Securely Processing Range Predicates On Cloud Databases

A PROJECT REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

Master of Engineering

IN

Faculty of Engineering

BY

Manish Kesarwani

Computer Science and Automation
Indian Institute of Science
Bangalore – 560 012 (INDIA)

June, 2015

Declaration of Originality

I, **Manish Kesarwani**, with SR No. **04-04-00-10-41-13-1-10208** hereby declare that the material presented in the thesis titled

Securely Processing Range Predicates On Cloud Databases

represents original work carried out by me in the **Department of Computer Science and Automation** at **Indian Institute of Science** during the years **2013-2015**.
With my signature, I certify that:

- I have not manipulated any of the data or results.
- I have not committed any plagiarism of intellectual property. I have clearly indicated and referenced the contributions of others.
- I have explicitly acknowledged all collaborative research and discussions.
- I have understood that any false claim will result in severe disciplinary action.
- I have understood that the work may be screened for any form of academic misconduct.

Date: Student Signature

In my capacity as supervisor of the above-mentioned work, I certify that the above statements are true to the best of my knowledge, and I have carried out due diligence to ensure the originality of the report.

Advisor Name: Prof. Jayant R. Haritsa Advisor Signature

DEDICATED TO

My Family and Friends
for their love and support

Acknowledgements

I am deeply grateful to Prof. Jayant R. Haritsa for his unmatched guidance, enthusiasm and supervision. He has always been a source of inspiration for me. I have been extremely lucky to work with him.

I am thankful to Prof. Sanjit Chatterjee for providing the references during the Topics in Cryptology course which were very helpful in my project work. I am also very thankful to Prof. Bhavana Kanukurthi, Mr. Akshar Kaul, Mr. Gagandeep Singh, Mr. Anshuman Dutt and Dr. Prasad M. Deshpande for sharing the helpful references and time to time useful discussions that happened during the project work.

My sincere thanks goes to my fellow lab mates for all the help and suggestions. Also I thank my CSA friends who made my stay at IISc pleasant, and for all the fun we had together.

Finally, I am indebted with gratitude to my parents and brothers for their love and inspiration that no amount of thanks can suffice. This project would not have been possible without their constant support and motivation.

Abstract

The economics and flexibility of the Cloud have made it attractive for enterprises to leverage the "Database-as-a-Service" (DBaaS) model of cloud computing. The main challenge in the DBaaS model is to simultaneously provide data security with efficient query execution. In decision support systems, the bulk of the data processing activities comprises of queries having range predicates. Current solutions for secure processing of these predicates resort to order-preserving encryption (OPE) or prefix-preserving encryption (PPE), but these schemes have only been analysed for "Honest-but-Curious" server attack model.

In this thesis, we present "SPLIT Encryption Scheme", a new scheme for securely processing range predicates and ensuring strong security guarantees against more powerful attack models, where the "Honest-but-Curious" server has the ability to inject artificial queries into the system. SPLIT is a deterministic encryption scheme that takes any OPE or PPE ciphertext and splits them into two parts. Further these two ciphertexts are stored in separate database tables and any tuple correspondence between the two tables is removed. At query processing time, a range predicate is rewritten into an equivalent set of sub-range predicates and these sub-range predicates are directly processed from the ciphertext tables. Evaluation of SPLIT on the IBM DB2 system indicates that query execution time over SPLIT ciphertext is within *3 times* slowdown as compared to the query execution over plaintext database.

Contents

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

Cloud computing has leaded to the emergence of a new business model known as "Database-as-a-Service" (DBaaS) model. The well-known economics and flexibility of DBaaS model have made it attractive for enterprises to consider outsourcing their database query processing activities to computing environments hosted by third-party service providers such as Amazon EC2 and S3 [1], Microsoft Azure [2] and IBM Cloudant [3]. A crippling deterrent to this move, however, is the well-taken concern of ensuring the security of confidential data hosted in remote locations. Encryption techniques for protecting such data have been worked on for several decades now. However, the difficulty is that these techniques may not directly support query processing in the encrypted domain, forcing clients to download and decrypt the entire data before executing the query. The net effect is that the Cloud becomes reduced to a mere storage repository, and ceases to be a first-class data processing engine in its own right.

In this report we look at the problem of processing range predicates in SQL queries over encrypted cloud databases. Our technique to process range predicates securely and efficiently can also be used to handle equality predicates, group by and equi-join operators. Queries containing range predicates, equality predicates, equi-joins of relations and grouping are very common in both transaction systems as well as analytical systems. Hence it is evident that secure and efficient processing of range predicates in SQL queries is very critical for building a secure Database-as-a-Service system.

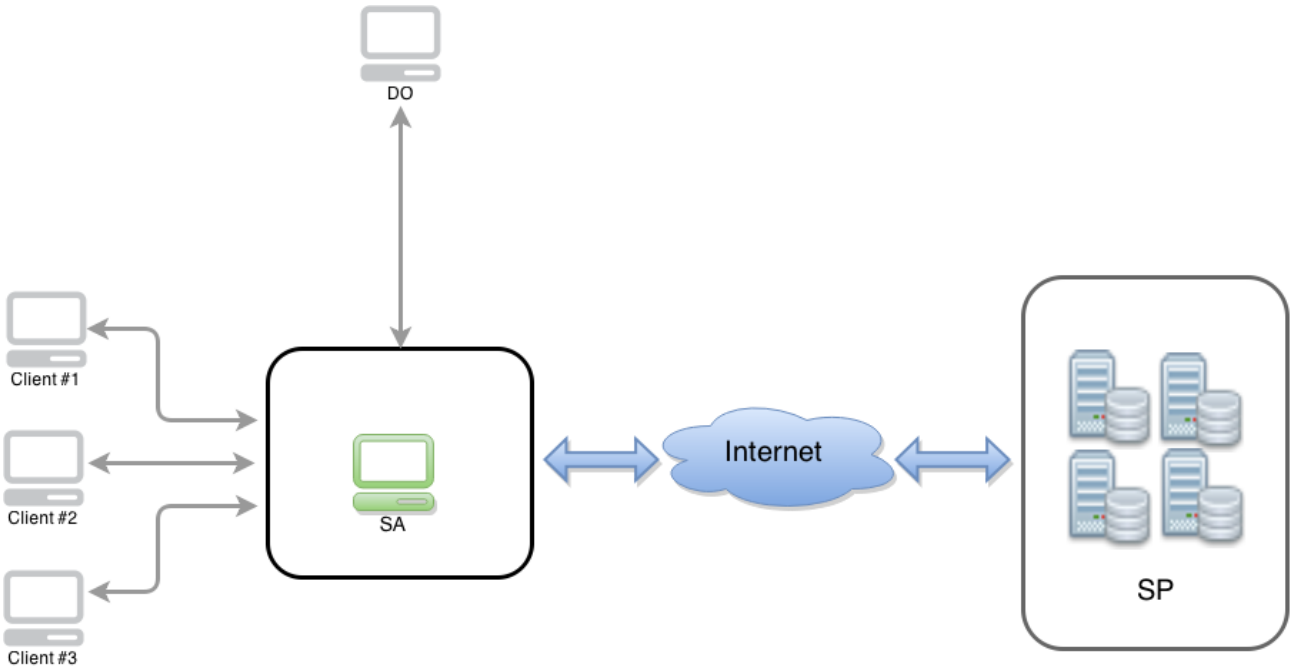


Figure 1.1: System Entities

The cloud computing model which we consider consists of *four* entities as shown in Figure 1.1. *Service Provider (SP)* which provides cloud infrastructure, *Data Owner (DO)* who owns the data and wants to store it on infrastructure hosted by *SP*, *Clients (C)* who are authorized to issue queries on data stored by *DO* at *SP*'s infrastructure and the *Security Agent (SA)* who acts as the bridge between *DO and SP* or *C and SP*. For example, a bank (DO) stores his data on IBM Cloudant (SP), and all the employees of the bank that perform analysis over bank's database will serve as the clients (C) in the system. *SA* is a transparent entity that helps to translate the plaintext queries into the corresponding query in the ciphertext space.

The adversarial model which describes the powers and objective of an adversary of the system is very closely related to these entities. Any entity which is inherently not trusted or which can be compromised adds to the power of the adversary. In our cloud computing model, the *Data Owner* and the *Security Agent* are considered trusted, while the *Service Provider* is always untrusted. One assumption we make here is that the database engine is not corrupted by the adversary (i.e. SP is honest in executing protocols). The *Clients* in our model can either be trusted or un-trusted. This gives rise to two adversarial model. A) *Honest-but-Curious* model: If the client is trusted then only *Service Provider* can be compromised. All the attacks in this scenario will be *passive* attacks. B) *Honest-but-Active* model: If the client is not-trusted or is compromised or colludes with the *Service Provider*, then adversary gains a lot of power. He can launch *active* attacks on the system. He can adaptively issue queries by acting as a

Client and then can see how that query is processed at *SP's* site.

The objective of adversary which we consider is *decrypting a selected encrypted value*. The intuition behind this is that adversary sees a lot of encrypted data being stored at the server. He chooses a particular encrypted data item (how he chooses is immaterial) and then tries to know the plaintext value of this data item. While trying to decrypt this ciphertext value he may learn plaintext values of some other ciphertext but that is not considered a high threat (since the ciphertext being decrypted is not in his control). All the previous work that allows the secure processing of range predicates in DBaaS model look at Honest-but-Curious model. However as we show in Section 3.3, Honest-but-Active model is a very realistic model and any DBaaS system of future should consider its security in this model rather than a weaker Honest-but-Curious model.

1.1 Background Review

Numerous attempts have been made to answer range predicates over encrypted database in DBaaS model. In *Bucketing Scheme* [10, 8] whole data domain is partitioned into buckets. Each bucket has a unique ID. This ID is stored at the server along with encrypted data. The input range query is mapped to bucket ids and all the rows which fall into these buckets are returned. This scheme leads to false positives if the bucket boundary does not match with range boundary. In *Order-Preserving Encryption Schemes (OPE)* [18, 7, 6, 17], encryption function preserves the order of the plaintext i.e. if a $\leq$ b then E(a) $\leq$ E(b). Any range predicate over plaintext can be easily converted into range predicate over encrypted values. Various implementations of order preserving encryptions have been proposed but they are not secure in Honest-but-Active model. The fundamental reason for this is that they leak the order of the plaintext values and a adversary in Honest-but-Active model can perform a binary search over the ciphertext values and successfully decrypt any target encrypted value. *Prefix-Preserving Encryption (PPE)* [13, 12] preserve the querying ability on the prefix of input data. Any range predicate on plaintext data is converted into queries having equality over prefixes. This encryption scheme is also not secure in Honest-but-Active model. A similar binary search over the ciphertext values can be performed by Honest-but-Active adversary to learn the plaintext value of any target ciphertext. There are also solutions [19] which work towards providing a new index structure to handle the range predicates in a secure manner. However these kind of solutions require change in database systems which is not a trivial task and faces stiff resistance from industry, further they are also not analysed in Honest-but-Active adversarial model. In SIGMOD 2014 Wong et. at. [20] have came up with a new encryption scheme SDB to securely process SQL queries over encrypted database, but in Chapter 5 we have shown attack against

SDB in Honest-but-Curious adversarial model itself. In Chapter 4 we will give examples to specifically show the weaknesses of OPE and PPE encryption scheme against Honest-but-Active adversary.

1.2 Our Contribution

In this thesis we propose a new encryption scheme, which takes any *order-preserving encryption* scheme or *prefix-preserving encryption* scheme and makes it secure in the sense that any Honest-but-Active adversary will not be able to compute the plaintext from its corresponding ciphertext despite of its adaptive querying power. Specifically we will show that, if N is the plaintext domain size, then an adversary can decrypt any ciphertext encrypted using OPE or PPE scheme in $\mathbf{log(N)}$ *adaptive queries*, while it has to make $\sqrt{\mathbf{N}}$ *adaptive queries* to decrypt any ciphertext encrypted using our proposed encryption scheme.

The OPE or PPE can be visualized using a complete binary tree over the ciphertext domain. Each encrypted value stores the whole path information from root of this ciphertext tree to the leaf node representing the input data. This property of having all the information in a single place leads to the possibility of binary search over the ciphertext domain. In our work we split the information of any ciphertext into two parts. Geometrically what we do it to divide all the levels in the complete binary tree representing the ciphertext into two contiguous parts, one part corresponds to the contiguous top levels of the tree while the other corresponds to the contiguous bottom levels of the tree. While encrypting a single data value we produce two ciphertexts. These two ciphertexts are then stored separately in different tables. Any association between the two tables is broken by randomizing the physical layout of the rows in the ciphertext tables. This makes our scheme secure in Honest-but-Active model. We call this scheme *SPLIT*. The scheme and its details are explained in Chapter 6.

Our SPLIT scheme can be implemented on top of current existing database engines. This is very important since it implies that SPLIT can be used in current DBaaS solutions. Through experimentation we have found that our scheme is efficient and practical. The SQL query execution in our scheme is *3 times* slow as compared to execution in plaintext database, which is a reasonable price to pay for getting higher security.

