Exploiting Data Sensitivity on Partitioned Data

Sharad Mehrotra(✉), Kerim Yasin Oktay, and Shantanu Sharma(✉)

Department of Computer Science, University of California, Irvine, USA
sharad@ics.uci.edu, shantanu.sharma@uci.edu

Abstract. Several researchers have proposed solutions for secure data outsourcing on the public clouds based on encryption, secret-sharing, and trusted hardware. Existing approaches, however, exhibit many limitations including high computational complexity, imperfect security, and information leakage. This chapter describes an emerging trend in secure data processing that recognizes that an entire dataset may not be sensitive, and hence, non-sensitivity of data can be exploited to overcome some of the limitations of existing encryption-based approaches. In particular, data and computation can be partitioned into sensitive or non-sensitive datasets – sensitive data can either be encrypted prior to outsourcing or stored/processed locally on trusted servers. The non-sensitive dataset, on the other hand, can be outsourced and processed in the cleartext. While partitioned computing can bring new efficiencies since it does not incur (expensive) encrypted data processing costs on non-sensitive data, it can lead to information leakage. We study partitioned computing in two contexts - first, in the context of the hybrid cloud where local resources are integrated with public cloud resources to form a effective and secure storage and computational platform for enterprise data. In the hybrid cloud, sensitive data is stored on the private cloud to prevent leakage and a computation is partitioned between private and public clouds. Care must be taken that the public cloud cannot infer any information about sensitive data from inter-cloud data access during query processing. We then consider partitioned computing in a public cloud only setting, where sensitive data is encrypted before outsourcing. We formally define a *partitioned security* criterion that any approach to partitioned computing on public clouds must ensure in order to not introduce any new vulnerabilities to the existing secure solution. We sketch out an approach to secure partitioned computing that we refer to as *query binning* (QB) and show how QB can be used to support selection queries. We evaluate conditions under which partitioned computing approaches

The full approaches proposed in this chapter may be found in [33,36]. This material is based on research sponsored by DARPA under agreement number FA8750-16-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government. This work is partially supported by NSF grants 1527536 and 1545071.

P. Samarati et al. (Eds.): Jajodia Festschrift, LNCS 11170, pp. 274–299, 2018.
https://doi.org/10.1007/978-3-030-04834-1_15

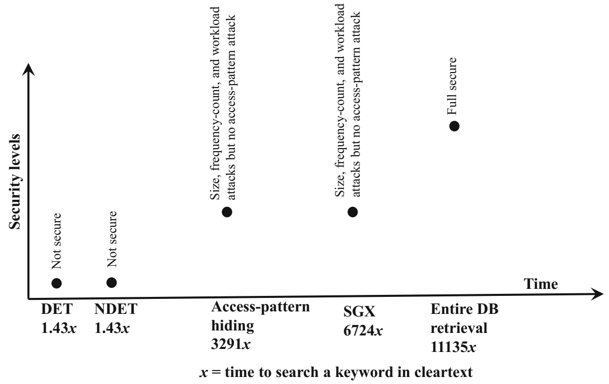
such as QB can improve the performance of cryptographic approaches that are prone to size, frequency-count, and workload attacks.

1 Introduction

Organizations today collect and store a large volume of data, which is analyzed for diverse purposes. However, in-house computational capabilities of organizations may become obstacles for storing and processing data. Many *untrusted cloud computing* platforms (*e.g.*, Amazon AWS, Google App Engine, and Microsoft Azure) offer database-as-a-service using which data owners, instead of purchasing, installing, and running data management systems locally, can outsource their databases and query processing to the cloud. Such cloud-based services available using the pay-as-you-go model offers significant advantages to both small, medium and at times large organizations. The numerous benefits of public clouds impose significant security and privacy concerns related to sensitive data storage (*e.g.*, sensitive client information, credit card, social security numbers, and medical records) or the query execution. The untrusted public cloud may be an *honest-but-curious* (or passive) adversary, which executes an assigned job but tries to find some meaningful information too, or a malicious (or active) adversary, that may tamper the data or query. Such concerns are not a new revelation – indeed, they were identified as a key impediment for organizations adopting the database-as-as-service model in early work on data outsourcing [25,26]. Since then, security/confidentiality challenge has been extensively studied in both the cryptography and database literature, which has resulted in many techniques to achieve *data privacy*, *query privacy*, and *inference prevention.* Existing work can loosely be classified into the following three categories:

1. **Encryption based techniques.** *E.g.*, order-preserving encryption [3], deterministic encryption (Chap. 5 of [24]), homomorphic encryption [21], bucketization [25], searchable encryption [41], private informational retrieval (PIR) [8], practical-PIR (P-PIR) [42], oblivious-RAM (ORAM) [23], oblivious transfers (OT) [39], oblivious polynomial evaluation (OPE) [34], oblivious query processing [5], searchable symmetric encryption [13], and distributed searchable symmetric encryption (DSSE) [27].
2. **Secret-sharing** [40] **based techniques.** *E.g.*, distributed point function [22], function secret-sharing [7], functional secret-sharing [30], accumulating-automata [18,19], Obscure [46], and others [20,31,32].
3. **Trusted hardware-based techniques.** They are either based on a secure coprocessor or Intel SGX, *e.g.*, [4,6]. The secure coprocessor and Intel SGX [12] allow decrypting data in a secure area and perform some computations.

While approaches to compute over encrypted data and systems supporting such techniques are plentiful, secure data outsourcing and query processing remain an open challenge. Existing solutions suffer from several limitations.



The x-axis shows the ratio between the selection query execution time on encrypted data using a cryptographic technique and on cleartext data for a fixed dataset on a specific database system (in both cases), and The y-axis shows the security levels. Weak cryptographic techniques (*e.g.*, deterministic encryption (DET)) are very fast but provide no security (against output size, frequency-count, access-patterns, and workload attacks), while access-pattern hiding techniques are relatively secure but slow. The completely secure technique may retrieve the entire dataset and process at the user-side but this technique is very slow. For join queries, weak cryptographic techniques are efficient since they can exploit hash/merge join. However, more secure techniques, since they need nested loop join, tends to become worse. NDET denotes non-deterministic encryption.

Fig. 1. Comparing different cryptographic techniques.

First, cryptographic approaches that prevent leakage, *e.g.*, fully homomorphic encryption coupled with ORAM, simply do not scale to large data sets and complex queries for them to be of practical value. Most of the above-mentioned techniques are not developed to deal with a large amount of data and the corresponding overheads of such techniques can be very high (see Fig. 1 comparing the time taken for TPC-H selection queries under different cryptographic solutions). To date, a scalable non-interactive mechanism for efficient evaluation of join queries based on homomorphic encryption that does not leak information remains an open challenge. Systems such as CryptDB [38] have tried to take a more practical approach by allowing users to explore the tradeoffs between the system functionality and the security it offers. Unfortunately, precisely characterizing the security offered by such systems given the underlying cryptographic approaches have turned out to be extremely difficult. For instance, [28,35] show that when order-preserving and deterministic encryption techniques are used together, on a dataset in which the entropy of the values is not high enough, an attacker might be able to construct the entire plaintext by doing a frequency analysis of the encrypted data. While mechanisms based on secret-sharing [40] are potentially more scalable, splitting data amongst multiple non-colluding cloud operators (an assumption that is not valid in a general setting) incurs significant communication overheads and can only support a limited set of selection and aggregation queries efficiently.

