Yash Pragnesh Gandhi

1150578261

Survey on Homomorphic Encryption for Secured Cloud Computing and its alternatives for Secured Data Sharing and Computing

CSci530 Computer Secrity Systems

I have read the Giude to Avoiding Plagiarism published by the student affairs office. I understand what is expected of me with respect to properly citing sources, and how to avoid representing the work of others as my own. The material in this paper was written by me, except for such material that is quoted or indented and properly cited to indicated the sources of the material. I understand that using the words of others, and simply tagging the sentence, paragraph, or section with a tag to the copied source does not constitute proper citation and that if such materiel is used verbatim or paraphrased it must be specifically conveyed (such as through the use of quotation marks or indentation) together with the citation. I further understand that overuse of properly cited quotations to avoid conveying the information in my own words, while it will not subject me to diciplinary action, does convey to the instructor that I do not understand the material enough to explain it in my own words, and will likely result in a lesser grade on the paper.

Signed:_____Yash Pragnesh Gandhi_____

Survey on Homomorphic Encryption for Secured Cloud Computing and its alternatives for Secured Data Sharing and Computing

Yash Pragnesh Gandhi
Department of Computer Science,
University of Southern California
Los Angeles, USA
yashprag@usc.edu

Clifford Neuman
Department of Computer Science,
University of Southern California
Los Angeles, USA
bcn@isi.edu

***Abstract*— Over the past decade, there has been a huge transition of companies from on-prem to Cloud storage. With billions of data generated everyday, cloud-based platforms provide an opportunity to manage, store and compute the data more efficiently and effectively. It may be secured internal company data or customers data encrypted using state-of-the-art encryption, but it is stored on Cloud. The encryption and decryption keys are also often stored on the cloud using the key management services provided by different Cloud Service Providers (CSP). We need to decrypt the cipher-text into raw data whenever there is a need to perform certain operations on it. This leads to sharing the encryption key materials with people across the company's hierarchical level and also with the CSP. Hence, recently there has been a need to apply homomorphic encryption methods which allow us to operate on the data without actually decrypting it. This technique allows the data owners to not reveal the encryption material and still allow access to their subordinate to work on the data or store only the cipher text on cloud. In this paper we attempt to review such methods and the challenges which they pose.**

Keywords—Cloud Computing, Homomorphic Encryption, Functional Encryption, CryptDB

I. INTRODUCTION

Database systems are the fundamental building block of any system. Before 2010, companies used to have on-premise databases. However, as the internet became widely available, it was becoming increasingly difficult to revamp resources to store billions of data. It becomes cumbersome to manage such data locally. Companies started adopting cloud services, primarily to host their data. Along with that, with Cloud Service Providers introducing Infrastructure as a Service (IAAS), Platform as a Service (PAAS) and (Software as a Service), attracted more customers to host all their services using the CSP. Cloud Computing is seeing a huge boost recently as it allows companies to better collaborate among the organization. It increases the accessibility and mobility of data. With reduced cost, companies can better manage their services and data. It provides higher availability and scalability [1].

However, such advantages come at a cost. As per [2] CSA identifies 12 threats which are faced by the companies. Data breach is the most encountered threat. Now when the data is stored on a third party CSP be it AWS, Azure or GCP there is always a certain hesitation with storing plaintext data on the cloud. However, even if the data is encrypted, we need the assurance from the CSP's that no adversaries who are a part of the CSP can snoop on the data. Applications need to share the decryption keys with CSP's in order to perform cloud computing. Hence, data security is and performing cloud computing without sharing key materials with the CSP is a really hot topic.

1. Data Breaches
2. Weak Identity, Credential and Access Management
3. Insecure APIs
4. System and Application Vulnerabilities
5. Account Hijacking
6. Malicious Insiders
7. Advanced Persistent Threats (APTs)
8. Data Loss
9. Insufficient Due Diligence
10. Abuse and Nefarious Use of Cloud Services
11. Denial of Service
12. Shared Technology Issues

Fig 1. Threats to cloud computing as per [2]

Not just the CSP, it is important that a company's private customer data should also be unavailable to the individuals within an organization. For example, in a banking environment, the Database Administrator should not be able to access the private account information of the customers or read data about balances of a particular user. Database Administrators in a hospital should not be able to fetch a patient's health data. However, in a bank, the DBA or any other developer might need to perform certain queries in order to get general idea about the data (e.g. finding the number of customers who are due to pay their monthly loan instalments). Medical DBA or medical staff might need to perform statistical analysis about the patient's data. In such cases, we need to create a boundary to determine the access of a certain data point and how to perform secured data

sharing. We cannot provide them with the decryption keys but they also cannot perform their queries without knowing the actual plaintext data. There is a need to find techniques which can allow such operations to take place wherein we can perform tasks on the encrypted data without actually sharing the keys.

