Palestine Polytechnic University
Deanship of Graduate Studies and Scientific Research
Master of informatics

Towards a Hybrid Data Partitioning Technique for Secure Data Outsourcing

Sultan Khalil Badran

Supervisor	Co- Supervisor
Prof. Nabil Arman	Dr. Mousa Farajalla

Thesis submitted in partial fulfillment of requirements of the degree "Master in Informatics."

June 2021

The undersigned hereby certify that they have read, examined, and recommended to the Deanship of Graduate Studies and Scientific Research at Palestine Polytechnic University the approval of a thesis entitled: **Towards a Hybrid Data Partitioning Technique for Secure Data Outsourcing**. Submitted by **Sultan Khalil Badran** in partial fulfillment of the requirements for the degree of Master in Informatics.

Graduate Advisory Committee:

Prof. Nabil Arman (Supervisor), Palestine Polytechnic University
Signature:_____
Date:_____

Dr. Mousa Farajalla (Supervisor), Palestine Polytechnic University
Signature:_____
Date:_____

Dr. Mahmoud Al-Saheb, (Internal committee member), Palestine Polytechnic University
Signature:_____
Date:_____

Dr. Husam Suwad, (External committee member), Palestine Technical University – Kadoorie
Signature:_____
Date:_____

Thesis Approved

Dr. Murad Abu Sbeih

Dean of Graduate Studies and Scientific Research

Palestine Polytechnic University

Signature:_____
Date:_____

DECLARATION

I declare that the Master Thesis entitled "Towards a Hybrid Data Partitioning Technique for Secure Data Outsourcing" is my original work. Besides, I hereby certify that unless stated, all work contained within this thesis is my independent research and has not been submitted for the award of any other degree at any institution, except where due acknowledgment is made in the text.

Sultan Khalil Badran

Signature:

Date: 30/06/2021

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for the master's degree in Informatics at Palestine Polytechnic University, I agree that the library shall make it available to borrowers under the library's rules. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgment of the source is made.

Permission for extensive quotation from, reproduction, or publication of this thesis may be granted by my supervisor, or in his absence, by the Dean of Graduate Studies and Sciatic Research when, in the opinion of either, the proposed use of the material is for scholarly purposes.

Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Sultan Khalil Badran

Signature:

Date: 30/06/2021

DEDICATION

My wife, whose words of encouragement and inspiration are still ringing in my ears and who has always supported me and encouraged me to obtain my master's degree. To her for being there throughout the entire master's program. I also dedicate this work and give special thanks to my friends, who have been a constant source of support and encouragement during the challenges I faced. I also dedicate this thesis to my colleagues at the UNRWA ICT office, who have supported me throughout the process.

To all the people in my life who have touched my heart, I dedicate this research; I will always appreciate all the help and love I received.

ACKNOWLEDGEMENT

With great pleasure, I would like to acknowledge the support, assistance, and contribution provided by individuals since the beginning of this work, people who secured me with access, data, and information to all those who supported the completion of this thesis.

Foremost, I would like to express my sincere gratitude to my advisors Prof. Nabil Arman and Dr. Mousa Farajallah, for the continuous support for my master's study and research, for their patience, motivation, enthusiasm, and immense knowledge. Their guidance helped me throughout the research and during the writing process of this thesis. Thank you for your critical eye and constructive criticism, your optimistic attitude, and encouragement, your awareness of my situation has been invaluable during the whole process.

Besides, I would like to thank the examining committee, my colleagues in the master's program for all their help, support, access, and assistance that made my work possible.

الملخص

في ضوء التقدم الذي أحرزه قطاع التكنولوجيا في مجالات سرعة الإنترنت وتطوير الخدمات السحابية، بالإضافة إلى المزايا الأخرى التي توفرها السحابة مثل الموثوقية وسهولة الوصول من أي مكان وفي أي وقت، رأى معظم مالكي البيانات فرصة في الاستفادة من السحابة لتخزين البيانات. ومع ذلك، سيواجه مالكو البيانات تحديًا في مواجهتهم في الاستعانة بمصادر خارجية، وهو حماية البيانات الحساسة من التسرب. ووجد الباحثون أن تقسيم البيانات إلى أقسام بناءً على حساسية البيانات، يمكن أن يحمي البيانات من التسرب ويزيد من الأداء عن طريق تخزين القسم الذي يحتوي على بيانات حساسة في شكل مشفر.

تقدم هذه الرسالة طريقة هجينة لتقسيم البيانات لتأمين البيانات الحساسة وتحسين أداء الاستعلام. في هذا النهج الجديد، يتم تنفيذ تقسيم البيانات العامودي والأفقي باستخدام خادم Microsoft SQL لإنشاء علاقات مقسمة. يتم تطبيق مجموعة من القواعد المقترحة على عملية طلب الاستعلام؛ Query binning (QB) وبيانات وصفية متعلقة بالتقسيم (Metadata). وتم التحقق من صحة النهج المقترح باستخدام التجارب التي تنطوي على مجموعة من البيانات التي تم تجميعها من نتائج تنفيذ اجراء مخزن (stored procedure). وكانت نتائج النهج المقترح مرضية فيما يتعلق بتحقيق خصائص تعريف أمن البيانات: عدم الارتباط وعدم القدرة على التمييز. وكانت نتائج النهج المقترح مرضية. كان الاداء من حيث زمن تنفيذ الاستعلامات ممتازًا أكثر من ٥٠% في المتوسط، وهو أفضل من نتائج أداء PANDA.

Abstract

In light of the progress achieved by the technology sector in the areas of internet speed and cloud services development, and in addition to other advantages provided by the cloud such as reliability and easy access from anywhere and anytime, most data owners saw an opportunity in taking advantage of the cloud to store data. However, data owners will face a challenge in confronting them in outsourcing, protecting sensitive data from leakage. Researchers found that partitioning data into partitions, based on data sensitivity, could protect data from leakage and increase performance by storing the partition, which contains sensitive data in an encrypted form.

This thesis is presenting a hybrid data partitioning approach for secure sensitive data and improve query performance. In this novel approach, vertical and horizontal data partitioning are implemented using a Microsoft SQL server to generate divided partitioned relations. A group of proposed rules is applied to the query request process; query binning (QB) and Metadata of partitioning. The proposed approach is validated using experiments involving a collection of data evaluated by outcomes of stored procedure. The suggested approach results are satisfactory in achieving the properties of defining the data security: non-linkability and indistinguishability. The results of the proposed approach were satisfactory. The performance of query execution time was more excellent than 50% on average, which is better than the results of PANDA performance.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AES	Advanced encryption standard
AV	Adversarial view
CCA	Chosen-ciphertext attack
CLR	Common language runtime
COA	Cipher-text only attack
CPA	Chosen plaintext attack
DB	Database
DES	Data encryption standard
EDB	Encrypted database
EMR	Electronic medical records
KPA	Known-plaintext attack
PANDA	PArtioNed DAta
QB	Query binning
R	Relation
SQL	Structured query language

Chapter 1

Introduction

Generally, data outsourcing is vulnerable to different types of attacks. Secure and efficient retrieval of outsourced data is still an open challenge. Data sensitivity is one of the most critical security issues that need to be investigated. Generally, the data owner avoids data outsourcing (sensitive and non-sensitive data). Alternatively, they are outsourcing all data in encrypted form to protect sensitive data. In this study, the data partitioning techniques based on data sensitivity are considered to secure data outsourcing.

Accordingly, hybrid data partitioning techniques (vertically and horizontally) are addressed. However, to improve data security against attacks, data partitioning techniques (PANDA) are proposed [1]. These techniques divide a relation into a set of relations based on data sensitivity. While good partitioning techniques prevent data leakage against inference attack, these techniques must have specific characteristics to be considered secure, such as non-linkability and indistinguishability.

1.1 Problem Statement

The data partitioning of relations for outsourcing purposes, without a reliable securing technique, leads to difficult prevention of leakage of sensitive data. Protecting sensitive data that outsourced to a untrusted database is a difficult task for database owners. The main problem is how to find the approach that can prevent leakage of sensitive data by achieving the data security properties (Non-linkability and indistinguishability) and improving the performance of query requests.

1.2 Motivation

Protecting sensitive data in outsourcing Locations usually needs a securing approach to prevent data leakage and keep the query execution time

acceptable for database owners. This thesis's primary motivation is to develop an approach to prevent data leakage of sensitive data from a untrusted database and improve performance using Hybrid data partitioning and query binning (QB) [2].

1.3 Contribution and Proposed Approach

This thesis aims to improve query execution time, secure the sensitive data against inference attacks, and prevent data leakage while outsourcing data to a untrusted database for storage.

In this thesis, a hybrid data partitioning technique is used to divide relations into vertical and horizontal relations based on data sensitivity. Query binning (QB) techniques [2] are applied for securing sensitive data and enhancing the performance of query execution time.

1.4 Thesis Organization

The first chapter of this thesis discusses the problem description and motivation. The theories and fundamental concepts needed to understand the thesis are discussed in Chapter 2, while Chapter 3 contains the previous partitioning methods' literature review. The methodology used for hybrid partitioning data is presented and discussed in Chapter 4. Chapter 5 shows experiments demonstration and evaluation. The discussion and conclusions are presented in Chapter 6 with the potential future work.

Chapter 2

Background

Data outsourcing is vulnerable to different types of attacks. Secure and efficient retrieval of outsourced data is still an open challenge. Data sensitivity is one of the most critical security issues that need to be investigated.

In the past, the data owner chose one of two scenarios. First, avoided data outsourcing (sensitive and non-sensitive data). Second, encrypted all data stored in the cloud to protect the sensitive data. However, to improve data security against attacks, data partitioning techniques are proposed [1], [2], and [3]. These techniques divide a relation into a set of relations based on data sensitivity.

One of the significant concerns of data owners today is the leakage of data to external parties, mainly sensitive data. On the other hand, the data owners need to outsource the data in a particular situation to benefit from cloud features such as accessing the services or data from anywhere/anytime. The user pays only for used services or data. Increasing data reliability supports parallel and distributed computing.

The challenges in security in outsourcing data lead researchers to find solutions to improve the security aspect. One of these solutions is partitioning data, where the data in a relation is divided into more minor relations depending on selective attributes or specific tuple values such as sensitive data. Then store the new relation in different data centers that may be in different locations.

Generally, to get the partitioning categories, there are many different data partitioning approaches such as Data Sensitivity Partition, Frequency of Use Based Partition, and Space-Based Partition.

This research is motivated by developing information systems solutions such as health record systems, which have sensitive data. In [1], data outsourcing was used in several application contexts. Data partitioning can be used to partition data into sensitive and non-sensitive relations [1].

This thesis discusses the different types of partitioning techniques and different approaches that researchers have proposed to come up with the partitioning categories. It highlights how researchers used those partitioning techniques to increase security and protect sensitive data. A hybrid data partitioning approach is proposed to improve these techniques in terms of security and performance.

2.1 Data Outsourcing

Information outsourcing and dissemination services have recently seen widespread diffusion because of the dramatically growing costs of on-premises storage and management of a large set of sensitive data and the requests for both storage capacity and skilled administrative personnel [4]. Even though data outsourcing offers many advantages like reducing management costs, higher availability and more effective disaster protection than in-house operations provide, data are not under their owners' control.

A novel technology for data outsourcing has been developed because of the fast evolution of storage, processing, and communication technologies [4]. The technology is called cloud, and the cloud services providers are responsible for storage, management, and dissemination for any companies or users who give their data to them. Using data outsourcing decreases the cost of the software and the hardware and provides high availability. On the other hand, data are not under the data owner's control; neither companies nor users and attackers can violate their data. This will affect their confidentiality and integrity.

2.2 Data Partitioning

Nowadays, the size of data grows dramatically in many large-scale solutions, so the data is divided into partitions that can be managed and accessed separately [5]. Partitioning can be used for many purposes: it can be used to improve scalability and security and optimize performance [1] [5] [6]. The cheap data storage is usually used to partition and archive the older data [5].

However, the partitioning techniques must be chosen carefully to maximize the benefits and minimize adverse effects [5].

2.2.1 How to design data partitions?

There are three typical partitioning techniques for data: horizontal partitioning, vertical partitioning, and functional partitioning [1] [5] [6]. The selected technique of partitioning depends on two factors: the first is the purpose of partitioning, and the second is what the owner wants to improve in the system (security, performance, or both).