1.3 Organization

The rest of the thesis is organized as follows. Chapter 2 introduces some preliminaries. Chapter 3 introduces our problem framework and the attack model. Chapter 4 shows attacks against existing solutions in Honest-but-Active attack model. Chapter 5 contains an attack against the SDB encryption algorithm [20] in Honest-but-Curious adversarial model. Chapter 6 describes

our basic SPLIT encryption scheme and its security while Chapter 7 proposes extension to our scheme to general range predicate queries. In Chapter 8 we have presented the experimental evaluation of SPLIT. Chapter 9 concludes the report and describes the future work.

Chapter 2

Preliminaries

1. **Notations:** Let an integer λ be the security parameter. The running time of the adversary (and the running time of the honest parties) as well as the adversary's success probability are all viewed as functions of security parameter. The algorithms in this paper implicitly takes the security parameter as an input.

   If N is an integer, then $[N]$ denotes the set $\{1, \cdots, N\}$. For a set S, $x \xleftarrow{\$} S$ denote that x is uniformly randomly selected from set S, $|x|$ denotes its length in bits. By $x_1||\cdots||x_n$ we denote an encoding of $x_1, \cdots, x_n$ from which $x_1, \cdots, x_n$ are uniquely recoverable. $x[m \cdots n]$ denotes extracting bits m to n from the bits representation of x, where m lies towards the most significant bit of x. Let $\mathcal{P}$ denote the domain of plaintext message, then $[x]_n \xleftarrow{\$} \mathcal{P}$ denote that a set of n plaintexts are selected uniformly at random from domain $\mathcal{P}$.

2. **Negligible Success Probability**: (*Definition 3.4 of [11]*) Any encryption scheme is said to be secure if for any *probabilistic polynomial-time adversary* (adversaries running for polynomial number of steps in the security parameter λ), there exists an integer N such that for all integers $\lambda > N$ the probability that the adversary succeeds in breaking the scheme is $f(\lambda) < 1/p(\lambda)$, where $p(\cdot)$ is any polynomial in λ.

   It means that for every constant c the adversary's success probability is smaller than λ^{-c} for large enough values of λ. A function that grows smaller than any inverse polynomial is called *negligible* and the above function $f(\lambda)$ denotes the *negligible success probability* of the adversary.

2.1 Order-Preserving Encryption

Order-Preserving encryption (OPE) [18, 7, 6, 17] is an important class of encryption functions to support range query processing over encrypted databases. These are deterministic encryption schemes that preserve the numerical ordering of plaintext in the encrypted domain. Following definition of OPE is taken from [7].

Definition 2.1.1 (*Order-Preserving Encryption*) For $A, B \subseteq \mathbb{N}$ with $|A| \leq |B|$, a function $f : A \to B$ is order-preserving (aka. strictly-increasing) if for all $i, j \in A$, $f(i) > f(j)$ iff $i > j$. We say that deterministic encryption scheme $\mathcal{SE} = (\mathcal{K}, Enc, Dec)$ with plaintext and ciphertext-spaces $\mathcal{D}, \mathcal{R}$ is order-preserving if $Enc(K, \cdot)$ is an order-preserving function from $\mathcal{D}$ to $\mathcal{R}$ for all K output by $\mathcal{K}$ (with elements of $\mathcal{D}, \mathcal{R}$ interpreted as numbers, encoded as strings).

2.2 Prefix-Preserving Encryption

This PPE scheme was initially proposed by Xu et al. [13] for prefix-preserving IP address anonymization. This encryption scheme also supports the efficient evaluation of range predicates over encrypted databases.

Definition 2.2.1 (*Prefix-Preserving Encryption*) *([13])* We say that two n-bit numbers $a = a_1a_2...a_n$ and $b = b_1b_2...b_n$ share a k-bit prefix ($0 \leq k \leq n$), if $a_1a_2 \cdots a_k = b_1b_2 \cdots b_k$, and $a_{k+1} \neq b_{k+1}$ when $k < n$. An encryption function E_p is defined as a one-to-one function from $\{0,1\}^n$ to $\{0,1\}^n$. An encryption function E_p is said to be prefix-preserving, if, given two numbers a and b that share a k-bit prefix, $E_p(a)$ and $E_p(b)$ also share a k-bit prefix.

Chapter 3

Problem Framework

In this chapter we are going to state the problem we are going to solve, explain about the entities in our cloud framework and also define the adversary against our scheme.

3.1 Solution Objective

To provide an efficient, secure and practical implementation for processing the range predicates in SQL queries over encrypted cloud database in Honest-but-Active adversary model.

Service Provider should do all the processing pertaining to the evaluation of range predicates in a SQL query. Along with this query processing, some measurable data security should be achieved which restricts the learning of adversary about the encrypted data.

3.2 System Entities

Figure 1.1 shows various entities involved in cloud environment:

1. **Service Provider (SP)**: It is a third party that provides the storage and computation capability as a service to its clients. For our scenario, Service Provider can be a system where any present-day state of the art database engine is running. For example, IBM Cloudant. We assume that SP is honest in executing the protocols correctly, but he may be interested in the plaintext of the encrypted data stored at it's site, either because it is curious or on account of being compromised.

2. **Data Owner (DO)**: A company or an individual who is having a proprietary right to the data such as, a Bank. Data Owner wants to outsource its data storage and computation to some Service Provider. Data Owner is inherently trusted. This is because he does not have to mount any attack on the system to gain access to the data.

3. **Client (C)**: The authorized users of the data who can issue SQL queries on the data stored on SP's infrastructure. For example, employees of the bank performing analytics over the data to make some new policies. Trustworthiness of Client depends upon the adversarial model. In Honest-but-Curious model he is trusted whereas in Honest-but-Active model he is not trusted and may collude with SP to break the system.

4. **Security Agent (SA)**: It acts as the proxy system for communication between Data Owner and Service Provider or between Client and Service Provider. All the secret keys which are used for encrypting the data are stored at Security Agent. It performs necessary query rewriting of range predicates in SQL queries coming from Clients and also decrypts the results returned for the respective query from Service Provider. Since keys are stored at Security Agent it should be managed by someone trusted by Data Owner or by Data Owner himself.

3.3 Adversary Model

The adversarial model which we follow is Honest-but-Active adversary. In this model Data Owner and Security Agent are trusted entities whereas Service Provider and Clients are not trusted. Service Provider and Clients can also collude in order to break the system. The powers of adversary in this model are as follows:

1. Adversary has access to all data which is stored at Service Provider.

2. Adversary can monitor all the computations done over data by the database engine.

3. Adversary knows all the details of the encryption schemes which the Data Owner and Security Agent have used to encrypt the data. Only the keys used are secret.

4. Adversary can adaptively ask range predicate queries through the colluded Client.

The main distinguishing factor of this model from the Honest-but-Curious model is the point 4 above. Now lets define what kind of active attacks can be mounted on the system. Usually the clients query the database not by writing SQL queries but via some applications. These applications provide a form based interface to the client. The client can fill in various values in the forms and the application takes care of forming the appropriate SQL query and displaying the output. Thus adversary can iteratively ask queries by changing the form parameters according to the result of the previous query. Additionally we assume that the Adversary knows the query template which is fired for each form.

Adversary's Objective

As mentioned in [14], the most basic and often sufficient requirement is security against *Message Recovery (MR), under an adaptive adversary.* Therefore the objective of Honest-but-Active adversary considered in our system is to *decrypt a chosen ciphertext.* Now lets define what this means in terms of DBaaS model. Data Owner has stored data related to various Clients on the Service Provider. An Adversary has access to subset of this data depending upon the access control and query templates used by the form interface. The adversary now chooses a tuple to which he cannot fire a direct SQL query and then picks a cell from this tuple. The goal of the adversary is to decrypt this chosen cell value.

Note, here we are not specifying how the adversary chooses the target tuple and cell which he wants to break. But we require that he chose this cell before mounting an attack on the system.

3.3.1 Attack Models

In this subsection we will describe various attack models that are possible by varying the powers of Honest-but-Active adversary.

In all the below attack models $\mathcal{A}$ will represent the adversary against the deterministic encryption scheme $\mathcal{SE} = (KeyGen, Enc, Dec)$ and λ will denote the security parameter. The function *negl(λ)* will represent a negligible function in the security parameter (λ).

3.3.1.1 Ciphertext Only Attack Model

The adversary $\mathcal{A}$ is given a set of z ciphertexts of (uniformly) random messages from the challenger $\mathcal{C}$ and is asked to come up with the plaintext of any one of them. In this model the adversary is *passive* and can only see the ciphertexts of unknown messages. It can be considered as more general form of standard one-wayness (*Definition 6.1 of [11]*). This model can be more formally described in the form of below experiment against the deterministic encryption scheme $\mathcal{SE}$:

The One-Wayness Experiment $Exp^{One-Way}_{\mathcal{SE},\mathcal{A}}(\lambda)$

1. $\mathcal{C}$ *computes* $K \leftarrow KeyGen(\lambda)$.
2. $\mathcal{C}$ *chooses input* $[x]_n \xleftarrow{\$} \mathcal{D}$, *computes* $y_i \leftarrow Enc_K(x_i),\ \forall i \in \{1, \cdots, n\}$.
3. $\mathcal{A}$ *is given* λ *and* $[y]_n$ *as input, and outputs* x'.

4. *The output of the experiment is defined to be 1 if* $Enc_K(x') = y_i$*, for some* $i \in \{1, \cdots, n\}$*, and 0 otherwise.*

The *one-wayness advantage* of the adversary $\mathcal{A}$ against $\mathcal{SE}$ is

$$Adv_{\mathcal{SE},\mathcal{A}}^{One-Way}(\lambda) = |Pr[Exp_{\mathcal{SE},\mathcal{A}}^{One-Way}(\lambda) = 1]| \tag{3.1}$$

$\mathcal{SE}$ is said to be secure against *one-wayness adversary* $\mathcal{A}$, if

$$Adv_{\mathcal{SE},\mathcal{A}}^{One-Way}(\lambda) \leq negl(\lambda) \tag{3.2}$$

3.3.1.2 Non-Adaptive Attack Model

The adversary $\mathcal{A}$ *on input the security parameter* λ*, decides at the beginning of its execution the sequence of queries it will ask and gets their corresponding ciphertext from the challenger* $\mathcal{C}$*, along with this knowledge of pairs of <plaintext,ciphertext>,* $\mathcal{A}$ *is also given a challenge ciphertext* y^**. The objective of the adversary* $\mathcal{A}$ *is to come up with the plaintext of the challenge ciphertext.* This model can be thought as *non-adaptive Chosen Plaintext Attack* model with the adversary having the knowledge of some known plaintexts. We will call this as *KPA* Model. This model can be more formally described in the form of below experiment against the deterministic encryption scheme $\mathcal{SE}$:

The Non-Adaptive Experiment $Exp_{\mathcal{SE}}^{KPA}(\lambda)$

1. $\mathcal{C}$ *computes* $K \leftarrow KeyGen(\lambda)$*.*
2. $\mathcal{A}$ *on input* λ*, selects* $[x]_n \leftarrow \mathcal{D}$*, and gives* $[x]_n$ *to* $\mathcal{C}$*.*
3. $\mathcal{C}$ *computes* $y_i \leftarrow Enc_K(x_i)$*,* $\forall i \in \{1, \cdots, n\}$*, also chooses a challenge plaintext* $x^* \in \mathcal{D}$ *and computes the corresponding challenge ciphertext* $y^* \leftarrow Enc_K(x^*)$*, where* $x^* \notin \{x_1, \cdots, x_n\}$ *and* $x^* \in \mathcal{D}$*.*
4. $\mathcal{A}$ *is given* $[y]_n$*, the corresponding ciphertext to* $[x]_n$*, and the challenge ciphertext* y^* *and outputs* x'*.*
5. *The output of the experiment is defined to be 1 if* $x' = x^*$*, and 0 otherwise.*

Note, since we are using deterministic encryption scheme, the challenge plaintext does not lies in the $\mathcal{A}'s$ chosen list of plaintext, otherwise it would have been trivial for the adversary to come

up with the correct plaintext. The *KPA advantage* of the adversary $\mathcal{A}$ against $\mathcal{SE}$ is defined as

$$Adv^{KPA}_{\mathcal{SE},\mathcal{A}}(\lambda) = |Pr[Exp^{KPA}_{\mathcal{SE},\mathcal{A}}(\lambda) = 1]| \tag{3.3}$$

$\mathcal{SE}$ is said to be secure against *KPA adversary* $\mathcal{A}$, if

$$Adv^{KPA}_{\mathcal{SE},\mathcal{A}}(\lambda) \leq negl(\lambda) \tag{3.4}$$

3.3.1.3 Adaptive Attack Model

The adversary $\mathcal{A}$ is given the security parameter λ, the challenge ciphertext y^* and is allowed to make polynomial number of queries in the security parameter to encryption oracle. The objective of the adversary $\mathcal{A}$ is to come up with the plaintext of the challenge ciphertext. This model represents the basic *Chosen Plaintext Attack* without the indistinguishability test. We will call this model as *CPA* model. This model can be more formally described in the form of below experiment against the deterministic encryption scheme $\mathcal{SE}$:

The Adaptive Experiment $Exp^{CPA}_{\mathcal{SE}}(\lambda)$

1. $\mathcal{C}$ *computes* $K \leftarrow KeyGen(\lambda)$*, chooses a challenge plaintext* $x^* \in \mathcal{D}$ *and computes the corresponding challenge ciphertext* $y^* \leftarrow Enc_K(x^*)$*.*

2. $\mathcal{A}$ *is given* λ *and* y^* *as input. Now* $\mathcal{A}$ *adaptively asks the encryption of polynomial number of points from* $\mathcal{C}$ *and gets the corresponding encryptions. At the end* $\mathcal{A}$ *outputs* x'*.*

3. *The output of the experiment is defined to be 1 if* $x' = x^*$*, and 0 otherwise.*

The *CPA advantage* of the adversary $\mathcal{A}$ against $\mathcal{SE}$ is defined as

$$Adv^{CPA}_{\mathcal{SE},\mathcal{A}}(\lambda) = |Pr[Exp^{CPA}_{\mathcal{SE},\mathcal{A}}(\lambda) = 1]| \tag{3.5}$$

$\mathcal{SE}$ is said to be secure against *CPA adversary* $\mathcal{A}$, if

$$Adv^{CPA}_{\mathcal{SE},\mathcal{A}}(\lambda) \leq negl(\lambda) \tag{3.6}$$

3.3.1.4 Query Injecting Attack Model

The adversary $\mathcal{A}$ is given the security parameter λ, the challenge ciphertext y^* and is allowed to make polynomial number of range queries in the security parameter to encryption oracle. The objective of the adversary $\mathcal{A}$ is to come up with the plaintext of the challenge ciphertext.