While the race to develop cryptographic solutions that (*i*) are efficient, (*ii*) support complex SQL queries, (*iii*) offer provable security from the application's perspective is ongoing, this chapter departs from the above well-trodden path by exploring a different (but complementary) approach to secure data processing by partitioning a computation over either the hybrid cloud or the public cloud based on the data classification into sensitive and non-sensitive data. We focus on an approach for situations when only part of the data is sensitive, while the remainder (that may consist of the majority) is non-sensitive. In particular, we consider a **partitioned computation model** that exploits such a classification of data into sensitive/non-sensitive subsets to develop efficient data processing solutions with **provable security guarantees**. Partitioned computing potentially provides significant benefits by (*i*) avoiding (expensive) cryptographic operations on non-sensitive data, and, (*ii*) allowing query processing on non-sensitive data to exploit indices.

The data classification into sensitive or non-sensitive may seem artificial/limiting at first, we refer to the readers to the ongoing dialogue in the popular media [1,2] about cloud security and hybrid cloud that clearly identify data classification policies to classify data as sensitive/non-sensitive as a key strategy to securing data in a cloud. Furthermore, similar to the model considered in this chapter, such articles emphasize either storing sensitive data on a private cloud while outsourcing the rest in the context of hybrid cloud or encrypting only the sensitive part of the data prior to outsourcing. Also, note that data classification based on column-level sensitivity is not a new concept. Papers [9–11,15–17] have explored many ways to outsource column-level partitioned data to the cloud. However, these papers does not dictate a joint query execution on two relations. Some recent database systems such as Jana[1] and Opaque [45] are exploring architectures will allow for only some parts of the data (that is sensitive) to be encrypted while the remainder of the (non-sensitive) data remains in plaintext, thereby supporting partitioned computing. That organizational data can actually be classified as sensitive/non-sensitive is not difficult to see if we consider specific datasets. For instance, in a university dataset, data about courses, catalogs, location of classes, faculty and student enrollment would likely be not considered sensitive, but information about someone's SSN, or grade of the student would be considered sensitive.

Contribution. Our contributions in this chapter are twofold:

Partition computation on the hybrid cloud. Our work is motivated by recent works on the hybrid cloud that has exploited the fact that for a large class of application contexts, data can be partitioned into sensitive and non-sensitive components. Such a classification was exploited to build hybrid cloud

[1] https://galois.com/research-development/cryptography/.

solutions [29,36,37,43,44] that outsource only non-sensitive data and enjoy both the benefits of the public cloud as well as strong security guarantees (without revealing sensitive data to an adversary).

Partition computation on the public cloud. In the setting of the public cloud, sensitive data is outsourced in an appropriate encrypted form, while non-sensitive data can be outsourced in cleartext form. While partitioned computing offers new opportunities for efficient and secure data processing due to avoiding cryptographic approach on the non-sensitive data, it raises several challenges when used in the public cloud. Specifically, the partitioned approach introduces a new security challenge – that of leakage due to simultaneous execution of queries on the encrypted (sensitive) dataset and on the plaintext (non-sensitive) datasets. In this chapter, we will study such a leakage (Sect. 3), a partitioned computing security definition in the context of the public cloud (Sect. 3), and a way to execute partitioned data processing techniques for selection queries (Sect. 4) that support partitioned data security while exploiting existing cryptographic mechanisms for secure processing of sensitive data and cleartext processing of non-sensitive data. Note that the proposed approach can also be extended to other operations such as join or range queries, which are provided in [33].

2 Partitioned Computations at the Hybrid Cloud

In this section, our goal is to develop an approach to execute SQL style queries efficiently in a hybrid cloud while guaranteeing that sensitive data is not leaked to the (untrusted) public machines. At the abstract level, the technique partitions data and computation between the public and private clouds in such a way that the resulting computation (*i*) minimizes the execution time, and (*ii*) ensures that there is no information leakage. Information leakage, in general, could occur either directly by exposing sensitive data to the public machines, or indirectly through inferences that can be made based on selective data transferred between public and private machines during the execution.

The problem of securely executing queries in a hybrid cloud naturally leads to two interrelated subproblems:

Data distribution: How is data distributed between private and public clouds? Data distribution depends on factors such as the amount of storage available on private machines, expected query workload, and whether data and query workload is largely static or dynamic.

Query execution: Given a data distribution strategy, how do we execute a query securely and efficiently across the hybrid cloud, while minimizing the execution time and obtaining the correct final outputs?

Since data is stored on public cloud in the clear text, data distribution strategy must guarantee that sensitive data resides only on private machines. Non-sensitive data, on the other hand, could be stored on private machines, public machines, or be replicated on both. Given a data distribution, the query processing strategy will split a computation between public and private machines while simultaneously meeting the goals of good performance and secure execution.

2.1 Split Strategy

In order to ensure a secure query execution, we develop a *split strategy* for executing SQL queries in the hybrid cloud setting. In a split strategy, a query Q is partitioned into two subqueries that can be executed *independently* over the private and the public cloud respectively, and the final results of the query can be computed by appropriately merging the results of the two sub-queries. In particular, a query Q on dataset D is split as follows:

$$Q(D) = Q_{merge}\Big(Q_{priv}(D_{priv}), Q_{pub}(D_{pub})\Big)$$

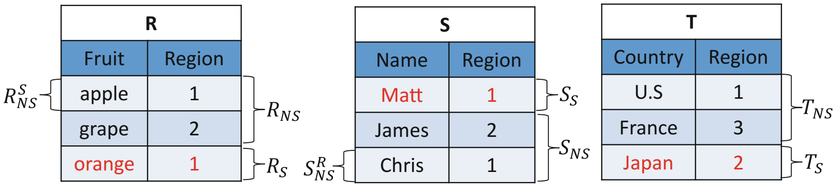
where Q_{priv} and Q_{pub} are private and public cloud sub-queries respectively. Q_{priv} is executed on the private subset of D (i.e., D_{priv}); whereas Q_{pub} is performed over the public subset of D (i.e., D_{pub}). Q_{merge} is a private cloud merge subquery that reads the outputs of former two sub-queries as input and creates the outputs equivalent to that of original Q. We call such an execution strategy as *split-strategy.*

Two aspects of *split-strategy* are noteworthy:

1. It offers full security, since the public machines only have access to D_{pub} that do not contain any sensitive data. Moreover, no information is exchanged between private and public clouds during the execution of Q_{pub}, resulting in the execution at the public cloud to be *observationally equivalent* to the situation where D_{priv} could be any random data.
2. Split-strategy gains efficiency by executing Q_{priv} and Q_{pub} in parallel at the private and public cloud respectively, and furthermore, by performing intercloud data transfer at most once throughout the query execution. Note that the networks between private and public clouds can be significantly slower compared to the networks used within clouds. Thus, minimizing the amount of data shuffling between the clouds will have a big performance impact.