Horizontal partitioning technique: in this technique, the original relation, and all partitions have the same relation schema. Each partition contains a specific subset of the tuples [1] [5]. Figure 2.1 shows horizontal partitioning. In this example, Staff member's data is divided into two relations based on the department value. Each relation holds the tuples for a contiguous range of shard values (IT with HR in one relation and second relation include marketing department tuples), organized alphabetically by ID attribute.

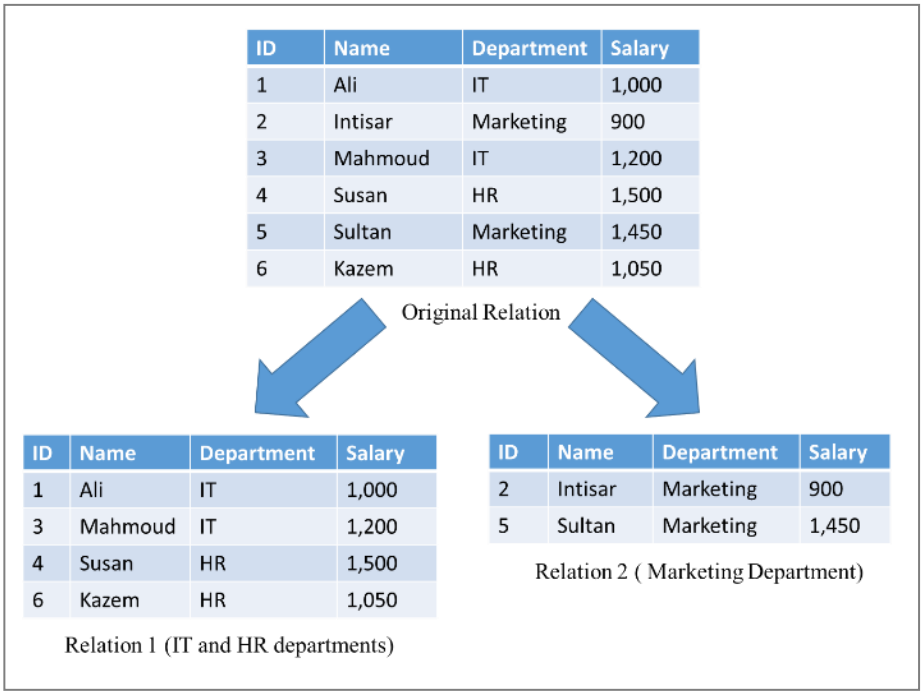


Figure 2.1. Horizontal partitioning of data

Vertical partitioning technique: in this technique, each partition holds a subset of relation attributes. The attributes are divided according to their pattern of use or property. For example, Sensitive attributes are placed in a relation, and non-sensitive attributes are placed in another relation [1] [7]. Figure 2.2 shows an example of vertical partitioning. In this example, different attributes are stored in different relations. One relation holds tuples that are sensitive data, including ID and Salary. Another relation holds non-sensitive data (ID, Name, and Department) [5]. In the above example, the ID belongs to the divided relations to re-join the relations to produce the original relation.

Functional partitioning: in this technique, data is divided into different relations according to the system's context. For example, in an HR system, the Staff member's data is stored in one relation and payroll data in another. Figure2.3 shows how the data is partitioned into two relations, where each partition is stored in different locations.

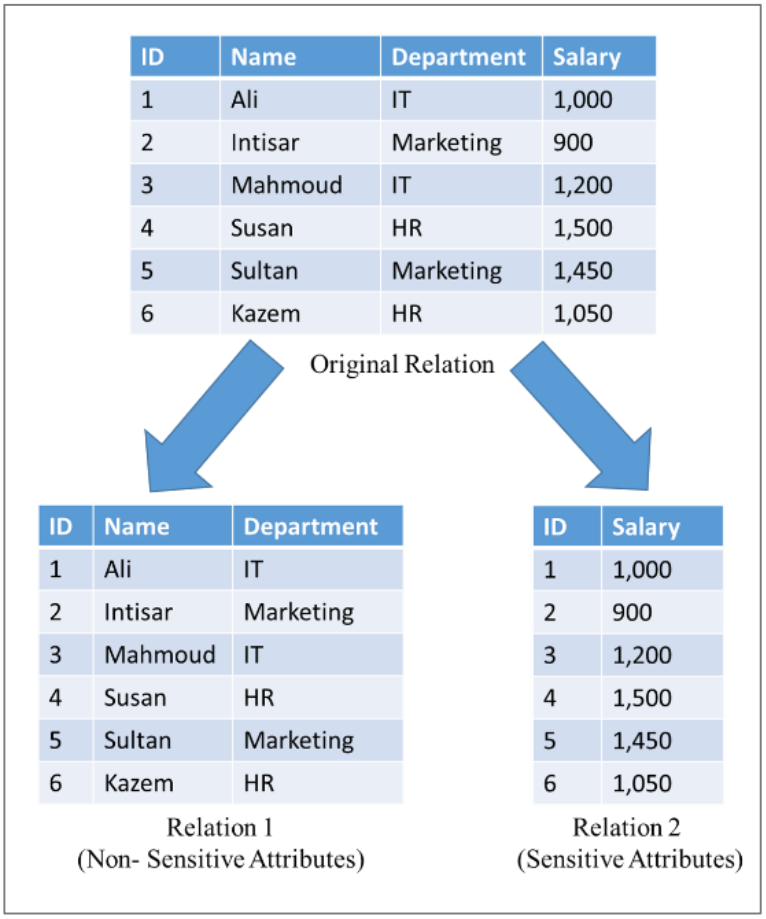


Figure 2.2. Vertically partitioning data

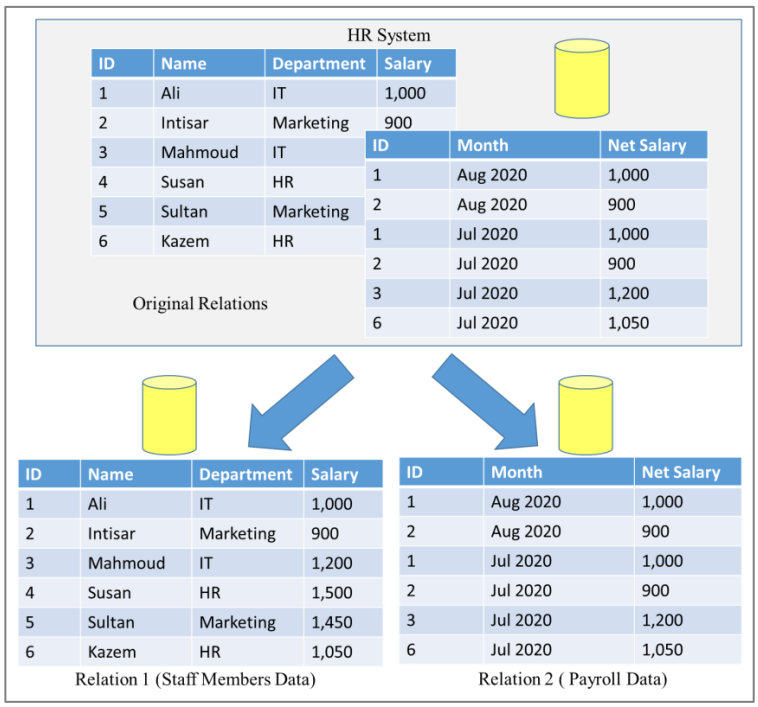


Figure2.3. Functionally partitioning data

These techniques can be combined, and it recommended that all of them be considered when designing a partitioning scheme. For example, dividing data into horizontal or functional partitioning and then use vertical partitioning to subdivide data in each horizontal or functional partitioning.

2.2.2 Why Using Data Partitioning?

Improve scalability. The single database system has a limitation when considering the used hardware resources. On the other hand, using the data partitioning techniques leads to optimal use of resources. Due to duplication of hardware resources, the single database is divided and distributed over more than one data center/location.

Improve performance. Using data partitioning saves time during the selection of query transactions. Data partitioning could make query processing more efficient. Query transactions that affect more than one partition are executed in parallel.

Improve security. Data partitioning is used to enhance security, such as dividing data into different databases or locations based on sensitivity and applying different security policies to sensitive data.

Improve availability. Partitioning data across multiple data centers avoids a single point of failure. If one center failed, only the data in that center is unavailable, but reduce availability for the system. Transactions on other data centers can still operate.

2.2.3 What are the Types of Partitioning?

There are many approaches to perform data partitioning:

- Data Sensitivity Partition

In this approach, the partitioned data is divided into a set of relations based on the data sensitivity. For example, suppose a relation includes a sensitive attribute such as a PIN code to access the account profile. In this case, a new relation is created for the PIN codes attribute and anoth-

er relation for the rest of the attributes. Keeping a link between tuples in the new relations created is needed to perform a joining operation to retrieve the original relation [1] [2] [8].

- Frequency of use Based Partition

In this partitioning approach, the transaction log file, and precisely the WHERE clause conditions in each SQL SELECT statement, is monitored. A statistical matrix for selected attributes used in the WHERE clause conditions is produced. This matrix includes the frequency of use of the WHERE clause's attribute values such as (WHERE SALARY Between 1000 and 2000). After the matrix is created, some cleaning to exclude the smaller frequency counts is performed, and then a partition is created for each frequency in the final matrix [9].

- Space-Based Partition

This partitioning approach starts by creating a statistical table that includes each value's frequency in a specific attribute and then continues with dividing data according to bucket size. The partitioned relations created are based on comparing the value frequency against the size of the bucket. For example, if the frequency of the first value in the statistical table equals the bucket size, a new partition is created and with all tuples that belong to the first value. If the sum of the second and third values' frequency is equal, a new partition is created with all tuples that belong to the first and third values. Generally, the new partitions can contain tuples from one or more values, and tuples may belong to values distributed to one or more partitions based on the frequency size and bucket size [9].

- Mondrian or Bisection Tree-Based Partition

In this approach, an attribute is selected to be used to partition data, then the tuples are ordered by the selected attribute. After that, the median is calculated, and the two relations are created. The first relation includes

the tuples in the right median, and the second relation includes the tuples in the left median. The approach is repeated for each new relation until the partition satisfies certain conditions [9].

- Histogram Based Partition

This approach is used to display statistical information. An Equi-width technique is one of its types. This technique divides the values into stacks of equal width. Each stack presents a new relation. This method simply subtracts the minimum value from the attribute's maximum value and is used to divide the results by the number of stacks [9].

2.3 Data Partitioning Security

The data security level should provide non-linkability and cipher-text indistinguishability to protect data [1] [10]. Figure 2.4 show the two properties that needed to be involved in the encryption algorithms:

- Non-linkability: the adversary does not learn the relationship between any encrypted and plaintext value.
- Cipher-text indistinguishability: the adversary does not learn any relationship between encrypted values.

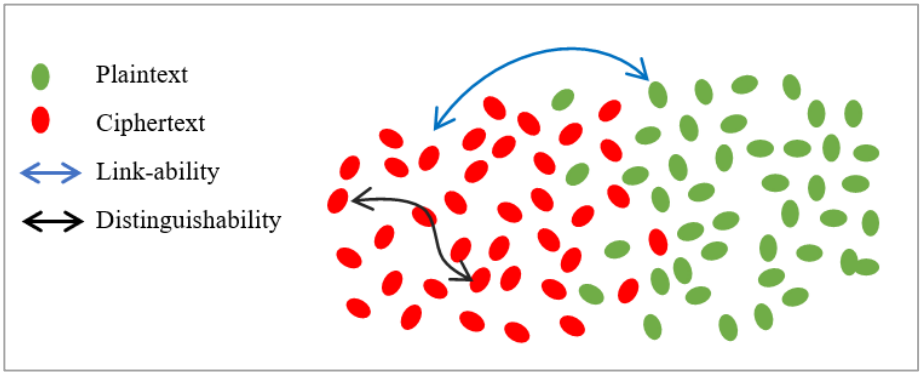


Figure 2.4 Data partitioning security [1]

2.4 Query Inference Attacks

The adversary uses query inference attacks to recover information about the data or queries by linking encrypted information with openly available information [11]. The most well-known types of inference attacks:

- Frequency attack, i.e., an adversary can conclude how many tuples have a similar value [11].
- The workload-skew attack, i.e., an adversary, knowing frequent selection queries by watching many queries, can guess which encrypted tuples potentially meet the frequent section in selection queries [1].

