|17050674072815906897|[illegible]
|??:j>'?????3???u????\#??M?t$?????Z\P??6?H???&?s>|1661486213064979374 |11016002168365069026|[illegible]
|)*?o?U#?Di??"o~|?\???\![?2H?????!???4'-)???9???+|11932918608891453270|12269998171 62456031|[illegible]
|?dz???x!(?XW?+g=?05?C ?{?@????,yE?>??????o?|?9?%|11357857265470477310|16233130577053224636|[illegible]
|??w;??K;??l?L?&g4l????SX?????R???x?-d?6?d???4?(?|17332999091483159506|11856261970723446943|[illegible]
|??P?????@??q2?]#?Y?h???C??4??????????G???????)??|5844520132205464779 |12833417213248648 99|[illegible]
|?????i0?????????1K??????#?M?9???????&????x?(yM$? |15346825464867339941|6746977149734246618 |[illegible]
[illegible]
```

4. Homomorphic Addition

Aiming to obtain uniform distribution for input messages, random samples were generated beforehand for each test. The most time consuming process in all schemes is the key generation. The reason Fast Additive algorithm provides better results than Paillier algorithm is that in FAHE1 and FAHE2, only single prime of size η is created, whereas Paillier involves the generation of two primes of size around n/2.

Figure 13. CryptDB - Aggregation (Sum) - With FAHE1



```
mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> select * from users;
+------+------+
| name | age  |
+------+------+
| a    | 75   |
| b    | 85   |
| c    | 95   |
+------+------+
3 rows in set (0.01 sec)

mysql> select sum(age) from users;
+----------+
| sum(age) |
+----------+
| 255      |
+----------+
1 row in set (0.01 sec)

mysql> insert into users values('d',80);
Query OK, 1 row affected (0.01 sec)
```

5.1 Metrics for Evaluation

The performance of the program is analysed by counting the cycles. Blocks that affect the performance of the algorithm are split and individual cycles of the blocks are counted. For counting the cycles, a python package named “hwcounter” is used.

i) $start = count$

is initialized at the start of the block for which the cycles have to calculated.

ii) $elapsed = count\_end() - start$

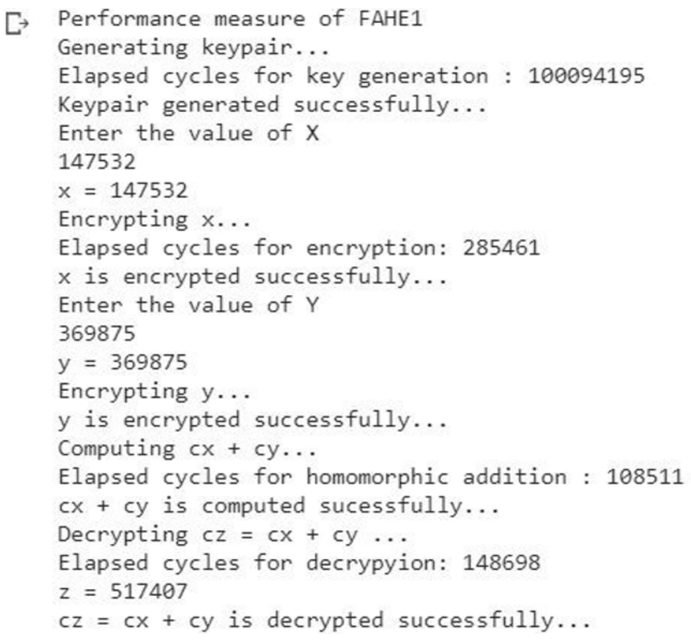
at the end of the block can calculate the total number of cycles taken for the individual block.

For calculating the total cycles taken for the execution of the program, individual blocks are executed multiple times and average of them are taken and added with other blocks of the program. Finally, the cycles taken for the program are benchmarked against the Paillier system and performance gain is calculated.

In classical crpytDB implementation, the homomorphic layer is implemented using the Paillier cryptosystem, thus enabling additions on encrypted data. In this scenario, FAHE1 and FAHE2 can be seen as efficient and effective drop-in replacements for Paillier.

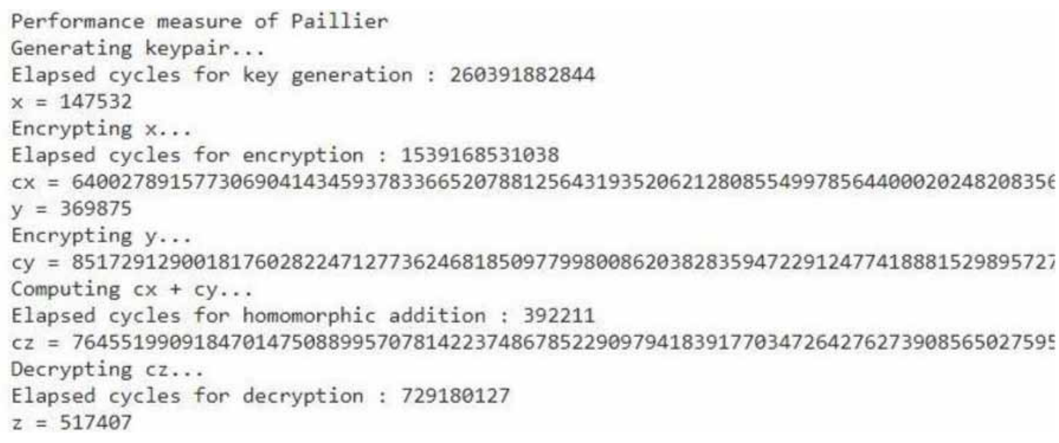
Finally the runtime of the encrypted queries using both paillier system and FAHE system is calculated to analyze the performance gain the latter has over the former. The outputs of performance testing of both the Paillier and FAHE1 are shown in the below figures. From the Figures 14 and 15 it can be observed that the initial step of the algorithm which is the key generation process takes 26 thousand crore cycles wheares the FAHE takes only 10 crore cycles. The encryption and decryption stages of Paillier algorithm takes around 1.5 lakh crore cycles and 72 crore cycles respectively. On the other hand, FAHE takes only 3 lakhs and 1.5 lakhs cycles for the encryption and decryption. This produces an enormous performance gain with FAHE when compared to the classical Paillier algorithm. In addition to these, FAHE is also quantum resistant which will make the algorithm highly secure comparing with the Paillier cryptography which is not quantum resistant in nature. Figure 16 shows the bar graph of the cycles taken by the algorithms for the process of key generation. It is clear from the figure that FAHE1 is 2600 times faster than the classical Paillier algorithm. FAHE2 is also 790 times faster than the Paillier cryptography.

Figure 14. Performance measure of FAHE



