SECURELY SEARCH ON ENCRYPTED DATA

Final Thesis submitted to the

Indian Statistical Institute, Kolkata

For award of the degree

Master of Technology in Cryptology & Security

by

Soumyadip Guria

(Roll No. CrS2216)

Under the guidance of

Primary Supervisor: **Captain Ritesh Wahi**

DDG(HOD), WESEE, Ministry of Defence

Co Supervisor: **Lt. Cdr. Keval Krishan**

SM, WESEE, Ministry of Defence

Secondary Supervisor: **Dr. Mriganka Mandal**

Assistant Professor

Cryptology & Security Research Unit (CSRU)

Indian Statistical Institute, Kolkata

to

CRYPTOGRAPHY & SECURITY RESEARCH UNIT

INDIAN STATISTICAL INSTITUTE

DECLARATION

I, **Soumyadip Guria (Roll No: CrS2216)**, hereby declare that, this report entitled **"Securely Search on Encrypted Data"** submitted to Indian Statistical Institute, Kolkata towards the fulfilment of the requirements for the degree of **Master of Technology** in **Cryptography & Security**, is an original work carried out by me under the supervision of **Captain Ritesh Wahi, Lt. Cdr. Keval Krishan and Dr. Mriganka Mandal** and has not formed the basis for the award of any degree or diploma, in this or any other institution or university. I have sincerely tried to uphold academic ethics and honesty. Whenever a piece of external information or statement or result is used then, that has been duly acknowledged and cited.

Soumyadip Guria 19/08/24

Kolkata - 700 108 **Soumyadip Guria**

August 2024

Certificate

This is to certify that the work contained in this project report entitled "**Securely Search on Encrypted Data**" submitted by **Soumyadip Guria** (Roll No. **CrS2216**) to the Indian Statistical Institute, Kolkata towards the fulfilment of the requirements for the degree of **Master of Technology** in **Cryptology & Security** has been carried out by him under my supervision and that it has not been submitted elsewhere for the award of any degree.

Captain Ritesh Wahi

Primary Supervisor

DDG(HOD), WESEE, Ministry of Defence

Mriganka Mandal.

Dr. Mriganka Mandal

Internal Supervisor

CRSU, ISI Kolkata

ACKNOWLEDGEMENT

I thank everyone who has assisted me in seeing this project through to its completion. I would like to first express my profound gratitude and deepest regards to my Guides, WESEE and ISI Kolkata, and sincerely wish to acknowledge their vision, guidance, valuable feedback and constant support throughout this project.

I am indebted to my friends for their steadfast encouragement and time. I am lastly grateful to the Indian Statistical Institute, Kolkata for providing the necessary resources and facilities to complete this project to the best of my ability.

WESEE, New Delhi - 110066 **Soumyadip Guria**

August 2024

ABSTRACT

In our usual life, data is all around us, everywhere we look! Now a days every individuals and organizations continue to generate, store, and share vast amounts of data and as a result the role of cloud storage becomes important to us. In the context of cloud computing, where vast amounts of data are stored and processed remotely, ensuring the privacy and security of sensitive information becomes even more challenging. Traditional approaches to data storage and retrieval often require decrypting the data on the cloud server, which introduces the security concerns of the data. One promising solution to address this concern is to enable secure search operations on data that is stored in the cloud in an encrypted form. Implementing a searchable encryption scheme using Homomorphic Encryption (HE) allow users to perform searches on encrypted data without compromising its confidentiality. By developing such a scheme, this thesis aims to contribute to the advancement of privacy-preserving data management in cloud environments.
With HE, complex mathematical operations can be conducted on ciphertexts, preserving the privacy of the underlying data throughout the computation. This opens up a path for searching and retrieving data from an encrypted dataset, as it enables to operate directly on encrypted inputs and produce encrypted outputs, thereby ensuring end-to-end data confidentiality. However, the adoption of HE on encrypted data is not without its challenges. The computational overhead associated with HE operations, coupled with performance limitations, poses significant obstacles to scalability and efficiency.
In light of these considerations, the aim of this thesis is to build a practical scheme using Homomorphic Encryption (HE) for encrypted searching. By investigating the applications, challenges, and potential contributions of HE in secure search

operations, this research endeavors to advance our understanding of privacy-preserving data management and pave the way for practical implementations in real-world scenarios.

Keywords:

Fully Homomorphic Encryption(FHE), Order Preserving Encryption(OPE), Fully Homomorphic Order Preserving Encryption(FHOPE)

Contents

List of Figures

List of Tables

Notations and Abbreviations

RND: Randomized

DET: Deterministic

$\mathbb{Z}_q$: Set of all integers modulo q

$\mathbb{Z}$: Set of all integers

$\mathbb{Z}[X]$: Polynomial ring over $\mathbb{Z}$

$\xleftarrow{R}$: Randomly chosen

Chapter 1

Introduction

As technology is evolving rapidly, the volume of data stored, generated and transmitted is increasing exponentially day by day. As the amount of data grows the concern about the security of data also increases. As a result, cryptography is also gaining more attention day by day.

1.1 Conventional Cryptography

There are mainly two types of encryption schemes in cryptography. One is *Symmetric* encryption and another one is *Asymmetric* encryption.