2.5 Data Encryption

DB owners are concerned about their data when data are hosted and managed in Untrusted databases. They tend to encrypt the data before sending it to the untrusted database to prevent the provider from accessing data hosted on their machines. Data encryption can be performed by using symmetric or asymmetric encryption schemes. Furthermore, data encryption can be performed based on a deterministic or non-deterministic framework [12]. The main difference between the two frameworks is evident when encrypting identical values. If two identical values are encrypted, the deterministic approach encrypts the values, and the output is two identical encrypted cipher-texts. On the other hand, the non-deterministic approach output is two non-identical encrypted cipher-texts.

Chapter 3

Literature Review

In this chapter, a summary of the main studies related to our research is being presented. The literature studies about data partitioning can be divided into two topics; the partitioning approach and the partitioning design. As stated in many related studies, the first topic usually depends on how relations are divided, For example, attribute sensitivity and frequency of use Based Partition. The second topic depends on the partitioned relation layout and the inference attack, which is the most crucial topic in protecting sensitive data.

Because of the importance of data security, especially these days, most companies and organizations are turning to cloud solutions. However, the companies and organizations are concerned about their data- stored in the cloud-from leakage and loss of privacy. Therefore, researchers have developed different techniques to improve the database's security and the queries' performance. In the literature, the security of sensitive data was achieved using data partitioning techniques.

Several researchers have used encryption to secure data, such as Searchable Encryption in [13]. Several data partitioning techniques have recently used encryption to secure data in database queries and avoid data leakage. Data partitioning was used to improve the data security and the performance in [1], [2], [3], [7], [8], and [9].

The authors in [1], [2] and [3] have used data partitioning (vertically and horizontally) based on data sensitivity for classifying tuples into sensitive and non-sensitive tuples and attributes. The proposed solution assumes that the entire database may not contain sensitive tuples, and non-sensitive tuples can be exploited to handle the limitations of encryption-based approaches. They proposed a new definition to secure data based

on data sensitivity and improve the security against inference attacks such as frequency-count and workload-skew attacks by proposing an approach called query binning (QB) to process joint queries through the sensitive and non-sensitive tuples. Furthermore, the data partitioning and QB enhance the performance, improve security, and prevent data leakage. However, the used techniques are unable to use more than one value for criteria in queries.

The authors in [7] have proposed asymmetric encryption techniques to protect data privacy during privacy-preserving data mining (PPDM). The method is implemented using vertically partitioned relations based on data sensitivity. The technique used different encryption algorithms to encrypt the sensitive attributes in different relations at the same time. The algorithms were applied to three data sets that were prepared as MS Excel files with different sizes. In addition, four symmetric encryption techniques were implemented using sensitive attributes only: AES, DES, Rijndael, and RC2. The proposed techniques' privacy results were better than using just one encryption algorithm on each partitioned relation.

In [8], the authors presented an algorithm to prevent the leakage of sensitive data or loss of privacy from the relations in a database stored in the cloud. They have developed a model intending to offer secure data management capability in cloud databases. That model used two approaches to partition data: the first uses attribute relationship to divide database relations, and the second uses the data sensitivity to divide the relations. They proposed a model to deal with the partitioning model results, which are used to store partitioned relations in the data center. The model can be stored in one cloud data center, which provides good performance and security, or store the partitioned relations in different cloud data centers and locations that improve the security. They distribute data into different data centers in different locations to improve security and do not consider performance.

In [9], the authors presented Data Partitioning Methods to Process Queries on Encrypted Databases on the cloud. They explored a technique to improve the query processing performance and at the same time keep the database relations secure on a cloud by encrypting relations from any leakage or attack. They point to protecting data from any leakage or attack by designing encrypted databases that process the SQL queries on encrypted relations. This solution's main idea is handling the query on encrypted data stored on the cloud without a need to decrypt it. The result of the query is decrypted at the client side. Furthermore, four different techniques are developed to index and partition the data as follows:

1. Frequency of use Based Partition.
2. Space-Based Partition.
3. Mondrian or Bisection Tree-Based Partition.
4. Histogram Based Partition.

They compared the efficiency of the first three techniques with the Histogram-Based partition. The indexes and partitions are used to process the query and select part of the cloud data. The 'indexes' data can be stored on the cloud or on-premises server with the encrypted database relation. This leads to a reduction in the overall processing time, which contains the data transfer time from the cloud to the query requester site, as well as the data decoding and processing time for the requester. Moreover, these techniques are used to compare encrypted relation and unencrypted relation. The comparison results including the running times for retrieving a different number of tuples in relation to different tuple sizes. They found that the encrypted relation with the Frequency-of-Use-Based partition and the encrypted relation with Bisection-Tree-Based partition is the most efficient partitions. In addition, they explain a particular issue that shows how combining Frequency-of-Use-Based and Bisection-Tree-Based enhances performance through data partitioning methods.

In addition to Data Partitioning techniques, researchers have explored Inference Attacks to understand how adversaries attack the encrypted databases and how the leakage of sensitive information occurred [11].

The authors of [11] have studied the inference attacks against encrypted database (EDB) systems based on property-preserving encryption (PPE) scheme, which is CryptDB. They presented a series of attacks that discover the plaintext from cipher-text encrypted using deterministic encryption (DTE) and order-preserving encryption (OPE) encrypted database attributes. They studied four different attacks. Two are well-known attacks: frequency analysis attack and sorting attack. Moreover, two are new attacks: ℓ_p-optimization attack and Cumulative attack. They conducted experiments in electronic medical records (EMR) to evaluate these attacks. The EMR data belong to actual patients from 200 US hospitals. They assume that the adversary has access to EDB in a steady state to perform the attack experiments. Besides, the adversary has access to some auxiliary information about the system and/or the data, such as application details, public statistics, and prior versions. They simulate the EMR-EDB to develop the experiment scenario to try the considered attacks and then analyze them. Their experimental results illustrate that the sensitive information can be recovered when the adversary has background knowledge about EDB data and properties. In numbers, their attacks recovered more than 80% of the patient records out of 95% of the hospitals when using OPE-encryption to encrypt attributes (e.g., age and disease severity). On the other hand, when using DTE in specific encrypted attributes (e.g., sex, race, and mortality risk), the attacks recovered more than 60% of the patient records out of 60% of the hospitals.

Chapter 4

Research Approach

In this chapter, the proposed approach presents in detail the process of the Hybrid Data Partitioning Model for partitioning relations (presented in section 4.1). The new divided relations utilize the Hybrid Data Partitioning Technique [14]. Once this is performed, a novel technique called Query Binning (QB) is implemented in [1]. The QB is explained how increase the security in section 0 and How prevent data leakage against inference attacks presented in section 4.3.

4.1 Hybrid Data Partitioning Model

In our research, we consider the following two entities in our model, Trusted Database and Untrusted Database, shown in Figure 4.1 and descript below:

(1) Trusted Database on-premises contains the whole data in plaintext format and executes queries, and sends query requests to untrusted DB on the cloud. We assume that a relation R has attributes, say A_1, A_2, A_3 . . . , A_n, containing all sensitive and non-sensitive values in tuples t_1, t_2, t_3…, t_m. According to values stored with specific attributes, the DB owner determines which attributes are sensitive or not and lays rules that determine when the tuple is sensitive.

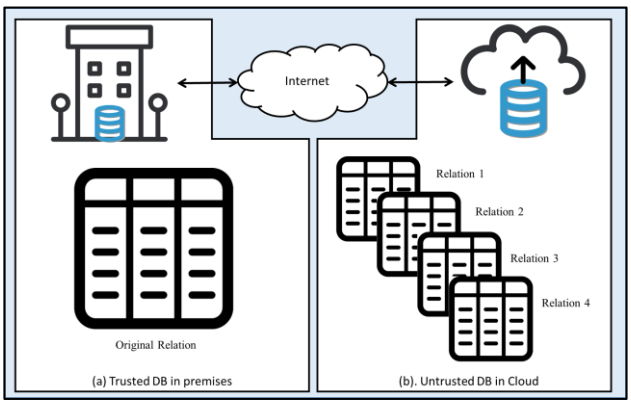


Figure 4.1 The context of the proposed approach

The DB owner divides relation R into several relations based on the tuples' data sensitivity using a hybrid technique that divides the original tuple into three tuples at max; each divided tuple is stored in a different relation. The first part contains the values stored in the attributes marked as sensitive. The second part contains the values stored in the attributes marked as non-Sensitive, and the rest of the values may either include sensitive or non-sensitive values. This means that the third tuple may either be considered sensitive or non-sensitive.

The DB owner outsources the relations that contain non-sensitive data to a cloud in plaintext form. The tuples of the relations that contain sensitive data are encrypted using any existing non-deterministic encryption mechanism before outsourcing to the same cloud.

In our model setting, the DB owner must store metadata such as a mapping relation that stores the original tuple ID with the new tuple IDs in each of the divided relations. The Metadata is used for appropriate query formulation using the Query Binning (QB) proposed in [2] and explained in section 0.

(2) Untrusted Database on Cloud that hosts the databases contains the partitioned relations, executes queries, and provides answers to the trusted DB stored on-premises.

To explain query execution in our model, let us assume a query σ over the relation R and p is a preposition, denoted by $\sigma_p(R)$. The query is executed on trusted DB with no limitation on the number of attributes in the WHERE condition clause. The results of the query include four attributes:

i. Tuple ID, the original ID for each tuple.

ii. ID_E, tuple ID representing the primary tuple key in new relation for sensitive data (R_E).

iii. ID_P, tuple ID representing the primary tuple key in new relation for non-sensitive data (R_P).

iv. ID_{P_E}, tuple ID representing the primary tuple key stored in relation R_{P_E} in plaintext or relation R_{E_P} in encrypted form.

After that, the query process splits the execution of σp(R) into four subqueries. As presented in Equation 4.1, each subquery is sent to an untrusted DB to be executed. Then the results of subqueries are returned to the Trusted DB. Then inside the trusted DB, there are two subqueries ((R_{P_E} and R_{E_P})) that have the same scheme, for which a union operation is performed. Then join operations are performed to join the union result with R_P and R_E. In particular, the query σ on a relation R is executed, as:

$$\sigma_p(R) = \sigma_p(R_E) \bowtie \sigma_p(R_P) \bowtie (\sigma_p(R_{P\_E}) U \sigma_p(R_{E\_P}))$$

Equation 4.1

Example: Let us illustrate the hybrid data partitioning model through the following example:

	Attributes No					
	a_1	a_2	a_3	a_4	a_5	a_6
Tuple No	**ID**	**Name**	**Department**	**Salary**	**Location**	**Password**
t_1	1	Ali	IT	1,000	Jerusalem	********
t_2	2	Intisar	Marketing	900	Jerusalem	********
t_3	3	Mahmoud	IT	1,200	Hebron	********
t_4	4	Susan	Marketing	1,500	Ramallah	********
t_5	5	Sultan	Marketing	1,450	Bethlehem	********
t_6	6	Kazem	HR	1,050	Nablus	********
t_7	7	Alaa	Marketing	1,460	Bethlehem	********
t_8	8	Ahmad	HR	980	Nablus	********

Table 4.1 Employee relation

Table 4.1 considers an Employee relation R. Note that the notation a_i ($1 \leq i \leq 6$) is an attribute in the relation; it indicates the i^{th} attribute. In this relation, note that the notation t_j ($1 \leq j \leq 8$) of the relation; we used this to indicate the j^{th} tuple. In this relation, the DB owner considers that the password attribute values are not outsourced data, and the salary attribute values are sensitive. Moreover, all values in department attribute that meet Department = "Marketing" are sensitive. In such a case, and after applying the Hybrid partitioning calculation, the metadata are generated as shown in Table 4.2. Metadata includes four attributes as described before. It is worth mentioning that the data type of ID_E, ID_P, and ID_{P_E} attributes is a unique identifier data type. This data type generates unique key values that contain 36 characters.

Tuple No	**Tuple ID**	**ID_E**	**ID_P**	**ID_{P_E}**
t_1	1	848CC055...A	43AACEF7...P	F0D9C43C...R
t_2	2	DF8BC1A8...C	2CF79E45...O	485F36AB...J
t_3	3	03E47A30...E	1AC4E44F...Y	CAF5A05C...Q
t_4	4	5E1A2955...A	990D4BF7...I	17EDA383...8
t_5	5	EF036F92...F	BA921C43...G	F1859688...Y
t_6	6	CB1CCD4D...K	4276A931...K	A03E7373...D
t_7	7	116DB16E...H	10E7C843...U	14C0E88B...X
t_8	8	F2220062...P	892285C5...D	05B4FA48...Z

Table 4.2 Metadata table for relation R

The Employee relation may be stored on the cloud as:

1. Relation 1contains all sensitive values in Salary's attribute and stores values in encrypted form, as shown in Table 4.3.