This model is a generalized version of above *Adaptive Attack Model*, here the adversary asks for the encryption of a range of values rather than points. This model represents the real life threat under Honest-but-Active adversary where the adversary $\mathcal{A}$ accesses the encrypted cloud database through some user interface where it can alter the range of queries in the where clause. We will call this model as *QI* model. This model can be more formally described in the form of below experiment against the deterministic encryption scheme $\mathcal{SE}$:

The Query-Injecting Experiment $Exp^{QI}_{\mathcal{SE}}(\lambda)$

1. $\mathcal{C}$ *computes* $K \leftarrow KeyGen(\lambda)$*, chooses a challenge plaintext* $x^* \in \mathcal{D}$ *and computes the corresponding challenge ciphertext* $y^* \leftarrow Enc_K(x^*)$.

2. $\mathcal{A}$ *is given* λ *and* y^* *as input. Now* $\mathcal{A}$ *adaptively asks the encryption of polynomial number of range queries from* $\mathcal{C}$ *and gets the corresponding encryptions. At the end* $\mathcal{A}$ *outputs* x'.

3. *The output of the experiment is defined to be 1 if* $x' = x^*$*, and 0 otherwise.*

The *QI advantage* of the adversary $\mathcal{A}$ against $\mathcal{SE}$ is defined as

$$Adv^{QI}_{\mathcal{SE},\mathcal{A}}(\lambda) = |Pr[Exp^{QI}_{\mathcal{SE},\mathcal{A}}(\lambda) = 1]| \tag{3.7}$$

$\mathcal{SE}$ is said to be secure against *QI adversary* $\mathcal{A}$, if

$$Adv^{QI}_{\mathcal{SE},\mathcal{A}}(\lambda) \leq negl(\lambda) \tag{3.8}$$

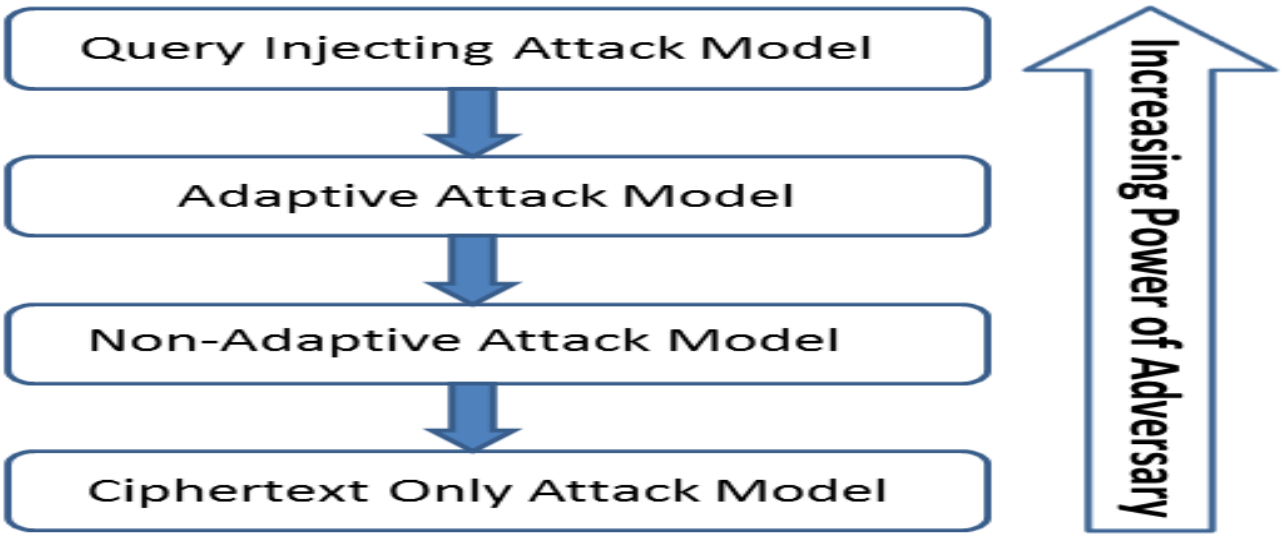


Figure 3.1: Hierarchy of Attack Models

We can easily show that there is a strict hierarchy among the adversarial models as shown in Figure 3.1. The security in higher attack models implies security in lower models, but not vice-versa. Later in Chapter 4, we will show that all the existing schemes that allow the evaluation of range predicates over encrypted data are not secure against atleast one of the attack models while in Chapter 6 we will show that our SPLIT encryption scheme is secure against *QI-adversary* if the plaintext domain is sufficiently large, which implicitly implies security in other attack models. Figure 3.2 shows a comparison of security of SPLIT scheme with the existing schemes.

Scheme / Attack Models	OPE	PPE	PBTree	SDB	SPLIT
Ciphertext Only	✓	✓	✓	✓	✓
Non Adaptive (KPA)	✓	✓	✓	✗	✓
Adaptive (CPA)	✗	✗	✓	✗	✓
Query Injecting (QI)	✗	✗	✗	✗	✓

Figure 3.2: Comparison of Security of SPLIT with Existing Schemes

3.4 Assumptions

There are various assumptions which are implicit in the adversarial model and DBaaS model which we use. We list them here:

1. There are certain tuples in the database to which Clients do not have direct access, either because of access control or because of the form interface provided by the application they are using to access the encrypted data.

2. The communication channel between client and server is secure. This can be ensured by using techniques such as TLS (Transport Layer Security) [15].

3. All the existing secure block cipher schemes which we use are unbreakable. If any of them is broken in future then we can replace it with another unbroken secure block cipher scheme having similar properties.

Chapter 4

Security of Existing Solutions

In this chapter we will analyse the security of existing solutions proposed in [18, 7, 17, 13, 12] under the attack model described in Section 3.3.1. Consider there is a Bank (say B) whose database schema for the customer section is shown in Figure 4.1.

Customer *(Customer_ID, Age)*
Account *(Customer_ID, Branch_ID, Balance)*

Figure 4.1: Customer Database Schema

Now suppose, in order to make new policies, the customer section of the bank wants to analyse the relation between the age and the account balances of the customers, so the bank may expose an interface to the analyst that issues the SQL query shown in Figure 4.2 to the customer database.

```
SELECT Branch_ID, Count(*)
FROM Customer, Account
WHERE Customer.Customer_ID = Account.Customer_ID
      AND Customer.Age ≥ 30
      AND Customer.Age ≤ 70
      AND Account.Balance ≥ 500000
      AND Account.Balance ≤ 5000000000
GROUP BY Branch_ID;
```

Figure 4.2: SQL query to "Count the number of customers in each branch having age between 30 and 70 and account balance between 500,000 to 5,000,000,000"

Say the Honest-but-Active adversary $\mathcal{A}$ has access to the form (through a colluded Client) used to issue the query presented in Figure 4.2. Now say $\mathcal{A}$ has selected a tuple from the *Account* table and it wants to decrypted the encrypted *Balance* cell in that tuple. Lets see how $\mathcal{A}$ can decrypt the target encrypted cell if the data is encrypted using one of the existing encryption schemes.

4.1 Order Preserving Encryption (OPE)

If the Bank's database is encrypted using any order preserving encryption scheme, then the Honest-but-Active adversary using the *Query Injecting Attack Model*, can perform a simple binary search over the encrypted data and decrypt the selected encrypted *Balance* cell.

In the particular example query shown in Figure 4.2, say the age of customer lies between $[0, 128]$, then $\mathcal{A}$ will first set the *Age* to this range to make sure that the target tuple is not pruned in its injected query because of incorrect *Age* value. Now $\mathcal{A}$ will do a binary search over the *Balance* column. Say negative balances are not allowed and the account balance of the customers can take any positive integer value of *64-bit*. Then the Honest-but-Active adversary will first set the balance to lie in range $[0, 2^{62} = 4611686018427387904]$ as shown in Figure 4.3, where *Customer_E and Account_E* are the encrypted *Customer and Account* tables under OPE scheme. Now if the target row is selected by this query then $\mathcal{A}$ will recur with binary search

```
SELECT Branch_ID, Count(*)
FROM Customer_E, Account_E
WHERE Customer_E.Customer_ID=Account_E.Customer_ID
      AND Customer_E.Age ≥ 0
      AND Customer_E.Age ≤ 128
      AND Account_E.Balance ≥ 0
      AND Account_E.Balance ≤ 4611686018427387904
GROUP BY Branch_ID;
```

Figure 4.3: SQL query over encrypted tables to "Count the number of customers in each branch having age between 0 and 128 and account balance between 0 to 2^{62}"

using range $[0, 2^{62} - 1]$, otherwise it will recur over range $[2^{62} + 1, 2^{63} - 1]$. Thus, in the worst case $\mathcal{A}$ will be able to decrypt the target tuple in just $\mathbf{log(2^{64}) = 64}$ queries.

If the underlying data is coming from uniform distribution, then the *interpolation search* [4] is also possible over the encrypted domain, in this case the average number of chosen range queries required by $\mathcal{A}$ to decrypt the target encrypted cell is $\mathbf{log(log(2^{64})) = 8}$.

Thus, we see that against any OPE scheme the Honest-but-Active adversary mounting a *Query Injecting attack* is able to decrypt any target encrypted data cell on average in $\mathbf{log(log(N))}$ queries and in worst case in $\mathbf{log(N)}$ queries, where N is the plaintext domain size.

4.2 Prefix Preserving Encryption (PPE)

If the Bank's database is encrypted using any prefix preserving encryption scheme, then as shown in Section 4.1 the Honest-but-Active adversary using the *Query Injecting Attack Model*, can perform a similar binary search over the encrypted data and decrypt the selected encrypted *Balance* cell.

Again, after observing the processing of the query shown in Figure 4.3(in this case *Customer_E and Account_E* are the encrypted *Customer and Account* tables under PPE scheme) the Honest-but-Active adversary $\mathcal{A}$ will be able to correctly learn the *most significant bit* of the target encrypted data cell. Now again depending on its observation $\mathcal{A}$ will be select the appropriate range for its next query and continue the *binary search* over the encrypted data.

Note, in case of PPE *interpolation search* is not possible, since the encrypted data is not ordered. Hence the Honest-but-Active adversary mounting a *Query Injecting attack* against any PPE scheme will be able to decrypt any target encrypted data cell on average and in worst case in $\mathbf{log(N)}$ queries, where N is the plaintext domain size.

4.3 PBtree Approach [19]

If the Bank's database is encrypted using *PBtree* approach [19] presented in VLDB, 2014, then the Honest-but-Active adversary using the *Query Injecting Attack Model*, can perform a simple binary search over the encrypted data and decrypt the selected encrypted *Balance* cell.

Similarly, adversary will first inject the query shown in Figure 4.3(here *Customer_E and Account_E* are the encrypted *Customer and Account* tables under PBtree scheme) and observe that whether its target tuple is getting selected in the result or not. Depending on its outcome the adversary can adaptively choose the range predicates for its next query and perform the binary search.

Hence the Honest-but-Active adversary mounting the *Query Injecting attack* against the *PBtree* scheme will be able to decrypt any target encrypted data cell on average and in worst case in $\mathbf{log(N)}$ range queries, where N is the plaintext domain size.

4.4 SDB Approach [20]

If the Bank's database is encrypted using the *SDB* approach [20] presented in SIGMOD, 2014, then the Honest-but-Active adversary using the *Non-Adaptive Attack Model*, *Adaptive Attack Model* or the *Query Injecting Attack Model* can decrypt the selected encrypted *Balance* cell. The authors themselves mention that they are not secure under any active attack models, but later in Chapter 5 we will explicitly show an attack scenario where the knowledge of only one pair of <*plaintext,ciphertext*>, will help the adversary to decrypt all the values in particular encrypted columns.

Chapter 5

Attack on SDB Approach [20]

In this chapter we will present an attack against the SDB approach in the Honest-but-Curious attack model with the adversary having the knowledge of one pair of known plaintext and ciphertext. For a background on SDB encryption algorithm, its security model and its various SQL operator implementation readers can refer [20].