Split strategy, and its efficiency, depends upon the data distribution strategy used to partition the data between private and public clouds. Besides storing sensitive data, the private cloud must also store part of non-sensitive data (called *pseudo sensitive data*) that may be needed on the private side to support efficient query processing. For instance, a join query may necessitate that non-sensitive data be available at the private node in case-sensitive records from one relation may join with non-sensitive records in another. Since in the split-execution strategy, the two subqueries execute independently with no communication, if we do not store non-sensitive data at the private side, we will need to transfer entire relation to the private side for the join to be computed as part of the merge query.

Split-Strategy for Selection or Projection. An efficient *split-strategy* for selection or projection operation is straightforward. In this case, Q_{priv} is equivalent to the original query Q, but is performed only over sensitive records in D_{priv}. Likewise, $Q_{pub} = Q$, but only runs over D_{pub}. Finally, $Q_{merge} = Q_{priv} \cup Q_{pub}$.



R	
Fruit	Region
apple	1
grape	2
orange	1

S	
Name	Region
Matt	1
James	2
Chris	1

T	
Country	Region
U.S	1
France	3
Japan	2

R^S_{NS}
R_{NS}
R_S
S^R_{NS}
S_S
S_{NS}
T_{NS}
T_S

Fig. 2. Example relations.

Split-Strategy for Equijoin. An efficient *split-strategy* for performing a join query such as $Q = R \underset{C}{\bowtie} S$ is more complex. To see this, consider the relations R and S as shown above in Fig. 2, where sensitive portions of R and S are denoted as R_s and S_s, respectively, and remaining fraction of them are non-sensitive, denoted as R_{ns} and S_{ns}, and the join condition is $C = (R.region = S.Region)$. Let us further assume that R_{ns} and S_{ns}, besides being stored in the public cloud are also replicated on the private cloud.

The *naive split-strategy* for $R \underset{C}{\bowtie} S$ would be:

- $Q_{pub} = R_{ns} \underset{C}{\bowtie} S_{ns}$
- $Q_{priv} = (R_s \underset{C}{\bowtie} S_s) \cup (R_s \underset{C}{\bowtie} S_{ns}) \cup (R_{ns} \underset{C}{\bowtie} S_s)$.

Note that if Q is split as above, Q_{priv} consists of three subqueries which scan 2, 3, and 3 tuples in R and S respectively resulting in 8 tuples to be scanned and joined. In contrast, if we simply executed the query Q on the private side (notice that we can do so, since R and S are fully stored on the private side), it would result in lower cost requiring scan of 6 tuples on the private side. Indeed, the overhead of the above split strategy increases even further if we consider multiway joins (e.g., $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$) compared to simply implementing the multiway join locally. Thus, if we use split-strategy for computing $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, where C' is $S.Region = T.Region$, then the number of tuples that are scanned/joined in the private cloud will be much higher than that of the original query.

A Modified Approach for Equijoin. The cost of executing Q in the private cloud can be significantly reduced by pre-filtering relations R and S based on sensitive records of the other relation. To perform such a pre-filtering operation, the tuples in the relations R_{ns} and S_{ns} have to be co-partitioned based on whether they join with a sensitive tuple from the other table under condition C or not.

Let R_{ns}^{S} be a set of non-sensitive tuples of R that join with any sensitive tuple in S. In our case, $R_{ns}^{S} = \langle$apple, 1$\rangle$. Similarly, let S_{ns}^{R} be non-sensitive tuples of S that join with any record from R_s, *i.e.*, $\langle$Chris, 1$\rangle$. In that case, the new private side computation can be rewritten as:

$$(R_s \cup R_{ns}^{S}) \underset{C}{\bowtie} (S_s \cup S_{ns}^{R}). \tag{1}$$

Thus, the scan and join cost of this new plan at the private cloud is 4, which is lower compared to computing the query entirely on the private side that had a cost of 6.

Guarded Join. The above mentioned modified strategy, nonetheless, introduces a new challenge. Since $R_{ns}^{S} \underset{C}{\bowtie} S_{ns}^{R}$ is both repeated at public and private cloud, the output of $R_{ns}^{S} \underset{C}{\bowtie} S_{ns}^{R}$, $\langle$apple, Chris, 1$\rangle$, is computed on both private and public clouds. To prevent this, we do a guarded join ($\bowtie'$) on the private cloud, which discards the output, if it is generated via joining two non-sensitive tuples. This feature can easily be implemented by adding a column to R and S that marks the sensitivity status of a tuple, whether it is sensitive or non-sensitive, and then by adding an appropriate selection after the join operation. In other words, the complete representation of private side computation for $R \underset{C}{\bowtie} S$ would be

$$\sigma_{R.sens=true \vee S.sens=true}((R_s \cup R_{ns}^{S}) \underset{C}{\bowtie} (S_s \cup S_{ns}^{R})) \tag{2}$$

where *sens* is a boolean column (or partition id) appended to relations R and S on the private cloud. Assume that it is set to true for sensitive records and false for non-sensitive records.

Challenges. There exist multiple challenges in implementing this new approach. First challenge is the cost of creating R_{ns}^{S} and S_{ns}^{R} beforehand. Extracting these partitions for a query might take as much time as executing the original query. However, the costs are amortized since these relations are computed once, and used multiple times to improve join performance at the private cloud.

The second challenge is the creation of co-partitioning tables for complex queries. For instance, in case of a query $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, the plan would be to first compute results of $R \underset{C}{\bowtie} S$, and then to join them with T. However, if we do the private side computation of $R \underset{C}{\bowtie} S$, based on Eq. 1 (no duplicate filtering) and join the results with T, then we will not be able to obtain the complete set of sensitive $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$ results.

To see this, consider the sensitive record (in Fig. 2) $\langle$Japan, 2$\rangle$ in T that joins with non-sensitive $\langle$grape, 2$\rangle$ tuple in $R - R_{ns}^{S}$ or joins with non-sensitive $\langle$James, 2$\rangle$ tuple from $S - S_{ns}^{R}$. Thus, the non-sensitive records of R and S has to be co-partitioned based on the sensitive records of T via their join paths from T. In $R \underset{C}{\bowtie} S \underset{C'}{\bowtie} T$, the join path from T to R is $T \underset{C'}{\bowtie} S \underset{C}{\bowtie} R$ and from T to S is $T \underset{C'}{\bowtie} S$. Similarly, the non-sensitive T records has to be co-partitioned based on the sensitive R and S records via join paths specified in the query.

Final challenge is in maintaining these co-partitions and feeding the right one when an arbitrary query arrives. Given a workload of queries and multiple possible join paths between any two relations, each relation R in the dataset may need to be co-partitioned multiple times. This implies that any non-sensitive record r of R might appear in more than one co-partition of R. So, maintaining each co-partition separately might be unfeasible in terms of storage. However, the identifiers of each co-partition that record r belongs to can be embedded into r as a new column. We call such a column as the *co-partition* (CPT) column. Note that CPT column is *only defined* on the private cloud data, since revealing it to public cloud would violate our security requirement.

CPT column initially will be set to null for sensitive tuples in the private side, since the co-partitions are only for non-sensitive tuples. Thus, it can further be used to serve another purpose, indicating the sensitivity status of a tuple r by setting it to "sens" only for sensitive tuples.