Attributes No	ID_E	a_4
Tuple No	**ID**	**Salary**
t1	848CC055...A	E(1000)

t2	DF8BC1A8...C	E(900)
t3	03E47A30...E	E(1200)
t4	5E1A2955...A	E(1500)
t5	EF036F92...F	E(1450)
t6	CB1CCD4D...K	E(1050)
t7	116DB16E...H	E(1460)
t8	F2220062...P	E(980)

Table 4.3 Relation 1

2. Relation 2 contain all non-sensitive values in all attributes marked as non-sensitive attributes and store values in plaintext form, as shown in Table 4.4.

Attributes No	ID_P	a_2	a_5
Tuple No	**ID**	**Name**	**Location**
t1	43AACEF7...P	Ali	Jerusalem
t2	2CF79E45...O	Intisar	Jerusalem
t3	1AC4E44F...Y	Mahmoud	Hebron
t4	990D4BF7...I	Susan	Ramallah
t5	BA921C43...G	Sultan	Bethlehem
t6	4276A931...K	Kazem	Nablus
t7	10E7C843...U	Alaa	Bethlehem
t8	892285C5...D	Ahmad	Nablus

Table 4.4 Relation 2

3. Relation 3 contain all tuples that the attributes include sensitive values. In the example, all sensitive values in Department attribute, where Department = Marketing", are stored in encrypted form, as shown in Table 4.5.

Attributes No	ID_{P_E}	a_3
Tuple No	**ID**	**Department**
t_2	CAF5A05C...Q	E(Marketing)
t_4	17EDA383...8	E(Marketing)
t_5	A03E7373...D	E(Marketing)
t_7	05B4FA48...Z	E(Marketing)

Table 4.5 Relation 3

4. Relation 4 contain all sensitive values in Name and Location attributes, where Department ="Marketing" and saved in plaintext form, as shown in Table 4.6.

Attributes No	ID_{P_E}	A_3
Tuple No	**ID**	**Name**
t_1	F0D9C43C...R	IT
t_3	485F36AB...J	IT
t_6	F1859688...Y	HR
t_8	14C0E88B...X	HR

Table 4.6 Relation 4

Hence, the sensitive data stored in Relation 1 and Relation 3 (Table 4.3 and Table 4.5) are encrypted before being outsourced to an untrusted database. In contrast, Relation 2 and Relation 4 (Table 4.4 and Table 4.6), including only non-sensitive data, are outsourced in plaintext form. The partitioning is executed on the tuple level, a trigger is fired and run the partitioning code in each time a tuple insertion, modification, or deletion operation occurs as mentioned in Algorithm 1 shown in Figure 4.2.

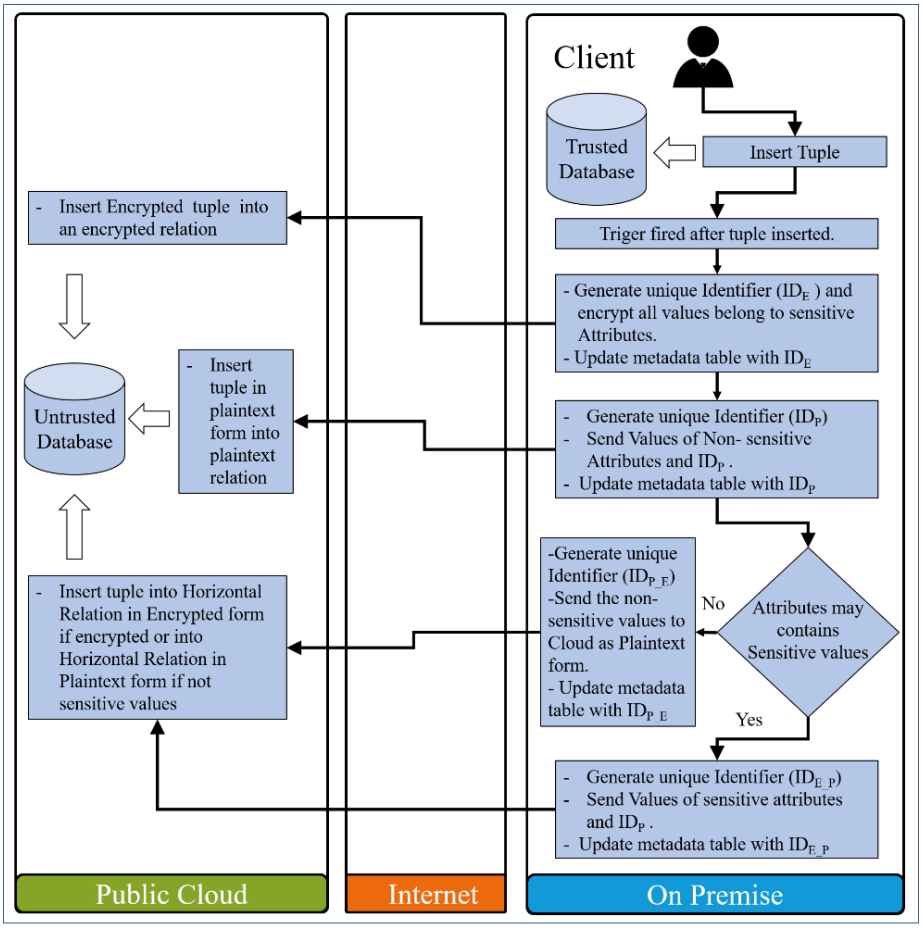


Figure 4.2 Insert tuple trigger and the partitioning computation

The example shows us the Hybrid approach used as the proposed solution in this thesis. The Relations in Table 4.3 and Table 4.4 are divided vertically. The first relation contains all values in an encrypted form that belong to sensitive attributes (all values are sensitive in attributes). The second relation contains values in plaintext that belong to attributes with all non-sensitive values. On the other hand, the relations in Table 4.5 and Table 4.6 are divided horizontally. The first relation contains sensitive values in encrypted form (we decrypt all values in tuple if one value is sensitive at least). The second relation contains the plaintext values that belong to the rest of the attributes and does not have any sensitive values.

Algorithm 1 Insert Tuple

Inputs: t: inserted/updated tuple.
Variable: Metadata: table to store metadata about t.

a[] list of attributes. v[] sensitive values list. ID_E, ID_P,ID_{P_E}

```
1   Function InsertTuple (t ) begin
2       a[]←Relation attributes v[]←Relation attributes Sensitive Values
3       ID_E ←Generate Unique Identifier key
4       t_E ← ID_E
5       t_E ←Encrypt all values store in attributes marked as sensitive in t
6       Send t_E to R_E in cloud
7       ID_p ←Generate Unique Identifier key
8       t_P ← ID_P
9       t_P ←all values store in attributes marked as non-sensitive in t
10      Sendt_P to R_P in cloud
11      ID_Temp←Generate Unique Identifier key
12      If the rest of the values in t marked as sensitive values
13          t_E_P ← ID_Temp
14          t_E_P ←Encrypt all values marked as sensitive in t
15          Send t_E_P to R_E_P in the cloud
16      Else
17          t_P_E ←ID_Temp
18          T_P_E ←all values marked as non-sensitive in t
19          Send t_P_Eto R_P_E in the cloud
20      Metadata ←t.ID, ID_P, ID_E,ID_Temp
21      Return
```

To continue with example 1, consider a query σ: SELECT Name, Department from Employee where location = N Jerusalem' . In the trusted DB, the query $\sigma_{Location = N'Jerusalem}(R)$ is executed on relation R, then as shown in Algorithm 2, the results of the query are joined in Metadata relation, after that, they produce four queries that are sent and executed in Untrusted DB below:

i) $\sigma_{IDe \text{ in (query results)}}(R_E)$ executes on R_E relation.

ii) $\sigma_{IDp \text{ in (query results)}}(R_P)$ executes on R_P Relation.

iii) $\sigma_{IDp\_e \text{ in (query results)}}(R_{P\_E})$ executes on R_{P_E} Relation.

iv) And the last query $\sigma_{IDe\_p \text{ in (query results)}}(R_{E\_P})$ executes on R_{E_P} Relation.

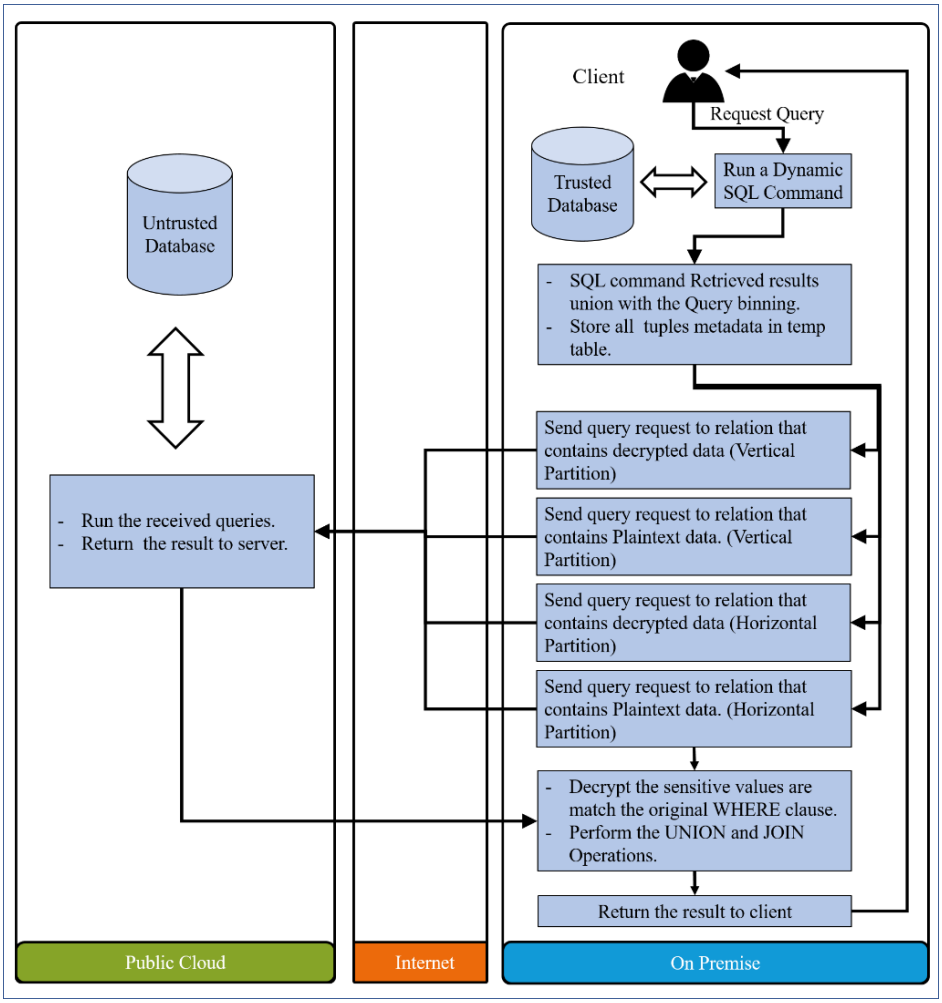


Figure 4.3 Query request

The query result is sent back to trusted DB, and SQL operation is performed as presented in Equation 4.1. Firstly, send and execute the queries. The UNION operation is performed between σ (R_{P_E}) and σ (R_{E_P}). Then the output is used in the joining operation of σ (R_P) and σ (R_E). The query retrieve tuples t_1 and t_2. Figure 4.3 and Algorithm 2 show how the query request process works.

It is worth mentioning that the partitioning computation occurs during the insertion of tuples into R relation. That saves time instead of doing the partitioning computation of the whole data at once, as in [2]. Figure 4.2 shows the steps of the trigger while inserting a tuple in R relation.