Claim 1: SDB approach is not secure under DB + QR Security Model, if the adversary has the knowledge of only one tuple of known plaintext and ciphertext. **Proof:** Let us consider that the cloud server acts as QR adversary (say $\mathcal{C}$) against the SDB approach. Now $\mathcal{C}$ will follow below steps to compute the plaintext values of encrypted column.

1. $\mathcal{C}$ will first observe a few number (>= 3) of comparison operations, on some encrypted column say A, that is involved in a correlated sub query doing comparison with unknown constants. We will use these comparisons to get the scaled value of random valued column R. This we will call as *Result1*.

2. Next $\mathcal{C}$ will use the instructions sent from client to server to answer $A > B$ comparison query, to curiously compute $A > -B$ comparison, where A is the encrypted column involved in the comparisons done in step 1 and B is any other column. This we will call as *Result2*.

3. Now after computing *Result1* and *Result2* from above steps, $\mathcal{C}$ will mathematically combine them to expose some scaled value of column A.

4. Now, given only one pair of plaintext and ciphertext (i.e. plaintext as well as encrypted value of only one tuple of A), $\mathcal{C}$ will be able to compute the exact values of all the tuples in column A.

Thus we see that by following above steps $\mathcal{C}$ is able to compute and reveal the plaintext values of all the tuples of encrypted column A, hence SDB cannot be accepted as a general purpose solution in real applications.
Let us see the above steps in greater details:

Step 1: Computing the scaled value of random valued column R.

Suppose over a period of time following comparison queries have been made by the client to the cloud server as part of a correlated query,

- $A > k_1$
- $A > k_2$
- $A > k_3$

In general, SP cannot distinguish whether an operation is done on A or on some constant multiple of A, so let us consider the general form of above equations:

- $\alpha A > k_1$
- $\alpha A > k_2$
- $\alpha A > k_3$

In the above comparisons A and B are encrypted database columns while $k_1,\ k_2,\ k_3\ and\ \alpha$ are unknown constants.

When above comparisons are performed, then according to the comparison protocol, following equations can be inferred by the server with their plaintext values,

$$[R \times (\alpha A - k_1 \times S)] = C_1\ (say) \tag{5.1}$$
$$[R \times (\alpha A - k_2 \times S)] = C_2\ (say) \tag{5.2}$$
$$[R \times (\alpha A - k_3 \times S)] = C_3\ (say) \tag{5.3}$$

In the above equations, all the values on the right hand side are available in plaintext at server side, while the left hand side in [ ] is inferred by the server because of its knowledge of the comparison protocol. R denotes the random value column and S denotes column with constant value 1, these are the two additional encrypted columns added by DO, before sending the encrypted database to the cloud server SP.
Although all the columns R, A and S as well as k_i's involved in the left hand side of above

equations within [ ] are encrypted and their actual values are not known to the server, but the server knows that the plaintext values on the right hand side of above equations are computed by evaluating the expressions on the plaintext values of respective columns and constants involved in that equation. Therefore we can rewrite above equations as:

$$[R \times (\alpha A - K_1)] = C_1 \tag{5.4}$$
$$[R \times (\alpha A - K_2)] = C_2 \tag{5.5}$$
$$[R \times (\alpha A - K_3)] = C_3 \tag{5.6}$$

where, K_1 is a vector with constant value k_1, K_2 is a vector with constant value k_2 and K_3 is a vector with constant value k_3.
Now, subtracting equation (5.4) from equation (5.5), we get,

$$[R \times K_1 - R \times K_2] = C_2 - C_1$$
$$\Rightarrow [R \times (K_1 - K_2)] = C_2 - C_1$$

Let $Y_1 = (K_1 - K_2)$ and $Z_1 = (C_2 - C_1)$, therefore we can write above equation as,

$$[R \times Y_1] = Z_1 \tag{5.7}$$

Suppose there are N tuples in the database, then we can write equation (5.7) as the element-wise product of two vectors:

$$\begin{pmatrix} r_1 \\ r_2 \\ r_3 \\ . \\ . \\ . \\ r_N \end{pmatrix} \times \begin{pmatrix} y_1 \\ y_1 \\ y_1 \\ . \\ . \\ . \\ y_1 \end{pmatrix} = \begin{pmatrix} z_{11} \\ z_{12} \\ z_{13} \\ . \\ . \\ . \\ z_{1N} \end{pmatrix}$$

$$\Rightarrow \begin{pmatrix} r_1 \times y_1 \\ r_2 \times y_1 \\ r_3 \times y_1 \\ . \\ . \\ . \\ r_N \times y_1 \end{pmatrix} = \begin{pmatrix} z_{11} \\ z_{12} \\ z_{13} \\ . \\ . \\ . \\ z_{1N} \end{pmatrix} \tag{5.8}$$

Lets rewrite above equation as:

$$\begin{pmatrix} r_1 \times y_1 & r_2 \times y_1 & r_3 \times y_1 & . & . & . & r_N \times y_1 \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} & z_{13} & . & . & . & z_{1N} \end{pmatrix} \tag{5.9}$$

Similarly, by subtracting equation (5.4) from equation (5.6) we get,

$$[R \times K_1 - R \times K_3] = C_3 - C_1$$
$$\Rightarrow [R \times (K_1 - K_3)] = C_3 - C_1$$

Let $Y_2 = (K_1 - K_3)$ and $Z_2 = (C_3 - C_1)$, therefore we can write above equation as,

$$[R \times Y_2] = Z_2 \tag{5.10}$$

Following the previous steps, we will get:

$$\begin{pmatrix} r_1 \times y_2 & r_2 \times y_2 & r_3 \times y_2 & . & . & . & r_N \times y_2 \end{pmatrix} = \begin{pmatrix} z_{21} & z_{22} & z_{23} & . & . & . & z_{2N} \end{pmatrix} \tag{5.11}$$

Now let's combine equation (5.9) and (5.11) in one matrix, we get

$$\begin{pmatrix} r_1 \times y_1 & r_2 \times y_1 & r_3 \times y_1 & . & . & . & r_N \times y_1 \\ r_1 \times y_2 & r_2 \times y_2 & r_3 \times y_2 & . & . & . & r_N \times y_2 \end{pmatrix} = \begin{pmatrix} z_{11} & z_{12} & z_{13} & . & . & . & z_{1N} \\ z_{21} & z_{22} & z_{23} & . & . & . & z_{2N} \end{pmatrix} \tag{5.12}$$

Now taking the column wise GCD of the above matrix we get,

$$\begin{aligned} GCD([r_1 \times y_1], [r_1 \times y_2]) &= GCD(z_{11}, z_{21}) \\ &= [r_1] \; if \; [y_1] \; and \; [y_2] \; are \; relatively \; prime \\ &= [r_1] \times GCD([y_1], [y_2]) \; otherwise \end{aligned}$$

Let us take the case when $[y_1]$ and $[y_2]$ are not relatively prime and let,

$$GCD([y_1], [y_2]) = [c]$$

where [c] is some constant.

$$\begin{aligned} \Rightarrow GCD([r_1 \times y_1], [r_1 \times y_2]) &= [r_1] \times [c] \\ &= [r_1 \cdot c] \end{aligned}$$

Similarly, we can compute the rest of $[r_i \cdot c]$, where $2 \leq i \leq n$. **Thus now we have $[R \cdot c]$ exposed in plaintext at the server. This we will call as *Result1*.**

Step 2: Computing $A > -B$ comparison from the instructions of $A > B$ comparison query.

To compute $A > B$ following steps are performed:

1. Client chooses a new column key $<m_C, x_C>$.
2. Now client issues the key update operations for columns A and B, to the new column key $<m_C, x_C>$.
3. Thus server now has:
   - κ(A, $<m_C, x_C>$) = $A_e^{'}$
   - κ(B, $<m_C, x_C>$) = $B_e^{'}$
4. Next client asks server to compute $C_e = A_e^{'} - B_e^{'}$.
5. Now following the comparison protocol, server performs EE multiplication between encrypted column R and newly constructed column C, thus server stores $D_e = R_e \times C_e$ and client sets column key for D as $ck_D = < m_D, x_D > = < m_R \cdot m_C, x_R + x_C >$.
6. Now server performs a key update on column D using column key $< 1, 0 >$.

By performing above steps client exposes $[R \times (A - B)]$ in plaintext to the server.
Now server can compute the result of $A > -B$ comparison by only modifying the step 4 of above comparison protocol, the modification is as follows:

- Instead of computing $C_e = A_e^{'} - B_e^{'}$, server computes $C_e = A_e^{'} + B_e^{'}$.

Rest all the steps 1, 2, 3, 5 and 6 are followed as described previously.
We can easily verify that following above steps, the QR adversary can compute the plaintext values of $[R \times (A + B)]$. From the adversarial point of view, SP can only infer the plaintext values of following equations :

$$[R \times (\delta A - \beta B)] = C_5 \ (say), and \tag{5.13}$$

$$[R \times (\delta A + \beta B)] = C_6 \ (say) \tag{5.14}$$

where, δ and β are unknown constants and C_5 and C_6 are exposed in plaintext to the server. We will refer above equations as *Result2*.

Step 3: Exposing scaled value of column A.
Adding equations (5.13) and (5.14), we get,

$$[2 \times R \times \delta A] = C_5 + C_6 \tag{5.15}$$

Let $Z_3 = (C_5 + C_6)$ / 2,

$$\Rightarrow [R \times \delta A] = Z_3 \tag{5.16}$$

Dividing equation (5.16) by *Result1*, i.e. $[R \cdot c]$, we get,

$$\left[\frac{\delta A}{c}\right] = \frac{Z_3}{[R \cdot c]} \tag{5.17}$$

Let $\gamma = \frac{\delta}{c}$, then

$$[\gamma A] = \frac{Z_3}{[R \cdot c]} \tag{5.18}$$

Thus the above equation exposes the plaintext values of column A multiplied by some unknown constant γ at the server side. Similarly, following the same approach we can expose the scaled values of other columns also.

Step 4: Exposing exact values of columns.
Now if SP knows only one pair of plaintext and ciphertext value of any tuple in column A , then it can compute the value of γ, let a_i be the plaintext value of some tuple in column A, then γ will be computed as below,

$$\frac{\gamma \cdot a_i}{(a_i)} = \gamma \tag{5.19}$$

where $\gamma \cdot a_i$ is the corresponding tuple of column A exposed as the result of Step 3.

Once the server is able to compute the value of γ, then it can expose all the tuples in column A.

Chapter 6

Encryption with SPLIT

6.1 Basics

In this chapter we propose a deterministic encryption scheme for processing equality and range predicates in SQL queries over encrypted integer data. Our scheme can be built upon any OPE or PPE scheme proposed in the literature. We will prove that our proposed scheme is secure against Honest-but-Active adversary, mounting the QI attack shown in Section 3.3.1. We will call our scheme as SPLIT.

The vulnerability of the existing schemes is due to the binary search attack as shown in Section 4. The main idea of SPLIT is to break the chain of queries in binary search by,

1. Splitting a single ciphertext into two parts and storing them in different tables.
2. Removing correlation among the two ciphertext tables.

Before formally describing the SPLIT scheme, it would be helpful to consider the geometric interpretation of our scheme and how it achieves the above two objectives. If we consider the ciphertexts that can take any n-bit value, then the entire set of these ciphertexts can be inherently represented by a complete binary tree of height n. This we will call as *ciphertext tree (CT)*. For ease of explanation let's consider the PPE ciphertext tree for *4-bit* integers as shown in Figure 6.1(a). In this case n is 4 and CT contains nodes at 5 different levels. We will denote the leaf level consisting of 2^4 nodes as L_0 and the root of the tree consisting of a single node as L_4.

Every integer in the leaf level of CT can be associated with 4 bits of information depending upon its path from root to level L_0. Next, we will divide all the levels into two categories as follows:

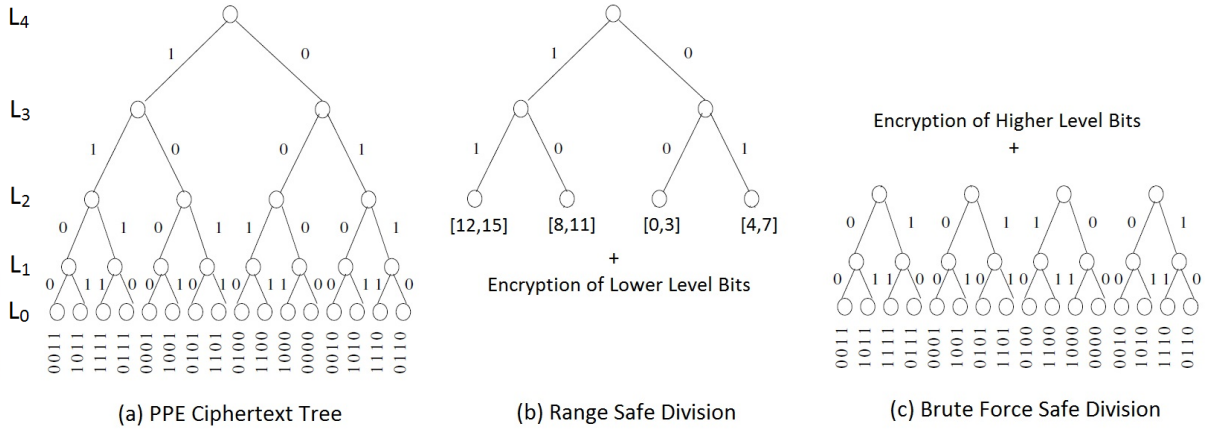


Figure 6.1: SPLIT Scheme Basic

1. **Range Safe (RS)**: This category will consist of some contiguous top levels from the tree, for example let, levels L_2 to L_4 comprise the RS range. This means that for every node at level L_0, the bits corresponding to levels L_2 to L_4 will preserve the semantics of the underlying PPE scheme while the remaining bits information coming from levels L_0 to L_2 is blinded using a *secure block cipher* (for eg. AES). Thus every leaf node in level L_0 is associated with a range at level L_2 instead of a value as shown in Figure 6.1(b). Here in our example the range represented by each leaf node is of 2^2 integers granularity. Thus the Honest-but-Active adversary mounting the *QI attack* against this part of the tree with the objective of *Message Recovery* from the ciphertexts will be able to identify the nodes at coarser granularity.