```
Performance measure of FAHE1
Generating keypair...
Elapsed cycles for key generation : 100094195
Keypair generated successfully...
Enter the value of X
147532
x = 147532
Encrypting x...
Elapsed cycles for encryption: 285461
x is encrypted successfully...
Enter the value of Y
369875
y = 369875
Encrypting y...
y is encrypted successfully...
Computing cx + cy...
Elapsed cycles for homomorphic addition : 108511
cx + cy is computed sucessfully...
Decrypting cz = cx + cy ...
Elapsed cycles for decrypyion: 148698
z = 517407
cz = cx + cy is decrypted successfully...
```

Figure 15. Performance measure of Paillier



```
Performance measure of Paillier
Generating keypair...
Elapsed cycles for key generation : 260391882844
x = 147532
Encrypting x...
Elapsed cycles for encryption : 1539168531038
cx = 640027891577306904143459378336652078812564319352062128085549978564400020248208356
y = 369875
Encrypting y...
cy = 8517291290018176028224712773624681850977998008620382835947229124774188815298957 27
Computing cx + cy...
Elapsed cycles for homomorphic addition : 392211
cz = 7645519909184701475088995707814223748678522909794183917703472642762739085650275 95
Decrypting cz...
Elapsed cycles for decryption : 729180127
z = 517407
```

Figure 16. Elapsed cycles of key generation

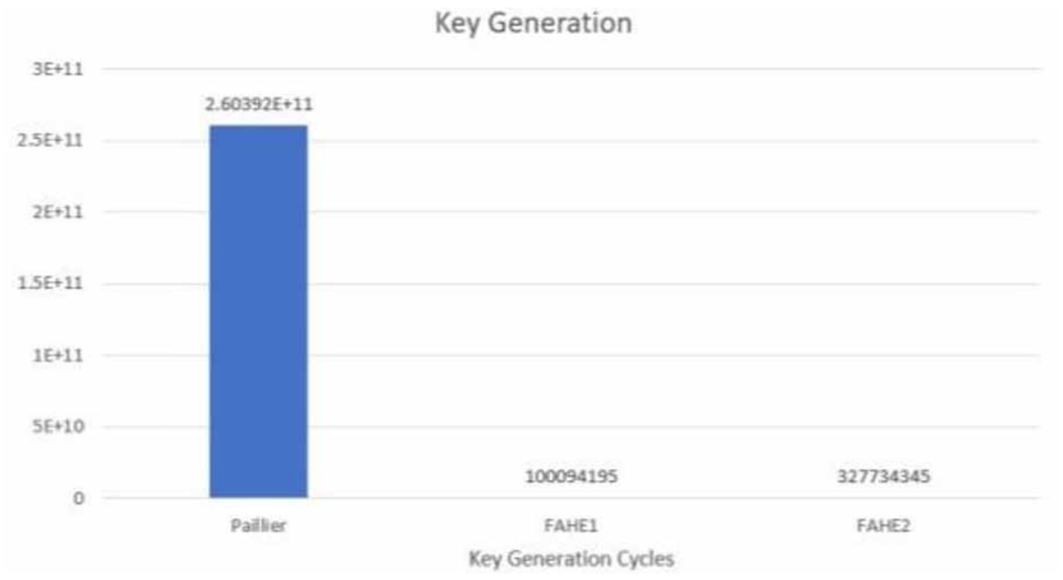


Figure 17 shows the elapsed cycles taken for encryption of the plain text. Encryption produces very significant improvement when compared to the Paillier. FAHE1 and FAHE2 take around 2.5 lakhs whereas the Paillier encryption takes nearly 1.5 lakh crore cycles. This shows that the FAHE's are 5 lakh times faster than the Paillier encryption process. FAHE2 also has an advantage of smaller ciphertext when compared to FAHE1.