Algorithm 2 Query Request

Inputs: SQLstr: Select query statement, Metadata: table to store metadata about tuples

Outputs: Results: Query results

Variable: T_R: temporary data table to store metadata about SQL results, Result1 temporary relation

1 Function run_SQL (SQLstr) begin

2 T_R←Execute(SQLstr)⋈ Metadata

3 Result1←Execute_on_Cloud(R_{E_P}, Domain(T_R.ID_{E_P}))∪ Execute_on_Cloud(R_{P_E}, Domain(T_R.ID_{P_E}))

4 Result1← Result1⋈ Execute_on_Cloud(R_P, Domain(T_R.ID_P))

5 Result1← Result1⋈ Execute_on_Cloud(R_E,Domain(T_R.ID_E))

6 Query results←retrieve tuple from Result1 match the original where clause

7 Return Query results

4.2 Query Binning Technique

In order to avoid the inference attacks in the partitioned computation, we need a helpful security definition. Later we will discuss the formal definition of partitioned data security in section 2.3, but first, we will provide a solution to avoid inference attacks. We use the Query Binning (QB) technique in [2] and describe how it works to link tuples.

QB involves two steps: first, the creation of the query bins. The second step consists of rewriting the query based on the binning.

We can say that the QB in the base case is a one-to-one relationship between one sensitive tuple and one non-sensitive tuple. Accordingly, this means that both tuples cannot be sensitive and non-sensitive.

Before describing QB, we need to present concept of approximate square factors of a number used to create the bins.

As defined in [2], "two numbers, say x and y, are approximately square factors of number n, where $n > 0$, if $x \times y = n$, and x and y are equal or close to each other. That difference between x and y is less than the difference between any two factors, say x' and y', of n such that $x' \times y' = n$".

In our research, the QB uses tuples stored in partitions divided horizontally to create the binning.

Continue with our example in section 4.1, to calculate the approximately square factors, let us consider that n = number of non-sensitive tuple = 4 tuples, according to the definition of Approximately square factors, x = 2 and y = 2, this satisfies the definition of the Approximately square factors. Now we have two sensitive bins and two non-sensitive bins.

After creating the bins, we need to fill them with tuples using the Algorithm described in [2] and shown in Appendix 2 that links between sen-

sitive tuples and non-sensitive tuples. The results of this operation are shown in Figure 4.4.

In the example shown below, we use the location attribute in WHERE clause and the same data in example1 (Table 4.1), and we retrieve tuples as follows:

i) Retrieve tuples corresponding to employees who work in Location =’Jerusalem’.

ii) Retrieve tuples corresponding to employees who work in Location =’Hebron’,

iii) And retrieve tuples corresponding to employees who work in Location =’Bethlehem’.

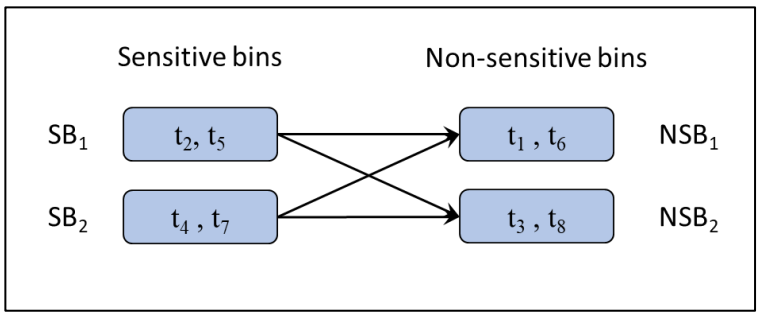


Figure 4.4 QB for four sensitive and four non-sensitive tuples.

Adversarial view

We assume that the adversary has access to Untrusted DB and to the transactions log file, which means that when answering a query, the adversary knows the retrieved encrypted tuples and the complete information of the retrieved non-sensitive tuples. This information is known to the adversary throw the adversarial view. Table 4.7 present the retrieved tuples without applying the QB.

Query value	Returned tuples/Adversarial view			
	Relation 1	Relation 2	Relation 3	Relation 4
Jerusalem	$E(t_1)$, $E(t_2)$	t_2	$E(t_2)$	t_1
Hebron	$E(t_3)$	t_3	Null	t_3
Bethlehem	$E(t_5)$, $E(t_7)$	t_5 , t_7	$E(t_5)$, $E(t_7)$	null

Table 4.7 Queries results, without apply QB

To apply the QB bins technique, we need to modify the query request performed, so Algorithm 3 shows how to query request work with QB.

Algorithm 3 Query Request with QB

Inputs: SQLstr: Select query statement, Metadata: table to store metadata about tuples
Outputs: Results: Query results
Variable: T_R_B: temporary data table with Bins, T_R_B: temporary data table without Bins, T_R: temporary data table to store metadata about SQL results, Result1 temporary relation

1 Function run_SQL (SQLstr) begin
2 T_R_W← Execute(SQLstr)
T_R_B← Retrieve_Bins(T_R_W)
T_R←T_R_B⋈ Metadata
3 Result1←Execute_on_Cloud(R_{E_P}, Domain($T\_R.ID_{E\_P)}$)∪ Execute_on_Cloud(R_{P_E}, Domain($T\_R.ID_{P\_E}$))
4 Result1← Result1⋈ Execute_on_Cloud(R_P, Domain($T\_R.ID_P$))
5 Result1← Result1⋈ Execute_on_Cloud(R_E, Domain($T\_R.ID_E$))
6 Query results← retrieve tuple from Result1 match the original where clause
7 Return Query results

To know how the QB affects the results of the query request process, the adversarial view is changed after Algorithm 3 is applied; Table 4.8 present the query request result for an adversary using the QB technique. In this example, we will use the same conditions in the previous example after applying the QB technique (show Figure 4.4).

Query value	Returned tuples/Adversarial view			
	Relation 1	Relation 2	Relation 3	Relation 4
Jerusalem	$E(t_1)$,$E(t_2)$, $E(t_5)$, $E(t_6)$	t_1,t_2,t_5,t_6	$E(t_2)$, $E(t_5)$	t_1,t_6
Hebron	$E(t_2)$,$E(t_3)$, $E(t_5)$, $E(t_8)$	t_2, t_3,t_5,t_8	$E(t_2)$, $E(t_5)$	t_3,t_8
Bethlehem	E(t2),E(t4), E(t5), E(t7), E(t3), E(t8)	t_2, t_3, t_4,t_5, t_7, t_8	$E(t_2)$,$E(t_4)$, $E(t_5)$, $E(t_7)$	t_3, t_8

Table 4.8 Query result using QB

4.3 Data Partitioning Security

Using non-deterministic encryption for sensitive data achieves the property of cipher-text indistinguishability (i.e., an adversary cannot distinguish between two cipher-texts) [2]. Hence, the same plaintext values have two different cipher-text values. Furthermore, the non-linkability is achieved in two positions, first in the untrusted database by using IDs for each tuple stored in each divided relation that is different from the original IDs in the private database. Second, in the query request process, this is achieved by using query binning (QB). Figure 4.5 Show the security context.

Adversarial view: We want to explain the adversarial view that assumes that the adversary has full access to Untrusted DB and the transactions log file. This means that when answering a query, the adversary can retrieve the all-Select SQL statements, and re-execute these statements and retrieve the encrypted tuples and the complete information of the retrieved non-sensitive tuples. The adversarial view lets the adversary knows this information. Moreover, the adversary has no access to Trusted DB.

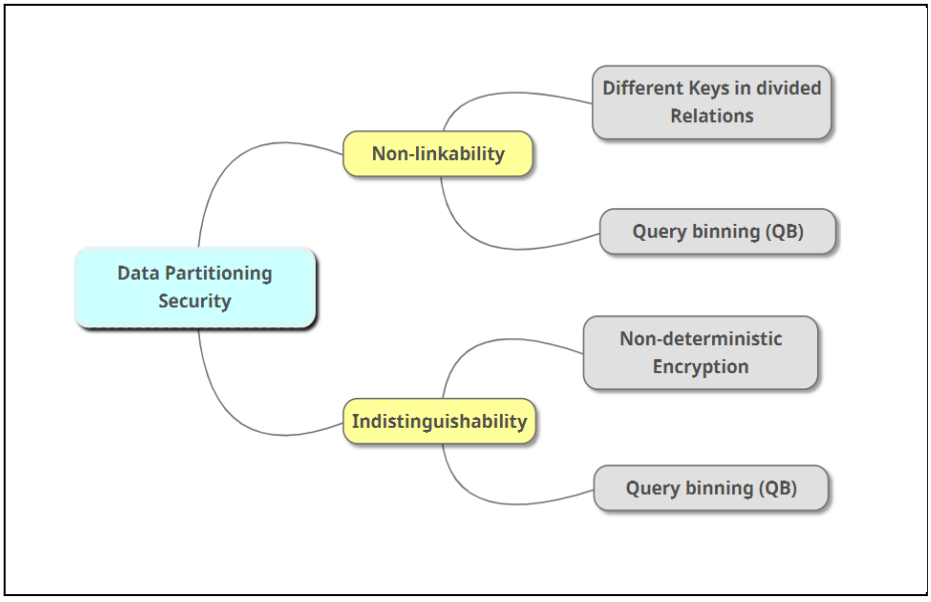


Figure 4.5 Map mind

Based on the adversarial view, we need proof of data security. For that, we should first explain the notion of partitioned data security used in PANDA [2] that is established when a partitioned computation over sensitive and non-sensitive data does not leak any sensitive information. Note that an adversary may infer sensitive information using the adversarial view that was created during query processing, knowledge of frequency counts, and workload characteristics. In PANDA, they begin by clarifying the concepts of associated values, associated tuples, and the relationship between counts of sensitive values.

The definitions used are the same notation used in [2] with additional notation added to prevent data leakage after hybrid partitioning:

1. $t_1, t_2. . .,t_m$ are tuples of a sensitive relation, say R_{E_P}. Thus, the relation R_{E_P} stores the encrypted tuples $E(t_1), E(t_2), . . . , E(t_m)$.
2. $s_1, s_2, . . . , s_{m'}$ are values of an attribute, say A, that appears in one of the sensitive tuples of R_{E_P}. Note that $m' \leq m$, since a number of tuples may have an identical value. Additionally, $s_i \in$ Domain(A),$i = 1, 2, . . . ,m'$.
3. $|s (s_i)|$, refer to the number of sensitive tuples with s_i as the value for attribute A. They further define $|s (v)| = 0, \forall v \in$ Domain(A),$v< s_1, s_2, . . . , s_{m'}$.
4. $t_1, t_2, . . . , t_n$ are tuples of a non-sensitive relation, say R_{P_E}.
5. $ns_1, ns_2, . . . , ns_{n'}$ are values of the attribute A that appears in one of the non-sensitive tuples of R_{P_E}. In equivalence with the case where the relation is sensitive, $n' \leq n$, and $ns_i \in$ Domain(A), $i = 1, 2, . . . ,n$.

Associated values. Let us say $e_i = E(t_i)[A]$ is the encrypted demonstration of an attribute value of A in a sensitive tuple of the relation R_{E_P}, and ns_j is a value of the attribute A for some tuple of the relation R_{P_E}. They said that e_i is associated with ns_j (denoted by $\stackrel{a}{=}$) if the plaintext value of e_i is

identical to the value ns_j. Because we used hybrid data partitioning, this association applies only to tuples divided horizontally.

Associated tuples. Let us say t_i is a sensitive tuple of the relation R_{E_P}(i.e., R_{E_P} stores encrypted representation of t_i) and t_j is a non-sensitive tuple of the relation R_{P_E}. We state that t_i is associated with t_j(for an attribute, say A) if the value of the attribute A in t_i is associated with the value of the attribute A in t_j(i.e., $t_i[A] \overset{a}{=} t_j[A]$). Note that this is the same as stating that the two values of attribute A are equal for both tuples.

Relationship between counts of sensitive values. Let $v_i$and v_j be two different values in Domain(A). They denote the relationship between the counts of sensitive tuples with these A values (i.e., $|s(v_i)|$(or $|s(v_j)|$)) by $v_i \overset{r}{\sim} v_j$.

Note that $\overset{r}{\sim}$ can be one of <, =, or > relationships. Such as, in our example, the $t_2 \overset{r}{\sim} t_4$ corresponds to =, since both values have exactly one sensitive tuple in relation divided horizontally R_{E_P} (see Table 4.5).

Given the above definitions, we can formally state the security requirements needed for selecting SQL queries over sensitive (encrypted values) and non-sensitive (plaintext values) data so that it does not leak any information. Before that, it is worthy of mentioning the security definition in our context. The inference attack in partitioned computing can be considered under the known-plaintext attack (KPA) category. The adversary could know some plaintext data hidden in a set of cipher-text. The adversary's goal in KPA is to designate cipher-text data that are related to a given plaintext, i.e., define a mapping between cipher-text and the corresponding plaintext data representing the same value. In the adversarial view, non-sensitive values are visible to the adversary in plaintext. However, the attacks are different since, unlike the case of KPA, in our setup, the cipher-text data might not contain any data value that is the same as some non-sensitive data visible to the adversary in

plaintext. That means assuming the using of non-deterministic encryption to encrypt the sensitive data. The adversary cannot launch the chosen-plaintext attack (CPA) and the chosen-cipher text attack (CCA). It is not subject to the cipher-text only attack (COA).