2. **Brute Force Safe (BS)**: This category will consists of the remaining contiguous levels of the CT tree from level L_0 up to the level where RS range ends. For example, after assigning L_2 to L_4 to RS range the levels L_0 to L_2 will comprise the BS range, while the bit information coming from higher levels is blinded using a *secure block cipher* (for eg. AES). Here the leaf nodes correspond to the actual integer values of CT tree but their higher order bit information is missing as shown in Figure 6.1(c). Thus the Honest-but-Active adversary with the objective of *Message Recovery* from the ciphertexts in this part of the tree will be forced to mount a brute force *chosen plaintext attack* over all the 2^2 nodes at level L_2 to guess the exact value of any node at level L_0.

Note that both the above categories consists of disjoint set of contiguous bits associated with the corresponding levels of ciphertext tree. The number of levels in BS or RS range is a configurable parameter for the DO to decide for every column on which the range predicate

query can occur. This parameter in turn decides the security of our scheme. In the coming sections we will assume that the number of levels in RS division is equal to the number of levels in BS division (meaning divide the OPE/PPE ciphertext into two equal halves).

6.2 Formal Definition

The SPLIT encryption scheme for plaintext domain $\mathcal{D}$ is a tuple of polynomial-time algorithms SPLIT = (KeyGen, Enc, Dec) run by the Data Owner (DO), Security Agent (SA) and the Client (C), where Keygen is probabilistic and the rest are deterministic.

1. ***Key Generation:*** *$sk \leftarrow KeyGen(\lambda)$. KeyGen* is the probabilistic algorithm that runs at the DO's site, takes as input the security parameter λ, and output the secret key *sk*. The *sk* consists of two equal length secret keys *(sk_1,sk_2)* of the underlying *OPE/PPE* encryption algorithm $\mathcal{E}$, upon which *SPLIT* will be defined and also two equal length secret keys *(k_1,k_2)* of a secure block cipher (AES in our case), i.e. $sk = (sk_1, sk_2, k_1, k_2)$. The *sk* is stored at the DO's as well as at the SA's site.

2. ***Encryption:*** *$c \leftarrow Enc(sk,m)$. Enc* is the deterministic encryption algorithm, composed of two separate algorithms Enc_{RS} and Enc_{BS}. Enc_{RS} is called to compute the Range Safe (RS) component of the ciphertext while Enc_{BS} is called to computes the Brute Force Safe (BS) component of the ciphertext. Each of these algorithm takes a part of the secret key *sk* and the plaintext message *m* and compute the ciphertext *c*. The output of *Enc* can be either RS component or BS component of the ciphertext depending upon which of the two algorithms is invoked. These algorithms can run at either the DO's or SA's site.

   - $c_{RS} \leftarrow Enc_{RS}(sk_1, k_1, m)$. Enc_{RS} first calls the underlying encryption algorithm $\mathcal{E}$ with the secret key sk_1 and generates the ciphertext $c'_{RS} \leftarrow \mathcal{E}(sk_1, m)$. Say $c'_{RS} \in \{0,1\}^n$. Let's represent $c'_{RS} = c'_{RS_l}||c'_{RS_r}$, where $c'_{RS_l}, c'_{RS_r} \in \{0,1\}^{n/2}$, then Enc_{BS} computes
   $$c_{RS} \leftarrow c'_{RS_l}||\mathcal{E}^{k_1}_{s\mathcal{E}}(c'_{RS_r}) \tag{6.1}$$
   In the above equation $\mathcal{E}^{k_1}_{s\mathcal{E}}$ represents a secure block cipher that is invoked with the secret key k_1. This algorithm sets $c \leftarrow c_{RS}$.

   - $c_{BS} \leftarrow Enc_{BS}(sk_2, k_2, m)$. Enc_{BS} first calls the underlying encryption algorithm $\mathcal{E}$ with the secret key sk_2 and generates the ciphertext $c'_{BS} \leftarrow \mathcal{E}(sk_2, m)$. Say $c'_{BS} \in \{0,1\}^n$. Let's represent $c'_{BS} = c'_{BS_l}||c'_{BS_r}$, where $c'_{BS_l}, c'_{BS_r} \in \{0,1\}^{n/2}$, then Enc_{BS} computes
   $$c_{BS} \leftarrow \mathcal{E}^{k_2}_{s\mathcal{E}}(c'_{BS_l})||c'_{BS_r} \tag{6.2}$$

In the above equation $\mathcal{E}_{s\mathcal{E}}^{k_2}$ represents a secure block cipher that is invoked with the secret key k_2. This algorithm sets $c \leftarrow c_{BS}$.

Note, in both the RS and BS part of the ciphertext only half of the bits follow the underlying property of the OPE/PPE scheme, while the other half is blinded using the secure block cipher. Further the RS and BS ciphertexts are stored in separate tables and the correlation of rows among the two tables is removed. We will further explain this in Section 6.3

3. ***Decryption:*** $m \leftarrow Dec(sk,c)$. *Dec* is the decryption algorithm, composed of two separate algorithms Dec_{RS} and Dec_{BS}. Dec_{RS} is used to decrypt the Range Safe (RS) component of the ciphertext while Dec_{BS} is used to decrypt the Brute Force Safe (BS) component of the ciphertext. Each of these algorithm takes a part of the secret key sk and the ciphertext message c and compute the plaintext message m. These algorithms can run at either the DO's or SA's site. The output of *Dec* is the plaintext message m corresponding to the ciphertext c.

   - $m \leftarrow Dec_{RS}(sk_1, k_1, c)$. The ciphertext c can be decomposed into two parts as $c \equiv c'_{RS_l}||\mathcal{E}_{s\mathcal{E}}^{k_1}(c'_{RS_r})$. Dec_{RS} first calls the decryption algorithm for the secure block cipher $\mathcal{D}_{s\mathcal{E}}^{k_1}$ and computes a part of the *OPE/PPE* ciphertext i.e. $c'_{RS_r} \leftarrow \mathcal{D}_{s\mathcal{E}}^{k_1}(\mathcal{E}_{s\mathcal{E}}^{k_1}(c'_{RS_r}))$. Let $c'_{RS} = c'_{RS_l}||c'_{RS_r}$. Then the underlying decryption algorithm $\mathcal{D}$ of the *OPE/PPE* scheme is called with the secret key sk_1 and the plaintext message m is computed as $m \leftarrow \mathcal{D}(sk_1, c'_{RS})$.
   - $m \leftarrow Dec_{BS}(sk_2, k_2, c)$. The ciphertext c can be decomposed into two parts as $c \equiv \mathcal{E}_{s\mathcal{E}}^{k_2}(c'_{BS_l})||c'_{BS_r}$. Dec_{RS} first calls the decryption algorithm for the secure block cipher $\mathcal{D}_{s\mathcal{E}}^{k_2}$ and computes a part of the *OPE/PPE* ciphertext i.e. $c'_{BS_l} \leftarrow \mathcal{D}_{s\mathcal{E}}^{k_2}(\mathcal{E}_{s\mathcal{E}}^{k_2}(c'_{BS_l}))$. Let $c'_{BS} = c'_{BS_l}||c'_{BS_r}$. Then the underlying decryption algorithm $\mathcal{D}$ of the *OPE/PPE* scheme is called with the secret key sk_2 and the plaintext message m is computed as $m \leftarrow \mathcal{D}(sk_2, c'_{BS})$.

   Note, calling the appropriate encryption and decryption algorithm is the responsibility of the DO or the SAs, but the Clients have no direct access to these modules.

Algorithm 1 ($\boldsymbol{KeyGen}(\lambda)$ - runs at DO)

1. $sk_1 \leftarrow OPE/PPE.KeyGen(\lambda)$
2. $sk_2 \leftarrow OPE/PPE.KeyGen(\lambda)$
3. $k_1 \leftarrow DET.KeyGen(\lambda)$
4. $k_2 \leftarrow DET.KeyGen(\lambda)$

Return $sk \leftarrow (sk_1, sk_2, k_1, k_2)$.

Figure 6.2: SPLIT Key Generation Algorithm

Algorithm 2 ($\boldsymbol{Enc}(\mathbf{sk}, \mathbf{m})$ - runs at DO or SA)
Composed of two separate algorithms

1. **($\boldsymbol{Enc}_{\mathbf{RS}}(\mathbf{sk_1}, \mathbf{k_1}, \mathbf{m})$)**
   - $c'_{RS} \leftarrow \mathcal{E}(sk_1, m)$
   - $c'_{RS} \equiv c'_{RS_l} || c'_{RS_r}$
   - $c_{RS} \leftarrow c'_{RS_l} || \mathcal{E}^{k_1}_{\mathcal{SE}}(c'_{RS_r})$

   Return $c \leftarrow c_{RS}$
2. **($\boldsymbol{Enc}_{\mathbf{BS}}(\mathbf{sk_2}, \mathbf{k_2}, \mathbf{m})$)**
   - $c'_{BS} \leftarrow \mathcal{E}(sk_2, m)$
   - $c'_{BS} \equiv c'_{BS_l} || c'_{BS_r}$
   - $c_{BS} \leftarrow \mathcal{E}^{k_2}_{\mathcal{SE}}(c'_{BS_l}) || c'_{BS_r}$

   Return $c \leftarrow c_{BS}$

Figure 6.3: SPLIT Encryption Algorithm

Algorithm 3 ($Dec(\mathbf{sk}, \mathbf{c})$ - runs at DO or SA)
Composed of two separate algorithms

1. $(Dec_{\mathbf{RS}}(\mathbf{sk_1}, \mathbf{k_1}, \mathbf{c}))$

   - $c \equiv c'_{RS_l} || \mathcal{E}^{k_1}_{s\xi}(c'_{RS_r})$
   - $c'_{RS_r} \leftarrow \mathcal{D}^{k_1}_{s\xi}(\mathcal{E}^{k_1}_{s\xi}(c'_{RS_r}))$
   - $c'_{RS} = c'_{RS_l} || c'_{RS_r}$

   Return $m \leftarrow \mathcal{D}(sk_1, c'_{RS})$.

2. $(Dec_{\mathbf{BS}}(\mathbf{sk_2}, \mathbf{k_2}, \mathbf{c}))$

   - $c \equiv \mathcal{E}^{k_2}_{s\xi}(c'_{BS_l}) || c'_{BS_r}$.
   - $c'_{BS_l} \leftarrow \mathcal{D}^{k_2}_{s\xi}(\mathcal{E}^{k_2}_{s\xi}(c'_{BS_l}))$
   - $c'_{BS} = c'_{BS_l} || c'_{BS_r}$.

   Return $m \leftarrow \mathcal{D}(sk_2, c'_{BS})$.

Figure 6.4: SPLIT Decryption Algorithm

The SPLIT algorithm is summarized in Figures 6.2, 6.3, 6.4.

6.3 Data Organization for 1D Range Predicate

In this section we will assume that only 1D range predicate queries are allowed. Later, in Section 7.1 we will remove this assumption and provide a general solution.

In SPLIT every data item is encrypted two times and two different ciphertexts are produced, further these ciphertexts are stored in two different tables. So, for every database table in the plaintext world two separate tables will be created in encrypted world. In particular, for the example schema shown in Figure 4.1, the encrypted database that will be created after applying SPLIT is shown in Figure 6.5. Here, we have assumed that range predicate query will appear only on column *Age* or on column *Balance* but not on both columns simultaneously in the same query. Since other columns namely, *Customer_ID and Branch_ID* are having unique numbers for every Customer or Branch in the bank, thus they can be encrypted using any secure block cipher like *AES or DES*. The encryption key of the block cipher used in *Age_BS, Age_RS, Balance_BS,*

Customer_BS$(Customer\_ID\_E_{K_1},\ Age\_BS)$
Customer_RS$(Customer\_ID\_E_{K_2},\ Age\_RS)$
Account_BS$(Customer\_ID\_E_{K_1}, Branch\_ID\_E_{K_3}, Balance\_BS)$
Account_RS$(Customer\_ID\_E_{K_2}, Branch\_ID\_E_{K_4}, Balance\_RS)$

Figure 6.5: Customer Database Schema, Encrypted using SPLIT Scheme

Balance_RS columns are different while the encryption keys for the two *Customer_ID* columns in *Customer_BS* and *Account_BS* tables or in *Customer_RS* and *Account_RS* is kept same to allow joins of these tables.