II. HOMOMORPHIC ENCRYPTION

In order to ensure that the decryption keys of the cipher texts stored on cloud remain with the respective owner and still allow cloud computing tasks to be performed on the data, homomorphic encryption came into place. The term "privacy homomorphism" was first coined by the inventors of RSA public key cryptography system Ron Rivest, Adi Shamir and Leonard Adleman in 1978. Derived from ancient Greek "Homo" means same and "morph" means shape. Homomorphic encryption in cryptography means that anyone in possession of the cipher can perform operations on it (Cloud Service Providers performing computing tasks) without having to access the decryption key. Further in 1982, Goldwasser and Micali invented a homomorphic cryptosystem capable of performing operations on just a single bit of data. In 1999, Pascal Pallier further advanced the research and put forward a partial homomorphic encryption scheme which allows performing additive operations on cipher texts and also multiplying a plaintext number with a ciphertext without a decryption key and still getting the accurate results. In the following years, many homomorphic encryption schemes have been proposed which support different numbers and types of operations. Around a decade ago in 2009, a full homomorphic scheme was proposed in [3] by Craig Gentry.

An encryption scheme is called homomorphic only if it satisfies equation (1) for any operation denoted by '*':

$$E(m_1) * E(m_2) = E(m_1 * m_2) \quad (1)$$

Where m1 and m2 belong to any plain text message M. Both symmetric and asymmetric cryptography approaches can be utilized to develop homomorphic schemes. There are primarily four steps for any HE algorithm. Key Generation, Encryption, Decryption and Evaluation. The first three steps are similar to that of any traditional cryptosystem symmetric or asymmetric. Evaluation refers to the step where the operations are executed on the ciphertext and we get the ciphertext output exactly same as derived by performing a function on plaintexts and then encrypting the result.

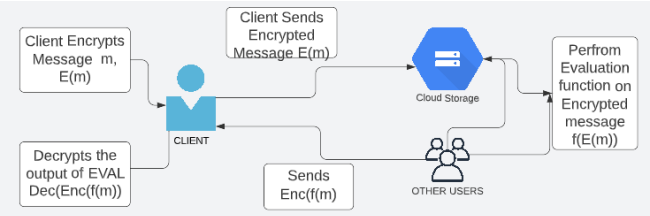

Fig 2. Homomorphic Encryption Scheme

The figure illustrates the steps for homomorphic encryption. The owner encrypts the message and stores it on a data server. Now the CSP or any other individual can execute an evaluation function without the decryption key. The client can then decrypt the resultant ciphertext without revealing the data to anyone else.

Based on the kind of arithmetic operations, symmetric or asymmetric cryptography and the number of operations allowed, the homomorphic encryption is classified into three types [4]:

A. Partial Homomorphic Encryption

PHE involves performing only one type of operation (additive or multiplicative). This can be done an infinite number of times.

Example:

1. RSA
   RSA which is a public key cryptosystem is one of the first schemes for applying PHE. The homomorphic encryption steps in RSA:

   a. Key Generation:
   We first find two large prime numbers such that n = p . q (where p and q are the prime numbers). ('.' denotes multiplication) We also calculate the value $\phi = (p-1) . (q-1)$ which helps to determine e and d such that $e . d = 1 \bmod(\phi)$. Here, e is the encryption key and d are the decryption key.

   b. Encryption using key e:
   Cipher text $c = m^e \bmod(n)$ where m is the plaintext message.

   c. Decryption using key d:
   The pair (d,n) are made public and sent to the intended recipient. One can retrieve the plaintext m by $m = c^d \bmod(n)$

   d. Evaluation:
   RSA algorithm satisfies the following homomorphic property for multiplicative operation:
   $E(m_1) . E(m_2) = E(m_1 . m_2)$ … from (1)
   (here * is replaced by . for multiplicative property)

Proof:
We can show that

$$E(m_1) . E(m_2) = m_1^e \bmod(n) . m_2^e \bmod(n)$$
$$= (m_1 . m_2)^e \bmod(n)$$
$$= E(m_1 . m_2)$$

As we can see in the above case RSA is homomorphic only for multiplication operation and not addition, hence, RSA is a PHE scheme. The multiplication operation can be performed infinitely many times.

