A Method for Secure Data Outsourcing Utilizing Hybrid Data Partitioning

ABSTRACT

Considering the recent advancements made in internet speed and the development of cloud services, along with the various benefits offered by the cloud such as reliability and convenient accessibility, it becomes necessary to incorporate additional features like flexibility and efficient resource utilization. However, in pursuit of these enhancements, data owners encounter challenges related to outsourcing and safeguarding sensitive data against unauthorized access.Researchers have discovered that segmenting data according to its sensitivity level can prevent data leakage and improve performance. This is achieved by encrypting the partition containing sensitive data along with portions of non-sensitive data using both vertical and horizontal partitioning techniques. A secure data outsourcing method is suggested, which involves a hybrid approach combining horizontal and vertical partitioning. This method employs a set of predefined rules during the query request process, utilizing Query Binning (QB) and metadata from hybrid partitioning. Data encryption is performed using the AES algorithm in compliance with the proposed rules. The effectiveness of this approach is verified through experiments conducted with generated sample data sets. Results demonstrate that our proposed approach surpasses the security performance of the PANDA approach, ensuring non-linkability and indistinguishability security requirements are met.

KEYWORDS

cloud services development,outsourcing, sensitive data,vertical and horizontal data partitioning, query binning, non-linkability, indistinguishably .

1. INTRODUCTION

The outsourcing of data presents a range of vulnerabilities, with the secure and efficient retrieval of outsourced data posing a significant challenge. Among the critical security concerns is the sensitivity of the data, which demands careful examination.Traditionally, data owners have faced two primary options: either refrain from outsourcing data entirely, regardless of its sensitivity, which is both inflexible and inefficient, or encrypt all data stored in the cloud to protect sensitive information, albeit at a potentially high cost.In the pursuit of bolstering data security against various attacks, strategies like data partitioning have been put forth [1, 2, 3, 4, 5, 6, 7, 8, 9]. These methods involve the segmentation of a dataset into multiple subsets based on the sensitivity of the data.
In the contemporary landscape, safeguarding data against unauthorized access, particularly sensitive information, is a paramount concern for data owners. However, the imperative to exploit cloud features, enabling ubiquitous access to services and data while optimizing costs and enhancing reliability, often necessitates data outsourcing. This paradigm shift also fosters the advancement of parallel and distributed computing capabilities.Amidst the challenges posed by data outsourcing, researchers are actively seeking innovative solutions to fortify security measures. Among these, data partitioning emerges as a promising strategy, wherein a dataset is fragmented into smaller subsets based on specific attributes or values, notably sensitive data. These subsets are then dispersed across diverse data centers, potentially spanning various geographical locations. Several distinct methodologies for data partitioning have been devised, encompassing approaches such as Data Sensitivity Partition, Frequency of Use Based Partition, and Space-Based Partition. Researchers have diligently explored a spectrum of techniques and strategies to delineate partitioning categories, aiming to heighten security protocols and shield sensitive data from breaches.This research endeavors to delve into the intricacies of different partitioning techniques, elucidating the methodologies that researchers have crafted to delineate partitioning categories. It sheds light on how these methodologies have been instrumental in fortifying security measures and safeguarding sensitive data. Furthermore, the study proposes a novel hybrid data partitioning approach poised to elevate both security and performance benchmarks.

Two prevalent methods for data segmentation are horizontal partitioning and vertical partitioning. The selection between these methods hinges on two key factors: the intended purpose of the partitioning and the specific objectives of the system owner, be it augmenting security, performance, or both [3, 10].

In the realm of data security, it is imperative to incorporate features such as non-linkability and ciphertext indistinguishability to fortify information protection [2, 11]. Figure 1 delineates the encryption algorithms requisite for ensuring these properties.

- Non-linkability: Ensuring that adversaries remain unaware of the relationship between any encrypted and plaintext values.
- Ciphertext indistinguishability: Preventing adversaries from discerning any relationships among encrypted values.

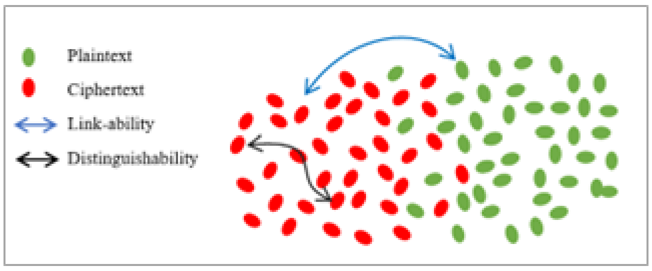


Figure 1: Properties of outsourcing data encryption algorithm

2. RELATED WORKS

Due to the escalating significance of data security, particularly in today's environment, numerous companies and organizations are embracing cloud-based solutions. However, apprehensions regarding potential risks like data leakage, security breaches, and privacy infringements associated with cloud data storage persist. Consequently, researchers have devised an array of techniques to fortify database security and optimize query performance. Within existing literature, safeguarding sensitive data has been addressed through methodologies such as data partitioning.Many researchers have turned to encryption as a shield for data, employing techniques such as Searchable Encryption. Recently, data partitioning techniques have integrated encryption to fortify database security and forestall data leaks during queries. In essence, data partitioning serves as a multifaceted solution, elevating both data security and performance [2, 3, 12, 13, 14, 15, 16].Data has been segregated into sensitive and non-sensitive categories using data partitioning, employing both vertical and horizontal strategies. A groundbreaking technique dubbed "Query Binning (QB)" has been introduced to bolster security against inference attacks like frequency count and workload skew attacks, facilitating joint query processing across sensitive and non-sensitive data. These methodologies enhance performance, fortify security, and mitigate the peril of data leakage. Nonetheless, the current approach struggles to accommodate multiple criteria values in queries [2, 3, 16].Vashi et al. proposed a symmetric encryption technique to uphold data privacy during Privacy-Preserving Data Mining (PPDM) [13]. This method entails implementing encryption on vertically partitioned data based on its sensitivity, employing various encryption algorithms to encrypt sensitive attributes across distinct relations simultaneously. The efficacy of this approach was demonstrated through its application to three datasets of varying sizes. Additionally, four symmetric encryption techniques (AES, DES, Rijndael, and RC2) were employed on sensitive attributes, yielding superior privacy results compared to utilizing a single encryption algorithm on each partitioned relation.Omran et al. introduced an algorithm aimed at thwarting the leakage of sensitive data and preserving privacy within cloud-stored database relations [15]. Their model concentrates on secure data management in cloud databases, employing two approaches to data partitioning: attribute relationship-based partitioning and sensitivity-based partitioning. They proposed a model for managing the outcomes of the partitioning process, involving the storage of partitioned relations in a data center. This model can be implemented in a single cloud data center for optimal performance and security, or the partitioned relations can be dispersed across multiple data centers in different locations to bolster security, albeit without addressing potential performance implications.Omran et al. [15] introduced innovative data partitioning methodologies aimed at facilitating query processing on encrypted databases in the cloud. Their work delved into enhancing query processing efficiency while simultaneously fortifying database relations against potential leaks or attacks in a cloud environment through encryption. Central to their approach was the concept of designing encrypted databases capable of executing SQL queries on encrypted relations without necessitating decryption processes. This paradigm shift ensured that queries could be handled directly on encrypted data stored in the cloud, with the subsequent decryption of query results occurring at the client side. Moreover, the researchers devised four distinct techniques for indexing and partitioning data: frequency of use based partitioning, space-based partitioning, Mondrian or bisection tree based partitioning, and histogram based partitioning.Efficiency evaluations of the first three techniques were juxtaposed with histogram-based partitioning to gauge their effectiveness. These indexes and partitions played a crucial role in query processing, enabling the selection of relevant data segments from the cloud. The researchers explored the storage of index data on either the cloud or on-premises servers alongside encrypted database relations, thereby reducing overall processing time. This encompassed data transfer time from the cloud to the query requester site, as well as the time required for data decoding and processing at the requester end. Furthermore, these techniques facilitated comparisons between encrypted and unencrypted relations, with analysis focusing on runtime and the number of tuples retrieved across relations with varying tuple sizes. The findings underscored the superior efficiency of encrypted relations employing frequency-of-use-based and bisection-tree-based partitioning methodologies.In tandem with data partitioning techniques, researchers have investigated inference attacks to discern how adversaries target encrypted databases and exploit potential information leaks [5]. Naveed et al. scrutinized inference attacks against encrypted database (EDB) systems utilizing property-preserving encryption (PPE) schemes such as CryptDB. Their study encompassed a series of attacks aimed at recovering plaintext from ciphertext encrypted using deterministic encryption (DTE) and order-preserving encryption (OPE) schemes for database attributes. Notably, four well-known attacks including frequency analysis and sorting were considered. Experimental evaluations were conducted on electronic medical records (EMR) data sourced from 200 U.S. hospitals, assuming that adversaries had steady-state access to the EDB and auxiliary information about the system or data. Results highlighted the vulnerability of sensitive information when adversaries possessed background knowledge about EDB data and properties. Specifically, the attacks succeeded in recovering substantial portions of patient records, exceeding 80% when OPE encryption was utilized for attributes like age and disease severity, and over 60% when DTE was applied to attributes such as sex, race, and mortality risk.