1. Symmetric key cryptography uses only one key for encryption and decryption. Here the key is shared beforehand between sender and receiver in some way and no person other than the sender and receiver knows the key. A

symmetric encryption scheme is a three-tuple of algorithms (`KeyGen`, `Enc`, `Dec`) defined over a three-tuple of spaces $(\mathcal{P}, \mathcal{C}, \mathcal{K})$, where

- $\mathcal{P}$: Plaintext Space.
- $\mathcal{C}$: Ciphertext Space.
- $\mathcal{K}$: Key Space.
- `KeyGen`: The Key-Generation Algorithm, a probabilistic algorithm that outputs a key k from $\mathcal{K}$ chosen according to some distribution.
- `Enc`: The Encryption Algorithm, $\texttt{Enc} : \mathcal{K} \times \mathcal{P} \rightarrow \mathcal{C} \quad \texttt{Enc}(k, m) = c$ $where \quad k \in \mathcal{K}, \quad m \in \mathcal{P}, \quad c \in \mathcal{C}.$
- `Dec`: The Decryption Algorithm, $\texttt{Dec} : \mathcal{K} \times \mathcal{C} \rightarrow \mathcal{P} \quad \texttt{Dec}(k, c) = m$ $where \quad k \in \mathcal{K}, \quad m \in \mathcal{P}, \quad c \in \mathcal{C}$

2. Asymmetric key cryptography uses different keys for encryption and decryption. Each user has a public key and private key pair called (pk, sk). Anybody can encrypt a message using public key(pk) of the receiver as $c \longleftarrow \texttt{Enc}(pk, m)$. Only receiver can decrypt it using its own private key(sk) as $m \longleftarrow \texttt{Dec}(sk, c)$.

1.2 Cloud Computing

Nowadays *cloud computing* started to gain an enormous amount of attention. Since cloud providers give nearly unlimited storage and computational power at very low cost, today most of the companies, hospitals and individual users outsource their data to the cloud providers. But there might be a huge amount of sensitive data

like medical records of patients, credit card details etc becomes available to everyone. Since our assumption is that the cloud service providers are honest but curious[8], we need to encrypt the sensitive data before uploading the data to the cloud server. But if encrypted data are uploaded to the cloud server many problems arise. Suppose the data owners want some of their data from a cloud database, then the cloud has to decrypt all the data to give a reply. But this also concerns security and a tremendous amount of computation cost because every time it has to decrypt the whole dataset. One solution might be executing SQL queries over the encrypted data in the cloud. However, encrypted data are not easy to handle. When *FHE* was first introduced it opened up many paths in this field because FHE allows to do computation over encrypted data.
Till date, many FHE schemes and OPE schemes have been proposed. However, FHE schemes are impractical for both cloud environments and end-parties because of their significant computational burden[14]. The OPE schemes disclose orders of the plaintext and some other private informations near to the cloud providers that allow SQL range queries.

1.3 Problem Statement

Given a large encrypted dataset uploaded in a cloud server, design a scheme for performing SQL queries over that encrypted dataset.

1.4 Organization of Dissertation

- **Chapter 2** describes about homomorphic encryption, types of homomorphic encryption and its applications.
- **Chapter 3** describes about order preserving encryption and security of order preserving encryptions.
- **Chapter4** describes some related work about the query over encrypted dataset.
- **Chapter 5** describes about fully homomorphic order preserving encryption and security of it.
- **Chapter 6** describes about the details of this thesis and its implementation.
- **Chapter 7** describes some future prospective of this area.

Chapter 2

Homomorphic Encryption

Homomorphic encryption is one special type of encryption which allows the computations to be performed on encrypted data without needing to decrypt it first. This concept is often regarded as the Holy Grail in the field of cryptography. Suppose we have a homomorphic cryptosystem and using that we have encrypted two integers. According to Fig 2.1, 10 and 15 are encrypted. Now when these two ciphertexts are added to give a new ciphertext, the resultant ciphertext when decrypted will be equal to the sum of plaintext integers(in this case 25).

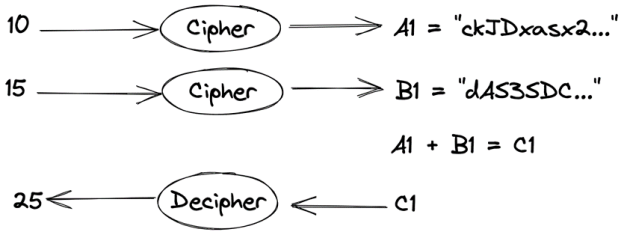

Figure 2.1: Concept of Homomorphic encryption.

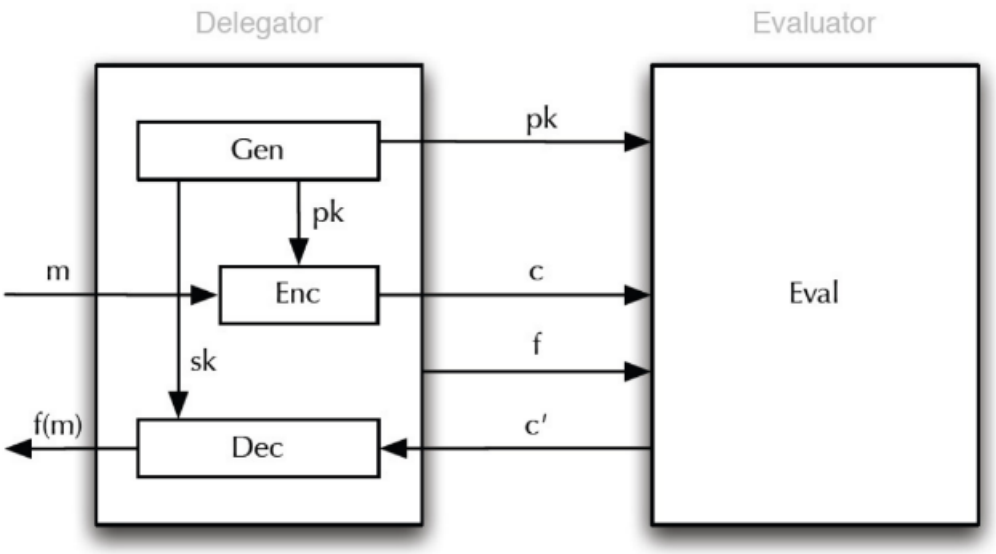

Figure 2.2: Homomorphic Encryption with Asymmetric keys.