In addition to creating two encrypted tables for every plaintext table we need to randomize the physical layout order of the rows in these tables. For example, of the two tables *Account_BS* and *Account_RS* created for *Account* table we will randomize the physical order of rows in one table, in order to remove the correlation among these two tables, such that the first tuple in *Account_BS* does not necessarily be present as the first tuple in physical layout of the *Account_RS* table.

Later in Section 6.5 we will show that by following above steps we have removed the attacks on SPLIT that were earlier possible on our underlying *OPE/PPE* encryption schemes as shown in Chapter 4.

6.4 Implementing 1D Range Predicate Queries over Encrypted Relations

In this section, we will show how a ID range predicate query will be executed in our framework after the data is encrypted and stored at the Service Provider's site.

SELECT *Count(*)*
FROM *Customer*
WHERE *Customer.Age* $\geq$ *3*
AND Customer.Age $\leq$ *9*

Figure 6.6: SQL query to "Count the number of Customers who have age between 3 and 9"

Suppose the underlying encryption mechanism used is *PPE* and the plaintext and ciphertext data comes from *4-bit* integer domain (PPE tree shown in Figure 6.1). Let us assign levels L_2 to

1. Starting from $k = 1$, find the most significant bit, numbered k, for which $a_k < b_k$.
2. If k is not found, i.e., for all $i \leq i \leq n$, $a_i = b_i$, then the interval can be denoted by prefix $a_1 a_2 \cdots a_n$. Return $a_1 a_2 \cdots a_n$.
3. If for all $k \leq i \leq n$, $a_i = 0$ and $b_i = 1$, then return $a_1 a_2 a_{k-1} *$ (return $*$ if $k = 1$).
4. Transform interval $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$ into $[a_1 \cdots a_{k-1} 0 a_{k+1} \cdots a_n, a_1 \cdots a_{k-1} 011 \cdots 1]$ $\cup$ $[a_1 \cdots a_{k-1} 100 \cdots 0, a_1 \cdots a_{k-1} 1 b_{k+1} \cdots b_n]$.
5. Run this algorithm with interval $[a_{k+1} \cdots a_n, 11 \cdots 1]$ as input, concatenate $a_1 \cdots a_{k-1} 0$ before all the returned prefixes. Then run this algorithm with interval $[00 \cdots 0, b_{k+1} \cdots b_n]$ as input, concatenate $a_1 \cdots a_{k-1} 1$ before all the returned prefixes. Return all the prefixes.

Figure 6.7: The algorithm for transforming interval $[a_1 a_2 \cdots a_n, b_1 b_2 \cdots b_n]$ into prefixes

level L_4 into the RS range while levels L_0 to L_2 into the BS range. Let's consider the encrypted database schema is as shown in Figure 6.5 and say the Client wants to "*Count the number of Customers who have age between 3 and 9*", the SQL query corresponding to the Client's request is shown in Figure 6.6.

Now following steps are performed:

1. *Client* will send the plaintext query (shown in Figure 6.6) to the *Security Agent (SA)*.
2. Now *SA* will first identify the subranges that will be answered from *Range Safe Account table* and the subranges that will be answered from *Brute Force Safe Account table*. This can done using the algorithm shown in Figure 6.7. The number of subrange predicates in this case will be bounded by $2 * (log(N) - 1)$ as shown in [13], where N is the plaintext domain size. The interval $[3, 9]$ will be transformed into below sub-intervals:

   - For RS table: $[4, 7]$.
   - For BS table: $[3]$, $[8, 9]$,

   Now *SA* will encrypt the end points using the Enc_{BS} and Enc_{RS} algorithms. Let's assume for this example, that the AES encryption of higher order bits of BS table is AES[00] = 10, AES[01] = 00, AES[10] = 11, AES[11] = 01. Now *SA* will generate queries shown in Figure 6.8 for the RS and BS table respectively.

```
SELECT Count(*)
FROM Customer_RS
WHERE Customer_RS.Age ≥ 8
      AND Customer_RS.Age ≤ 11
```

```
SELECT Count(*)
FROM Customer_BS
WHERE (Customer_BS.Age = 6 OR
      AND Customer_BS.Age ≥ 10)
      AND Customer_BS.Age ≤ 11)
```

Figure 6.8: SQL query for RS and BS table respectively

3. Now, *SA* will send the transformed queries shown in Figure 6.8 to the *Service Provider (SP)*.

4. *SP* will execute the received queries and will send the count of tuples in each query to the *SA*. If instead of the count, entire tuples of the tables are requested by the *Client*, then in this step *SP* would have returned encrypted tuples to *SA* as a result of query evaluation, instead of their count.

5. Now, *SA* will simply add the counts received from the two queries and send the final result back to *Client*. If instead of the count *SA* have received encrypted tuples as the query result from *SP*, then *SA* would have decrypted individual tuples and have taken the union of the tuples returned from the two queries.
   Note, *SA* is able to do simple addition of count or union of tuples from the two queries, since these queries access disjoint set of tuples from the encrypted tables.

6. *Client* receives the plaintext result of its issued query.

6.5 Security of SPLIT

In this section we will show the security of SPLIT scheme against Honest-but-Active adversary mounting a *QI* attack.

Lets first consider that the underlying encryption on which SPLIT is build upon is *OPE* and each plaintext and ciphertext data is a *64-bit* integer. Levels L_{32} to level L_{64} belong into the RS range while levels L_0 to L_{32} belong into the BS range. We will prove the security of SPLIT encryption scheme inclemently by proving that

1. BS and RS encryptions are independently secure.
2. For any plaintext table, there is no correlation between the corresponding BS and RS ciphertext tables.

Finally we will use the above knowledge to prove the overall security of SPLIT encryption scheme.

Claim 6.5.1 If AES is a secure block cipher, then any probabilistic polynomial time adversary $\mathcal{A}_{SPLIT_{BS}}$ will have negligible advantage in the adaptive Query Injecting experiment $Exp^{QI}_{SPLIT,\mathcal{A}}(\lambda)$ against $SPLIT_{BS}$ encryption scheme.

Proof: We will prove the claim by the method of contradiction. Assume that there exists a *QI*-adversary $\mathcal{A}_{SPLIT_{BS}}$ against $SPLIT_{BS}$ scheme with some non-negligible advantage ϵ. Consider that there exists a challenger $\mathcal{C}_{AES}$ for AES block cipher and there exists another adversary $\mathcal{A}_{AES}$ that wants to break the AES ciphertext. Now we will show that $\mathcal{A}_{AES}$ will act as the challenger ($\mathcal{C}_{SPLIT_{BS}}$) of $SPLIT_{BS}$ encryption scheme and will use $\mathcal{A}_{SPLIT_{BS}}$ to break the AES ciphertext with non-negligible advantage ϵ, which will contradict the assumption that AES is a secure block cipher. Hence the adversaries $\mathcal{A}_{AES}$ and $\mathcal{A}_{SPLIT_{BS}}$ does not exists and $SPLIT_{BS}$ is a secure encryption scheme.

The above argument can be better understood by a game between the challenger $\mathcal{C}_{AES}$, $\mathcal{A}_{AES}$ (also acts as $\mathcal{C}_{SPLIT_{BS}}$) and $\mathcal{A}_{SPLIT_{BS}}$ as shown in Figure 6.9, 6.10.

Since we have assumed that $\mathcal{A}_{SPLIT_{BS}}$ has a non-negligible advantage in the *QI-experiment* with the $\mathcal{C}_{SPLIT_{BS}}$,

$$Adv^{QI}_{\mathcal{C}_{SPLIT_{BS}},\mathcal{A}_{SPLIT_{BS}}}(\lambda) = \epsilon \tag{6.3}$$

hence $\mathcal{A}_{SPLIT_{BS}}$ will computes the correct value of the plaintext of the challenge ciphertext with ϵ probability. Now $\mathcal{A}_{SPLIT_{BS}}$ will compute the OPE encryption of that plaintext and the higher order *32-bits* of OPE ciphertext are its guess of the challenge *AES ciphertext*. Now if $\mathcal{A}_{SPLIT_{BS}}$ has correctly computed the plaintext then $\mathcal{A}_{AES}$ will win the AES challenge with the same non-negligible ϵ probability, i.e.

$$Pr[success\ of\ \mathcal{A}_{AES}] = \epsilon \tag{6.4}$$

which contradicts the assumption that AES is a secure block cipher. Hence the adversary $\mathcal{A}_{SPLIT_{BS}}$ against $SPLIT_{BS}$ encryption scheme with non-negligible advantage does not exists. □

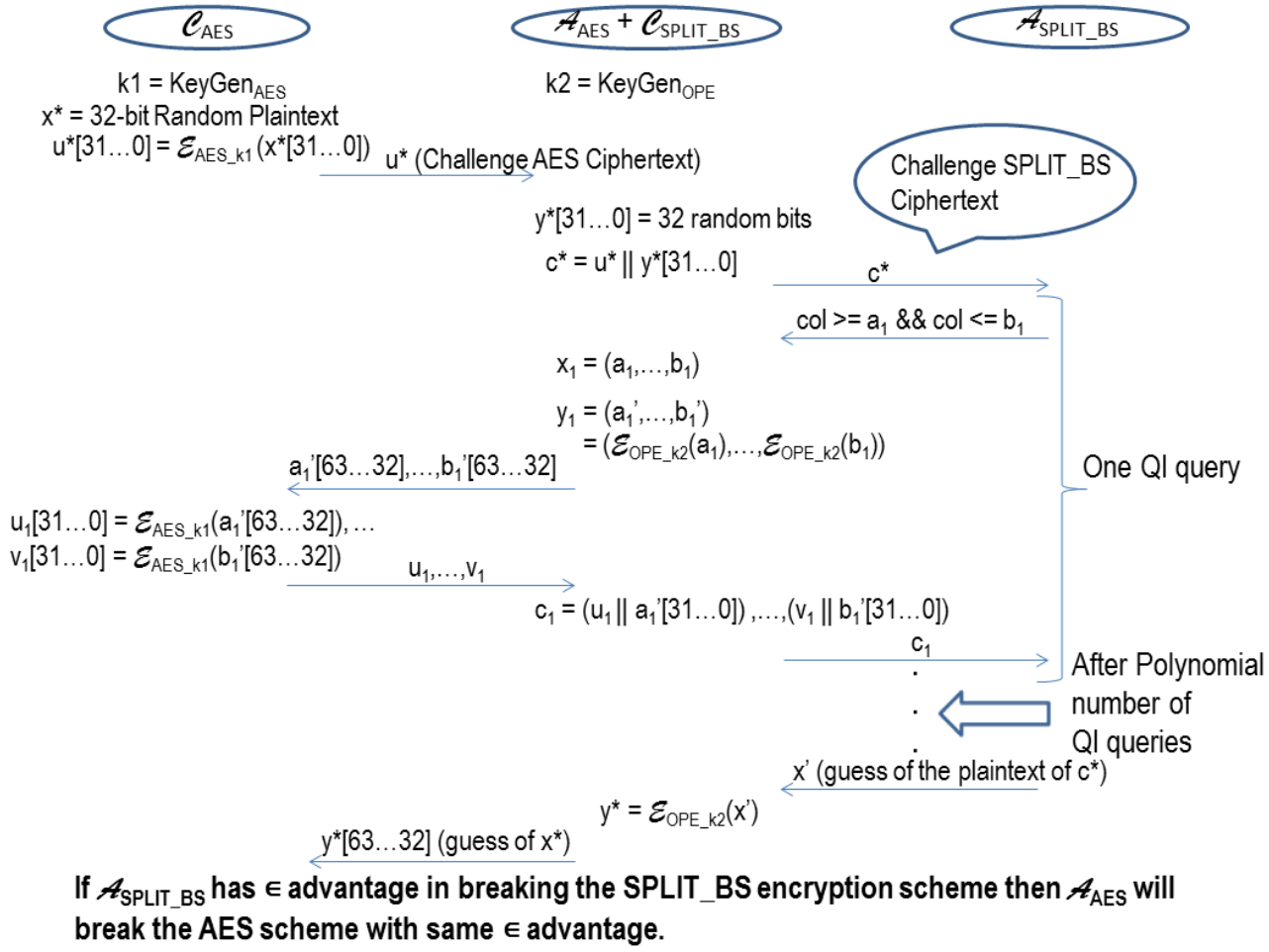


Figure 6.10: BS Security Game

Claim 6.5.2 If AES is a secure block cipher, then any probabilistic polynomial time adversary $\mathcal{A}$ will have negligible advantage in the adaptive Query Injecting experiment $Exp^{QI}_{SPLIT,\mathcal{A}}(\lambda)$ against $SPLIT_{RS}$ encryption scheme.