3. PROPOSED APPROACH

The proposed research model is outlined in Section 3. Section 3.1 introduces the hybrid data partitioning model, delineating the approach for partitioning relations. In Section 3.2, the proposed model is elaborated upon, providing insights into its design and implementation. The query binning technique, a pivotal aspect of the research, is elucidated in Section 3.3. Finally, Section 3.4 delves into the encryption technique employed within our approach.

3.1 Hybrid Data Partitioning Model

A trusted on-premises Database (DB) housing data in plaintext format executes queries and forwards query requests to an untrusted DB located on the cloud. Within this context, consider a relation R comprising attributes A1, A2, ..., An, encompassing both sensitive and non-sensitive tuples, denoted as t1, t2, ..., tm. The determination of attribute sensitivity is vested in the DB owner, who establishes rules dictating whether a tuple is deemed sensitive based on values associated with certain attributes.Employing a hybrid technique, the DB owner partitions relation R into multiple relations based on the sensitivity of the data. This technique involves dividing each original tuple into a maximum of three tuples, with each divided tuple allocated to a separate relation. The first tuple segment houses values from attributes marked as sensitive, while the second segment comprises values from attributes marked as non-sensitive. The remaining values, found in the third tuple segment, may encompass a mix of sensitive and non-sensitive data, rendering its classification as sensitive or non-sensitive ambiguous.The DB owner proceeds to outsource relations containing non-sensitive data to the cloud in plaintext format. However, tuples from relations containing sensitive data undergo encryption using a non-deterministic encryption mechanism before being outsourced to the same cloud. In our proposed model, the DB owner must maintain metadata, such as a mapping relation, which associates the original tuple ID with the new tuple IDs in each of the divided relations. This metadata plays a crucial role in formulating queries appropriately, leveraging the Query Binning (QB) technique proposed in [2].On the cloud, an untrusted DB hosts the partitioned relations, executing queries and furnishing responses to the trusted on-premises DB.

3.2 Proposed Model

To explain query execution within our model, let's consider a query denoted as σ over the relation R, where a predicate p is represented as $\sigma_p(R)$. This query is executed on the trusted DB without any restrictions on the number of attributes in the WHERE condition clause. The output of the query comprises four attributes: -

- Tuple ID: Represents the original ID assigned to each tuple.

- ID_E:Denotes the tuple ID serving as the primary key in the new relation for sensitive data(RE).
- ID_P: Indicates the tuple ID acting as the primary key in the new relation for non-sensitive(RP).
- IDP_E: Signifies the tuple ID stored in either the relation RP_E in plaintext or relation RE_P in the encrypted form.

Following this, the query process divides the execution of $\sigma_p(R)$ into four sub-queries. Each sub-query, as depicted in Equation 1, is dispatched to an untrusted DB for execution. Subsequently, the results of these sub-queries are returned to the Trusted DB. Within the trusted DB, two sub-queries ((R_{P_E} and R_{E_P}))exhibit identical schemas, necessitating a union operation. This union result is then subjected to join operations with R_P and R_E. Specifically, the query $\#sigma$ on a relation Rs executed as described in Equation 1.

$$\sigma_p(R) = \sigma_p(R_E) \bowtie \sigma_p(R_P) \bowtie \sigma_p(R_{P\_E}) \sqcup \sigma_p(R_{E\_P}) \tag{1}$$

Let's illustrate the proposed hybrid data partitioning model through an example. Consider Table 1, which represents an Employee relation R.Here,a_i $(1 \leq i \leq 6)$denotes an attribute in the relation, indicating the i-th attribute. Similarly, t_j $(1 \leq j \leq 8)$represents the j-th tuple in the relation.The database owner designates the password attribute values as non-outsourced data, while deeming the salary attribute values and all department attribute values corresponding to the "Marketing" department as sensitive.Following the application of the hybrid partitioning algorithm, metadata is generated, as illustrated in Table 2. This metadata comprises four attributes, as described previously. It's noteworthy that the data type of ID_E, ID_P, and ID_{P_E} attributes is a unique identifier data type, generating unique key values consisting of 36 characters.

Table 1: Employee relation

	Attributes No					
	a1	a2	a3	a4	a5	a6
Tuple No	ID	Name	Department	Salary	Location	Password
t1	1	Ali	IT	1,000	Jerusalem	********
t2	2	Intisar	Marketing	900	Jerusalem	********
t3	3	Mahmoud	IT	1,200	Hebron	********
t4	4	Susan	Marketing	1,500	Ramallah	********
t5	5	Sultan	Marketing	1,450	Bethlehem	********
t6	6	Kazem	HR	1,050	Nablus	********
t7	7	Alaa	Marketing	1,460	Bethlehem	********
t8	8	Ahmad	HR	980	Nablus	********

Table 2: Metadata table for relation R

Tuple No	Tuple ID	IDE	IDP	IDP_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

The Employee relation may be stored on the cloud as follows: Relation-1 contains all sensitive values in the Salary attribute, securely encrypted and represented in Table 3. Meanwhile, Relation 2 houses non-sensitive values, storing them in plaintext format across attributes marked as non-sensitive, as showcased in Table 4.Additionally, Relation 3 encompasses tuples containing sensitive data, specifically those where the Department attribute equals "Marketing". These sensitive values are encrypted and delineated in Table 5. On the other hand, Relation 4 caters to sensitive values in the Name and Location attributes, confined to instances where the Department is "Marketing". These values are stored in plaintext format, as outlined in Table 6.Therefore, sensitive data housed in Relation 1 and Relation 3 (refer to Table 3 and Table 5) undergo encryption before being transmitted to an untrusted database. Conversely, Relation 2 and Relation 4 (refer to Table 4 and Table 6), comprising exclusively non-sensitive data, are conveyed in plaintext format. The partitioning process operates at the tuple level, triggered to execute the partitioning code each time a tuple insertion, modification, or deletion operation occurs, as delineated in Algorithm 1 and visually depicted in Figure 2.