2.1 Types of Homomorphic Encryption

There are various kinds of homomorphic encryption based on their properties.

- *Somewhat Homomorphic Encryption(SHE)*: Some homomorphic encryption schemes support a limited number of additions and multiplications on ciphertexts. These type of homomorphic encryption schemes are called Somewhat Homomorphic Encryption schemes.

- *Partially Homomorphic Encryption(PHE)*: The homomorphic encryption schemes that allows only a single type of operation to be computed over encrypted data, typically either addition or multiplication(eg. RSA is partially homomorphic that allows to do multiplication on encrypted data) are called Partially Homomorphic Encryption schemes. However PHE

schemes have inherent limitations.. If a PHE scheme supports only addition and subtraction, it can compute only linear functions over its input, as a result this is not useful for general purposes.

- *Fully Homomorphic Encryption(FHE)*: HE schemes that supports both ring addition and multiplication over encrypted data are called FHE.
- *Levelled FHE*: Some FHE schemes support unlimited number of additions but only limited number(this is predefined) of multiplication. These type of FHE schemes are known as levelled FHE schemes.
- Unqualified FHE: FHE schemes that support any number of addition and any number of multiplication on encrypted data are called unqualified FHE schemes.
- *Multi-hop Homomorphic Encryption*: In some cases, the result of the `Eval` algorithm can be used as an input for another homomorphic computation. These type of homomorphic encryption schemes are called multi-hop homomorphic encryption scheme.

In 2009, Gentry[6] introduced the first practical schemes for Unlimited FHE by employing a technique known as *bootstrapping*. The essential concept of bootstrapping is to execute the decryption algorithm on a ciphertext within a leveled FHE framework. This effectively "refreshes" the ciphertexts which means the process removes the restrictions on the computations that can be carried out on encrypted data. However, bootstrapping is quite resource-intensive, as it requires converting each homomorphic operation into a substantial bootstrapping task.

2.2 Hard Problems

There are lots of mathematical NP-hard problems on which the security of FHE schemes lies. In this section, some hard problems will be discussed.

- **Approximate GCD**: Approximate Greatest Common Divisor(GCD) remains a complex task among the classes of symbolic-numeric algorithms. Different methods try to provide the closest approximate solution and the subsequent improvement on the degree of precision and accuracy of this solution. This is used to find the GCD of the numbers a and b which are the approximation of two given numbers a_0 and b_0.

- **Algebraic Number Theory(ANT)**: ANT is the field known to specialize in the area of algebraic structures of Integers. Suppose there is a ring $\mathcal{O}$ of algebraic integers within an algebraic number field. In this branch we looks into diverse features of this ring, such as behaviour of its ideals, factorization, field extensions etc.

- **Sparse Subset Sum Problem**: This is a very important problem in the context of *theory of complexity* and *cryptography*. The problem involves identifying a non-empty subset from a given set of integers that sums to zero. For example, in the set {-5, -4, -2, 5, 1}, there exists a subset {-5, 5} whose sum is equal to zero. So in this case the answer is yes. This problem is classified as NP-complete.

- **Learning with Error (LWE)**: Regev introduced the Learning with Errors(LWE)[12] problem and showed that, on average, it is as challenging as

solving multiple lattice problems that are standardized, in the worst-case scenario, even with quantum capabilities. LWE forms the basis of several cryptographic constructions. It can be viewed as a noisy generalization of the hidden lattice problem.

Definition 2.1 (Learning With Errors)**.** Let $\mathbf{s} \in \mathbb{Z}_q^n$ be a secret vector and let $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ be a uniformly random matrix. For integers $q = q(n) \geq 2$ and a noise distribution $\chi = \chi(n)$ over $\mathbb{Z}_q$, the LWE problem is to distinguish between the following two distributions:

$$\{\mathbf{A}, \mathbf{A}^T\mathbf{s} + \mathbf{e}\} \text{ and } \{\mathbf{A}, \mathbf{u}\},$$

where $m = \texttt{poly}(n), \mathbf{e} \leftarrow \chi^m$, and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$. We refer to the m columns of the matrix $\mathbf{A}$ as the LWE sample points.

Chapter 3

Order Preserving Encryption

Order-preserving encryption (OPE) is a deterministic method of encrypting data where the encryption process maintains the original numerical order of the plaintext values. A more formal exploration of the concept of order-preserving symmetric encryption (OPE) was introduced in the database community by Agrawal et al [1]. This kind of encryption scheme gains more limelight because they allow efficient range queries on encrypted data.

3.1 Definitions

Definition 3.1 (Order Preserving Function)**.** Let f be a function from $\mathcal{X}$ to $\mathcal{Y}$ where $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{N}$ with $|\mathbf{X}| \leq |\mathcal{Y}|$. The function f is said to be order preserving or strictly increasing if $f(i) > f(j) \iff i > j$ for all $i, j \in \mathcal{X}$.