Join Path. To formalize the concept of co-partitioning, we first need to define the notion of join path. Let R_i be a relation in our dataset D, and let Q be a query over the relation R_i. We say a join path exists from a relation R_j to R_i, if either R_i is joined with R_j directly based on a condition C, *i.e.*, $R_j \underset{C}{\bowtie} R_i$, or R_j is joined with R_i *indirectly* using other relations in Q. A join path p can be represented as a sequence of relations and conditions between R_j and R_i relations. Let $PathSet$ be the set of all join paths that are extracted either from the expected workload or a given dataset schema.

$$PathSet_i = \{\forall p \in PathSet : \text{path } p \text{ ends at relation } R_i\}. \tag{3}$$

Let $CP(R_i, p)$ be the set of non-sensitive R_i records that will be joined with at least one sensitive record from any other relation R_j via the join path p. Note that p starts from R_j and ends at R_i that can be used as an id to $CP(R_i, p)$. Any $CP(R_i, p)$ is called as "co-partition" of R_i. Given these definitions, the CPT column of a R_i record, say r, can be defined as:

$$r.CPT = \begin{cases} sens & \text{if } r \text{ is sens.} \\ \{\forall p \in PathSet_i : r \in CP(R_i, p)\} & \text{otherwise} \end{cases} \tag{4}$$

Figure 3 shows our example R, S and T relations with their CPT column. For instance, the join path $R \bowtie S$ will be appended to the CPT column of all the tuples in S_{ns}^{R}. Additionally, the CPT column of all tuples in R_s will be set to $sens$.

2.2 Experimental Analysis

To study the impact of table partitioning discussed in the previous section, we differentiate between two realizations of our strategy: in our first technique, entitled (*CPT-C*), every record in a table at the private cloud contains a CPT column and they are physically stored together; whereas in our second approach,

R		
Fruit	Region	CPT
apple	1	$S \bowtie R$
grape	2	$T \bowtie S \bowtie R$
orange	1	sens

S		
Name	Region	CPT
Matt	1	sens
James	2	$T \bowtie S$
Chris	1	$R \bowtie S$

T		
Country	Region	CPT
U.S	1	$S \bowtie T$, $R \bowtie S \bowtie T$
Japan	2	sens
France	3	null

Fig. 3. Example relations with the CPT columns.

entitled *CPT-P*, the tables are partitioned based on their record's CPT column and each partition is stored separately. Each partition file then appended to the corresponding Hive table as a separate partition, so at querying stage, Hive filters out the unnecessary partitions for that particular query.

Sensitive Data Ratio. For these experiments, we varied the amount of sensitive records $(1, 5, 10, 25, 50\%)$ in *customer* and *supplier* tables. Also, we set the number of public machines to 36. As expected, Fig. 4 shows that a larger percentage of sensitive data within the input leads to a longer workload execution time for both, CPT-C and CPT-P in Hadoop and Spark. The reason behind this is that a higher sensitive data ratio results in more computations being performed on the private side and implies a longer query execution time in *split-strategy*. When the sensitivity ratio increases, CPT-P's scan cost increases dramatically. Since the scan cost of queries is the dominant factor compared to other operators (join, filtering etc.) in Spark, CPT-C provides a very low-performance gain compared to All-Private in Spark. Because the scan cost of these two approaches is same. Overall, when sensitivity ratio is as low as 1%, CPT-P provides 8.7× speed-up in Hadoop and 5× speed-up in Spark compared to All-Private.

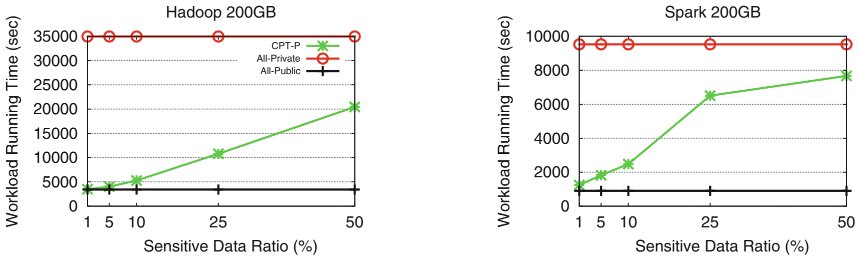


Fig. 4. Running times for different sensitivity ratios.

Recall that we created the CPT column using a Spark job for CPT-C solution. We then physically partitioned tables for CPT-P solution. Figure 5 shows how much time we spent in preparing private cloud data for both CPT-C and CPT-P. It also indicates the gains of these approaches compared to All-Private in terms of the overall workload execution time. As indicated in Fig. 5, until 25%

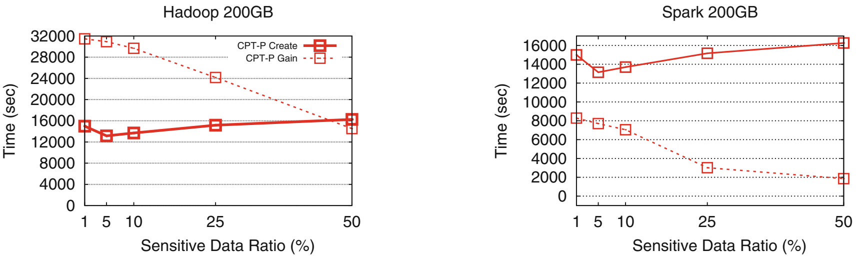


Fig. 5. The CPT column's creation for different sensitivity ratios.

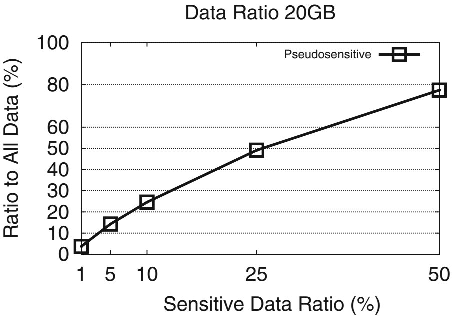


Fig. 6. Comparison of pseudo-sensitive data and sensitivity ratio.

sensitivity, CPT-P's data preparation time is less than that of performance gain in Hadoop; whereas in Spark, data preparation time is always higher than the performance gain for both CPT-P and CPT-C. Note that, we prepare the CPT column only once on a static data for an expected workload that will more likely be executed more than once with different selection and projection conditions. In Spark, if the sensitivity ratio is as high as 10%, executing the workload more than once will be enough for the performance gain of CPT-P solution to be higher than the overhead of data preparation time.

Size of Private Storage. Besides storing sensitive data, in our technique, we also store pseudo-sensitive data on the private cloud. This enables us to execute queries in a partitioned manner while minimizing expensive inter-cloud communication during query execution. In Fig. 6, we plot the size of pseudo-sensitive data as a percentage of total database size at different sensitivity levels. We note that even when sensitivity levels are as high as 5–10%, the pseudo-sensitive data remains only a fraction (15–25% of the total data). At smaller sensitivity levels, the ratio is much smaller.

2.3 Other Approaches to Partitioned Computing

The discussion above focused on partitioned computing in hybrid clouds in the context of SQL queries and is based primarily on the work that appeared in [36].