Table 3: Relation 1

Attributes No	IDE	a4
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: Relation 2

Attributes No	**IDP**	**a2**	**a5**
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 5: Relation 3

Attributes No	**IDP__E**	**a3**
Tuple No	**ID**	**Department**
t2	CAF5A05C...Q	E(Marketing)
t4	17EDA383...8	E(Marketing)
t5	A03E7373...D	E(Marketing)
t7	05B4FA48...Z	E(Marketing)

Table 6: Relation 4

Attributes No	**IDP__E**	**A3**
Tuple No	**ID**	**Name**
t1	F0D9C43C...R	IT
t3	485F36AB...J	IT
t6	F1859688...Y	HR
t8	14C0E88B...X	HR

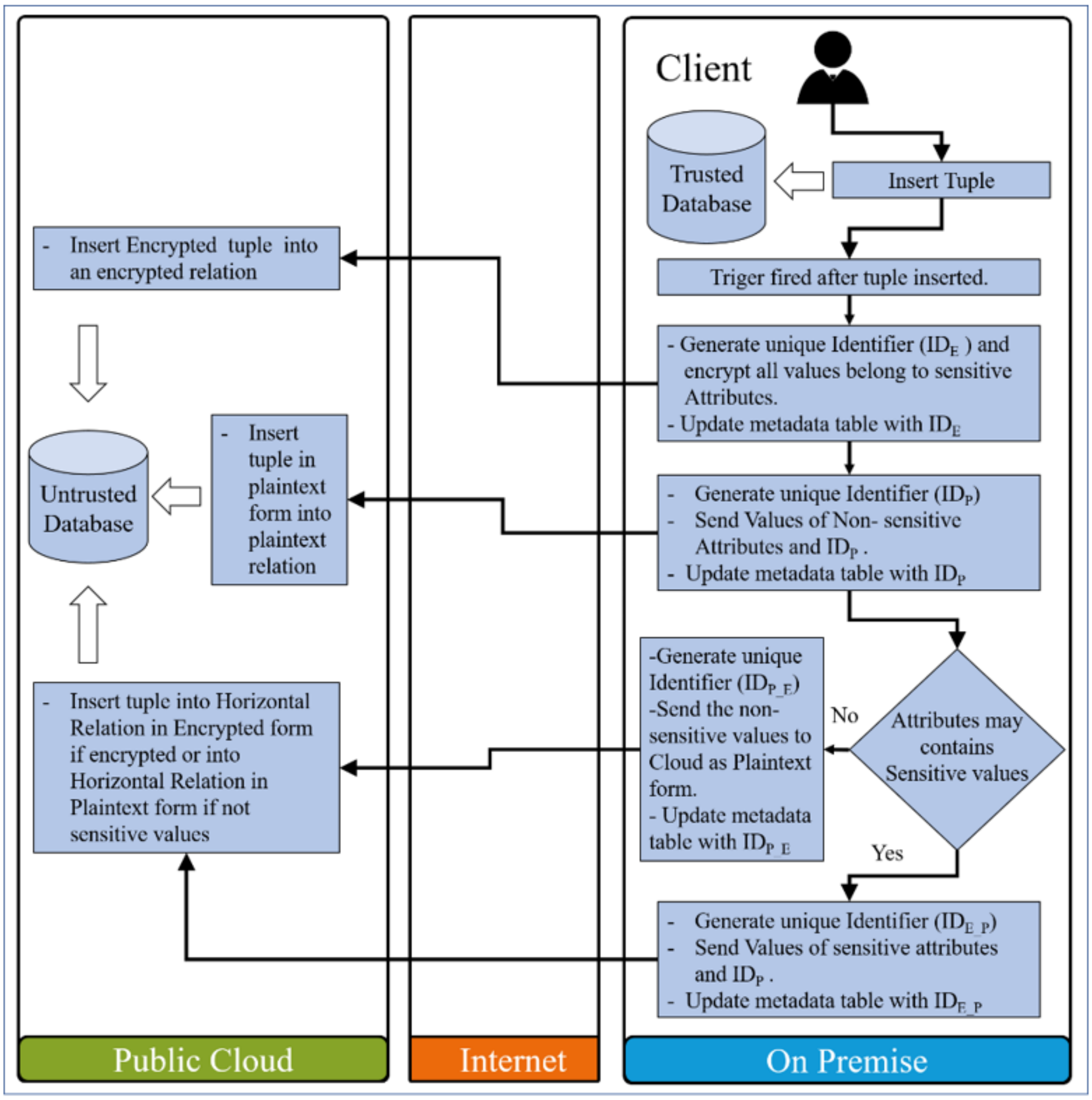

Client
Trusted Database
Insert Tuple
Triger fired after tuple inserted.
- Generate unique Identifier (ID_E) and encrypt all values belong to sensitive Attributes.
- Update metadata table with ID_E
- Generate unique Identifier (ID_P)
- Send Values of Non- sensitive Attributes and ID_P .
- Update metadata table with ID_P
Attributes may contains Sensitive values
No
Yes
-Generate unique Identifier (ID_P_E)
-Send the non-sensitive values to Cloud as Plaintext form.
- Update metadata table with ID_P_E
- Generate unique Identifier (ID_E_P)
- Send Values of sensitive attributes and ID_P .
- Update metadata table with ID_E_P
- Insert Encrypted tuple into an encrypted relation
Untrusted Database
- Insert tuple in plaintext form into plaintext relation
- Insert tuple into Horizontal Relation in Encrypted form if encrypted or into Horizontal Relation in Plaintext form if not sensitive values
Public Cloud
Internet
On Premise

Figure 2: Insert tuple trigger and the partitioning computation

The example showcases the hybrid approach, proposed as the solution in this research. Relations depicted in Table 3 and Table 4 are vertically partitioned. The former holds all values encrypted, pertaining to sensitive attributes, while the latter stores plaintext values associated with attributes containing solely non-sensitive data.Conversely, the relations showcased in Table 5 and Table 6 undergo horizontal partitioning. In the former, sensitive values are encrypted, encompassing tuples containing at least one sensitive value. Meanwhile, the latter houses plaintext values corresponding to the remaining attributes devoid of any sensitive data.

Continuing with Example 1, let's consider a query σ: SELECT Name, Department from Employee where location = N'Jerusalem'. Initially, in the trusted DB, the query $\sigma_{location}$ = N'Jerusalem(R) is executed on relation R. Following Algorithm 2, the results of this query are then joined in the Metadata relation. Subsequently, four queries are generated and dispatched to the Untrusted DB as follows:

- σID_e in (query results)(R_E) executes on R_E relation.
- σID_p in (query results)(R_p) executes on R_p relation.
- σID_{p_E} in (query results)(R_{p_E}) executes on R_{p_E} relation.
- σID_{E_P} in (query results)(R_{E_P}) executes on R_{E_P} relation.