Definition 3.2. Let $\varepsilon = (\texttt{KeyGen}, \texttt{Enc}, \texttt{Dec})$ be a symmetric encryption scheme with plaintext and ciphertext-spaces $\mathcal{M}, \mathcal{C}$. We say that the scheme is an order preserving encryption scheme if $\texttt{Enc}(k, \cdot)$ is an order-preserving function from $\mathcal{M}$ to $\mathcal{C}$ for all $k \in \mathcal{K}$ where $\mathcal{K}$ is the key space.

3.2 Notion of security for OPE

Since OPE schemes are deterministic and it also leaks the order-relations among the plaintexts, it cannot satisfy all the standard notions of security, such as indistinguishability against chosen-plaintext attack (IND-CPA). So the concept of IND-CPA security is weakened a little bit for this kind of encryption schemes.
Weaking IND-CPA security: In the context of deterministic symmetric encryption, a new concept known as indistinguishability under distinct chosen-plaintext attack (IND-DCPA) has been introduced, as formalized by[2]. The core idea is that because deterministic encryption schemes reveal equalit of plaintext, the adversary in the IND-CPA experiment must make only distinct queries. Specifically, if the adversary makes queries $(m_0^1, m_1^1), ..., (m_0^q, m_1^q)$, it must ensure that $m_b^1, m_b^2, ..., m_b^q$ are all distinct for $b \in \{0, 1\}$.

Similarly, any Order-Preserving Encryption (OPE) scheme inherently leaks the order relations among the plaintexts. To address this, the approach should be generalized. Furthermore, the queries made by the adversary must satisfy $m_0^i < m_0^j \iff m_1^i < m_1^j$ for all $1 \leq i \leq q$. An adversary meeting these criteria is termed an IND-OCPA adversary, reflecting indistinguishability under ordered chosen-plaintext attack. However, IND-OCPA is not particularly useful since an

OPE scheme can only achieve IND-OCPA if its ciphertext space is extraordinarily large, on the order of being exponential in the size of the plaintext space.

Chapter 4

Literature Survey

Recently there has been considerable research focus on ensuring the data security and processing them in encrypted format in cloud database environment. Many cryptographic techniques have been developed like FHE, multi party computation(MPC) etc. When Gentry proposed unlimited FHE schemes[6], it opens up many paths since the FHE supports computations over encrypted data. After that many improvements[15, 4, 13, 5] have been made to boost the performance. But, due to the low efficiency of current Fully Homomorphic Encryption (FHE) schemes, they are not practical for real-world applications. Various solutions have been suggested for performing queries on encrypted database [7, 10]. H. Hacigumus et al. proposed one of the most fundamental solution for performing queries on a database that is stored in encrypted format in [7]. It does the encryption of data at tuple level, and then used a set of attributes(predefined) in queries. But the first practically possible system for processing queries over the encrypted database is `CryptDB` [11].

4.1 CryptDB

In this system, there are three components namely user, proxy server and the server where the encrypted data is stored. The basic protocol of cryptdb is when a user does a query, the proxy server intercepts it. Then the proxy server rewrites the query and send the query to the server. The server executes the modified query and sends the encrypted result to the proxy server. Proxy decrypts it and sends the result to user. The workflow is also described in the Figure 4.1. The DBMS server never receives decryption keys to the plaintext so it never sees sensitive data, so a curious database adversary cannot gain access to private information.

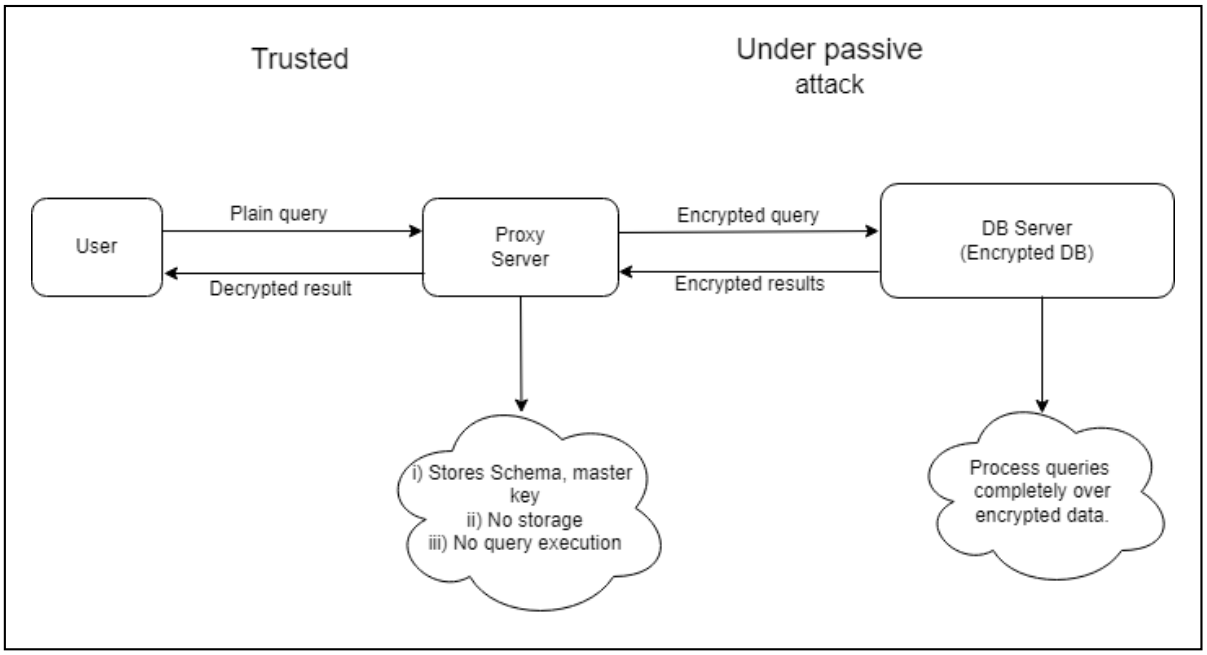

Figure 4.1: CryptDB setup.