Figure 18 displays the cycles needed to complete the decryption of the ciphertext generated by the respective algorithms. This also shows impressive results as FAHE1 and FAHE2 take around 1.5 lakh and 2 lakh cycles for the decryption process respectively. On the other hand, the Paillier algorithm takes on the range of 72 crore which stats that the FAHE is nearly 3500 times faster than the Paillier algorithm.

Figure 17. Elapsed cycles of encryption

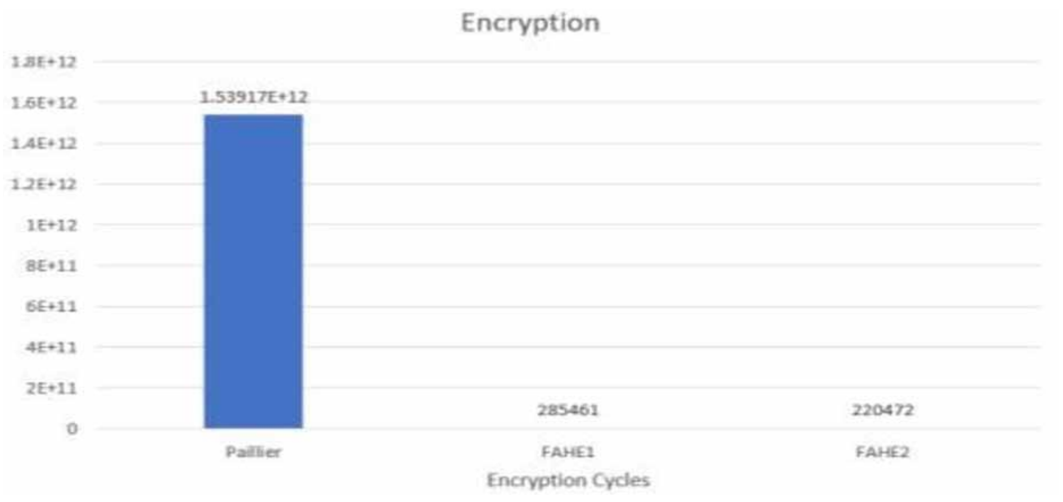


Figure 18. Elapsed cycles of decryption

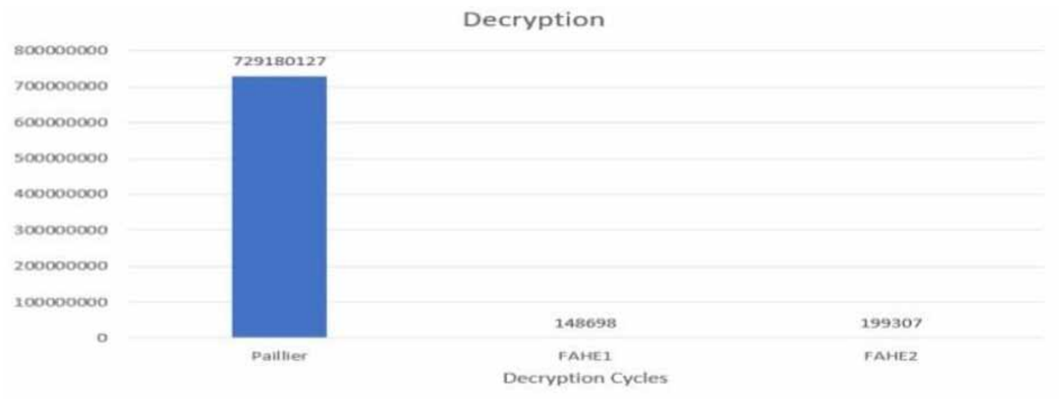


Figure 19 shows the bar graph of the cycles for the addition process. The time taken to complete the homomorphic addition in Paillier is slightly higher than other two algorithms.

Table 1 shows a representation of the comparison of the elapsed cycles for all the three algorithms, the Paillier, the FAHE1 and the FAHE2 of their Key generation, Encryption, Decryption and Homomorphic Addi tion.

Figure 20 shows that the CryptDB with the Paillier used for homomorphic operations takes around 0.28 seconds for the first insertion. Updating the HOM constants take around 0.04 to 0.05 seconds with the Pail lier algorithm.

When the CryptDB HOM layer is replaced with the FAHE1 algorithm, first insertion takes around 0.03 seconds and further insertions with the time of 0.01 second. This produces significant performance gain when compared with the Paillier algorithm. Updating a row also occurs at a time period of 0.02 seconds on an average. Figure 21 displays the time taken to complete the insertion and