The query result is sent back to trusted DB, and SQL operation is per-formed as presented in Equation-1. Initially, the queries are sent and executed. A UNION operation is conducted between $\sigma_p(R_{P\_E})$ and

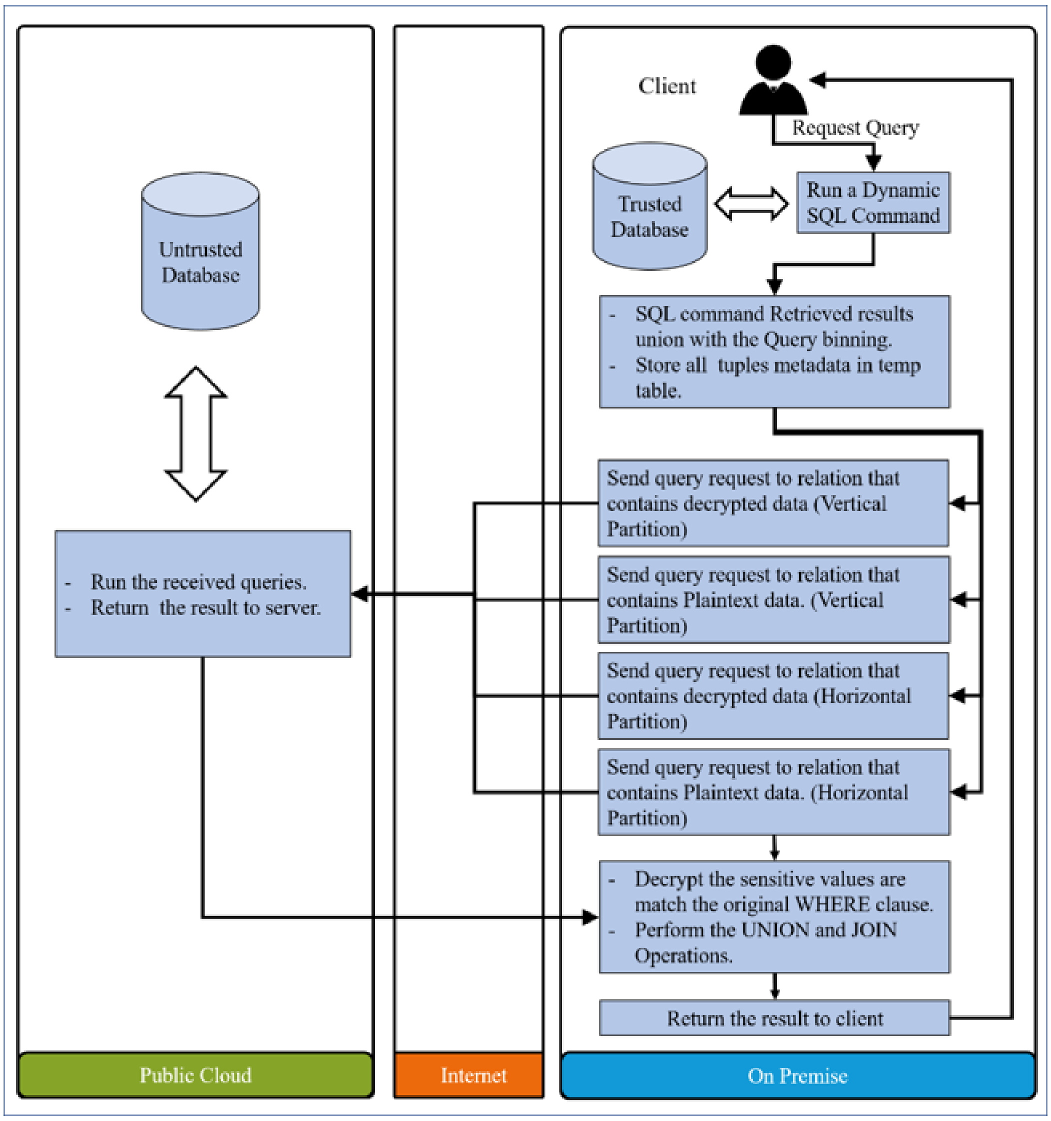
Algorithm 1 Insert Tuple

```
    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      Send$t_P$ 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 $t_{P\_E}$to $R_{P\_E}$ in the cloud
20      Metadata ←t.ID, $ID_P$, $ID_E$,$ID_{Temp}$
21      Return
```

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 re-
          sults, 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.$ID_{E\_P}$))U
                  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
```

$\sigma_p(R_{E\_P})$.Subsequently, the output is utilized in the join operation of $\sigma(R_P)$ and $\sigma(R_E)$.This process facilitates the retrieval of tuples1 t1 and t2. Figure 3 and Algorithm 2 elucidate the query request process.



Client
Request Query
Trusted Database
Run a Dynamic SQL Command
- SQL command Retrieved results union with the Query binning.
- Store all tuples metadata in temp table.
Untrusted Database
Send query request to relation that contains decrypted data (Vertical Partition)
Send query request to relation that contains Plaintext data. (Vertical Partition)
Send query request to relation that contains decrypted data (Horizontal Partition)
Send query request to relation that contains Plaintext data. (Horizontal Partition)
- Run the received queries.
- Return the result to server.
- Decrypt the sensitive values are match the original WHERE clause.
- Perform the UNION and JOIN Operations.
Return the result to client
Public Cloud
Internet
On Premise

Figure 3: Query request scenario

It is noteworthy that the partitioning computation takes place concurrently with the insertion of tuples into the R relation. This approach not only saves time but also avoids the need for performing partitioning computation for the entire dataset at once, as demonstrated in [2]. Figure 2 illustrates the steps of the trigger during the insertion of a tuple into the R relation.

3.3 Query Binning Technique

A solution should be proposed in order to avoid the inference attacks in the partitioned computation. The Query Binning (QB) technique in [2] involves two steps: first, the creation of the query bins. The second step consists of rewriting the query based on the binning. The QB in the base case can be considered as a one-to-one relationship between one sensitive tuple and one non-sensitive tuple. Accordingly, this means that both tuples cannot be sensitive or non-sensitive.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.In relation to the previous example, to calculate the approximately square factors, let us consider that $n =$ number of non-sensitive tuples $= 4$ tuples. According to the definition of approximately square factors, $x = 2$ and $y = 2$, this satisfies the definition of 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] that links between sensitive tuples and non-sensitive tuples. The results of this operation are shown in Figure-4.

The adversary has access to untrusted DB and also 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-7 presents the retrieved tuples without applying the QB technique.To implement the QB bins technique, adjustments are required in the query request process, as depicted in Algorithm 3, which outlines the query request workflow with QB. To understand the impact of QB on the query request results, the adversarial view is altered after the application of Algorithm 3. Table-8 showcases the query request result for an adversary utilizing the QB technique. In this instance, we maintain the same conditions as in the previous example after applying QB (refer to Figure-4). Employing non-deterministic encryption for sensitive data ensures ciphertext indistinguishability, meaning an adversary cannot discern between two ciphertexts [2]. Consequently, the same plaintext values produce two distinct ciphertext values. Additionally, non-linkability is established in two instances: firstly, within the untrusted database by assigning unique IDs to each tuple stored in every divided relation, distinct from the original IDs in the private database; secondly, in the query request process, achieved through query binning (QB). Figure-5 illustrates the security context. In the adversarial view, the adversary has complete access to the Untrusted DB and the transactions log file, consistent with Kerckhoffs's security principle. This implies that while responding to a query, the adversary can retrieve all SELECT SQL statements, re-execute these statements, and obtain the encrypted tuples alongside comprehensive information about the retrieved non-sensitive tuples. This information is accessible to the adversary via the adversarial view. Moreover, the adversary lacks access to the Trusted DB.

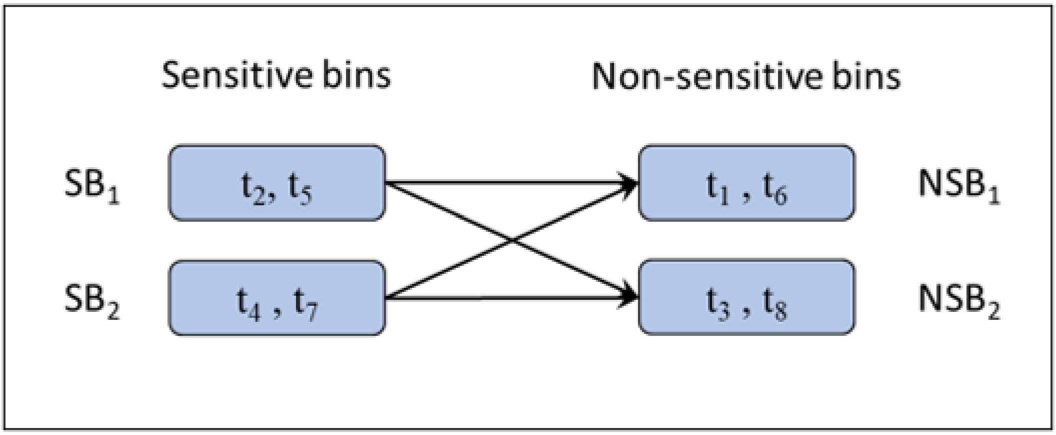


Figure 4: QB of four sensitive and four non-sensitive tuples

Table 7: Queries results, without apply QB

Query value	**Returned tuples/Adversarial view**			
	Relation 1	Relation 2	Relation 3	Relation 4
Jerusalem	E(t1), E(t2)	t2	E(t2)	t1
Hebron	E(t3)	t3	Null	t3
Bethlehem	E(t5), E(t7)	t5 , t7	E(t5), E(t7)	null

Table 8: Query result using QB

Query value	**Returned tuples/Adversarial view**			
	Relation 1	Relation 2	Relation 3	Relation 4
Jerusalem	E(t1),E(t2), E(t5), E(t6)	t1,t2,t5,t6	E(t2), E(t5)	t1,t6
Hebron	E(t2),E(t3), E(t5), E(t8)	t2, t3,t5,t8	E(t2), E(t5)	t3,t8
Bethlehem	E(t2),E(t4), E(t5), E(t7), E(t3), E(t8)	t2, t3, t4,t5, t7, t8	E(t2),E(t4), E(t5), E(t7)	t3, t8

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 $\leftarrow$ Execute( SQLstr )
        T_R_B $\leftarrow$ Retrieve_Bins( T_R_W)
        T_R $\leftarrow$ T_R_B $\bowtie$ Metadata
3       Result1 $\leftarrow$ Execute_on_Cloud($R_{E\_P}$, Domain(T_R.$ID_{E\_P}$)) $\cup$
                    Execute_on_Cloud($R_{P\_E}$, Domain(T_R.$ID_{P\_E}$))
4       Result1 $\leftarrow$ Result1 $\bowtie$ Execute_on_Cloud($R_P$, Domain(T_R.$ID_P$))
5       Result1 $\leftarrow$ Result1 $\bowtie$ Execute_on_Cloud($R_E$, Domain(T_R.$ID_E$))
6       Query results $\leftarrow$ retrieve tuple from Result1 match the original where
                    clause
7       Return Query results
```