4.1.1 How it actually works?

CryptDB divides the SQL queries in different types eg. equality query, range query, search query etc and it uses different onion layer encryptions for queries on encrypted dataset One more thing is it uploads the whole encrypted dataset for each type of query.

Encryption adjustment: It uses different layers of encryption for encrypting the whole dataset. For equality and range queries it uses one deterministic encryption scheme as first layer and a randomized encryption scheme as its second layer as described in Figure 4.2. Whenever user does a query, proxy intercepts it and does the following:

- Strip off layers of onions(Proxy gives keys to server using a SQL user defined function and Proxy remembers onion layer for columns.)
- Do not put back onion layers.

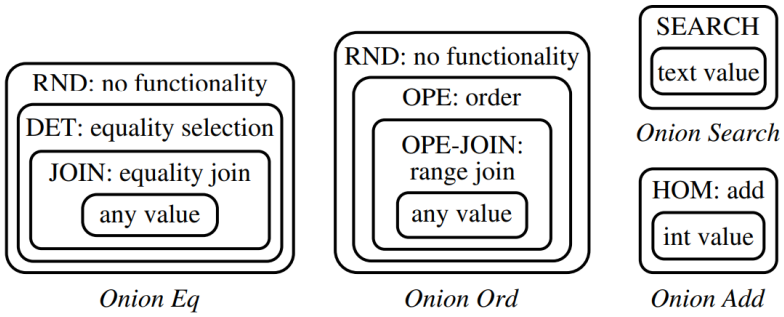

Figure 4.2: Onion encryption layers and the classes of computation they allow.

Example: Suppose we have a dataset 4.1 containing 3 columns 'Rank', 'Name' and 'Salary'.

Rank	Name	Salary
CEO	...	...
Worker	...	...

Table 4.1: Dataset

Now user does a query SELECT * FROM emp WHERE Rank = 'CEO'.
Now at the cloud's end the dataset is uploaded in encrypted format. CryptDB encypts the dataset for each onion encryption. So the dataset at cloud is shown below:

Col1_OnionEq	Col2_OnionEq	Col3_OnionEq	Col1_OnionOrder	...
RND	RND	Search	RND	...
RND	RND	Search	RND	...

Table 4.2: Encrypted dataset(not completed)

where Col1, Col2, Col3 are the mapping of Rank, Name and Salary respectively. OnionEq means the corresponding query is equlity query and similar for others. Now Proxy intercepts the query and watches that it is an equality query. So Proxy sends the following query to cloud: UPDATE table1 SET Col1_OnionEq = DECRYPT_RND(key, Col1_OnionEq). Cloud runs the query. So this query removes the randomized layer of the column1 for equality query. As a result encryption of that particular column becomes deterministic. So the dataset becomes

Col1_OnionEq	Col2_OnionEq	Col3_OnionEq	Col1_OnionOrder	...
DET	RND	Search	RND	...
DET	RND	Search	RND	...

Table 4.3: Encrypted dataset(not completed)

Now Proxy send another query: SELECT * FROM table1 WHERE Col1 OnionEq = Enc(CEO). Cloud runs the query and send encrypted result to proxy. Proxy decrypts this and sends it to the application.

4.1.2 Problems

The scheme has the following limitations:

- It uses deterministic encryption scheme as one of its onion layer for equality query and range query. But we know that deterministic encryption schemes are not Chosen Plaintext Attack (CPA) Secure.
- CryptDB does not support queries that contains multiplication (eg. CryptDB can't be able to execute the query SELECT * FROM table WHERE att_i = 100 AND att_i + att_j * att_k > att_l).
- It encrypts the whole dataset for every onion classes. For example, for range queries it encrypts the whole dataset with OPE and for aggregate queries, it encrypts the dataset with homomorphic encryption. This increases the overall computation cost.

where the PRF key s is held by the server and the set of

Chapter 5

Fully Homomorphic Order Preserving Encryption Scheme

5.1 FHOPE

Definition 5.1 (FHOPE)**.** A FHOPE scheme consists of four algorithms (`KeyGen`, `Enc`, `Dec`, `Comp`). They are described in details:

- `KeyGen`: $sk \longleftarrow$ `KeyGen`(1^{λ}). `KeyGen` takes security parameter λ as input and outputs a private key sk.
- `Enc`: $c \longleftarrow$ `Enc`(sk, v). `Enc` takes the secret key sk and sensitive data v as input and outputs the ciphertext c.
- `Dec`: $v \longleftarrow$ `Dec`(sk, c). `Dec` takes the secret key sk and ciphertext c as input and outputs the original data v.

- `Comp`: res $\longleftarrow$ `HAMOE`$(c_1, c_2, ..., c_n)$. `HAMOE` takes ciphertexts $c_1, c_2, ..., c_n$ as input and can perform addition, multiplication, order comparison, and equality checks over the ciphertext and then output the result of the computation.

5.2 Security of FHOPE

Definition 5.2. We say a FHOPE scheme is IND-HOCPA secure, if for any PPT adversary $\mathcal{A}$ has only a negligible advantage $\text{Adv}_{FHOPE,\mathcal{A}}^{IND-HOCPA}$ to win in the below game.