2. El-Gamal:
   Based on Diffie-Hellman Key Exchange, this algorithm is mainly used to encrypt the keys of symmetric cryptography. It is also a PHE scheme as it allows only multiplication operations.
   a. Key Generation:
   The sender selects a large prime number p and generator g such that GCD(p,g) = 1. Further, it selects

a secret key d such that g<d<p. We calculate value e = g^d mod(p) and publish the public key (g,p,e)

b. Encryption:
To encrypt a message m which has to be less than p, we select a random number k such that k belongs to any Integer. Now we produce two cipher texts:
$c_1 = g^k$ mod(p)
$c_2 = m \cdot e^k$ mod(p)

c. Decryption:
The user with secret key d will compute the plaintext.
Plaintext = $(c_2 \cdot c_1^d)^{-1}$ mod(p)
$= m \cdot e^k$ mod(p) . g^{-kd} mod(p)
$= m \cdot g^{kd}$ mod (p) . g^{-kd} mod(p) = m

d. Evaluation:
The El-Gamal algorithm satisfies the homomorphic property [1] for the multiplicative operation.
$E(m_1) \cdot E(m_2) = E(m_1 \cdot m_2)$ … from (1)
(here * is replaced by . for multiplicative property)
Proof:
We can show that
$E(m_1) \cdot E(m_2) = (g^{k1}$ mod(p), $m_1 \cdot e^{k1}$ mod(p)) .
(g^{k2} mod(p), $m_2 \cdot e^{k2}$ mod(p))
$=(g^{k1+k2}$ mod(p), $(m_1 \cdot m_2) \cdot e^{k1+k2}$ mod(p))
$= E(m_1 \cdot m_2)$

3. Paillier:
Paillier is a public key cryptosystem which is based on the composite residuosity problem.

a. Key Generation:
First pick two large prime numbers p and q such that GCD(pq, (p-1).(q-1)) = 1. Then find n = LCM (p-1 , q- 1) LCM is Least Common Multiple. Select a random integer g from the set of Integers: {1 , … , p.q} such that GCD(p.q, L(g. $^{\text{mod(p.q)}}$) = 1, where L(x) = (x-1)/(p.q) is called the modular multiplicative inverse. Hence, we have the public keys (n,g) where n = p.q and (p,q) are the secret keys of the owner.

b. Encryption:
For a message m, calculate ciphertext as follows:
$c = g^m \cdot r^n$ mod(n^2) where n = p.q and r is random number picked from 1 to n.

c. Decryption:
For a ciphertext c, calculate the plaintext as follows:
Plaintext = m = (L (c mod(n^2) / L(g mod(n^2))) mod(n)
Where L(x) = (x-1)/(p.q)

d. Evaluation:
Pallier satisfies the homomorphic property for the additive operation.

Proof:

$E(m_1) \cdot E(m_2) = g^{m1} \cdot r_1^n$ mod(n^2) . $g^{m2} \cdot r_2^n$ mod(n^2)
$= g^{m1+m2} \cdot (r_1 \cdot r_2)^2$ mod(n^2)
$= E(m_1 + m_2)$

Pallier also allows multiplication of plaintext with the ciphertext.

4. Goldwasser – Micali:
This scheme proposes a probability public key encryption algorithm based on the quadratic residuosity problem.

a. Key Generation:
We first calculate a number n = p . q such that p and q are two large prime numbers. Then we find a number x such that x^2 = a mod(n). Here a is the quadratic residue modulo of n. The public key (x,n) is published and (p,q) is the secret key with the owner.

b. Encryption:
Convert message m to string of bits. Calculate y (quadratic non residue) such that GCD(y,n) = 1. Now encrypt each bit of message m:
$c_i = E(m_i) = y_i \cdot x^{mi}$ mod(n) …… m_i is the string of bits in 0 and 1.
Similarly find c_1, c_2, c_3 … for m_1, m_2 , m_3.
Here x is different for each encryption and belongs to the set of integers {1,....n-1} which are quadratic residues modulo n.

c. Decryption:
If c_i is a quadratic residue modulo n then m_i would return 0 or else 1.

d. Evaluation:
For each bit of the message m, this scheme satisfies the homomorphic property only for additive operations on plaintext binary numbers:
E(m1) . E(m2) = ($y_1^2 \cdot x^{m1}$ mod(n)) .($y_2^2 \cdot x^{m2}$ mod(n))
$= (y_1 \cdot y_2) \cdot x^{m1+m2}$ mod (n)
= E (m1 + m2)

The figure 3 provided from [5] shows all the PHE schemes till date .