Figure 19. Elapsed cycles of homomorphic addition

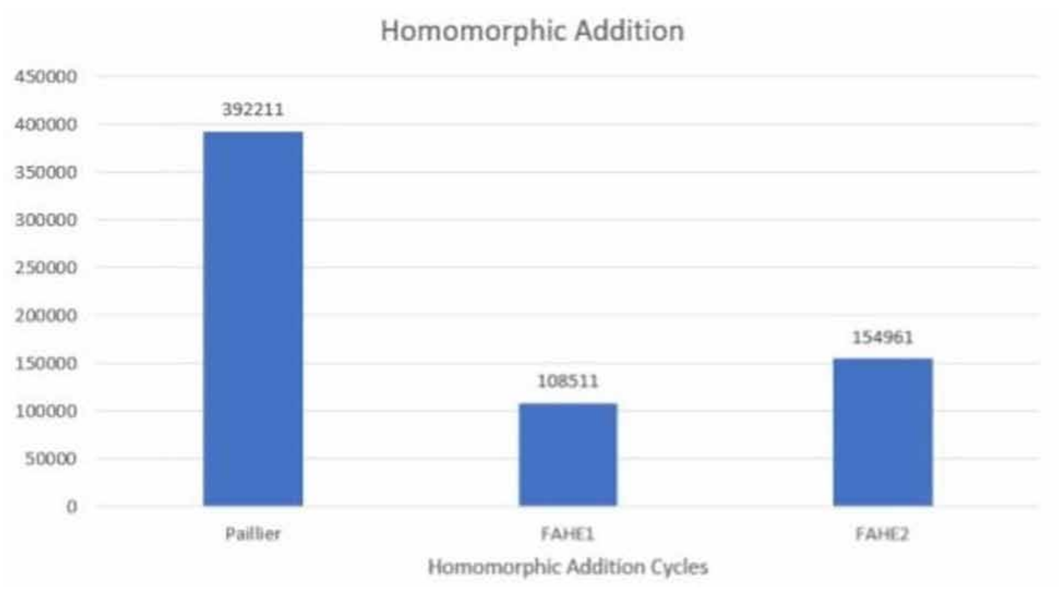


Table 1. Comparing the algorithms

Operations / Algorithms (in Elapsed Cycles)	Paillier	FAHE1	FAHE2
Key Generation	260391882844	100094195	327734345
Encryption	1539168531038	285461	220472
Decryption	729180127	148698	199307
Homomorphic Addition	392211	108511	154961

updation in the HOM layer when FAHE2 is used to achieve the homomorphic operations. This also executes the queries in the time period as same as the FAHE1 with an advantage of smaller cipher text when compared with FAHE1.

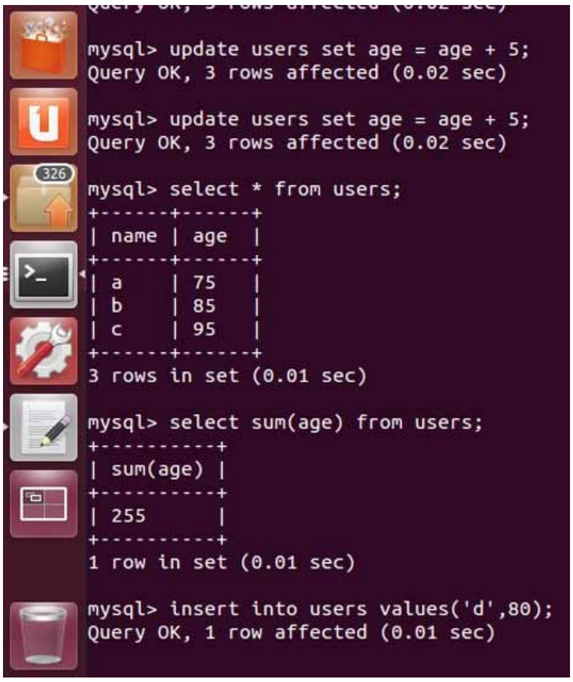
5.2 Test Cases

CryptDB is an encrypted database which takes plain text queries as input from the front end. The front end then encrypts the queries and sends to the database server. All the encryption and decryption are performed by the CryptDB server lying between the end - user and the database server. Any subsequent query to the system is interpreted by the CrybtDB server and the output is produced to the front end after fetching the necessary data from the encrypted database.

5.2.1 System Unit Testing

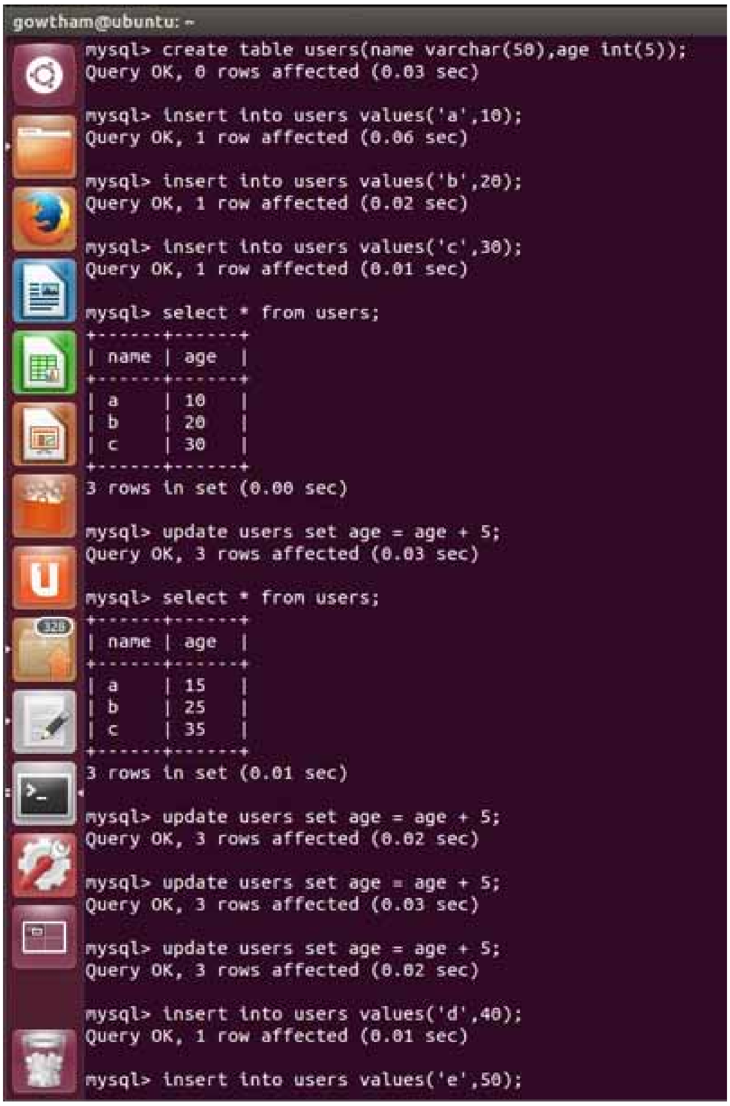
The application is unit tested using the python package called "unittest". This can be installed with the pip command called "pip install unittest". All the modules of both the application are covered by generating random inputs followed by their encryption and addition of the encrypted values. These

Figure 20. Execution time in CryptDB with FAHE1