Several other approaches to partitioned computing in the hybrid cloud have also been developed in the literature that, similar to the above-mentioned method, offer security by controlling data distribution between private and public clouds. Many of these approaches [29,37,43,44] have been developed in the context of MapReduce job execution, and they address security at a lower level compared to the approach defined above, which is at SQL level. Note that one could, potentially, transform SQL/Hive queries into lower level MapReduce jobs and run such MapReduce jobs using privacy preserving extensions. There are several limitations of such an approach, however, and we refer the reader to [36] for a detailed discussion of the limitations of such an approach and to [14] for a detailed survey on the hybrid cloud based MapReduce security.

3 Partitioned Computations at the Public Cloud and Security Definition

In this section, we define the partitioned computation, illustrate how such a computation can leak information due to the joint processing of sensitive and non-sensitive data, discuss the corresponding security definition, and finally discuss system and adversarial models under which we will develop our solutions.

Partitioned Computations

Let R be a relation that is partitioned into two sub-relations, $R_e \supseteq R_s$ and $R_p \subseteq R_{ns}$, such that $R = R_e \cup R_p$. The relation R_e contains all the sensitive tuples (denoted by R_s) of the relation R and will be stored in encrypted form in the cloud. Note that R_e may contain additional (non-sensitive) tuples of R, if that helps with secure data processing). The relation R_p refer to the sub-relation of R that will be stored in plaintext on the cloud. Naturally, R_p does not contain any sensitive tuples. For the remainder of the chapter, we will assume that $R_e = R_s$ and $R_p = R_{ns}$, though our approach will be generalized to allow for a potentially replicated representation of non-sensitive data in encrypted form, if it helps to evaluate queries more efficiently. Let us consider a query Q over relation R. A partition computation strategy splits the execution of Q into two independent sub-queries: Q_s: a query to be executed on $E(R_e)$ and Q_{ns}: a query to be executed on R_p. The final results are computed (using a query Q_{merge}) by appropriately merging the results of the two sub-queries at the trusted database (DB) owner side (or in the cloud, if a trusted component, *e.g.*, Intel SGX, is available for such a merge operation). In particular, the query Q on a relation R is partitioned, as follows:

$$Q(R) = Q_{merge}\Big(Q_s(R_e), Q_{ns}(R_p)\Big)$$

Let us illustrate partitioned computations through an example.

	EId	FirstName	LastName	SSN	Office#	Department
t_1	E101	Adam	Smith	111	1	Defense
t_2	E259	John	Williams	222	2	Design
t_3	E199	Eve	Smith	333	2	Design
t_4	E259	John	Williams	222	6	Defense
t_5	E152	Clark	Cook	444	1	Defense
t_6	E254	David	Watts	555	4	Design
t_7	E159	Lisa	Ross	666	2	Defense
t_8	E152	Clark	Cook	444	3	Design

Fig. 7. A relation: *Employee*.

Example 1. Consider an *Employee* relation, see Fig. 7. In this relation, the attribute *SSN* is sensitive, and furthermore, all tuples of employees for the *Department* = “`Defense`” are sensitive. In such a case, the *Employee* relation may be stored as the following three relations: (*i*) *Employee1* with attributes *EId* and *SSN* (see Fig. 8); (*ii*) *Employee2* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* =“`Defense`” (see Fig. 9); and (*iii*) *Employee3* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* <> “`Defense`” (see Fig. 10). Since the relations *Employee1* and *Employee2* (Figs. 8 and 9) contain only sensitive data, these two relations are encrypted before outsourcing, while *Employee3* (Fig. 10), which contains only non-sensitive data, is outsourced in clear-text. We assume that the sensitive data is strongly encrypted such that the property of *ciphertext indistinguishability* (*i.e.*, an adversary cannot distinguish pairs of ciphertexts) is achieved. Thus, the two occurrences of `E152` have two different ciphertexts.

	EId	SSN
t_1	E101	111
t_2	E259	222
t_3	E199	333
t_5	E152	444
t_6	E254	555
t_7	E159	666

Fig. 8. A sensitive relation: *Employee1*.

Consider a query `Q: SELECT FirstName, LastName, Office#, Department from Employee where FirstName = ''John''`. In partitioned computation, the query `Q` is partitioned into two sub-queries: Q_s that executes on `Employee2`, and Q_{ns} that executes on `Employee3`. Q_s will retrieve the tuple t_4 while Q_{ns} will retrieve the tuple t_2. Q_{merge} in this example is simply a union

	EId	FirstName	LastName	Office#	Department
t_1	E101	Adam	Smith	1	Defense
t_4	E259	John	Williams	6	Defense
t_5	E152	Clark	Cook	1	Defense
t_7	E159	Lisa	Ross	2	Defense

Fig. 9. A sensitive relation: *Employee2*.

	EId	FirstName	LastName	Office#	Department
t_2	E259	John	Williams	2	Design
t_3	E199	Eve	Smith	2	Design
t_6	E254	David	Watts	4	Design
t_8	E152	Clark	Cook	3	Design

Fig. 10. A non-sensitive relation: *Employee3*.

operator. Note that the execution of the query `Q` will also retrieve the same tuples.

Inference Attack in Partitioned Computations

Partitioned computations, if performed naively, could lead to inferences about sensitive data from non-sensitive data. To see this, consider following three queries on the *Employee2* and *Employee3* relations: (i) retrieve tuples of the employee `Eid = E259`, (ii) retrieve tuples of the employee `Eid = E101`, and (iii) retrieve tuples of the employee `Eid = E199`. We consider an *honest-but-curious* adversarial cloud that returns the correct answers to the queries but wishes to know information about the encrypted sensitive tables, *Employee1* and *Employee2*.

Table 1 shows the adversary's view based on executing the corresponding Q_s and Q_{ns} components of the above three queries assuming that the tuple retrieving cryptographic approaches are not hiding access-patterns. During the execution, the adversary gains complete knowledge of non-sensitive tuples returned, and furthermore, knowledge about which encrypted tuples are returned as a result of Q_s ($E(t_i)$ in the table refers to the encrypted tuple t_i).

Given the above adversarial view, the adversary learns that employee `E259` has tuples in both D_s ($= D_e$) and D_p ($= D_{ns}$). Coupled with the knowledge about data partitioning, the adversary can learn that `E259` works in both sensitive and non-sensitive departments. Moreover, the adversary learns which sensitive tuple has an *Eid* equals to `E259`. From the 2nd query, the adversary learns that `E101` works only in a sensitive department, (since the query did not return any answer from the Employee3 relation). Likewise, from the 3rd query, the adversary learns that `E199` works only in a non-sensitive department.

In order to prevent such an attack, we need a new security definition. Before we discuss the formal definition of partitioned data security, we first provide

Table 1. Queries and returned tuples/adversarial view.

Query value	Returned tuples/adversarial view	
	Employee2	Employee3
E259	$E(t_4)$	t_2
E101	$E(t_1)$	Null
E199	Null	t_3

intuition for the definition. Observe that before retrieving any tuple, under the assumption that no one except the DB owner can decrypt an encrypted sensitive value, say $E(s_i)$, the adversary cannot learn which non-sensitive value is identical to cleartext value of $E(s_i)$; let us denote s_i as cleartext of $E(s_i)$. Thus, the adversary will consider that the value s_i is identical to one of the non-sensitive values. Based on this fact, the adversary can create a complete bipartite graph having $|S|$ nodes on one side and $|NS|$ nodes on the other side, where $|S|$ and $|NS|$ are a number of sensitive and non-sensitive values, respectively. The edges in the graph are called *surviving matches of the values*. For example, before executing any query, the adversary can create a bipartite graph for 4 sensitive and 4 non-sensitive values of `EID` attribute of Example 1; as shown in Fig. 11.