1. Suppose λ is the security parameter, the challenger $\mathcal{CH}$ runs `KeyGen` and generates the key as, sk $\longleftarrow$ `KeyGen`(1^λ).
2. The challenger $\mathcal{CH}$ and the adversary $\mathcal{A}$ interacts with each other for polynomial number of rounds. In round i,
   - the adversary $\mathcal{A}$ takes two messages $v_i^0, v_i^1 \in \mathcal{P}$ randomly where $|v_i^0| = |v_i^1|$ where $|v_i^b|$ represents the length of the message v_i^b and sends those two messages to the challenger $\mathcal{CH}$.
   - the challenger $\mathcal{CH}$ randomly picks $b \in \{0, 1\}$ and encrypts v_i^b using the secret key sk and sends all the ciphertexts to the adversary $\mathcal{A}$.
3. The adversary $\mathcal{A}$ outputs b'.

The adversary $\mathcal{A}$ wins the game if:

(i) it does correct guess(ie, b' = b) and

(ii) the sequences $\{p_i^0\}_i$ and $\{p_i^1\}_i$ have the same order relations(ie, $\forall$ i, j, $p_i^0 < p_j^0 \iff p_i^1 < p_j^1$). That is, $\mathcal{A}$ wins the above game if $\mathrm{Adv}_{FHOPE,\mathcal{A}}^{IND-HOCPA}$(k) is nonnegligible, where the adversary's advantage $\mathrm{Adv}_{FHOPE,\mathcal{A}}^{IND-HOCPA}$(k) is defined as:

$$\mathrm{Adv}_{FHOPE,\mathcal{A}}^{IND-HOCPA}(\mathrm{k}) = \left|\tfrac{1}{2} - Pr[win^(\mathcal{A}, k)]\right|$$

where $win^(\mathcal{A}, k)$ is a random variable which indicates the success of the adversary in the above game.

Chapter 6

Our Work

Our protocol uses a FHOPE scheme to process the encrypted data and uses hashes and polynomials to process the equality query. This is implemented over MySQL database server. There are three parties in this protocol. The parties are:

1. Client: Those who are interested in retrieving some data from the cloud.
2. Proxy Server: This is a trusted server. Proxy stores the secret key for encryption and decryption. Proxy also maintains a table where it stores hash value of all possible "attribute = value" pairs present in the dataset and a polynomial whose roots are the indices where that particular value is present in that attribute. Also it stores a mapping of table name and column names.
3. MySQL database server: It stores and manages the encrypted data.

The workflow of this protocol is similar to CryptDB that is mentioned in the section 4.1. The Workflow described in Figure 6.1 is as follows:

- Client does an SQL query for retrieving some data from the server.
- Whenever client does a query, proxy first intercepts the query. Then proxy checks the type of the query.
- After checking the query it modifies the query accordingly and sends the modified query to MySQL server.
- MySQL server runs the modified query over the encrypted dataset.
- Then the server sends the encrypted result to proxy.
- Proxy decrypts the result and send the decrypted result to client.

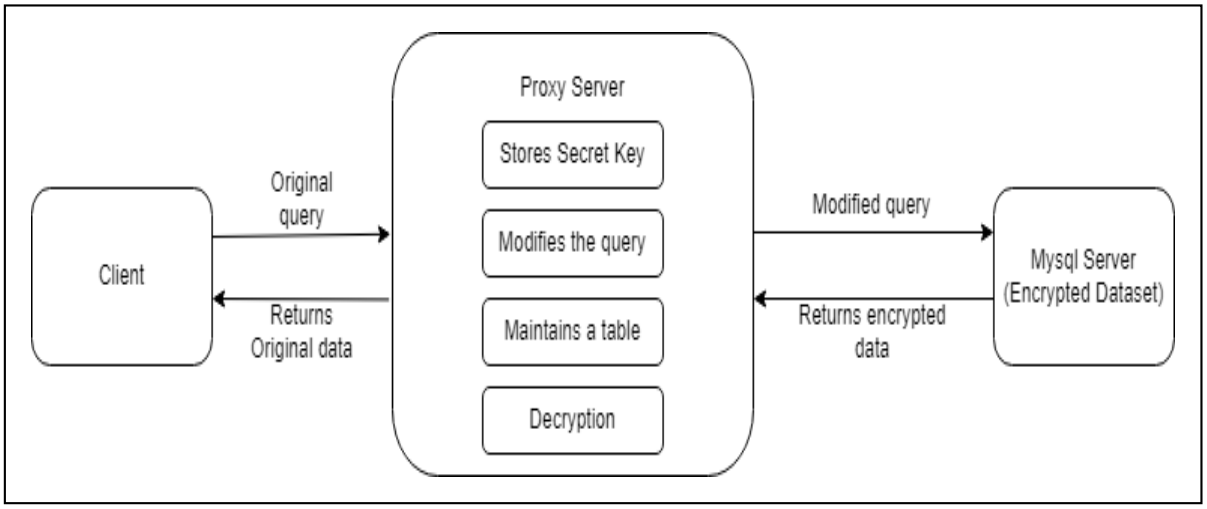

Figure 6.1: Setup.

6.1 Encryption Scheme Details

Here one Fully Homomorphic Order Preserving Encryption Scheme [9] is used to encrypt the dataset.

The encryption scheme is described below:

- `KeyGen`: The `KeyGen` algorithm generates the secret key as:

$$\begin{aligned} \mathrm{K(n)} &= [\vec{k_1}, \vec{k_2}, \ldots, \vec{k_n}] \\ &= [(u_1, v_1), (u_2, v_2), \ldots, (u_n, v_n)] \end{aligned}$$

  where $(u_i, v_i)(1 \leq i \leq n)$ is a list of pair of large prime numbers, $u_i * v_i > 0$, $n > 1$, $u_i \neq 0$ for all i, $u_2 = -u_1$, $v_1 + v_2 + ... + v_{n-1} \neq 0$ and $v_n \neq 0$.