```
mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> select * from users;
+------+------+
| name | age  |
+------+------+
| a    | 75   |
| b    | 85   |
| c    | 95   |
+------+------+
3 rows in set (0.01 sec)

mysql> select sum(age) from users;
+----------+
| sum(age) |
+----------+
| 255      |
+----------+
1 row in set (0.01 sec)

mysql> insert into users values('d',80);
Query OK, 1 row affected (0.01 sec)
```

Figure 21. Execution time in CryptDB with FAHE2



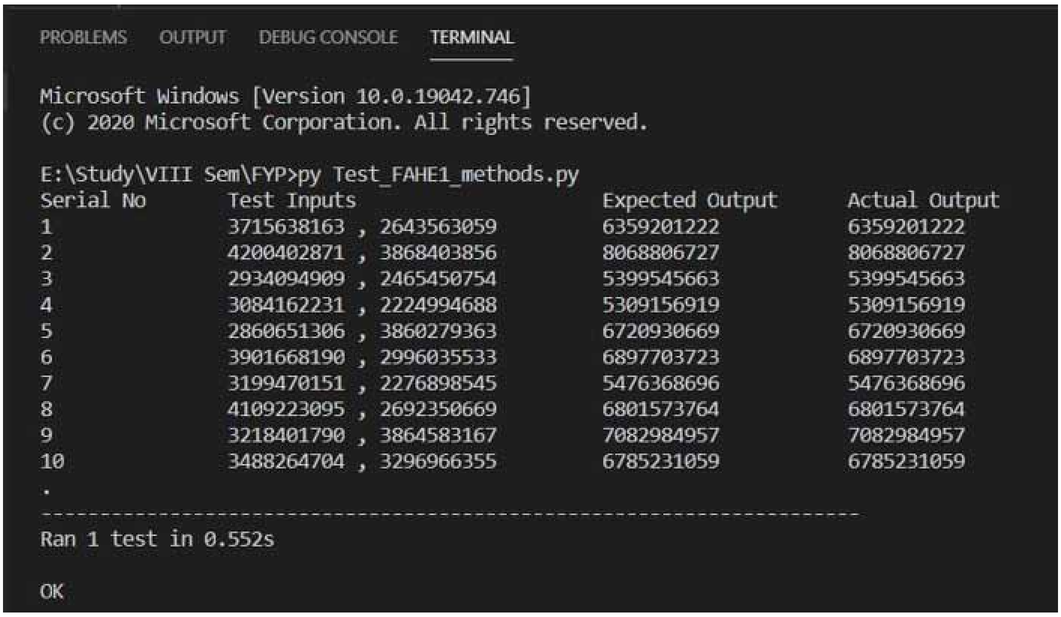
```
gowtham@ubuntu: ~
mysql> create table users(name varchar(50),age int(5));
Query OK, 0 rows affected (0.03 sec)

mysql> insert into users values('a',10);
Query OK, 1 row affected (0.06 sec)

mysql> insert into users values('b',20);
Query OK, 1 row affected (0.02 sec)

mysql> insert into users values('c',30);
Query OK, 1 row affected (0.01 sec)

mysql> select * from users;
+------+------+
| name | age  |
+------+------+
| a    | 10   |
| b    | 20   |
| c    | 30   |
+------+------+
3 rows in set (0.00 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.03 sec)

mysql> select * from users;
+------+------+
| name | age  |
+------+------+
| a    | 15   |
| b    | 25   |
| c    | 35   |
+------+------+
3 rows in set (0.01 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.03 sec)

mysql> update users set age = age + 5;
Query OK, 3 rows affected (0.02 sec)

mysql> insert into users values('d',40);
Query OK, 1 row affected (0.01 sec)

mysql> insert into users values('e',50);
```

values are decrypted with the application module and are compared with the expected output for the generated inputs. The results are displayed in Figure 22 and Figure 23.

6. CONCLUSION

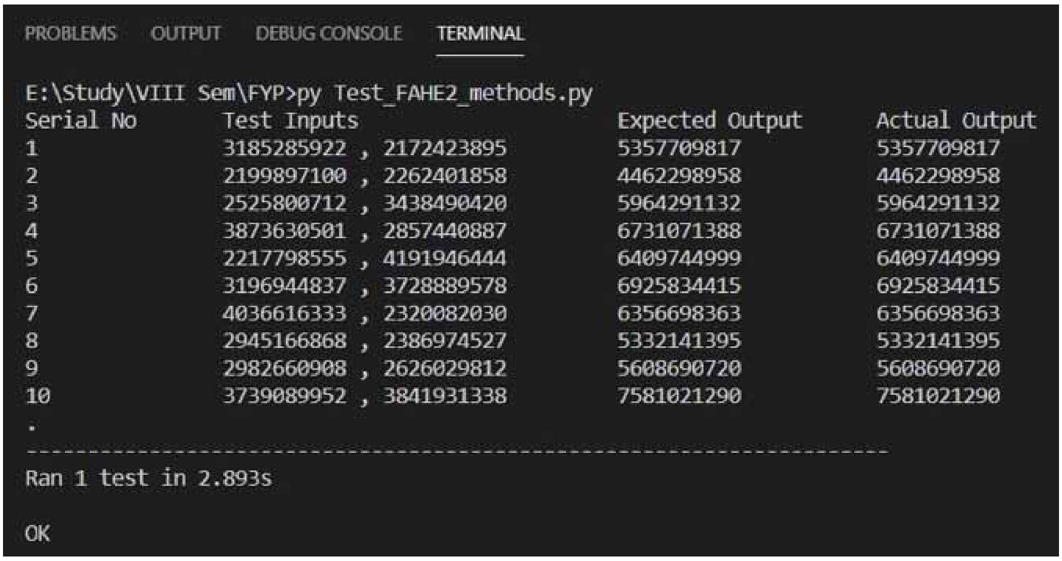
From what the industry have seen so far, the Fast Additive Homomorphic Encryption algorithms are way faster than the Paillier algorithm in each and every aspect. In key generation there is a difference of two fifty billion in terms of elapsed cycles and in encryption it goes further and reaches a whopping difference of 1 trillion cycles. The closest Paillier encryption can come to FAHE is in Homomorphic addition. But the least difference is two hundred thousand. One cannot decide efficiency based on only Algorithm's performance. There has to be some application which can employ both the encryption methods and tell the difference in real world applications.

Figure 22. FAHE1 - System testing



```
Microsoft Windows [Version 10.0.19042.746]
(c) 2020 Microsoft Corporation. All rights reserved.