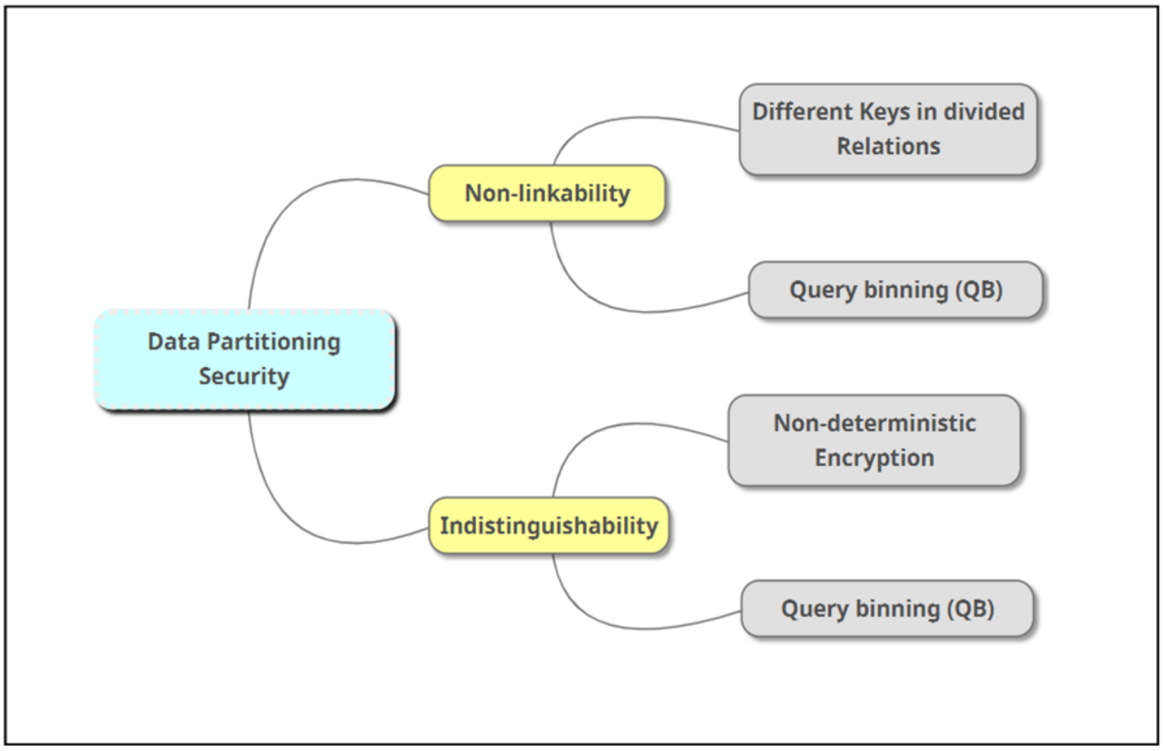

Data Partitioning Security
Non-linkability
Different Keys in divided Relations
Query binning (QB)
Indistinguishability
Non-deterministic Encryption
Query binning (QB)

Figure 5: Map mind

3.4 Encryption Technique

The proposed solution adopts the AES encryption algorithm to encrypt the selected bytes. Algorithm 4 outlines the process of applying encryption

```
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
```

4. RESULTS AND DISCUSSIONS

In this section, we outline our experimental setup, followed by an evaluation of our proposed algorithm's effectiveness in preventing non-linkability and maintaining indistinguishability properties to resist inference attacks. Additionally, we compare the performance of our proposed encryption method with similar and previously published works.

4.1 Experimental Tools

The tools utilized for implementing and testing our proposed solution include Microsoft SQL Server 2014 installed on a Windows Server 2012 R2 platform, serving as the database storage and development environment. Additionally, we employed stored procedures to monitor the performance of query requests effectively. Furthermore, Microsoft Visual Studio 2015 was employed to develop SQL assembly files responsible for the encryption and decryption processes.The experimental setup specifications for evaluating our solution encompass a server equipped with an Intel(R) Xeon(R) CPU E5-2620 v2 @ 2.10GHz (2 CPUs), 32 GB RAM, and a 512 GB hard disk. This server runs Microsoft SQL Server 2014 and operates on the Windows Server 2012 R2 Standard 64-bit operating system.