- `Encryption`: Encrypt the plaintext FHOPE(K(n), p) = $(c_1, c_2, ..., c_n)$, where p is a plaintext; the ciphertext is a n-tuple, Each c_i is defined as follows:

$$c_i = Enc_i(\mathrm{K(n)}, p) + Noise_i(\mathrm{K(n)}, R) + \xi$$

  where,

$$Enc_i(\mathrm{K(n)}, p) = u_i * v_i * p$$

$$\mathrm{Noise}_i(\mathrm{K(n)}, R) = \begin{cases} \dfrac{u_1 p_1}{u_2} - \dfrac{u_1 r_n}{u_n} + r_1 - p_n & \text{if } i = 1 \\ \dfrac{u_i p_i}{u_{i+1}} - \dfrac{u_i r_{i-1}}{u_n} + r_i - p_{i-1} & \text{if } 2 \leq i \leq n-1 \\ \dfrac{u_n p_n}{u_1} - \dfrac{u_n r_{n-1}}{u_{n-1}} + r_n - p_{n-1} & \text{if } i = n \end{cases}$$

where, R = $\{(r_1, p_1), ..., (r_n, p_n)\}$ is defined in a finite integer domain. The noise function should satisfy the below condition:

$$0 < Noise_i(\mathrm{K(n)}, R) < (Enc_i(\mathrm{K(n)}, p + \mathcal{S}) - (Enc_i(\mathrm{K(n)}, p))$$

where $\mathcal{S}$ is the sensitivity of the dataset defined below:

Definition 6.1 (Sensitivity)**.** Let $\mathcal{P} = \{p_1, p_2, ..., p_n\}$ be the set of all input plaintext values. The sensitivity of $\mathcal{P}$ is the minimum element in the set $\{|x - y| : x \in \mathcal{P}, y \in \mathcal{P}, x \neq y\}$. That is the sensitivity $\mathcal{S}$ is defined as

$$\mathcal{S} = \min_{\substack{x,y \in \mathcal{P} \\ x \neq y}} |x - y|.$$

and ξ denotes random noise. ξ is randomly sampled from $(-\infty, +\infty)$, ξ is added to only the first two subciphertexts and for the others, value of ξ is 0.

- `Decryption:` Decrypt a ciphertext $C = (c_1, ..., c_n)$, and get the plaintext p.

$$\mathtt{Dec}(\mathrm{K(n)}, (c_1, c_2, ..., c_n)) = p$$

where K(n) = $[(u_1, v_1), (u_2, v_2), ..., (u_n, v_n)]$ is a secret key and p is a plaintext. The decryption algorithm is:

$$\sum_{i=1}^{n} Dec_i(u_i, v_i) * c_i = p$$

where Dec_i function is defined as:

$$Dec_i(u_i, v_i) = \frac{1}{u_i * \sum_{i=1}^{n} v_i}$$

The above mentioned FHOPE scheme is based on the hardness of Approximate GCD problem mentioned is section 2.2. This scheme supports additive and multiplicative homomorphism, order comparisons (for all $p_1, p_2 \in \mathcal{P}$, if $p_1 > p_2$, then $C_1 > C_2$ and for any two ciphertexts C_1, C_2, $C_1 > C_2$ means $c_{1i} > c_{2i}$, where c_{1i} and c_{2i} are the i-th subciphertexts of C_1 and C_2 respectively).

Some Results about the security of this encryption Scheme:

- Recovering $\vec{k_i}$ from a key K(n) is challenging, even when an unlimited number of ciphertexts encrypted using FHOPE with K(n) are available.
- This scheme satisfies IND-HOCPA security.

6.2 Using FHOPE to Database Server

Setup: Using this FHOPE scheme the scheme requires the following setup:

i. The client.

ii. Proxy server.

iii. Database server.

Insert Query: To understand how INSERT queries work suppose the client wants to execute the query INSERT INTO table_name (column1, column2, column3, ...) VALUES (value1, value2, value3, ...); Proxy intercepts the query and changes the column names according to the mapping present near to it and encrypts each value

present in the query and send the query INSERT INTO (NewColumn1, NewColumn2, NewColumn3, ...) VALUES (FHOPE(K(n), value1), FHOPE(K(n), value2), FHOPE(K(n), value3), ...) to the MySQL server. At the same time, the proxy calculates the hash of each new "attribute = value" expression and if the hash value is already present in the table it updates the polynomial and if the hash value is not present it adds a new one degree polynomial in the table. The MySQL server executes the encrypted query and adds the new encrypted data to the database.

Select Query: When the client does the query SELECT * FROM table_name, proxy just change the table name using the mapping gets all the encrypted data from server decrypts it and send to the client.