E:\Study\VIII Sem\FYP>py Test_FAHE1_methods.py
Serial No       Test Inputs                  Expected Output     Actual Output
1               3715638163 , 2643563059      6359201222          6359201222
2               4200402871 , 3868403856      8068806727          8068806727
3               2934094909 , 2465450754      5399545663          5399545663
4               3084162231 , 2224994688      5309156919          5309156919
5               2860651306 , 3860279363      6720930669          6720930669
6               3901668190 , 2996035533      6897703723          6897703723
7               3199470151 , 2276898545      5476368696          5476368696
8               4109223095 , 2692350669      6801573764          6801573764
9               3218401790 , 3864583167      7082984957          7082984957
10              3488264704 , 3296966355      6785231059          6785231059
.
----------------------------------------------------------------------
Ran 1 test in 0.552s

OK
```

Figure 23. FAHE2 - System testing



```
E:\Study\VIII Sem\FYP>py Test_FAHE2_methods.py
Serial No       Test Inputs                  Expected Output     Actual Output
1               3185285922 , 2172423895      5357709817          5357709817
2               2199897100 , 2262401858      4462298958          4462298958
3               2525800712 , 3438490420      5964291132          5964291132
4               3873630501 , 2857440887      6731071388          6731071388
5               2217798555 , 4191946444      6409744999          6409744999
6               3196944837 , 3728889578      6925834415          6925834415
7               4036616333 , 2320082030      6356698363          6356698363
8               2945166868 , 2386974527      5332141395          5332141395
9               2982660908 , 2626029812      5608690720          5608690720
10              3739089952 , 3841931338      7581021290          7581021290
.
----------------------------------------------------------------------
Ran 1 test in 2.893s