Definition: Partitioned Data Security. We use the same notation in [2] for defining the partitioned data security with some additions:

1. R is a relation containing sensitive and non-sensitive tuples.
2. R_E is the sensitive relations for vertical partitioning.
3. R_P is the non-sensitive relations for vertical partitioning.
4. R_{E_P} and R_{P_E} are the sensitive and non-sensitive relations, respectively.
5. AV is an adversarial view generated for a query $q(w)(R_E, R_P, R_{E\_P}, R_{P\_E})[A]$, where the query, q, for a value w of the attribute A of the R_s and R_{ns} relations.
6. ⋈ It is a joining operation between relations.
7. R_i and R_j are divided relations from R.
8. X is the auxiliary information about sensitive data.
9. Pr_{Adv} is the probability of the adversary knowing any information.

A query execution mechanism ensures the partitioned data security if the following three properties hold:

1. $Pr_{Adv}[R_i \bowtie R_j|X, AV] = 0$, where R_i and $R_j \in$ Divided relations on R and ⋈ is a joining operation using the ID attribute, which represents the primary key for the divided relation from R.

Equation 4.2

2. $Pr_{Adv}[e_i \stackrel{a}{=} ns_j|X] = Pr_{Adv}[e_i \stackrel{a}{=} ns_j|X, AV]$, where $e_i = E(t_i)[A]$ is the encrypted representation for attribute value A for any tuple t_i of the relation R_s and ns_j is a value for attribute A for any tuple of relation R_{ns}. [2]

Equation 4.3

3. $Pr_{Adv}[v_i \overset{r}{\sim} v_j|X] = Pr_{Adv}[v_i \overset{r}{\sim} v_j|X, AV]$, for all $v_i, v_i \in$ Domain(A). [2]

Equation 4.4

The use of the hybrid partitioning technique raises a new security challenge, which presents a security gap that lets the adversary guess which values in the plaintext are stored in divided relations that belong to encrypted values stored in encrypted relation for the same tuples (t_i in Original R). The security gap covered by Equation 4.2 represents the 'zero' probability of knowing the original relation by joining the divided relations. This equation is achieved by using different Identifier keys in the divided relations from the original Keys in R.

The second Equation 4.3 presents an initial probability of associating a sensitive tuple with a non-sensitive tuple after executing a query on the divided relations. Therefore, an adversary cannot know anything from an adversarial view. Satisfying this condition also prevents the adversary from succeeding in KPA. The last Equation 4.4 presents that the probability of an adversary learning information about the relative frequency of sensitive values does not increase after the query execution is applied to the divided relations.

In our example, the execution of any queries for any values in the WHERE clause without using different ID keys does not satisfy the first equation. i.e., the query for t_2 retrieves the original tuple stored in R; the adversary knows that Intisar works for a sensitive department. Furthermore, execution of any of three queries (for values 2, 3, or 4) without using QB does not satisfy the second equation. For example, the query for 3 in the domain "ID" retrieves the only tuple from non-sensitive relation. That makes the probability of estimating whether 3 is sensitive or non-sensitive zero compared to an initial probability of the exact estimation, which was 1/4. Hence, the execution of the three queries violates partitioned data security. However, the query execution for "2" and "4"

satisfies the last equation since the count of returned tuples from R_{E_P} is equal. Hence, the adversary cannot distinguish between the count of the values ("2" and "4") in the domain of "ID" of R_{E_P} relation.

4.4 Encryption Technique

In the proposed solution, we create Microsoft .NET Framework common language runtime (CLR) functions to encrypt and decrypt data. CLR function is created as a database object inside the SQL Server instance as a programmed assembly files (DLLs files). CLR function built using Microsoft visual studio 2015 with C# language, and the encryption implemented using the AES encryption technique. Algorithm 4 shows how the encryption is applied.

Algorithm 4 Encryption

```
    Inputs: Tuple_ID, Attributes_Value
    Outputs: Cipher-Text
    Variable: Encryption_Key
1   Function Encryption (Tuple_ID, Attributes_Value ) begin
2       Encryption_Key←GenerqateKey(Tuple_ID)
3       Cipher-Text←AES_Encryption(Attributes_Value, Encryp-
                    tion_Key)
4       Return  Cipher-Text
```

Chapter 5

Implementation, Results, and Discussion

This chapter presents the implementation's practical approach by applying Hybrid Partitioning and QB to the trusted Databases and untrusted Databases, respectively.

The effectiveness of the proposed approach is demonstrated by testing it against inference attacks. Testing the proposed approach against inference. The results are discussed at the end of this chapter.

The rest of this chapter is organized as follows; Section 5.1 introduces the tools used for implementing our approach. Section 5.2 describes the steps of implementing the proposed approach. Section 5.3 presents Experiment Results and Discussion. Finally, Section 5.4 concludes the experiment results.

5.1 Experimental Tools

This section introduces the tools used to implement and test the proposed solution; we used Microsoft SQL server 2014 installed on Windows Server 2012 R2 to store the database and build the proposed solution. In addition, we used a stored procedure as a tool to log the performance of query requests. Moreover, we used Microsoft Visual studio 2015 to write SQL assembly files for encryption and decryption of data.

The experiment environment specification used to evaluate the proposed approach: Server includes Processor Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz (2 CPUs), Installed Memory 32 GB RAM, Hard Disk 512 GB, Microsoft SQL Server 2014, and Windows Server 2012 R2 Standard 64-bit.

5.2 Implementation of Proposed Approach

This section explains the practical approach and the implementation of the Hybrid partitioning technique used. Figure 5.1 shows a general overview of the proposed approach architecture where the two database servers host the trusted and untrusted databases, respectively. The first database server is connected to the internet and private network and hosts the trusted database. The second database server is connected to the internet and is hosting the untrusted database. The Client's devices are connected to the private network.

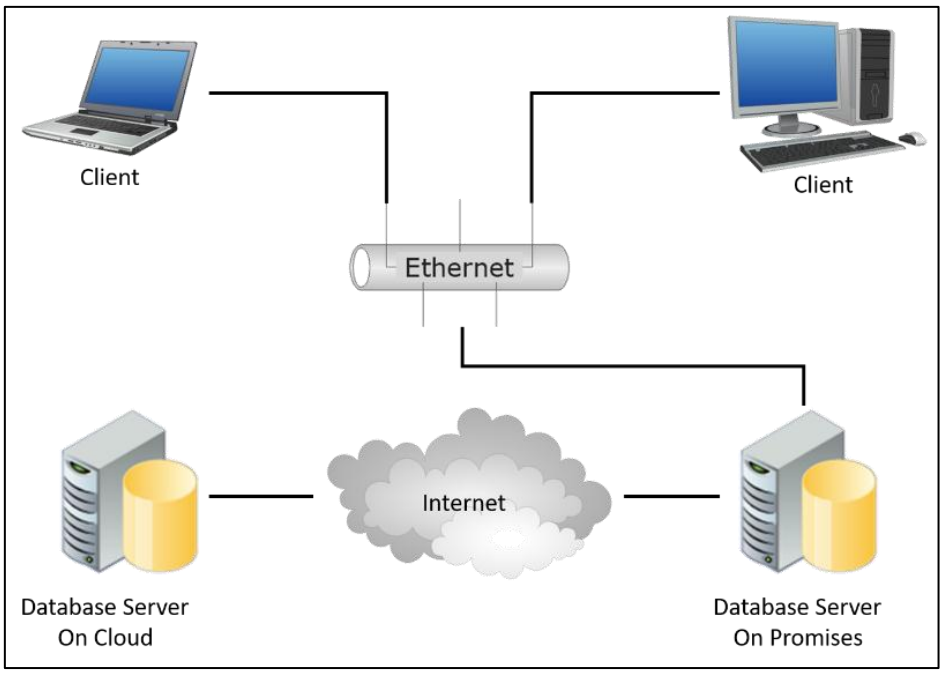


Figure 5.1 General architecture of the proposed approach

The rest of this section describes how the proposed approach is implemented.

5.2.1 Data Partitioning

A stored procedure in SQL server built to implement the Hybrid data partitioning. The stored procedure called after inserting, updating, and deleting the tuples in the original relation (see Table 4.1).

A trigger executes the "Apply Data Partitioning" Stored procedure. While calling the procedure, the type of operation, and the tuple ID (for example, if the operation is an insertion, we send "I" as a char to procedure and “U” as char if the operation is updating a tuple).

In the same stored procedure, "Apply Data Partitioning" we apply the Query Binning described in-depth in subsection0 4.2.

There are two relations stored in trusted databases used to store metadata about sensitive attributes and sensitive values in original relations. For example, Table 5.1 presents the attribute names and whether the attribute is sensitive or not, and Table 5.2 present the sensitive values for the attribute “Position”. These relations are used in the "Apply Data Partitioning" procedure, and the DB owner can add, remove and modify those values.

AttributeName	**IsSensitive**
Employee_Name	FALSE
Position	TRUE
DOB	FALSE
Sex	FALSE
MaritalStatus	FALSE
Salary	TRUE
Location	FALSE
Address1	FALSE
Address2	FALSE
Password	TRUE

Table 5.1 Sensitive attributes

AttributeName	**SensitiveValue**
Position	Marketing Director
Position	Marketing
Position	Senior Marketing

Table 5.2 Sensitive attribute values

5.2.2 Encryption Data

Data encryption is implemented by writing two functions in C# language using Microsoft Visual Studio 2015 and Advanced Encryption Standard (AES). As shown in Appendix 1, the first function is used to encrypt data, while the second is used to decrypted data, as shown in Appendix 1.

5.3 Experiment Results and Discussion

The proposed approach is evaluated by conducted experiments with a different number of tuples retrieved from the database, starting from 2000 tuples, and then the number is increased by 2000 until it reaches 20,000 tuples. In each experiment, the number of attributes that contain sensitive values is gradually increased to 10. As well, these attributes contain 50% of the sensitive values only. The following subsections describe the results of the trials in more detail.

5.3.1 Security Proof

In [2], the authors proved that QB is secure against inference attacks and satisfies the definition of partitioned data security and proved that all the sensitive bins are associated with all the non-sensitive by proving that Equation 4.3 and Equation 4.4 has satisfied the data security properties (non-linkability and indistinguishability). Furthermore, after using Hybrid partitioning, a new security gap is raised: the adversary can learn and link the encrypted values (sensitive attributes) with values that are not encrypted (non-sensitive attributes) that belong to the same tuple. This gap does not satisfy Equation 4.2.

Using different keys in the untrusted database for each tuple satisfies Equation 4.2. It is worth mentioning that the adversary cannot learn anything from the encrypted data since the DB owner is the only party who knows the keys and the Metadata. The Metadata relation is hidden from the adversary.

All experiments show that Equation 4.1 to retrieve the original relation from divided relations on untrusted database equals 0 tuples all the time. That bridges the gap and satisfies Equation 4.2 and, therefore, satisfies the data security property (non-linkability).

5.3.2 Performance Experiments

The first experiment that is being discussed is a query to retrieve 2000 tuples. Table 5.3 present the experimental results of the comparison between our approach and PANDA. Experiment results of query execution performance to retrieve 2000 tuple (%50 of values in sensitive attributes are sensitive).

# of Sensitive attributes	Technique		Enhancement percentage
	Our approach (Seconds)	PANDA (Seconds)	
1	2.89	9.08	68%
2	4.00		56%
3	4.97		45%
4	5.54		39%
5	6.10		33%
6	7.09		22%
7	7.52		17%
8	8.33		8%
9	9.49		~0%
10	9.88		~0%

Table 5.3 Query execution experiment results for 2000 tuples, 50% of values are sensitive in each attribute

In PANDA experiments, the execution time represent in one number (9.08 Second) because, there is no difference in sensitivity status of tuples when number of sensitive attributes are changed.

Equation 5.1 presents how the enhancement percentage of any proposed algorithm regarding the exiting algorithm is evaluated.

$$Enhancement\ percentage = \left(1 - \frac{Our\ approach\ time}{\text{Exiting algorithm } time}\right) \times 100\%$$

Equation 5.1

Figure 5.2 Show the performance of query execution time for ten different partitioned relations according to the number of sensitive attributes in the original relation, 2000 tuples and 50% of tuples contain sensitive values in our approach and PANDA. Unites are measured in seconds.