PHE Scheme	Homomorphic Property and Equation		
	(+)	(×)	Equation
RSA		✓	$(m_1{}^e \bmod n) \times (m_2{}^e \bmod n) = (m_1 \times m_2)^e \bmod n$
GM	✓		$((x^{m_1} r_1{}^2) \bmod n) \times ((x^{m_2} r_2{}^2) \bmod n) = (x^{m_1+m_2}(r_1 r_2)^2) \bmod n$
ElGamal		✓	$(g^{r_1}, m_1 h^{r_1}) \times (g^{r_2}, m_2 h^{r_2}) = (g^{r_1+r_2}, m_1 \times m_2 h^{r_1+r_2})$
Benaloh	✓		$((g^{m_1} r_1{}^b) \bmod n) \times ((g^{m_2} r_2{}^b) \bmod n) = (g^{m_1+m_2}(r_1 \times r_2)^b) \bmod n$
NS	✓		$((g^{m_1}, r_1{}^\sigma) \bmod n) \times ((g^{m_2}, r_2{}^\sigma) \bmod n) = ((g^{m_1+m_2}), (r_1 r_2)^\sigma) \bmod n$
OU	✓		$((g^{m_1} h^{r_1}) \bmod n) \times ((g^{m_2} h^{r_2}) \bmod n) = (g^{m_1+m_2} h^{r_1+r_2}) \bmod n,$
Pailier	✓		$(g^{m_1} r_1^n \bmod n^2) \times (g^{m_2} r_2^n \bmod n^2) = (g^{(m_1+m_2)}(r_1 \times r_2)^n) \bmod n^2$
DJ	✓		$(g^{m_1} r_1^{n^s} \bmod n^{s+1}) \times (g^{m_2} r_2{}^{n^s} \bmod n^{s+1}) = (g^{(m_1+m_2)}(r_1 \times r_2)^{n^s}) \bmod n^{s+1}$

Fig 3. All PHE schemes

Comparison:
The paper [6] compares three of the widely used PHE algorithms based on several factors which include secrecy, encryption and decryption time, memory usage and

throughput. The algorithms include Paillier, El-Gamal and RSA.

Throughput in the paper is described as:
Throughput = Length(Encrypted text) / Total encryption time

More the throughput, more efficiently the particular algorithm is performing in encryption.
In table II, we can observe that the throughput decreases for Paillier as it has extensive encryption time.

TABLE I. Encryption Time shown in [6] for plaintext of same size

Method	Key size			
	256	*512*	*1024*	*2048*
RSA	1000	15623800	62495700	390597900
Paillier	15623700	78119400	453073000	3204100500
ElGamal	15617600	15627000	62497400	281250400

TABLE II. Encryption Throughput shown in [6]

Method	File size			
	29 bytes	*57 bytes*	*73 bytes*	*103 bytes*
RSA	5.42724597 28e-7	8.25882281 3e-7	9.34325852e -7	1.648163498 e-6
Paillier	6.38020290 17e-8	1.08364256 7e-7	1.37056591e -7	1.663313462 e-8
ElGamal	4.63993318 49e-7	9.12030644 2e-7	1.16807288e -6	1.929583322 e-6

Based on the experimental results [6] which were derived by performing encryption and decryption using the algorithms in simple form on short plaintexts, we find that:

Paillier being additively homomorphic provides semantic security but is considerably slow in encryption and decryption as compared to the other two. RSA is deterministic encryption and Paillier and El-Gamal are probabilistic. Hence, RSA is semantically unsecured. Paillier has the largest key size among all. RSA has a better throughput as compared to El-Gamal. El-Gamal being multiplicatively homomorphic is less efficient as the ciphertext produced is twice the size of the plaintext and hence it becomes more cumbersome with multiple multiplicative operations on the ciphertext. We can conclude that Paillier is more secure but loses in terms of efficiency especially on the additive operation.