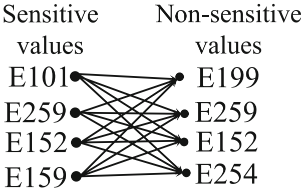


Fig. 11. A bipartite graph showing an initial condition sensitive and non-sensitive values before query execution.

The query execution on the datasets creates an adversarial view that guides the adversary to create a (new) bipartite graph of the same number of nodes on both sides. The requirement is to preserve all the edges of the initial bipartite graph in the graph obtained after the query execution, leading to the initial condition that the cleartext of the value $E(s_i)$ is identical to one of the non-sensitive values. Note that if the query execution removes any surviving matches of the values, it will leak that the value s_i is not identical to those non-sensitive values.

We also need to hide occurrences of a sensitive value. Before a query execution, due to ciphertext indistinguishability, all occurrences of a single sensitive value are different, but a simple search or join query may reveal how many tuples have the same value. Based on the above two requirements, we can define a notion of *partitioned data security*.

Partitioned Data Security at the Public Cloud

Let R be a relation containing sensitive and non-sensitive tuples. Let R_s and R_{ns} be the sensitive and non-sensitive relations, respectively. Let $q(R_s, R_{ns})[A]$ be a query, q, over an attribute A of the R_s and R_{ns} relations. Let X be the auxiliary information about the sensitive data, and Pr_{Adv} be the probability of the adversary knowing any information. A query execution mechanism ensures the partitioned data security if the following two properties hold:

- $Pr_{Adv}[e_i \overset{\text{a}}{=} ns_j|X] = Pr_{Adv}[e_i \overset{\text{a}}{=} ns_j|X, q(R_s, R_{ns})[A]]$, where $e_i = E(t_i)[A]$ is the encrypted representation for the attribute value A for any tuple t_i of the relation R_s and ns_j is a value for the attribute A for any tuple of the relation R_{ns}. The notation $\overset{\text{a}}{=}$ shows a sensitive value is identical to a non-sensitive value. This equation captures the fact that an initial probability of linking a sensitive tuple with a non-sensitive tuple will be identical after executing several queries on the relations.
- $Pr_{Adv}[v_i \overset{\text{r}}{\sim} v_j|X] = Pr_{Adv}[v_i \overset{\text{r}}{\sim} v_j|X, q(R_s, R_{ns})[A]]$, for all $v_i, v_i \in Domain(A)$. The notation $\overset{\text{r}}{\sim}$ shows a relationship between counts of the number of tuples with sensitive values. This equation states that the probability of adversary gaining information about the relative frequency of sensitive values does not increase by the query execution.

The definition above formalizes the security requirement of any partitioned computation approach. Of course, a partitioned approach, besides being secure, must also be correct in that it returns the same answer as that returned by the original query Q if it were to execute without regard to security.

4 Query Binning: A Technique for Partitioned Computations Using a Cryptographic Technique at the Public Cloud

In this section, we will study query binning (QB) as a partitioned computing approach. QB is related to bucketization, which is studied in past [25]. While bucketization was carried over the data in [25], QB performs bucketization on queries. In general, one may ask more queries than original query while adding overhead but it prevents the above-mentioned inference attack. We study QB under some assumption and setting, given below.[2].

Problem Setup. We assume the following two entities in our model: (i) *A database (DB) owner*: who splits each relation R in the database having attributes R_s and R_{ns} containing all sensitive and non-sensitive tuples, respectively. (ii) *A public cloud*: The DB owner outsources the relation R_{ns} to a public cloud. The tuples in R_s are encrypted using any existing mechanism before

[2] Some of these assumptions are made primarily for ease of the exposition and will be relaxed in [33].

outsourcing to the same public cloud. However, in the approach, we use non-deterministic encryption, *i.e.*, the cipher representation of two occurrences of an identical value has different representations.

DB Owner Assumptions. In our setting, the DB owner has to store some (limited) metadata such as searchable values and their frequency counts, which will be used for appropriate query formulation. The DB owner is assumed to have sufficient storage for such metadata, and also computational capabilities to perform encryption and decryption. The size of metadata is exponentially smaller than the size of the original data.

Adversarial Model. The adversary (*i.e.*, the untrusted cloud) is assumed to be honest-but-curious, which is a standard setting for security in the public cloud that is *not trustworthy.* An honest-but-curious adversarial public cloud, thus, stores an outsourced dataset without tampering, correctly computes assigned tasks, and returns answers; however, it may exploit side knowledge (*e.g.*, query execution, background knowledge, and the output size) to gain as much information as possible about the sensitive data. Furthermore, the adversary can eavesdrop on the communication channels between the cloud and the DB owner, and that may help in gaining knowledge about sensitive data, queries, or results. The adversary has full access to the following information: (*i*) all non-sensitive data outsourced in plaintext, and (*ii*) some *auxiliary* information of the sensitive data. The auxiliary information may contain the metadata of the relation and the number of tuples in the relation. Furthermore, the adversary can observe frequent query types and frequent query terms on the non-sensitive data in case of selection queries. The honest-but-curious adversary, however, cannot launch any attack against the DB owner.

Assumptions for QB. We develop QB initially under the assumption that queries are only on a single attribute, say A. The QB approach takes as inputs: (*i*) the set of data values (of the attribute A) that are sensitive along with their counts, and (*ii*) the set of data values (of the attribute A) that are non-sensitive, along with their counts. The QB returns a partition of attribute values that form the query bins for both the sensitive as well as for the non-sensitive parts of the query.

In this chapter, we also restrict to a case when a value has at most two tuples, where one of them must be sensitive and the other one must be non-sensitive, but both the tuples cannot be sensitive or non-sensitive. The scenario depicted in Example 1 satisfies this assumption. The *EId* attribute values corresponding to sensitive tuples include ⟨`E101`, `E259`, `E152`, `E159`⟩ and from the non-sensitive relation values are ⟨`E199`, `E259`, `E152`, `E254`⟩. Note that all the values occur only one time in one set.

Full Version. In this chapter, we restrict the algorithm for selection query only on one attribute. The full details of the algorithm, extensions of the algorithm for values having a different number of tuples, conjunctive, range, join, insert queries, and dealing with the workload-skew attack is addressed in [33]. Further, the computing cost analysis and efficiency analysis of QB at different or identical-levels of security against a pure cryptographic technique is given in [33].

The Approach. We develop an efficient approach to execute selection queries securely (preventing the information leakage as shown in Example 1) by appropriately partitioning the query at a public cloud, where sensitive data is cryptographically secure while non-sensitive data stays in cleartext. For answering a selection query, naturally, we use any existing cryptographic technique on sensitive data and a simple search on the cleartext non-sensitive data. Naturally, we can use a secure hardware, *e.g.*, Intel SGX, for all such operations; however, as mentioned in Sect. 1 Fig. 1, SGX-based processing takes a significant amount of time, due to limited space of the enclave.