4.2 Proposed Approach Scenario

The practical implementation of the Hybrid partitioning technique involves two distinct database servers designated for hosting trusted and untrusted databases, respectively. The first server, linked to both the internet and a private network, serves as the repository for the trusted database. Conversely, the second server, also connected to the internet, is dedicated to hosting the untrusted database. Devices belonging to clients are connected exclusively to the private network.Within the trusted databases, two relations are established to store metadata concerning sensitive attributes and the corresponding values within original relations. For instance, Table-9 delineates attribute names along with their sensitivity status, while Table-10 outlines sensitive values associated with the "Position" attribute. These relations play a pivotal role in the "Apply Data Partitioning" procedure, affording the database owner the flexibility to add, remove, or modify values as needed.

4.3 Experiment Results and Discussion

The proposed approach undergoes evaluation through a series of experiments involving varying numbers of tuples retrieved from the database. Initially, the experiments start with 2000 tuples and increment by 2000

Table 9: Sensitive attributes

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

Table 10: Sensitive attributes

AttributeName	SensitiveValue
Position	Marketing Director
Position	Marketing
Position	Senior Marketing

until reaching 20,000 tuples. Within each experiment, the number of attributes containing sensitive values gradually increases to 10. Moreover, these attributes exclusively contain 50% of the sensitive values.In [2], it was demonstrated that QB offers security against inference attacks and fulfills the criteria of partitioned data security. The study also establishes that all sensitive bins are associated with non-sensitive ones based on the fulfillment of data security properties, as indicated by Equation 3 and Equation 4. However, following the implementation of Hybrid partitioning, a new security concern arises: adversaries may gain insights and establish links between encrypted values (sensitive attributes) and non-encrypted values (non-sensitive attributes) belonging to the same tuple. This gap fails to satisfy Equation 2.

$$PrAdv[R_i \bowtie R_j | X, AV] = 0, \tag{2}$$

Where R_i and R_j $\in$ Divided relations on R and $\bowtie$ is a joining operation using the ID attribute, which represents the primary key for the divided relation from R.

$$PrAdv[e_i \overset{a}{=} ns_j | X] = PrAdv[e_i \overset{a}{=} ns_j | X, AV] \tag{3}$$

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, ns_j is a value for attribute A for any tuple of relation R_{ns}.

$$PrAdv[v_i \overset{r}{\sim} v_j | X] = PrAdv[v_i \overset{r}{\sim} v_j | X, AV] \tag{4}$$

For all v_i and v_i $\in$ Domain(A).

Utilizing distinct keys for each tuple in the untrusted database effectively adheres to Equation 2. It's noteworthy that encrypted data remains inscrutable to adversaries as only the database owner possesses the keys and metadata. The metadata relation remains concealed from adversaries.All experiments consistently demonstrate that Equation 1's capability to retrieve the original relation from partitioned relations on the untrusted database consistently yields 0 tuples. This effectively bridges the gap and upholds Equation 2, thereby ensuring the data security property of non-linkability.The initial experiment under discussion entails a query aimed at retrieving 2000 tuples. Table 11 showcases the experimental findings, comparing our approach with PANDA. The experiment evaluates query execution performance to retrieve 2000 tuples, where 50% of the values in sensitive attributes are deemed sensitive.

In the PANDA experiments, the execution time consistently yields a single value (9.08 seconds) since there is no variation in the sensitivity status of tuples when the number of sensitive attributes is altered.Equation 5 outlines the methodology for evaluating the enhancement percentage of any proposed algorithm concerning the existing algorithm.

Table 11: Query execution experiment results for 2000 tuples, 50% of values are sensitive in each attribute

# of Sensitive attributes	Technique: Our approach (Seconds)	Technique: PANDA (Seconds)	Enhancement percentage
1	2.89	9.08	68%
2	4.00		56%
3	4.97		44%
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%

$$Enhancement\ percentage = (1 - \frac{proposed\ algorithm\ time}{presented\ algorithm\ time}) \times 100\% \qquad (5)$$

The figure depicted in Figure 6 illustrates the query execution time performance across ten distinct partitioned relations. These results are based on varying numbers of sensitive attributes present in the original relation. Each scenario considers 2000 tuples, with 50% of them containing sensitive values, in both our approach and PANDA. The units of measurement are in seconds.

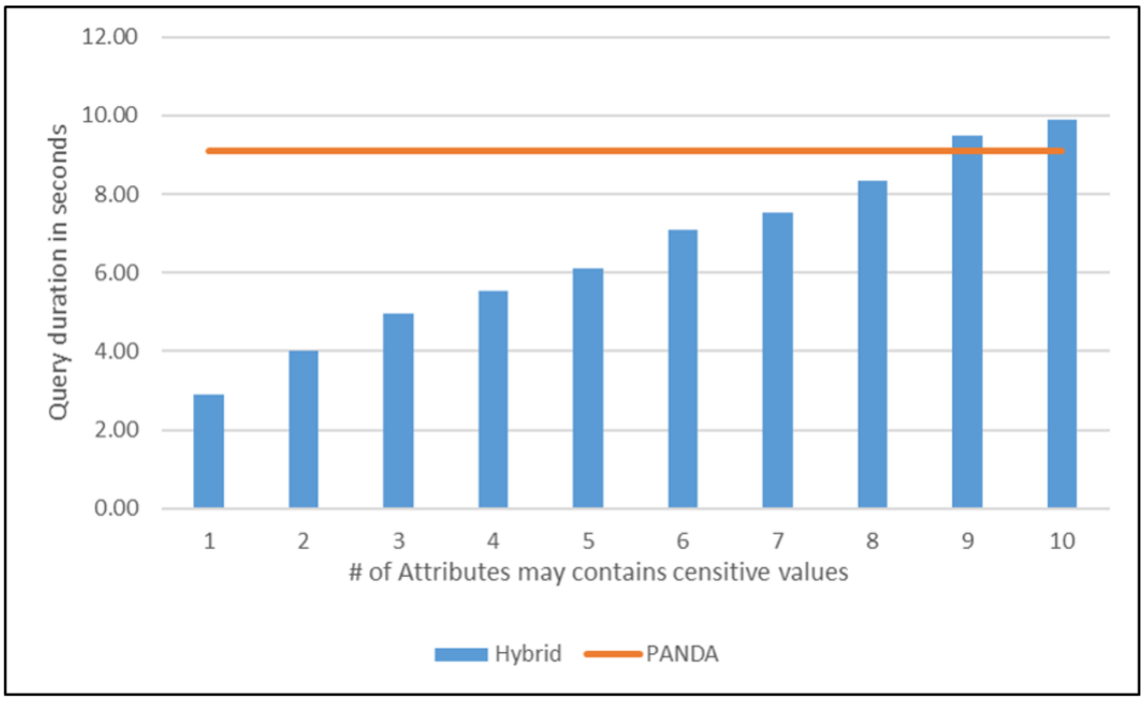


Figure 6: Query execution experiment results for 2000 tuples, 50% values are sensitive in each attribute

In general, our approach outperforms the PANDA technique in terms of query execution time across the same range of attributes. Both methods exhibit prolonged query execution times when all attributes contain sensitive values, notably when the attribute count reaches nine or ten. However, the most substantial disparity in performance between the two techniques becomes evident when only one sensitive attribute is present.For scenarios with one or two sensitive attributes, our approach demonstrates significantly faster query execution times, averaging around 3 to 4 seconds compared to PANDA's 9 seconds. Similarly, our approach exhibits higher query execution times when the attribute count exceeds 8, reaching approximately 9 to 9.5 seconds.Moving on to the second experiment, which involves querying to retrieve 4000 tuples, Table 12 presents the comparative results between our approach and PANDA. These findings pertain to the query execution performance for retrieving 4000 tuples, with 50% of the values in sensitive attributes marked as sensitive.