Applications:
PHE has been adopted to provide confidentiality in Secured Electronic Voting Systems as proposed in [5]. Voters can vote for their candidate and the data is stored on cloud. The data is encrypted and stored as cipher text which provides confidentiality from the administrators. It utilizes the homomorphic property of Paillier to perform computations on the data store on cloud and count the votes. It claims to achieve 100% accurate results in determining the votes for each candidate. However, as discussed earlier there are concerns about the performance of the system with respect to the encryption and decryption time taken.

B. Somewhat Homomorphic Encryption

SWHE scheme involves performing both addition and multiplication operations on the cipher text data but only for a limited number of times as far as the system is able to decrypt the resultant cipher text after the evaluation step.

Example:

1. Boneh-Goh-Nissim:
   This scheme is based on subgroup decision problems. It is the first proposed Somewhat Homomorphic scheme which allows both addition and multiplication.

a. Key Generation:
   Select two distinct prime numbers p and q. Find n = p.q. Select g and u as generators from n. Set $h = u^q$. Publish (n,h,g).

b. Encryption:
   For a plaintext m, cipher text $c = g^m . h^r \bmod(n)$ where r is a random number from 1 to n-1.

c. Decryption:
   First find $c^* = c^p$ and $g^* = g^p$.
   Now decrypted plaintext $m = D(c) = \log_{g^*} (c^*)$.
   The drawback here is that the message space has to be small in order to decrypt quickly as the log function is expensive.

d. Evaluation:
   BGN satisfies homomorphic properties for both addition and multiplication. For plaintext m_1 and m_2 and corresponding ciphertexts c_1 and c_2.
   $c = c_1 . c_2 . h^r = (g^{m1} .h^{r1}) . (g^{m2} .h^{r2}) h^r = g^{m1+m2} h^{r*}$
   Where $r^* = r1+r2$ and so we can easily find the addition of the plaintexts m1+m2.
   Now,
   $c = e (c^1, c^2) h_1{}^r = e (g^{m1}. h^{r1} ,g^{m2} . h^{r2})h_1{}^r$
   $= g_1{}^{m1. m2} . h_1{}^{m1r2+r2m1 + \alpha q2r1r2+r} = g^{m1 . m2} . h_1{}^{r*}$
   Hence, we can see that the multiplicative property is also satisfied.

2. Other schemes:
   Yao's technique is based on a problem called the millionaire's problem which is determining who is rich among two people without actually knowing their wealth. However, it is computationally very expensive as compared to BGN. IP which is based on branching problems and SYY are also expensive as compared to BGN as the ciphertext keeps increasing with each operation performed.

Comparison
We can see the comparison of different SWHE schemes in the TABLE III. The size of the ciphertext keeps growing with every operation in all the algorithms except in BGN.

TABLE III. Comparision of SWHE schemes [5].

SWHE Scheme	Evaluation Size	Evaluation Circuit	Ciphertext Size
Yao	Arbitrary	Garbled circuit	Grows at least linearly
SYY	Polly-many AND & one OR/NOT	NC1 circuit	Grows exponentially
BGN	Unlimited add & 1 multiplication	2-DNF formulas	Constant
IP	Arbitrary	Branching programs	Doesn't depend on the size of function

Application:
SWHE schemes are rarely used as the homomorphic functions can be applied only for a limited number of times which is not useful for cloud computing or to perform queries on the database. There has been an attempt [7] to use SWHE scheme to store IOT sensor-based data with encryption on cloud. It decreases the encryption time and allows transmission of healthcare-based data securely.

C. *Fully Homomorphic Encryption*

FHE involves any number of evaluation operations which can be performed an infinite number of times. In 2009, Gentry published the first potential FHE scheme which produces the results of the evaluation in the ciphertext space, but requires more computation power.

Example:

1. Gentry:
   Based on lattice based cryptographic algorithms, Gentry was proposed in 2009 and is the first proposed fully FHE scheme. With homomorphic evaluation, the cipher texts generated after multiple operations include noise which makes it difficult to decrypt the final result and get an accurate plaintext. Hence, Gentry tries to solve this problem by including methods like squashing and bootstrapping. Squashing involves reducing the complexity of the decryption algorithm so as to make the ciphertext bootstrappable. Bootstrapping involves re-crypting the ciphertext to remove the noise and decrypt the corresponding plaintext.