OK
```

There comes Cryptdb, a database which can hold thousands of bits of encryted text. It has modules which could be used to operate on data without decrypting them. One of those modules is paillier, which is used to perform multiplication and addition on encrypted data. So FAHE was replaced to find out the performance gain in real world applications and it did well as expected. But FAHE and cryptdb itself has some drawbacks.

Noticeably there are some drawbacks to the FAHE algorithms. FAHE out performed the Paillier in every aspect but the main drawback is it doesn't support multiplication. As the name suggests it is additive. But with Paillier supporting both addition and multiplication, one should doubt the replacement cost of paillier. Not only FAHE is defective, but cryptdb itself has some. Cryptdb is developed to operate and store encrypted data. It was developed in 2012 by MIT. But after that the support to cryptdb was stopped and community became inactive. So with today's system architectures one may find it difficult to implement and run without any community support. Cryptdb is developed to implement very few SQL operations. This might be a great problem for those who want to implement a full-fledged secure application.

CONFLICTS OF INTEREST

We wish to confirm that there are no known conflicts of interest associated with this publication and there has been no significant financial support for this work that could have influenced its outcome.

FUNDING STATEMENT

No funding was received for this work.

Correspondence should be addressed to Angelin Gladston; angelingladston@gmail.com

REFERENCES

Cardoso dos Santos, L., Bilar, G. R., & Fabio, D. P. (2015). Implementation of the fully homomorphic encryption scheme over integers with shorter keys. *7th International Conference on New Technologies, Mobility and Security (NTMS)*. IEEE. doi:10.1109/NTMS.2015.7266495

Chaudhary, P., Gupta, R., Singh, A., & Majumder, P. (2019). *Analysis and Comparison of Various Fully Homomorphic Encryption Techniques*. 2019 International Conference on Computing, Power and Communication Technologies (GUCON), NCR New Delhi, India.

Chaudhary, P., Gupta, R., Singh, A., & Majumder, P. (2019). Analysis and Comparison of Various Fully Homomorphic Encryption Techniques. *International Conference on Computing, Power and Communication Technologies (GUCON)*. INSPEC

Sah, C. & Gupta, P. (2019). Comparative Analy sis of Zero-Knowledge Proofs Technique using Quadratic Residuosity Problem. *6th International Conference on Computing for Sustainable Global Development (INDIACom)*. Springer. ISBN:978-1-5386-9271-4

Du, C., & Bai, G. (2016). Towards efficient polynomial multiplication for lattice-based cryptography. *IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. doi:10.1109/ISCAS.2016.7527456

Cominetti, E. & Simplicio, M. (2020). Fast Additive Partially Homomorphic Encryption From the Approximate Common Divisor Problem. *IEEE Transactions on Information Forensics and Security*. IEEE. 10.1109/TIFS.2020.2981239

Fujita, T., Kogiso, K., Sawada, K., & Shin, S. (2015). Security enhancements of networked control systems using RSA public- key cryptosystem. *10th Asian Control Conference (ASCC)*. IEEE. doi:10.1109/ASCC.2015.7244402

Gu, C. (2010). Public Key Cryptosystems from the Multiplicative Learning with Errors. *International Conference on Multimedia Information Networking and Security*. IEEE. doi:10.1109/MINES.2010.102