Proof: The prove for this claim will be exactly same as the prove of Claim 6.5.1, with a slight modification, that instead of higher order bits lower order bits of the OPE ciphertext will be selected for AES encryption. We are omitting the proof because of space restriction. □

Till now we have seen that BS and RS encryptions are individually secure, but according to the SPLIT encryption scheme, the BS and RS encryptions of every plaintext data exist together

1. The Challenger $\mathcal{C}_{AES}$ generates the key $k_1 \leftarrow KeyGen_{AES}$ for the AES block cipher and selects a random *32-bit* challenge plaintext $x^* \xleftarrow{\$} \{0,1\}^{32}$ and computes the challenge ciphertext $u^*[31 \cdots 0] \leftarrow \mathcal{E}_{AES_{k_1}}(x^*)$, and forwards u^* to the AES adversary $\mathcal{A}_{AES}$.

2. The AES adversary $\mathcal{A}_{AES}$ also acts as the challenger $\mathcal{C}_{SPLIT_{BS}}$ for the $SPLIT_{BS}$ encryption scheme and generates the key $k_2 \leftarrow KeyGen_{OPE}$ for the OPE encryption scheme. Now it randomly selects *32-bits*, $y^*[31 \cdots 0] \leftarrow \{0,1\}^{32}$, compute the $SPLIT_{BS}$ challenge ciphertext $c^* = u^*||y^*[31 \cdots 0]$ and forward c^* to adversary $\mathcal{A}_{SPLIT_{BS}}$.

3. Now adversary $\mathcal{A}_{SPLIT_{BS}}$ will ask for the encryption of a range of values from $\mathcal{C}_{SPLIT_{BS}}$. Say it has send the range from $[a_1, b_1]$.

4. Upon receiving the query, $\mathcal{C}_{SPLIT_{BS}}$ will compute the OPE ciphertext for the complete range $[a_1, b_1]$ i.e. $y_1 = (a_1', \cdots, b_1') = (\mathcal{E}_{OPE_{k_2}}(a_1), \cdots, \mathcal{E}_{OPE_{k_2}}(b_1))$ and sends the most significant *32-bits* of the OPE ciphertext $((a_1'[63 \cdots 32], \cdots, b_1'[63 \cdots 32]))$ to $\mathcal{C}_{AES}$.

5. $\mathcal{C}_{AES}$ computes the AES encryptions of the requested list of values, $u_1[31 \cdots 0] = \mathcal{E}_{AES_{k_1}}(a_1'[63 \cdots 32]), \cdots, v_1[31 \cdots 0] = \mathcal{E}_{AES_{k_1}}(b_1'[63 \cdots 32])$, and sends $u_1, \cdots, v_1$ to $\mathcal{A}_{AES}$ (playing the role $\mathcal{C}_{SPLIT_{BS}}$).

6. $\mathcal{C}_{SPLIT_{BS}}$ computes the requested ciphertext $c_1 = (u_1||a_1'[31 \cdots 0]), \cdots, (v_1||b_1'[31 \cdots 0])$ of the $SPLIT_{BS}$ scheme, and sends c_1 to $\mathcal{A}_{SPLIT_{BS}}$. This completes the processing of one range predicate query of $\mathcal{A}_{SPLIT_{BS}}$.

7. Now adversary $\mathcal{A}_{SPLIT_{BS}}$ can ask for more range predicate queries (polynomial in the security parameter) and $\mathcal{C}_{SPLIT_{BS}}$ will construct their response as shown in steps 4-6 above.

8. At the end $\mathcal{A}_{SPLIT_{BS}}$ will output its guess x' as the plaintext for its challenge ciphertext c^*.

9. Now $\mathcal{C}_{SPLIT_{BS}}$ will compute $y^* \leftarrow \mathcal{E}_{OPE_{k_2}}(x')$, and sends $y^*[63 \cdots 32]$ as its guess of x^* to AES challenger $\mathcal{C}_{AES}$.

Figure 6.9: AES adversary $\mathcal{A}_{AES}$ using the $SPLIT_{BS}$ adversary $\mathcal{A}_{SPLIT_{BS}}$ to solve the AES challenge

in the database, but in separate database tables and the physical layout of the rows of these tables is also made different. Say a function F_k is used to produce the different permutation

of rows in the encrypted database tables, and since this permutation is never required to be inverted hence a new secret key is generated every time permutation of rows is required and the secret key is discarded after words.

Claim 6.5.3 If F_k is a strong pseudorandom permutation, then any probabilistic polynomial time adversary $\mathcal{A}$ will have negligible advantage in correlating the BS and RS encryptions of the same plaintext present in their respective tables.

Proof: For ease of explanation, consider the read-only database over which only 1D range predicate queries are allowed, later in Section 7 we will show, how our scheme handles higher dimensional range predicate queries and also updates in the database. Whenever DO wants to store some encrypted data at SP's site the function F_k is invoked with a new secret key for each ciphertext table to decide the layout of tuples in that table. Since the secret key is never reused, hence the generated permutation of tuples is equivalent to selecting a random permutation. Now if the Honest-but-Active adversary selects any tuple from BS (or RS) table it equally likely maps to all the distinct tuples (since our scheme is deterministic) of the RS (or BS) table. Say $\mathcal{M}$ is the number of distinct tuples in RS table, then the probability of correlating any tuple of BS table with RS table tuple is given by

$$Pr[corelating\ tuples\ in\ BS\ and\ RS\ tables] = \frac{1}{M}$$

Since adversary $\mathcal{A}$ is not able to prune any tuple of RS table as the corresponding tuple of BS table and vice versa, hence such an attack comes in the category of brute force search, therefore not considered as a threat to encryption scheme. □

Till now we have seen that BS and RS encryptions are independently secure, and there is no correlation between the tuples of BS and RS ciphertext tables containing the encryption of same plaintext table.

Claim 6.5.4 If AES is a secure block cipher and F_k is a strong pseudorandom permutation, then any probabilistic polynomial time adversary $\mathcal{A}_{SPLIT}$ will have negligible advantage in the adaptive Query Injecting experiment $Exp_{\mathcal{A}}^{QI}(\lambda)$ against $SPLIT$ encryption scheme.

Proof: Since half of the bits of BS or RS ciphertext are equal to the underlying OPE/PPE encryption scheme, hence the adversary can perform a similar binary search over these bits as shown in Section 4. But as proved in Claims 6.5.1 and 6.5.2 adversary $\mathcal{A}_{SPLIT}$ will not be able to decrypt any challenge BS or RS ciphertext, since AES is a secure block cipher. Further as proved in Claim 6.5.3 there is no correlation between the tuples of BS and RS ciphertext

tables, since the key used by the function F_k to produce the permutation of tuples in ciphertext tables is never reused again and hence the permutation produced is equivalent to a random permutation, therefore $\mathcal{A}_{SPLIT}$ will not be able to continue its binary search over the challenge ciphertext from BS table to RS table and vice versa. Thus the adversary $\mathcal{A}_{SPLIT}$ will not be able to win the QI experiment with non-negligible advantage against SPLIT scheme. □

Above we have proved that *QI-adversary* is not able to decrypt the challenge ciphertext in SPLIT encryption scheme. But there is some partial leakage of information associated with BS and RS ciphertexts.

1. **Leakage from Range Safe Ciphertext:** If we look at the algorithm Enc_{RS} shown in Figure 6.3, we will notice that after we get the *OPE* ciphertext $c'_{RS} \in \{0,1\}^{64}$, *bits 0 to 31* are blinded using a strong pseudorandom permutation while *bits 32 to 63* will still preserve the *OPE* property. Thus the Honest-but-Active adversary will be able to mount the attack shown in Section 4.1 only on *bits 32 to 63*, and will only be able to identify the underlying integer at the granularity of 2^{32}.

2. **Leakage from Brute Force Safe Ciphertext:** If we look at the algorithm Enc_{BS} shown in Figure 6.3, we will notice that after we get the *OPE* ciphertext $c'_{RS} \in \{0,1\}^{64}$, *bits 32 to 63* are blinded using a strong pseudorandom permutation while *bits 0 to 31* will still preserve the *OPE* property. Now because of the blinding of higher order bits each pair of nodes at level L_{31} equally likely belong to all the 2^{32} nodes present at level L_{32}. Thus the Honest-but-Active adversary can only mount a brute force attack to identify the exact plaintext value of any chosen ciphertext.

For an example, consider the encrypted bank's database schema shown in Figure 6.5. Now suppose, Honest-but-Active adversary $\mathcal{A}_{SPLIT}$ has selected a tuple in *Account_BS* table and wants to decrypt the *Balance_BS* cell in that tuple. If it starts with the query shown in Figure 4.3, then again part of its query will be answered from *Account_BS* table and part from *Account_RS* table, and even if its target encrypted cell gets selected, $\mathcal{A}_{SPLIT}$ will not be able to guess the 2^{32} granularity node to which its target cell value belongs. Now, suppose $\mathcal{A}_{SPLIT}$ starts to do a binary search over lower order bits i.e. *bits 0 to 31*, then $\mathcal{A}_{SPLIT}$ will be able to correctly guess the *bits 0 to 31* of its target cell in just *32 QI queries*, but now to guess all the bits it needs to launch a bruteforce search over all the equally likely 2^{32} nodes at level L_{32} of the ciphertext tree, i.e. $\mathcal{A}_{SPLIT}$ has to make $2^{32} = \sqrt{2^{64}}$ *QI* queries, to successfully break any target encrypted data cell, where 2^{64} is the plaintext domain size

Thus, the security of SPLIT depends on two factors

- The number of levels in the RS and BS divisions, i.e. the split point.
- The plaintext domain size (say N).

Thus, *if the split point is chosen such that number of levels in the RS division is equal to the number of levels in the BS division and N is a sufficiently large plaintext domain size such that making $\sqrt{N}$ QI queries becomes infeasible in the security parameter, then SPLIT is secure against the Honest-but-Active adversary mounting a QI attack.*

Chapter 7

Extension to SPLIT

7.1 Support for MultiD Range Predicates

Till now we have seen, how our scheme takes the plaintext table and generates encrypted database and how the 1D range predicate queries are handled. In this chapter we will first show how to extend our scheme to support 2D range predicate queries and similarly we can extend it for other higher dimensions.

Suppose in our example database shown in Figure 4.1, *Client* wants to ask a query having simultaneous range predicates over *Age* and *Balance* columns. To securely support such queries *Data Owner* needs to create four replication of each table corresponding to all the BS and RS enumerations of the two columns on which range predicates can occur simultaneously (in our example the *Age* and *Balance* columns). The encrypted database to allow such queries is shown in Figure 7.1

Customer_BS_BS$(Customer\_ID\_E_{K_1}, Age\_BS)$
Customer_BS_RS$(Customer\_ID\_E_{K_2}, Age\_BS)$
Customer_RS_BS$(Customer\_ID\_E_{K_3}, Age\_RS)$
Customer_RS_RS$(Customer\_ID\_E_{K_4}, Age\_RS)$
Account_BS_BS$(Customer\_ID\_E_{K_1}, Branch\_ID\_E_{K_5}, Bal\_BS)$
Account_BS_RS$(Customer\_ID\_E_{K_2}, Branch\_ID\_E_{K_6}, Bal\_RS)$
Account_RS_BS$(Customer\_ID\_E_{K_3}, Branch\_ID\_E_{K_7}, Bal\_BS)$
Account_RS_RS$(Customer\_ID\_E_{K_4}, Branch\_ID\_E_{K_8}, Bal\_RS)$

Figure 7.1: Customer Database Schema, Encrypted using SPLIT Scheme for 2D range predicates

In general, if there are T tables and total n columns in the database, and the *Data Owner*

wants to support simultaneous range predicate over all of the n *columns*, then every database table will be enumerated 2^n times, corresponding to all the combinations of BS and RS split of each column, therefore there will $T * 2^n$ tables in the encrypted database. In Section 7.3 we will show how we can reduce this exponential space blow up.

7.2 Support of Equi-Join, Group By and count based Having Clause in SPLIT Scheme

In SPLIT scheme, only tables corresponding to the same enumeration of the BS and RS division can be joined, further this is also sufficient to produce the correct result. For instance, in the schema shown in Figure 7.1, *Customer_BS_BS* can be joined with *Account_BS_BS* on the *Customer_ID* column, since same encryption key is used for this column data in both the tables, but join of *Customer_BS_BS* with *Account_BS_RS* is not allowed, otherwise Honest-but-Active adversary can trivially launch a similar binary search attack over our scheme as shown in Section 4 for existing solutions.

Since SPLIT scheme is a deterministic encryption scheme, it trivially supports the *Group By* clause, but there is a possibility that same groups may appear as the result of different sub queries being executed on different tables. These groups can be trivially merged by the *SA* after decrypting the independent results.

Similarly, count based *Having* clause can be supported in SPLIT scheme. Here also if same groups appear in the result of different sub queries executed on different tables, *SA* needs to simply add the counts before returning the final result to the *Client*.

7.3 Optimizations and Improvements of SPLIT

In this section we will show how we can *reduce the required storage overhead* for the MultiD case, how we can *enhance the overall security*, and how we can *reduce the query execution time* in SPLIT scheme.