2. Other FE schemes have also been developed over the course of the last decade. Van Dijk proposed a scheme based on solving the Approximate-Greatest Common Divisor (AGCD) problems. Oded Regev introduced a Learning with error(LWE) based FHE scheme as a part of learning from parity with error problem.

   Table IV. provides the comparison between three widely studied FHE schemes.

Table IV. Comparison of all FHE schemes

Metrics	Lattice Based	Integer Based	LWE ring bases
Security	High	low	Medium
Efficiency	Medium	Least	Highest
Simplicity	Difficult to implement	Easier to implement	Medium
Drawbacks	Large Key size	Less security	None
Problem Basis	Subset sum problem	Approximate GCD problem	Rings Learning with error problem

Applications:
Due to its ability to perform infinite number of operations, FHE has been adopted for privacy preserving healthcare data exchange applications [11]. It has also been utilized to perform Data Mining [12] and Data Analytics [13] on cloud data.

III. Drawbacks of Homomorphic Encryptions

Generally, the encryption schemes are judged upon three metrics. Security: All the above schemes provide reasonable security as an attacker cannot fetch any plaintext data from the homomorphic operations. Efficiency: With multiple operations HE performs slowly and it becomes difficult to convert the resultant ciphertext to plaintext. Simplicity: Some of the schemes can be difficult to implement due to the complexity of the algorithm, size of the keys, size of the cipher texts, computing power required, etc. and might not be feasible without enough knowledge.

IV. Alternatives

There has been constant research and development on finding better ways of data sharing and providing secured access to encrypted data to perform cloud computing without leaking the data content. Some of them are:

A. *Functional Encryption:*

Functional Encryption (FE) proposed in [8] provides an alternative to fully homomorphic encryption scheme. In FHE, our main goal is to obscure the data sent to a third party and perform operations on the encrypted data. However, in order to find the result of the operation the resultant ciphertext still needs to be sent to the owner of the decryption keys. Functional Encryption tries to provide a different approach where the owner of the data produces a master key sk_k. We can now execute a function F(k,x) on an encrypted data without knowing anything about x and still get the result using the key sk_k. (k is the key space and x is the plaintext data). Steps in functional encryption [9]:

a. Setup:
   Create a secret key (sk) and public key (pk).

b. Key Generation:
   Use sk and a function f to calculate the corresponding decryption key sk[f].

c. Encryption:
   Encrypt the data using pk. E(pk,x) where x is the plaintext data.

d. Decryption:
   Decryption using sk[f] of cipher text c produced by the function f on encrypted data c* = E(pk, x) produces f(x).

Here an attacker or a third person who has sk[f] will not be able to understand what the output f(x) means as it does not reveal anything about the original plaintext x. This allows secured data sharing without leaking data contents.

Application:
A user creates a secret key sk[f] for a function f which determines whether an encrypted mail using public key pk sent by a sender is spam or not. If f(x) = 1 then it is spam, else it is not spam. This function may be user defined. However, the key is sent to the CSP or the Mail Service Provider (MSP) which holds the encrypted mail and performs the function on the cipher text. Now without knowing the contents of the mail, the MSP will be able to know if a mail is spam or not.

Drawback:
It is required to perform research to create the public and secret keys for each function without using current tools which use Bilinear maps based on elliptic curves. Functional Encryption also does not provide comparable efficacy to homomorphic encryption as it varies depending on the cryptographic construction of the keys

B. CryptDB

CryptDB [10] is a state-of-the-art system for confidentiality of SQL-data by performing SQL query processing on encrypted data. It is an attempt to prevent any Database administrator from having access to the private information on the database. It also solves the issue of a compromised application and database server. The way it does this is by introducing a proxy server of CryptDB in between the application and the DBMS i.e., in our case it can be the Cloud Service Provider.