Jinsu Kim, J. (2015). A Hybrid Scheme of Public-Key Encryption and Somewhat Homomorphic Encryption. *IEEE Transactions on Information Forensics and Security*, *10*(5). 10.1109/TIFS.2015.2398359

Li, S., Zhao, S., Min, G., Qi, L., & Liu, G. (2022, August 15). Lightweight Privacy-Preserving Scheme using Homomorphic Encryption in Industrial Internet of Things. *IEEE Internet of Things Journal*, *9*(16), 14542–14550. doi:10.1109/JIOT.2021.3066427

Mahmood, Z. H., & Ibrahem, M. K. (2018). *New Fully Homomorphic Encryption Scheme Based on Multistage Partial Homomorphic Encryption Applied in Cloud Computing*. 2018 1st Annual International Conference on Information and Sciences (AiCIS), Fallujah, Iraq. doi:10.1109/AiCIS.2018.00043

Mahmood, Z. H., & Ibrahem, M. K. (2018). New Fully Homomorphic Encryption Scheme Based on Multistage Partial Homomorphic Encryption Applied in Cloud Computing. 1st Annual International Conference on Information and Sciences (AiCIS). IEEE. doi:10.1109/AiCIS.2018.00043

Mazza, S., Patel, D., & Viola, I. (2021, February). Homomorphic-Encrypted Volume Rendering. *IEEE Transactions on Visualization and Computer Graphics*, *27*(2), 635–644. doi:10.1109/TVCG.2020.3030436 PMID:33048733

Nassar, M., Erradi, A., & Malluhi, Q. M. (2015). Paillier's encryption: Implementation and cloud applications. *2015 International Conference on Applied Research in Computer Science and Engineering (ICAR)*, Beirut. doi:10.1109/ARCSE.2015.7338149

Joshi, S. (2020). *An Efficient Paillier Cryptographic Technique for Secure Data Storage on the Cloud*. 2020 4th International Conference on Intelligent Computing and Control Systems (ICICCS), Madurai, India. doi:10.1109/ICI- CCS48265.2020.9121105

Pradhan, P. K., Rakshit, S., & Datta, S. (2019). Lattice Based Cryptography: Its Applications, Areas of Interest Future Scope. *3rd International Conference on Computing Methodologies and Communication (ICCMC)*. IEEE. doi:10.1109/ICCMC.2019.8819706

Ryu, J., Kim, K., & Won, D. (2023). *A Study on Partially Homomorphic Encryption.* 2023 17th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Korea. doi:10.1109/IMCOM56909.2023.10035630

Shihab, T. V. J., & Liji, P. I. (2017). *Simple and secure internet voting scheme using generalized paillier cryptosystem*. 2017 International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT), Kannur. doi:10.1109/ICICICT1.2017.8342623

Yang, H., Xia, Q., Wang, X., & Tang, D. (2012). A New Somewhat Homomorphic Encryption Scheme over Integers. *2012 International Conference on Computer Distributed Control and Intelligent Environmental Monitoring*, Hunan. doi:10.1109/CDCIEM.2012.21

Zhang, P., Sun, X., Wang, T., Gu, S., Yu, J., & Xie, W. (2016). *An acceleratedfully homomorphic encryption scheme over the integers*. 2016 4th International Conference on Cloud Computing and Intelligence Systems (CCIS), Beijing. doi:10.1109/CCIS.2016.7790295

Angelin Gladston is working as Associate Professor at the Department of Computer Science and Engineering, Anna University, Chennai. Her research interests include software engineering, software testing, image processing, social network analysis and data mining.

NaveenKumar is undergraduate student of Department of Computer Science and Engineering, Anna University. His research interests are encryption and network security.

Sanjeev is undergraduate student of Department of Computer Science and Engineering, Anna University. His research interests are encryption and network security.

Gowthamraj is undergraduate student of Department of Computer Science and Engineering, Anna University. His research interests are encryption and network security.