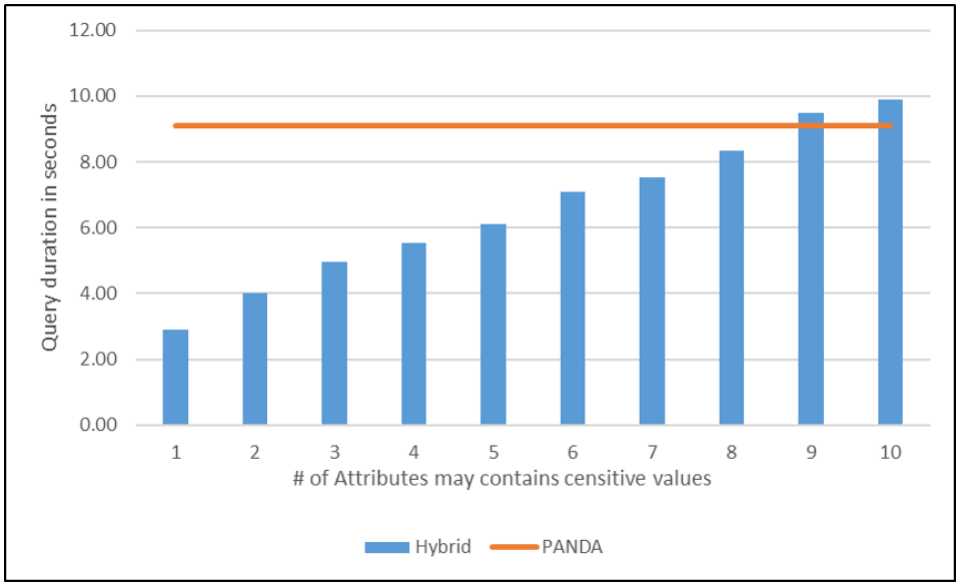


Figure 5.2 Query execution experiment results for 2000 tuples, 50% values are sensitive in each attribute.

Overall, the PANDA technique takes more query execution time than our approach in the given attributes range. Both PANDA and our approach spend most of their query execution time when all attributes contain sensitive values. That happens when the number of attributes is nine and ten too. Furthermore, the most significant difference in performance between the two techniques is apparent when the number of sensitive attributes is one.

In case number of sensitive attributes, one and two, query execution time in our approach is about 3 to 4 seconds as opposed to the PANDA, which is 9 seconds. Similarly, our approach query execution time was higher when the number of attributes was more significant than 8 (around 9 seconds and 9.5 seconds, respectively).

The second experiment is a query to retrieve 4000 tuples. Table 5.4 present the experiment results of a comparison between our approach and PANDA. The experiment results for query execution performance to retrieve 4000 tuples (%50 of values in sensitive attributes are sensitive).

# of Sensitive attributes	Technique		Enhancement percentage
	Our approach (Seconds)	PANDA (Seconds)	
1	3.69	16.51	78%
2	5.52		67%
3	7.29		56%
4	8.13		51%
5	9.87		40%
6	11.17		32%
7	12.53		24%
8	14.42		13%
9	16.13		2%
10	17.45		~0%

Table 5.4 Query execution experiment results for 4000 tuples, 50% values are sensitive in each attribute.

In PANDA experiments, the execution time represent in one number (16.51 Second) because, there is no difference in sensitivity status of tuples when number of sensitive attributes are changed.

Figure 5.3 shows the performance of query execution time for ten different partitioned relations according to the number of sensitive attributes in original relation, 4000 tuples and 50% of tuples contain sensitive values in PANDA and our approach, unites are measured in seconds.

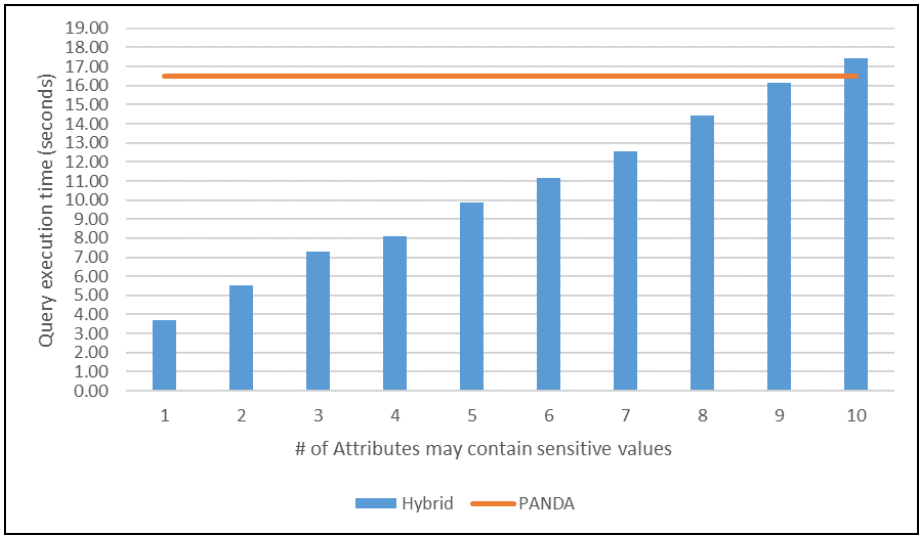


Figure 5.3 Query execution experiment results for 4000 tuples, 50% values are sensitive in each attribute.

In general, the PANDA technique takes more query execution time than our approach in the attributes to the range given. Both PANDA and our approach spend most of their query execution time when all attributes contain sensitive values. That happens when the number of attributes is 10. Furthermore, the most significant difference in performance between the two techniques was when the sensitive attribute number is one.

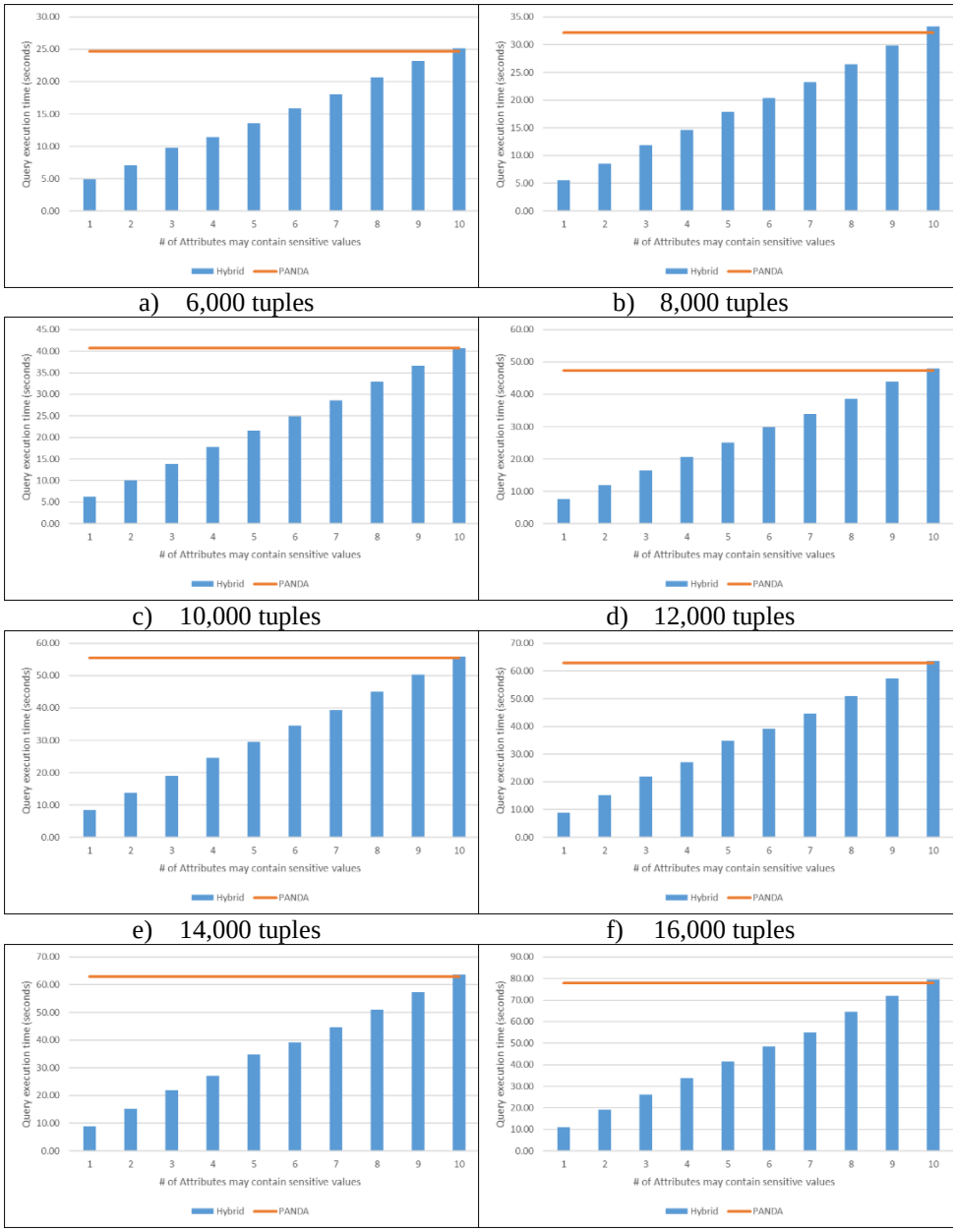
In case the number of sensitive attributes is 1 to 7, query execution time in our approach spend about 3.5 seconds to 12.5 seconds. In this, as opposed to the PANDA's 16 seconds. In this range, the performance improves more than 25%. Similarly, our approach query execution time is higher when the number of attributes is 10.

Finally, the rest of experiments are to retrieve 4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, 20,000 tuples. All experiment results are present in Table 5.5. Figure 5.4 shows the performance of query execution time for eight experiments. For each experiment, a different number of tuples and ten different partitioned relations according to the original relation's number of sensitive attributes. The sensitive attributes have 50% of tuples with sensitive PANDA values, and our approach, where unites are measured in seconds.

# of tuples	Technique	Number of Attributes that contains 50% sensitive values									
		1	2	3	4	5	6	7	8	9	10
2000	Our approach	2.89	4.00	4.97	5.54	6.10	7.09	7.52	8.33	9.49	9.88
	PANDA	9.08	9.08	9.08	9.08	9.08	9.08	9.08	9.08	9.08	9.08
4000	Our approach	3.69	5.52	7.29	8.13	9.87	11.17	12.53	14.42	16.13	17.45
	PANDA	16.51	16.51	16.51	16.51	16.51	16.51	16.51	16.51	16.51	16.51
6000	Our approach	4.87	7.01	9.77	11.43	13.59	15.88	18.02	20.66	23.20	25.18
	PANDA	24.67	24.67	24.67	24.67	24.67	24.67	24.67	24.67	24.67	24.67
8000	Our approach	5.51	8.53	11.84	14.63	17.93	20.40	23.24	26.46	29.83	33.28
	PANDA	32.19	32.19	32.19	32.19	32.19	32.19	32.19	32.19	32.19	32.19
10000	Our approach	6.21	10.03	13.89	17.84	21.61	24.93	28.59	32.88	36.58	40.64
	PANDA	40.72	40.72	40.72	40.72	40.72	40.72	40.72	40.72	40.72	40.72
12000	Our approach	7.57	11.82	16.49	20.70	25.09	29.80	33.83	38.62	43.90	48.02
	PANDA	47.27	47.27	47.27	47.27	47.27	47.27	47.27	47.27	47.27	47.27
14000	Our approach	8.52	13.67	19.02	24.49	29.45	34.58	39.32	45.10	50.40	55.90
	PANDA	55.51	55.51	55.51	55.51	55.51	55.51	55.51	55.51	55.51	55.51
16000	Our approach	8.94	15.16	21.88	27.15	34.76	39.14	44.68	51.09	57.23	63.70
	PANDA	62.90	62.90	62.90	62.90	62.90	62.90	62.90	62.90	62.90	62.90
18000	Our approach	9.99	16.83	24.15	30.83	37.35	43.55	50.68	57.87	63.89	72.47
	PANDA	70.29	70.29	70.29	70.29	70.29	70.29	70.29	70.29	70.29	70.29
20000	Our approach	10.88	19.11	26.22	33.71	41.38	48.63	54.86	64.60	71.86	79.55
	PANDA	77.86	77.86	77.86	77.86	77.86	77.86	77.86	77.86	77.86	77.86

Table 5.5 Experiments Results of query execution times (in seconds) for PANDA and our approach

Overall, the PANDA technique takes more query execution time than our approach in each experiment's attributes range. Both PANDA and our approach spend most of their query execution times in each experiment when sensitive attributes are nine. Furthermore, the most significant difference in performance between the two techniques is when the number of sensitive attributes is one. On the other hand, our approach increases slightly on the number of attributes 10.



a) 6,000 tuples b) 8,000 tuples

c) 10,000 tuples d) 12,000 tuples

e) 14,000 tuples f) 16,000 tuples

g) 18,000 tuples h) 20,000 tuples

Figure 5.4 Query execution experiments result from 6,000to 20,000 tuples, 50% values are sensitive in each attribute.