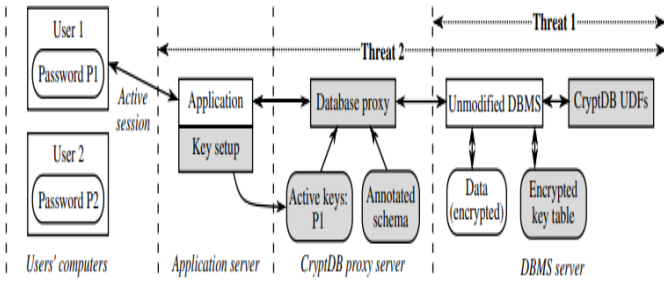

Fig. 4. Architecture of CryptDB from [10]

Threat 1: Conceal data from the Database Administrator

Database Administrators have access to the database and may be required to perform queries to fetch general information about the data. It is possible that an attacker gets access to the database through a DBA and can snoop on the private data or even a mischievous DBA can look into the private information in the database. Hence, CryptDB solves this problem by performing sql queries on encrypted data by using the CryptDB UDF (User-Defined Functions). It performs the same queries as that of the plaintext on the encrypted data like joins, aggregates, projections, etc. with some modified operators. The proxy server stores a Master Key MK, encryption information about the columns and the data schema. It takes the query from the application server and converts it into a query for the encrypted data using the master key and also encrypts the constants present in the normal query. It then uses UDF to change the encryption layer of the columns required for the query. DBMS server then performs the query and sends the decrypted result using the crypt proxy to the application. Each data column in the DBMS has an onion of encryption. Onions are a way to compactly store multiple ciphertext for the same plaintext. Based on the required operation we change the layer of encryption and fetch corresponding ciphertext for that column. Here the layer of encryption means the encryption scheme used. Depending on the type of query (join, search, aggregate), some encryption schemes are more efficient than others and such a process is called Adjustable query-based encryption. The keys for the encryption are derived from Master Key MK. It can be seen below.

Encryption schemes for SQL processing operations:

The following are the encryption schemes used to perform the query on the encrypted data.

a. RND - Random:
Utilizes Blowfish and AES in CBC mode. Such a scheme is useful to sort the data in a column. Typically, when there are no comparisons are required in a column.

b. DET - Deterministic:
Uses AES with CMC mode. DET is used for comparisons of data in a column eg. for columns that require equality checks.

c. OPE - Order Preserving Encryption:
This scheme is responsible for ORDER BY , MIN and MAX queries.

d. HOM - Homomorphic Encryption:
Utilizes Homomorphic properties of FHE and SHE for multiplying two ciphertexts. Useful for computing averages in a column.

e. Along with these JOIN and Word Search are another available scheme in CryptDB as seen in.

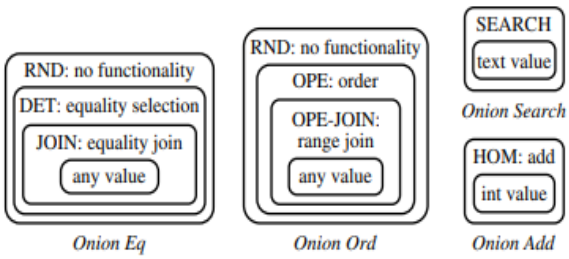

Fig 5. Onions of Encryptions in CryptDB [10]

Threat 2: In cases of compromise of application and database server

In case there is an attacker who has established access to the application and database server, he will only be able to access the data of logged in users. This is because CryptDB uses chain encryption keys. The data can be decrypted by the keys which are derived from the passwords of the users. Hence, if the attacker does not know the user passwords, then he won't be able to access the data of logged out users.

The drawbacks of this approach are the latency caused due to the CryptDB proxy introduced in between as we can see the throughput is decreased in the figure.

Application**:**
Such a SQL based system is very useful in a healthcare setup. It keeps the patient's data confidential from the DBA and also remains resistant against application and server attacks.

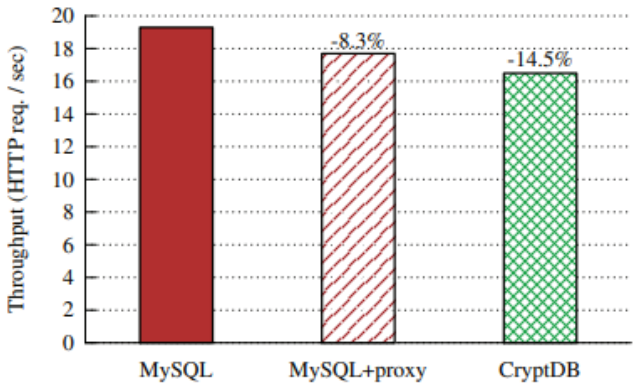

Fig 6. Throughput of Simple MySQL to compare with CryptDb from [10]