Sum Query: If the client issues the query SELECT * FROM Table1 WHERE col1 + col2 = col3, the proxy simply changes the column names and sends the query to MySQL server. MySQL runs the query over the ciphertexts using some User Defined Functions(UDFs) and returns the encrypted result to proxy, proxy decrypts it and send the result to client. Similar things happen when client does a query to find the sum of elements of some column.
Let C_1 and C_2 be two ciphertexts such that $C_1 = (c_{11}, c_{12}, ..., c_{1n})$ and $C_2 = (c_{21}, c_{22}, ..., c_{2n})$ respectively. The sum of two ciphertexts means sum of each element coordinatewise.

$$C_1 + C_2 = (c_{11}, c_{12}, ..., c_{1n}) + (c_{21}, c_{22}, ..., c_{2n}) = (c_{11} + c_{21}, c_{12} + c_{22}, ..., c_{1n} + c_{2n})$$

Multiplication Query: If the client issues the query SELECT * FROM Table1 WHERE $col1 \times col2 = col3$, the proxy simply changes the column names and sends

the query to MySQL server. MySQL runs the query over the ciphertexts using some UDFs and returns the encrypted result to proxy, proxy decrypts it and send the result to client. Let C_1 and C_2 be two ciphertexts such that $C_1 = (c_{11}, c_{12}, ..., c_{1n})$ and $C_2 = (c_{21}, c_{22}, ..., c_{2n})$ respectively. The multiplication of two ciphertexts means:

$$C_1 \times C_2 = \begin{bmatrix} c_{11} \times c_{21} & c_{11} \times c_{21} & .... & c_{11} \times c_{2n} \\ ... & ... & .... & ... \\ c_{1n} \times c_{21} & c_{1n} \times c_{22} & .... & c_{1n} \times c_{2n} \end{bmatrix}$$

Equality Query:

- When the client issues the query SELECT * FROM Table1 WHERE $a_i = v_i$;, proxy intercepts the query and first changes the table name from the mapping. Then proxy calculates the hash of $a_i = v_i$ and checks whether the hash value is present in its table or not. If it is not present proxy sends an empty result to the client. If it is present in the table proxy retrieves the corresponding polynomial from the table and finds the roots of the polynomial. After getting the roots proxy modifies the query as SELECT * FROM NewTable1 WHERE id IS IN (roots) and sends to the server. The server runs the query and returns the encrypted result to the proxy. Proxy decrypts the result and sends it to the client.

- When the client issues the query SELECT * FROM Table1 WHERE $a_1 = v_1$ AND $a_2 = v_2$ AND ... AND $a_t = v_t$;, proxy intercepts the query and first changes the table name. Then proxy calculates the hash of $a_i = v_i$ for all i and checks whether all the hash values are present in its table or not. If any

one of them is not present proxy sends an empty result to the client. If all the hash values are present in the table proxy retrieves all the corresponding polynomials from the table. Then proxy generates a new polynomial by adding all the polynomials. Since the rows that satisfy all the conditions should be the result so the index of that row should be the root of all those polynomials. We know that if r is a common root of two polynomials a(x) and b(x), then r is also a root of a(x) + b(x) [because, a(r) = 0, b(r) = 0. Now (a + b)(r) = a(r) + b(r) = 0 + 0 = 0] and if r is not a common root of a(x) and b(x) then r is not a root of a(x) + b(x)[because r is not a common root, without loss of generality let r be a root of a(x) and r is not a root of b(x). So a(r) = 0 and $b(r) \neq 0$. So (a + b)(r) = $a(r) + b(r) \neq 0$]. So proxy then calculates the roots of the new polynomial and picks the proper roots. After picking the roots proxy modifies the query as SELECT * FROM NewTable1 WHERE id IS IN (roots) and sends to the server. The server runs the query and returns the encrypted result to the proxy. Proxy decrypts the result and sends it to the client.

- When the client issues the query SELECT * FROM Table1 WHERE $a_1 = v_1$ OR $a_2 = v_2$ OR ... OR $a_t = v_t$;, proxy intercepts the query and first changes the table name. Then proxy calculates the hash of $a_i = v_i$ for all i. Then proxy retrieves all the corresponding polynomials from the table whose hash values are present in the table. Then proxy generates a new polynomial by multiplying all the polynomials. Since the rows that satisfy any one of the conditions should be the result so the index of that row should be the root of at least any one of those polynomials. We know that if r is a root of any one of two polynomials a(x) and b(x), then r is also a root of a(x) * b(x) [without

loss of generality let r be a root of a(x) and r is not a root of b(x). So a(r) = 0 and $b(r) \neq 0$. So (a * b)(r) = $a(r) * b(r) = 0$]. So proxy then calculates the roots of the new polynomial. After getting the roots proxy modifies the query as SELECT * FROM NewTable1 WHERE id IS IN (roots) and sends it to the server. The server runs the query and returns the encrypted result to the proxy. Proxy decrypts the result and sends it to the client.

- If AND and OR both are present proxy calculates the roots according to the precision and does the same.

Max and Min Query: When the client is interested to know the maximum or minimum element of an attribute, it does the max/min query. After intercepting the query when proxy watches that it is a max or min query it first changes the table and column name and then issues a query to MySQL server that will check only the first sub ciphertext of that particular column and return the row with the maximum first sub ciphertext in case of MAX query and return the row with minimum first sub ciphertext in case of MIN query.

Chapter 7

Conclusion and Future Work

This homomorphic encryption supports homomorphic addition, homomorphic multiplication, order comparison. As a result this scheme supports basic MySQL functions, range queries, equality checks etc. This scheme can be implemented in a cloud environment which uses MySQL queries to retrieve data. Moreover the encryption scheme that is used here is secure with respect to IND-FHOPE adversary and the security of the key K(n) is based on the hardness of AGCD problem.

Future Work: For equality query when the proxy server sends the ids to MySQL server after finding the roots the server knows which data are retrieved. A private information retrieval(PIR) scheme can be integrated between proxy server and MySQL server to solve this problem where the proxy works as a client and MySQL server works as a server. Then the server does not know which indices are queried. This scheme does not support queries that contains regular expression. So this can be extended so that it supports all the queries containing regular expression.