Table 12: Query execution experiment results for 4000 tuples, 50% values are sensitive in each attribute.

# of Sensitive attributes	Technique Our approach (Seconds)	 PANDA (Seconds)	Enhancement percentage
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%

In the PANDA experiments, the execution time remains consistent at a single value (16.51 seconds). This is because there is no variation observed in the sensitivity status of tuples when the number of sensitive attributes is altered.Figure 7 illustrates the query execution time performance for ten distinct partitioned relations, based on varying numbers of sensitive attributes in the original relation. Each scenario involves 4000 tuples, with 50% of them containing sensitive values in both PANDA and our approach. The units of measurement are in seconds.

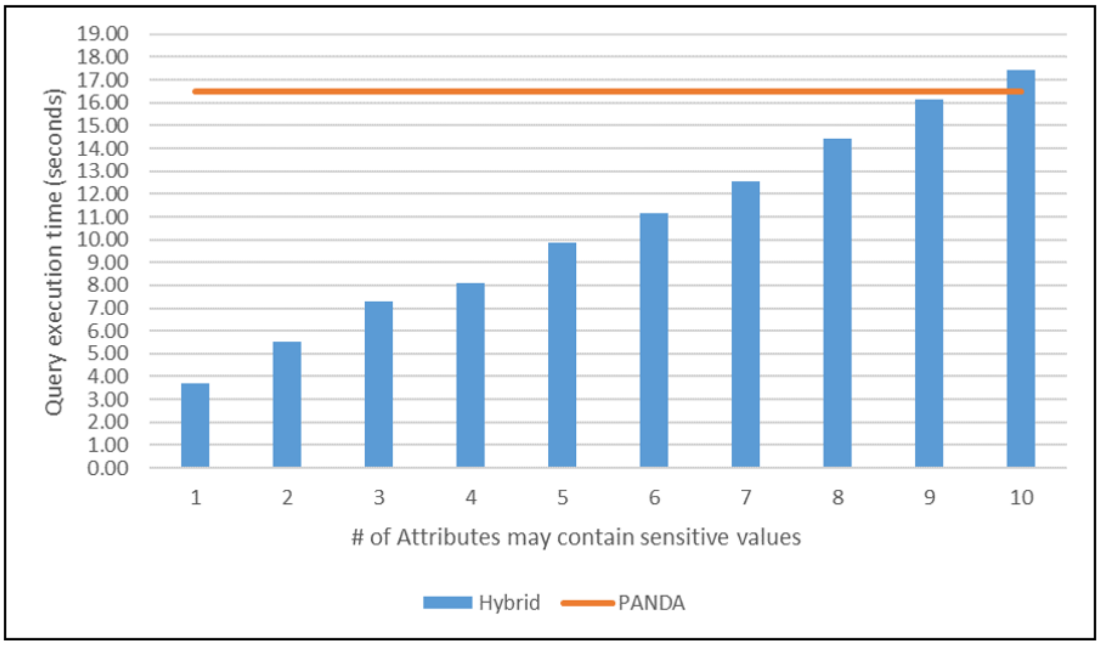


Figure 7: Query execution experiment results for 4000 tuples, 50% values are sensitive in each attribute.

In general, the PANDA technique exhibits longer query execution times compared to our approach across the given range of attributes. Both PANDA and our approach demonstrate prolonged query execution times, particularly when all attributes contain sensitive values, which occurs when there are 10 attributes in total. However, the most notable performance gap between the two techniques occurs when there is only one sensitive attribute present.For scenarios where the number of sensitive attributes ranges from 1 to 7, our approach achieves query execution times ranging from approximately 3.5 seconds to 12.5 seconds. In contrast, PANDA takes around 16 seconds to complete queries within this range. This signifies an improvement in performance of more than 25% with our approach. Conversely, our approach experiences higher query execution times when the number of attributes reaches 10.Finally, additional experiments involve retrieving 4000, 6000, 8000, 10000, 12000, 14000, 16000, 18000, and 20,000 tuples. The results of all experiments are presented in Table 13. Additionally, Figure 8 illustrates the query execution time performance for these eight experiments. Each experiment considers a different number of tuples and ten distinct partitioned relations based on the original relation's number of sensitive attributes. These sensitive attributes comprise 50% of tuples with sensitive values in both PANDA and our approach.