V. Conclusion and Future Works

In this paper we attempt to compare and review different homomorphic encryption schemes which have been in-demand lately due to their ability to solve data security issues. We see that Fully Homomorphic Encryption is computationally extensive. However, for smaller tasks Partial and Somewhat Homomorphic encryptions are useful. It is necessary to further research on FHE schemes to ensure that the resultant length of the ciphertext remains decryptable efficiently. We also saw few alternatives which are currently under development. Functional Encryption has a huge potential provided sufficient studies are conducted in order to make it more efficient and feasible. CryptDB is another alternative which seems to solve the issues of data security on SQL databases by providing a proxy to handle all the queries. It is necessary to utilize CryptDB approach on NoSQL data and more research work is required for that. However, more work needs to be done in order to ensure that the schemes which are adopted are also efficient.

References

[1] Tebaa M, Hajji SE. “Secure cloud computing through homomorphic encryptio”. arXiv preprint arXiv:1409.0829. 2014 Sep 2.

[2] Nalini Subramanian, Andrews Jeyaraj, “Recent security challenges in cloud computing”, Computers & Electrical Engineering,Volume 71,2018,Pages 28-42,ISSN 0045-7906,

[3] Craig Gentry. 2009. “Fully homomorphic encryption using ideal lattices”. In Proceedings of the forty-first annual ACM symposium on Theory of computing (STOC '09). Association for Computing Machinery, New York, NY, USA, 169–178. https://doi.org/10.1145/1536414.1536440

[4] Abbas Acar, Hidayet Aksu, A. Selcuk Uluagac, and Mauro Conti. 2018. “A Survey on Homomorphic Encryption Schemes: Theory and Implementation”. ACM Comput. Surv. 51, 4, Article 79 (July 2019), 35 pages.

[5] R. Awadallah and A. Samsudin, "Homomorphic Encryption for Cloud Computing and Its Challenges," *2020 IEEE 7th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2020, pp. 1-6, doi: 10.1109/ICETAS51660.2020.9484283.

[6] S. J. Mohammed and D. B. Taha, "Performance Evaluation of RSA, ElGamal, and Paillier Partial Homomorphic Encryption Algorithms," 2022 International Conference on Computer Science and Software Engineering (CSASE), 2022, pp. 89-94, doi: 10.1109/CSASE51777.2022.9759825.

[7] V.Subramaniyaswamy, V.Jagadeeswari, V.Indragandhi, Rutvij H. Jhaveri, V.Vijayakumar, Ketan Kotecha and Logesh Ravi, “Somewhat Homomorphic Encryption: Ring Learning with Error Algorithm for Faster Encryption of IoT Sensor Signal-Based Edge Devices”, 2022 February 24th, Security and Communication Networks, Hindawi.

[8] Boneh, D., Sahai, A., Waters, B. (2011). “Functional Encryption: Definitions and Challenges”. In: Ishai, Y. (eds) Theory of Cryptography. TCC 2011. Lecture Notes in Computer Science, vol 6597. Springer, Berlin, Heidelberg.

[9] Dan Boneh, Amit Sahai, and Brent Waters. 2012. Functional encryption: a new vision for public-key cryptography. Commun. ACM 55, 11 (November 2012), 56–64.

[10] Raluca Ada Popa, Catherine M. S. Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11). Association for Computing Machinery, New York, NY, USA, 85–100.

[11] Sendhil, R., Amuthan, A. Contextual fully homomorphic encryption schemes-based privacy preserving framework for securing fog-assisted healthcare data exchanging applications. *Int. j. inf. tecnol.* **13**, 1545–1553 (2021).

[12] Mohammed Golam Kaosar, Russell Paulet, Xun Yi,"Fully homomorphic encryption based two-party association rule mining", Data & Knowledge Engineering, Volumes 76–78, 2012, Pages 1-15, ISSN 0169-023X.

[13] Abdulatif Alabdulatif, Ibrahim Khalil, Xun Yi, "Towards secure big data analytic for cloud-enabled applications with fully homomorphic encryption", Journal of Parallel and Distributed Computing, Volume 137, 2020, Pages 192-204, ISSN 0743-7315,