Informally, QB distributes attribute values in a matrix, where rows are sensitive bins, and columns are non-sensitive bins. For example, suppose there are 16 values, say $0, 1, \ldots, 15$, and assume all the values have sensitive and associated non-sensitive tuples. Now, the DB owner arranges 16 values in a 4×4 matrix, as follows:

	NSB_0	NSB_1	NSB_2	NSB_3
SB_0	11	2	5	14
SB_1	10	3	8	7
SB_2	0	15	6	4
SB_3	13	1	12	9

In this example, we have four sensitive bins: SB_0 $\{11, 2, 5, 14\}$, SB_1 $\{10, 3, 8, 7\}$, SB_2 $\{0, 15, 6, 4\}$, SB_3 $\{13, 1, 12, 9\}$, and four non-sensitive bins: NSB_0 $\{11, 10, 0, 13\}$, NSB_1 $\{2, 3, 15, 1\}$, NSB_2 $\{5, 8, 6, 12\}$, NSB_3 $\{14, 7, 4, 9\}$. When a query arrives for a value, say 1, the DB owner searches for the tuples containing values 2,3,15,1 (viz. NSB_1) on the non-sensitive data and values in SB_3 (viz., 13, 1, 12, 9) on the sensitive data using the cryptographic mechanism integrated into QB. While the adversary learns that the query corresponds to one of the four values in NSB_1, since query values in SB_3 are encrypted, the adversary does not learn any sensitive value or a non-sensitive value that is identical to a clear-text sensitive value.

Formally, QB appropriately maps a selection query for a keyword w, say $q(w)$, to corresponding queries over the non-sensitive relation, say $q(W_{ns})(R_{ns})$, and encrypted relation, say $q(W_s)(R_s)$. The queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$, each of which represents a set of query values that are executed over the relation R_{ns} in plaintext and, respectively, over the sensitive relation R_s, using the underlying cryptographic method. The sets W_{ns} from R_{ns} and W_s from R_s are selected such that: (i) $w \in q(W_{ns})(R_{ns}) \cap q(W_s)(R_s)$ to ensure that all the tuples containing w are retrieved, and, (ii) the execution of the queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$ does not reveal any information (and w) to the adversary. The set of $q(W_{ns})(R_{ns})$ is entitled non-sensitive bins, and the set of $q(W_s)(R_s)$ is entitled

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
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 $NSB$
end
Function $allocateNS(SB[i][j])$ begin
    find a non-sensitive value associated with the $j^{th}$ sensitive value of the $i^{th}$ sensitive bin
end
```

sensitive bins. Algorithm 1 provides pseudocode of bin-creation method.[3] Results from the execution of the queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$ are decrypted, possibly filtered, and merged to generate the final answer.

Based on QB Algorithm 1, for answering the above-mentioned three queries in Example 1, given in Sect. 2, Algorithm 1 creates two sets or bins on sensitive parts: sensitive bin 1, denoted by SB_1, contains {E101, E259}, sensitive bin 2, denoted by SB_2, contains {E152, E159}, and two sets/bins on non-sensitive parts: non-sensitive bin 1, denoted by NSB_1, contains {E259, E254}, non-sensitive bin 2, denoted by NSB_2, contains {E199, E152}.

Table 2. Queries and returned tuples/adversarial view when following QB.

Query value	Returned tuples/adversarial view	
	Employee1	Employee2
E259	$E(t_4)$, $E(t_1)$	t_2, t_6
E101	$E(t_4)$, $E(t_1)$	t_3, t_8
E199	$E(t_4)$, $E(t_1)$	t_3, t_8

Algorithm 2 provides a way to retrieve the bins. Thus, by following Algorithm 2, Table 2 shows that the adversary cannot know the query value w or

[3] The function $approx\_sq\_factors$ in Algorithm 1 two factors x and y of a number n, such that either they are equal or close to each other so that the difference between x and y is less than the difference between any two factors of n (and $x \times y = n$).

Algorithm 2. Bin-retrieval algorithm.