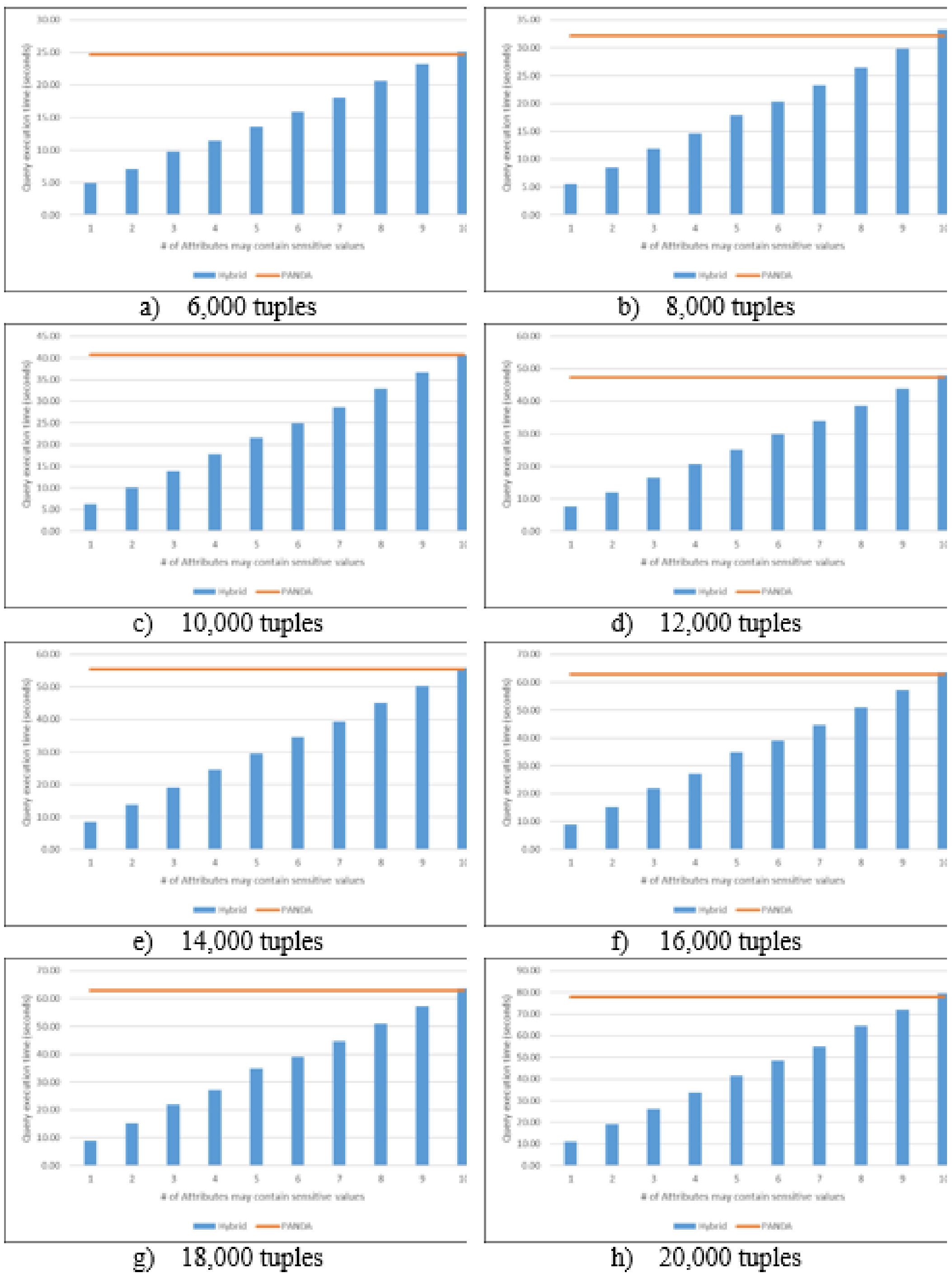


Figure 8: Query execution experiments result from 6,000to 20,000 tuples, 50% values are sen-sitive in each attribute.

In general, the PANDA technique consistently exhibits longer query execution times compared to our approach across all experiments' attributes ranges. Both PANDA and our approach tend to consume the majority of their query execution times when the number of sensitive attributes is nine. However, the most notable disparity in performance between the two techniques arises when there is only one sensitive attribute present. Conversely, our approach experiences a slight increase in query execution time when the number of attributes reaches 10.

Overall, as indicated in Table 14, there is a notable enhancement in the performance of query execution time. It's worth noting that most of the relations don't entirely consist of sensitive values.Upon implementing our approach technique, all experimental results consistently demonstrated a direct correlation between the increase in query execution time and the number of tuples and attributes containing sensitive values. The trend observed across all experiments regarding Hybrid performance reveals that an increase in the number of tuples leads to a subsequent rise in query execution time. Similarly, an increase in the number of attributes containing sensitive data also contributes to longer query execution times. This increase is attributed to the heightened decryption workload, translating ciphertext into plaintext, which escalates with the growing number of tuples or sensitive attributes. Consequently, this overhead on query requests is incurred.Moreover, the aforementioned security validation confirms that our proposed approach is reliable, effective, and capable of thwarting inference attacks. This underscores the utility of hybrid partitioning data for securing sensitive data while simultaneously enhancing performance.In general, when the number of sensitive attributes accounts for approximately half of the total attributes in a relation, our proposed approach surpasses PANDA by over 35% in terms of enhancement percentages. Furthermore, this enhancement percentage tends to increase with the growing number of tuples in the original relation.

Table 13: Experiments Results of query execution times (in seconds) for PANDA and our approach

		Number of Attributes that contains 50% sensitive values									
# of tuples	Technique	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

5. CONCLUSION

This paper introduces a hybrid approach for data partitioning aimed at securing sensitive data during the outsourcing process. The proposed approach plays a crucial role in safeguarding sensitive data when it is stored in an untrusted database. One of its key advantages lies in its ability to enhance query performance while ensuring the security of sensitive data against inference attacks. Additionally, a novel partitioning technique called Hybrid data partitioning, which combines vertical and horizontal approaches, is developed.To achieve robust security, the well-established and secure symmetric encryption algorithm AES is employed to encrypt sensitive data. The effectiveness of the proposed approach is evaluated through

Table 14: Average of performance enhancement of our approach

# 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%

a series of experiments conducted on partitioning data in an untrusted database. Moreover, comparisons with the PANDA technique are presented to provide insights into the performance of the proposed approach.The experimental results demonstrate the effectiveness of the proposed approach in fulfilling the key security properties of non-linkability and indistinguishability. Furthermore, the performance of query execution using the proposed approach outperforms that of the PANDA technique, highlighting its superior performance in practical scenarios.

REFERENCES

[1] Sultan Badran, Nabil Arman, and Mousa Farajallah. Towards a hybrid data partitioning technique for secure data outsourcing. In *2020 21st International Arab Conference on Information Technology (ACIT)*, pages 1–9. IEEE, 2020.

[2] Sharad Mehrotra, Shantanu Sharma, Jeffrey D Ullman, Dhrubajyoti Ghosh, Peeyush Gupta, and Anurag Mishra. Panda: Partitioned data security on outsourced sensitive and non-sensitive data. *ACM Transactions on Management Information Systems (TMIS)*, 11(4):1–41, 2020.

[3] Sharad Mehrotra, Shantanu Sharma, Jeffrey Ullman, and Anurag Mishra. Partitioned data security on outsourced sensitive and non-sensitive data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 650–661. IEEE, 2019.

[4] Sultan Badran, Nabil Arman, and Mousa Farajallah. An efficient approach for secure data outsourcing using hybrid data partitioning. In *2021 International Conference on Information Technology (ICIT)*, pages 418–423. IEEE, 2021.

[5] Muhammad Naveed, Seny Kamara, and Charles V Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655, 2015.

[6] Shantanu Sharma, Anton Burtsev, and Sharad Mehrotra. Advances in cryptography and secure hardware for data outsourcing. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 1798–1801. IEEE, 2020.

[7] Cheng Guo, Ruhan Zhuang, Yingmo Jie, Kim-Kwang Raymond Choo, and Xinyu Tang. Secure range search over encrypted uncertain iot outsourced data. *IEEE Internet of Things Journal*, 6(2):1520–1529, 2018.

[8] Adel Jebali, Salma Sassi, Abderrazak Jemai, and Richard Chbeir. Secure data outsourcing in presence of the inference problem: A graph-based approach. *Journal of Parallel and Distributed Computing*, 160:1–15, 2022.

[9] Adel Jebali, Salma Sassi, and Abderrazak Jemai. Secure data outsourcing in presence of the inference problem: issues and directions. *Journal of Information and Telecommunication*, 5(1):16–34, 2021.

[10] Meeta J Pajwani and Mansi P Bosamia. Overview of horizontal partitioning and vertical partitioning.

[11] Weipeng Lin, Ke Wang, Zhilin Zhang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, Cheng Long, and Chunyan Miao. Towards secure and efficient equality conjunction search over outsourced databases. *IEEE Transactions on Cloud Computing*, 2020.

[12] Mohamed Ahmed Abdelraheem, Tobias Andersson, Christian Gehrmann, and Cornelius Glackin. Practical attacks on relational databases protected via searchable encryption. In *International Conference on Information Security*, pages 171–191. Springer, 2018.

[13] Devendrasinh Vashi, HB Bhadka, Kuntal Patel, and Sanjay Garg. Implementation of attribute based symmetric encryption through vertically partitioned data in ppdm. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(1):868–874, 2019.

[14] Osama M Ben Omran and Brajendra Panda. A data partition based model to enforce security in cloud database. *Journal of Internet Technology and Secured Transaction*, 3(3):311–319, 2014.

[15] Osama Ben Omran. *Data partitioning methods to process queries on encrypted databases on the cloud*. University of Arkansas, 2016.

[16] Sharad Mehrotra, Kerim Yasin Oktay, and Shantanu Sharma. Exploiting data sensitivity on partitioned data. In *From Database to Cyber Security*, pages 274–299. Springer, 2018.