In General, according to Table 5.6, there is an enhancement in the performance of query execution time. It is worthy of mentioning that most of the relations do not fully contain sensitive values.

# of Sensitive Attributes	Enhancement Percentage
1	82%
2	72%
3	62%
4	54%
5	44%
6	35%
7	27%
8	16%
9	6%
10	~0%

Table 5.6 Average of performance enhancement of our approach

5.4 Summary and Generalization of Results

After implementing our approach technique, all experiments results showed that our approach's query execution time increases directly with increasing number of tuples and attributes that contain sensitive values. All experiments had the same trend about Hybrid performance. The increase in the number of tuples leads to an increase in the query execution time. Furthermore, the increase in the number of attributes containing sensitive data increases the query execution time. That increase is justified because either the number of sensitive values that need to be decrypted from cipher-text to plaintext increases in line with the increase in the tuples or sensitive attributes. That becomes overhead on query requests. As well, the proof in security in section 5.3.1 proves that our approach is reliable, effective, and can prevent inference attacks. That improves the use of the hybrid partitioning data, whether to secure sensitive data and improve performance.

In general, if the number of sensitive attributes is about half of the total number of attributes in a relation, our approach outperforms PANDA by more than 35% in terms of Enhancement percentages. This Enhancement percentage tends to increase as the number of tuples in the original relation increases.

Chapter 6

Conclusion and Future Work

In this chapter, section 6.1 presents the conclusion, and the future work is discussed in section 6.2.

6.1 Conclusion

In this thesis, a Hybrid approach for data partitioning aimed to secure sensitive data when outsourcing data is proposed. The proposed approach is essential to secure the sensitive data that is outsourced to a untrusted database. The proposed approach has the main advantage of improving query performance and securing sensitive data against inference attacks. Furthermore, a new partitioning approach called Hybrid data partitioning (Vertical and Horizontal) is developed. In addition, AES encryption is used to encrypt sensitive data.

The proposed approach has been evaluated using a set of experiments of partitioning data in an untrusted database. Also, comparisons of the results with the PANDA technique are presented. The results of the proposed approach were satisfactory in which the properties of defining the data security satisfy the non-linkability and indistinguishable. Furthermore, the proposed approach results are satisfactory, where the performance of query execution is better than the results of PANDA performance.

6.2 Future Work

This thesis's proposed approach is a Hybrid data partitioning approach to secure data and improve query request performance. A potential future enhancement is to have “join” and “union” operations. Another potential future work is using a dynamic query binning (QB) or finding a novel

technique replacement to the QB, which may improve the results significantly by finding a technique that keeps the sensitive data secure.

Bibliography

[1] S. Mehrotra, S. Sharma, J. D. Ullman and A. Mishra, "Partitioned Data Security on Outsourced Sensitive and Non-Sensitive Data," *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pp. 650-661, 2019.

[2] S. MEHROTRA, S. SHARMA, J. D. ULLMAN, D. GHOSH and P. GUPTA, "PANDA: Partitioned Data Security on Outsourced Sensitive and Non-sensitive Data," *ACM Transactions on Management Information Systems*, 05 2020.

[3] S. Mehrotra, Y. O. Kerim and S. Shantanu, "Exploiting Data Sensitivity on Partitioned Data," *From Database to Cyber Security*, pp. 274-299, 2018.

[4] P. Samarati and S. D. C. d. Vimercati, "Data protection in outsourcing scenarios: Issues and directions," in *Proceedings of the 5th International Symposium on Information, Computer and Communications Security, ASIACCS 2010*, 2010.

[5] "Horizontal, vertical, and functional data partitioning," Microsoft, 11 04 2018. [Online]. Available: https://docs.microsoft.com/en-us/azure/architecture/best-practices/data-partitioning. [Accessed 20 02 2020].

[6] J. P. Meeta and P. B. Mansi, "Overview of Horizontal Partitioning and Vertical Partitioning," in *National Conference on "Computer Science & Security" (COCSS-2013)*, SVIT, Vasad, 2013.

[7] D. Vashi, H. B. Bhadka, K. Patel and S. Garg, "Implementation of Attribute Based Symmetric Encryption through Vertically Partitioned Data in PPDM," *International Journal of Engineering*

and Advanced Technology (IJEAT), vol. 9, no. 1, pp. 868-874, 2019.

[8] O. M. Ben Omran and B. Panda, "A Data Partition Based Model to Enforce Security in Cloud Database," *Journal of Internet Technology and Secured Transaction*, vol. 3, no. 3, pp. 311-319, 09 2014.

[9] O. Omran and B. Panda, "Data Partitioning Methods to Process Queries on Encrypted Databases on the Cloud," University of Arkansas, Fayetteville, 2016.

[10] W. L. e. al., "Towards Secure and Efficient Equality Conjunction Search over Outsourced Databases," in *IEEE Transactions on Cloud Computing*, Early Access, 2020.

[11] M. Naveed, S. Kamara and C. V. Wright, "Inference attacks on property-preserving encrypted databases," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, Machinery, New York, NY, USA, 2015.

[12] S. M. Shafi Goldwasser, "Probabilistic encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270-299, 1984.

[13] M. A. Abdelraheem, T. Andersson, C. Gehrmann and C. Glackin, "Practical Attacks on Relational Databases Protected via Searchable Encryption," *International Conference on Information Security*, pp. 171-191, 9 September 2018.

[14] S. K. Badran, N. Arman and M. Farajallah, "Towards a Hybrid Data Partitioning Technique for Secure Data Outsourcing," in *The International Arab Conference on Information Technology ACIT*, Cairo, 2020.

[15] T. Peng, C. Xiang, S. Sen, C. Rui and S. Huaxi, "Differentially Private Vertically Partitioned Data Publishing," *IEEE Transactions on Dependable and Secure Computing*, 2019.

[16] M. A. Panhwar, S. A. Khuhro, G. Panhwar and K. A. Memon, "SACA: A Study of Symmetric and Asymmetric Cryptographic Algorithms," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND NETWORK SECURITY*, vol. 19, no. 1, pp. 48-55, 2019.

[17] B. Maram, J. M. Gnanasekar, G. Manogaran and M. Balaanand, "Intelligent security algorithm for UNICODE data privacy and security in IOT," *Service Oriented Computing and Applications*, 2018.

[18] S. V. Khedkar and A. Gawande, "Data partitioning technique to improve cloud data storage security," *International Journal of Computer Science and Information Technologies*, vol. 5, no. 3, pp. 3347-3350, 2014.

[19] J. M. Gnanasekar, "UNICODE Text Security Using Dynamic and Key-Dependent 16X16 S-box," *Service Oriented Computing and Applications*, 2016.

[20] P. Dhulavvagol, V. Bhajantri and S. Totad, "Performance Analysis of Distributed Processing System using Shard Selection Techniques on Elasticsearch," *Procedia Computer Science*, vol. 167, pp. 1626-1635, 2020.

[21] V. Ciriani, S. Vimercati, S. Foresti, S. Jajodia, S. Paraboschi and P. Samarati, "Fragmentation Design for Efficient Query Execution over Sensitive Distributed Databases," *2009 29th IEEE International Conference on Distributed Computing Systems*, pp.

32-39, 2009.

Appendixes

Appendix 1

Encryption and Decryption Functions

```
publicstaticstring EncryptAES(string str,string tuple_id)
{
string EncrptKey = GetKey(tuple_id);
byte[] byKey = { };
byte[] IV = { 18, 52, 86, 120, 144, 171, 205, 239, 18, 52, 86, 120, 144, 171, 205, 239 };
AesCryptoServiceProvider aes = new AesCryptoServiceProvider();
  byKey = System.Text.Encoding.UTF8.GetBytes(EncrptKey.Substring(0, aes.Key.Length));
 byte[] inputByteArray = Encoding.UTF8.GetBytes(str);
MemoryStream ms = new MemoryStream();
CryptoStream cs = new CryptoStream(ms, aes.CreateEncryptor(byKey, IV), CryptoStreamMode.Write);
  cs.Write(inputByteArray, 0, inputByteArray.Length);
  cs.FlushFinalBlock();
returnConvert.ToBase64String(ms.ToArray());
}
publicstaticstring DecryptAES(string str,string tuple_id)
      {
          str = str.Replace(" ", "+");
string DecryptKey = GetKey(tuple_id)
byte[] byKey = { };
byte[] IV = { 18, 52, 86, 120, 144, 171, 205, 239, 18, 52, 86, 120, 144, 171, 205, 239 };
byte[] inputByteArray = newbyte[str.Length];
AesCryptoServiceProvider aes = newAesCryptoServiceProvider();
          byKey = System.Text.Encoding.UTF8.GetBytes(DecryptKey.Substring(0, aes.Key.Length));
          inputByteArray = Convert.FromBase64String(str.Replace(" ", "+"));
MemoryStream ms = newMemoryStream();
CryptoStream cs = newCryptoStream(ms, aes.CreateDecryptor(byKey, IV), CryptoStreamMode.Write);
          cs.Write(inputByteArray, 0, inputByteArray.Length);
          cs.FlushFinalBlock();
          System.Text.Encoding encoding = System.Text.Encoding.UTF8;
return encoding.GetString(ms.ToArray());
      }
```

Appendix 2

Query Binning Algorithm Source ([2])

Algorithm 1: Bin-creation algorithm, the base case.

Inputs: $|NS|$: the number of values in the non-sensitive data, $|S|$: the number of values in the sensitive data.
Outputs: SB: sensitive bins; NSB: non-sensitive bins
Variable: $|NSB|$: non-sensitive values in a non-sensitive bin, $|SB|$: sensitive values in a sensitive bin.

Function *create_bins*(S, NS) **begin**
    Permute all sensitive values
    $x, y \leftarrow$ *approx_sq_factors*$(|NS|)$: $x \geq y$
    $|NSB| \leftarrow x$, $NSB \leftarrow \lceil |NS|/x \rceil$, $SB \leftarrow x$, $|SB| \leftarrow y$
    for $i \in (1, |S|)$ **do** $SB[i \text{ modulo } x][*] \leftarrow S[i]$
    for $(i, j) \in (0, SB - 1), (0, |SB| - 1)$ **do** $NSB[j][i] \leftarrow$ *allocateNS*$(SB[i][j])$
    for $i \in (0, NSB - 1)$ **do** $NSB[i, *] \leftarrow$ fill the bin if empty with the size limit to x
    return SB and NSB
Function *allocateNS*$(SB[i][j])$ **begin**
    find a non-sensitive value associated with the j^{th} sensitive value of the i^{th} sensitive bin

Appendix 3

Bin-Retrieval Algorithm Source ([2])

Algorithm 2: Bin-retrieval algorithm.

Inputs: w: the query value. SB and NSB: Sensitive and non-sensitive bins, created by Algorithm 1.
Outputs: SB_a and NSB_b: one sensitive bin and one non-sensitive bin to be retrieved for answering w.
Variables: $found \leftarrow$ **false**

```
1 Function retrieve_bins(q(w)) begin
2     for (i, j) ∈ (0, SB − 1), (0, |SB| − 1) do
3         if w = SB_i[j] then
4             return SB_i and NSB_j; found ← true; break
5     if found ≠ true then
6         for (i, j) ∈ (0, NSB − 1), (0, |NSB| − 1) do
7             if w = NSB_i[j] then
8                 return NSB_i and SB_j; break
9     Retrieve the desired tuples from the cloud by sending encrypted values of the bin SB_i (or SB_j) and cleartext
        values of the bin NSB_j (or NSB_i) to the cloud
```