1. **Equivalence Classes Partitioning of Columns:** Say $\{Col\}$ represents a set comprising of all the columns in the database, then if the *Data Owner (DO)* is able to:

   (a) Partition $\{Col\}$ into two disjoint sets, such that $\{Set_1\}$ is the set of database columns that will not be used as range predicate in any legal SQL query, and $\{Set_2\}$ is the set of all the columns on which range predicates can occur, formally

   - $\{Set_1\} \cap \{Set2\} = \phi$
   - $\{Set_1\} \cup \{Set_2\} = \{Col\}$

(b) Further *DO* can partition $\{Set_2\}$ into a number of equivalence classes, such that no two column in each equivalence class can occur simultaneously as range predicate in any legal SQL query against the database. Say *DO* is able to identify d equivalence classes, then

- $\{Set_{2i}\} \cap \{Set_{2j}\} = \phi \qquad \forall (i,j) \in \{1,...,d\}$
- $\{Set_{21}\} \cup \{Set_{22}\} \cup ... \cup \{Set_{2d}\} = \{Set_2\}$

Since only the columns in $\{Set_2\}$ participate in a range predicate and are further restricted to be one from each equivalence class at a time, hence the maximum dimension of simultaneous range predicates allowed in any legal SQL query against this database will be *d- dimensions*. Hence the total blowup in the number of table replication is $T * 2^d$.

For example, consider a test database shown in Figure 7.2. Suppose if the database has

Table1*(ColA, ColB, ColC)*
Table2*(ColD, ColE, ColF)*
Table3*(ColG, ColH, ColI)*

Figure 7.2: Test Database Schema

to support simultaneous range predicates over all the columns, then the total number of tables in the encrypted database is $3 * 2^9$.

Here $\{Col\} = \{ColA, ColB, ColC, ColD, ColE, ColF, ColG, ColH, ColI\}$, now suppose $\{Set_1\} = \{ColA,\ ColB,\ ColD,\ ColG,\ ColI\}$, then $\{Set_2\} = \{ColC,\ ColE,\ ColF,\ ColH\}$, further, say $\{Set_2\}$ can be partitioned into two equivalence classes, i.e. $\{Set_{21}\} = \{ColC,\ ColH\}$ and $\{Set_{22}\} = \{ColE,\ ColF\}$, then the total number of tables in the encrypted database is $3 * 2^2$. The encrypted database is shown in Figure 7.3.

For a real example, we can look at the TPCH benchmark [5] database and the recommended *22* benchmarking queries. Total number tables in TPCH database is *8* and total number of columns in the database is *61* but only *4* columns take part as the range predicate, i.e. $\{|Set_1|\} = 57$, and $\{|Set_2|\} = 4$. Namely, $\{Set_2\} = \{O\_ORDERDATE,\ L\_SHIPDATE,\ L\_RECEIPTDATE,\ L\_QUANTITY\}$. Further we can split $\{Set_2\}$ into two groups, viz. $\{Set_{21}\} = O\_ORDERDATE$ and $\{Set_{22}\} = L\_SHIPDATE,\ L\_RECEIPTDATE,\ L\_QUANTITY$. Thus total number of tables in the encrypted TPCH database is $8 * 2^2$, i.e. the encrypted database is *four* times the size of the plaintext database.

Table1_BS_BS*(ColA, ColB, ColC_BS)*
Table2_BS_BS*(ColD, ColE_BS, ColF_BS)*
Table3_BS_BS*(ColG, ColH_BS, ColI)*

Table1_BS_RS*(ColA, ColB, ColC_BS)*
Table2_BS_RS*(ColD, ColE_RS, ColF_RS)*
Table3_BS_RS*(ColG, ColH_BS, ColI)*

Table1_RS_BS*(ColA, ColB, ColC_RS)*
Table2_RS_BS*(ColD, ColE_BS, ColF_BS)*
Table3_RS_BS*(ColG, ColH_RS, ColI)*

Table1_RS_RS*(ColA, ColB, ColC_RS)*
Table2_RS_RS*(ColD, ColE_RS, ColF_RS)*
Table3_RS_RS*(ColG, ColH_RS, ColI)*

Figure 7.3: Encrypted Test Database Schema

2. **Enhancing the security of SPLIT:** Till now in SPLIT scheme we have produced two ciphertext for every plaintext data and kept them in two different tables corresponding to the BS and RS divisions, further since we are building upon *OPE and PPE* schemes, thus over half of the bits of any ciphertext one of the attack shown in Section 4 can be mounted. But on the other hand, if we increase the number of ciphertexts corresponding to every plaintext, for example, we can keep each level of the ciphertext tree into its own table instead of dividing them into two groups, i.e. if plaintext data is represented using *64-bits*, we will have *64* corresponding ciphertexts, and hence every plaintext table will have *64* corresponding ciphertext table. Thus in this case we can show that Honest-but-Active adversary will not be able to mount any of the attacks shown in Section 3.3.1 over any of the *64* ciphertext tables.

3. **Reducing the query execution time:** In SPLIT scheme we have seen that each plaintext table is replicated into multiple tables, further each plaintext query is also translated into multiple queries over ciphertext tables. But key observation is that,

   - Each ciphertext query access disjoint set of rows, and
   - Each ciphertext query access independent tables.

Hence, these queries can be executed in parallel and their outcome can be merged by the *SA* with simple addition (in case of count as query result) or union (in case of tuples as query result) operations.

7.4 Limitations of SPLIT

There are certain limitations of the SPLIT scheme, which are listed below:

1. The storage space required by SPLIT scheme is exponential in the number of equivalence classes of the database columns on which the range predicate queries are permitted. Further this is actually not a limitation since storage is cheap now-a-days.

2. Only batch updates are possible to ciphertext database encrypted using SPLIT encryption scheme. This requirement is essential to remove the correlation between the encrypted database tables.

7.5 Supporting More complicated Queries

Since SPLIT scheme does not return any false positive, hence it can be used along with other partial homomorphic encryption scheme to support a wide range of SQL queries. For example,

1. Along with the encrypted ciphertext corresponding to SPLIT scheme we can store the *paillier* encryption [16] of plaintext data, and can support SQL queries having *SUM* aggregate clause in there *SELECT* statement.
2. Similarly, we can store *ELGAMAL* encryption [9] of the data in parallel with the SPLIT scheme ciphertext and allow the *multiplication* of two columns in the database.

Chapter 8

Experimental Evaluation

To evaluate the performance of SPLIT scheme, we have conducted experiments on Google Cloud Platform running Ubuntu 14.04 LTS with 14.4GB memory and 2.5GHz Intel Xeon E5 v2 16 core processor. The database engine used was "IBM DB2 Enterprise Server Edition". We have evaluated the range predicate execution time on TPCH 1GB and TPCH 10GB benchmark databases. The metric for evaluation is the *ratio of the query execution time over the data encrypted using our scheme with respect to the execution time of the same query over plaintext database*. This metric will give the effective slowdown that is introduced in the query processing time due to our encryption framework.

Various experiments have been performed over both plaintext and encrypted database by considering the affect of following factors:

1. No index was created in both the plaintext and the encrypted database.
2. Index was created in both the databases over all the columns on which range predicate appears in the queries.
3. The executed queries have 100% selectivity.
4. The dimension of the query is also varied, upto 3 dimensional query has been evaluated.

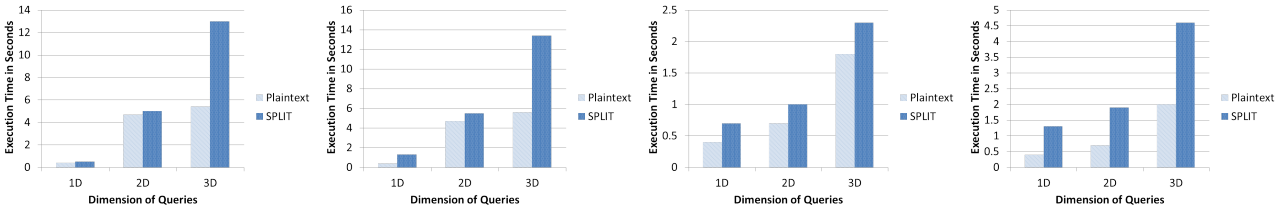
All the 1D queries use the SQL query template shown in Figure 8.1. Further 2D queries add the range predicate for column *L_Orderdate* and the 3D queries add the range predicates for columns *L_Orderdate and L_Orderkey* in the query template 8.1.

In all the executions the query *Q1* instantiates the query template with the range <*min_column_value, max_column_value*> for the respective columns, while the query *Q2* instantiates the query template with the range <*min_domain_value, max_domain_value*>. Further the subrange predicates are ordered from high selectivity to low selectivity in all the queries for the encrypted database.

SELECT *Count(*)*
FROM *Lineitem*
WHERE *L_Shipdate* $\geq$ "value1"
AND L_Shipdate $\leq$ "value2"

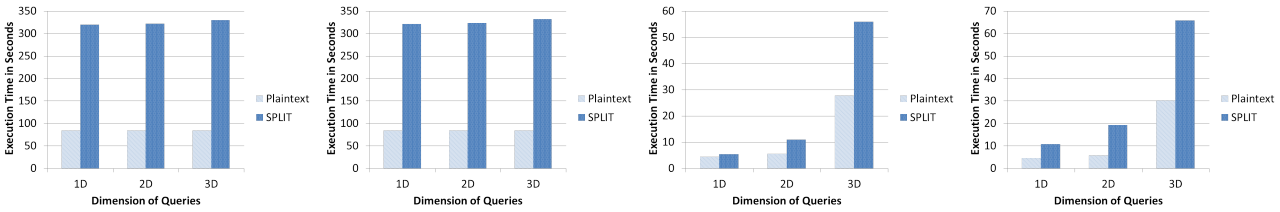
Figure 8.1: SQL query template for 1D range predicate query

The bar charts in Figure 8.2 shows the query execution time for 1GB TPCH database, while the Figure 8.3 shows the query execution time for 10GB TPCH database. In all the charts the *x-axis* shows the dimension of the query and *y-axis* shows the query execution time in seconds. Further the first two plots from the left in each figure shows the result of query execution without indexes, while the remaining two plots shows the query execution in the presence of indexes. Note that both single dimensional and multidimensional indexes were created.



(a) Q1 without index (b) Q2 without index (c) Q1 with index (d) Q2 with index

Figure 8.2: TPCH 1GB Experimental Results



(a) Q1 without index (b) Q2 without index (c) Q1 with index (d) Q2 with index

Figure 8.3: TPCH 10GB Experimental Results

As shown in Figures 8.2 and 8.3, all the queries for the encrypted database executed within the *3-times* slowdown of the plaintext query execution time.

Chapter 9

Conclusion and Future Work

In this thesis we have shown the presence of Honest-but-Active adversary in the DBaaS system and proposed an encryption scheme to securely and efficiently process range predicate over encrypted database in the presence of such adversaries. We have also shown that our scheme can efficiently handle equality predicates, group by and equi-join operators. Further, we have also implemented and evaluated our scheme on benchmark databases and the experiment results show that our scheme can efficiently support real time range predicate queries with strong security guarantees.

The future work includes constructing an encryption scheme to securely and efficiently handle all the SQL operators over the encrypted database against Honest-but-Active adversaries.

Bibliography

[1] http://aws.amazon.com/. 1

[2] http://azure.microsoft.com/en-in/services/sql-database/. 1

[3] http://www-01.ibm.com/software/data/cloudant/. 1

[4] http://en.wikipedia.org/wiki/Interpolation_search. 17

[5] http://www.tpc.org/tpch/. 44

[6] A. O'Neill A. Boldyreva, N. Chenette. Order-preserving encryption revisited: Improved security analysis and alternative solutions. In *CRYPTO*, 2011. 3, 7

[7] Y. Lee A. ONeill A. Boldyreva, N. Chenette. Order-preserving symmetric encryption. In *EUROCRYPT*, 2009. 3, 7, 16

[8] G. Tsudik B. Hore, S. Mehrotra. A privacy-preserving index for range queries. In *VLDB*, 2004. 3

[9] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *CRYPTO*, 1984. 46

[10] C. Li S. Mehrotra H. Hacigümüs, B. R. Iyer. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD*, 2002. 3

[11] Y. Lindell J. Katz. Introduction to modern cryptography. In *VLDB*. 6, 10

[12] E. R. Omiecinski J. Li. Efficiency and security trade-off in supporting range queries on encrypted databases. In *DBSec*, 2005. 3, 16

[13] M. H. Ammar A. B. Moon J. Xu, J. Fan. Prefix-preserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. In *In:*

Proceedings of the 10th IEEE International Conference on Network Protocols, 2002. 3, 7, 16, 34

[14] P. Rogaway T. Stegers M. Bellare, T. Ristenpart. Format-preserving encryption. In *Selected Areas in Cryptograpgy*, 2009. 10

[15] N. Ikram Mahboob, Athar. Transport layer security (tls)a network security protocol for e-commerce. In *PNEC Research Journal*, 2004. 15

[16] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999. 46

[17] N. Zeldovich R. A. Popa, F. H. Li. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy*, 2013. 3, 7, 16

[18] R. Srikant Y. Xu R. Agrawal, J. Kiernan. Order-preserving encryption for numeric data. In *SIGMOD*, 2004. 3, 7, 16

[19] A. L. Wang B. Bruhadeshwar R. Li, A. X. Liu. Fast range query processing with strong privacy protection for cloud computing. In *VLDB*, 2014. iv, 3, 18

[20] D. W. Cheung R. Li S. Yiu W. K. Wong, B Kao. Secure query processing with data interoperability in a cloud database environment. In *SIGMOD*, 2014. iv, 3, 4, 19, 20