```
Inputs: $w$: the query value.
Outputs: $SB_a$ and $NSB_b$: one sensitive bin and one non-sensitive bin to be
retrieved for answering $w$.
Variables: $found \leftarrow$ false
Function $retrieve\_bins(q(w))$ begin
    for $(i, j) \in (0, SB - 1), (0, |SB| - 1)$ do
        if $w = SB_i[j]$ then
            return $SB_i$ and $NSB_j$; $found \leftarrow$ true; break
        end
    end
    if $found \neq$ true then
        for $(i, j) \in (0, NSB - 1), (0, |NSB| - 1)$ do
            if $w = NSB_i[j]$ then
                return $NSB_i$ and $SB_j$; break
            end
        end
    end
    Retrieve the desired tuples from the cloud by sending encrypted values of
    the bin $SB_i$ (or $SB_j$) and clear-text values of the bin $NSB_j$ (or $NSB_i$) to
    the cloud
end
```

find a value that is shared between the two sets, when answering to the above-mentioned three queries. The reason is that the desired query value, w, is encrypted with other encrypted values of the set W_s, and, furthermore, the query value, w, is obscured in many requested non-sensitive values of the set W_{ns}, which are in cleartext. Consequently, the adversary is unable to find an intersection of the two sets, which is the exact value. Thus, while answering a query, the adversary cannot learn which employee works only in defense, design, or in both.

Correctness. The correctness of QB indicates that the approach maintains an initial probability of *associating* a sensitive tuple with a non-sensitive tuple will be identical after executing several queries on the relations.

We can illustrate the correctness of QB with the help of an example. The objective of the adversary is to deduce a clear-text value corresponding to an encrypted value of either {`E101, E259`} or {`E152, E159`}, since we retrieve the set of these two values. Note that before executing a query, the probability of an encrypted value, say E_i, (where E_i may be `E101`, `E259`, `E152`, or `E159`) to have the clear-text value is 1/4, which QB maintains at the end of a query. Assume that E_1 and E_2 are encrypted representations of `E101` and `E259`, respectively. Also, assume that v_1, v_2, v_3, v_4 are showing the cleartext value of `E259`, `E254`, `E199`, and `E152`, respectively.

When the query arrives for $\langle E_1, E_2, v_1, v_2 \rangle$, the adversary gets the fact that the clear-text representation of E_1 and E_2 cannot be v_1 and v_2 or v_3 and v_4.

If this will happen, then there is no way to associate each sensitive bin of the new bipartite graph with each non-sensitive bin. Now, if the adversary considers the clear-text representation of E_1 is v_1, then the adversary have four possible allocations of the values v_1, v_2, v_3, v_4 to E_1, E_2, E_3, E_4, such as: $\langle v_1, v_2, v_3, v_4\rangle$, $\langle v_1, v_2, v_4, v_3\rangle$, $\langle v_1, v_3, v_4, v_2\rangle$, $\langle v_1, v_4, v_3, v_2\rangle$.

Since the adversary is not aware of the exact clear-text value of E_1, the adversary also considers the clear-text representation of E_1 is v_2, v_3, or v_4. This results in 12 more possible allocations of the values v_1, v_2, v_3, v_4 to E_1, E_2, E_3, E_4. Thus, the retrieval of the four tuples containing one of the following: $\langle E_1, E_2, v_1, v_2\rangle$, results in 16 possible allocations of the values v_1, v_2, v_3, and v_4 to E_1, E_2, E_3, and E_4, of which only four possible allocations have v_1 as the clear-text representation of E_1. This results in the probability of finding $E_1 = v_1$ is 1/4.

Note that following this technique, executing queries under for each keyword will not eliminate any surviving matches of the bipartite graph, and hence, the adversary can find the new bipartite graph identical to a bipartite graph before the query execution. Figure 11 shows an initial bipartite graph before the query execution and Fig. 12 shows a bipartite graph after the query execution when creating bins on the values. Note that in Fig. 12 each sensitive bin is linked to each non-sensitive bin, that in turns, shows that each sensitive value is linked to each non-sensitive value.

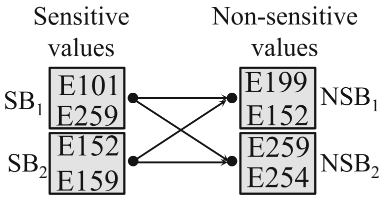


Fig. 12. A bipartite graph showing sensitive and non-sensitive bins after query execution, where each sensitive value gets associated with each non-sensitive value.

5 Effectiveness of QB

From the performance perspective, QB results in saving of encrypted data processing over non-sensitive data – the more the non-sensitive data, the more potential savings. Nonetheless, QB incurs overhead – it converts a single predicate selection query into a set of predicates selection queries over cleartext non-sensitive data, and, a set of encrypted predicates selection queries albeit over a smaller database consisting only of sensitive data. In this section, we compare QB against a pure cryptographic technique and show when using QB is beneficial.

For our model, we will need the following notations: (i) C_{com}: Communication cost of moving one tuple over the network. (ii) C_p (or C_e): Processing cost of a single selection query on plaintext (or encrypted data). In addition, we define three parameters:

α**:** is the ratio between the sizes of the sensitive data (denoted by S) and the entire dataset (denoted by $S + NS$, where NS is non-sensitive data).

β**:** is the ratio between the predicate search time on encrypted data using a cryptographic technique and on clear-text data. The parameter β captures the overhead of a cryptographic technique. Note that $\beta = C_e/C_p$.

γ**:** is the ratio between the processing time of a single selection query on encrypted data and the time to transmit the single tuple over the network from the cloud to the DB owner. Note that $\gamma = C_e/C_{com}$.

Based on the above parameters, we can compute the cost of cryptographic and non-cryptographic selection operations as follows:

$Cost_{plain}(x, D)$ is the sum the processing cost of x selection queries on plaintext data and the communication cost of moving all the tuples having x predicates from the cloud to the DB owner, *i.e.*, $x(\log(D)P_p + \rho DC_{com})$.

$Cost_{crypt}(x, D)$ is the sum the processing cost of x selection queries on encrypted data and the communication cost of moving all the tuples having x predicates from the cloud to the DB owner, i.e., $P_eD + \rho xDC_{com}$, where ρ is the selectivity of the query. Note that cost of evaluating x queries over encrypted data using techniques such as [20, 22, 41], is amortized and can be performed using a single scan of data. Hence, x is not the factor in the cost corresponding to encrypted data processing.

Given the above, we define a parameter η that is the ratio between the computation and communication cost of searching using QB and the computation and communication cost of searching when the entire data (viz. sensitive and non-sensitive data) is fully encrypted using the cryptographic mechanism.

$$\eta = \frac{Cost_{crypt}(|SB|, S)}{Cost_{crypt}(1, D)} + \frac{Cost_{plain}(|NSB|, NS)}{Cost_{crypt}(1, D)}$$

Filling out the values from above, the ratio is:

$$\eta = \frac{C_eS + |SB|\rho DC_{com}}{C_eD + \rho DC_{com}} + \frac{|NSB|\log(D)C_p + |NSB|\rho DC_{com}}{C_eD + \rho DC_{com}}$$

Separating out the communication and processing costs, η becomes:

$$\eta = \frac{S}{D}\frac{C_e}{C_e + \rho C_{com}} + \frac{|NSB|\log(D)C_p}{C_eD + \rho DC_{com}} + \frac{\rho DC_{com}(|NSB| + |SB|)}{C_eD + \rho DC_{com}}$$

Substituting for various terms and cancelling common terms provides:

$$\eta = \alpha\frac{1}{(1 + \frac{\rho}{\gamma})} + \frac{\log(D)}{D}\frac{|NSB|}{\beta(1 + \frac{\rho}{\gamma})} + \frac{\rho}{\gamma}\frac{|NSB| + |SB|}{(1 + \frac{\rho}{\gamma})}$$

Note that ρ/γ is very small, thus the term $(1 + \rho/\gamma)$ can be substituted by 1. Given the above, the equation becomes:

$$\eta = \alpha + \log(D)|NSB/D\beta + \rho(|NSB| + |SB|)/\gamma$$

Note that the term $\log(D)|NSB|/D\beta$ is very small since $|NSB|$ is the number of distinct values (approx. equal to $\sqrt{|NS|}$) in a non-sensitive bin, while D, which is the size of a database, is a large number, and β value is also very large. Thus, the equation becomes:

$$\eta = \alpha + \rho(|SB| + |NSB|)/\gamma$$

QB is better than a cryptographic approach when $\eta < 1$, *i.e.*, $\alpha + \rho(|SB| + |NSB|)/\gamma < 1$. Thus,

$$\alpha < 1 - \frac{\rho(|SB| + |NSB|)}{\gamma}$$

Note that the values of $|SB|$ and $|NSB|$ are approximately $\sqrt{|NS|}$, we can simplify the above equation to: $\alpha < 1 - 2\rho\sqrt{|NS|}/\gamma$. If we estimate ρ to be roughly $1/|NS|$ (i.e., we assume uniform distribution), the above equation becomes: $\alpha < 1 - 2/\gamma\sqrt{|NS|}$.

The equation above demonstrates that QB trades increased communication costs to reduce the amount of data that needs to be searched in encrypted form. Note that the reduction in encryption cost is proportional to α times the size of the database, while the increase in communication costs is proportional to $\sqrt{|D|}$, where $|D|$ is the number of distinct attribute values. This, coupled with the fact that γ is much higher than 1 for encryption mechanisms that offer strong security guarantees, ensures that QB almost always outperforms the full encryption approaches. For instance, the cryptographic cost for search using secret-sharing is ≈10 ms [20], while the cost of transmitting a single row (≈200 bytes for TPCH Customer table) is ≈4 µs making the value of $\gamma \approx 25000$. Thus, QB, based on the model, should outperform the fully encrypted solution for almost any value of α, under ideal situations where our assumption of uniformity holds. Figure 13 plots a graph of η as a function of γ, for varying sensitivity and $\rho = 10\%$.

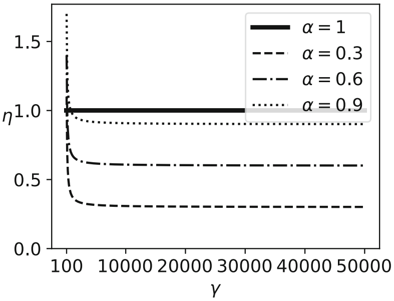


Fig. 13. Efficiency graph using equation $\eta = \alpha + \rho(|SB| + |NSB|)/\gamma$.

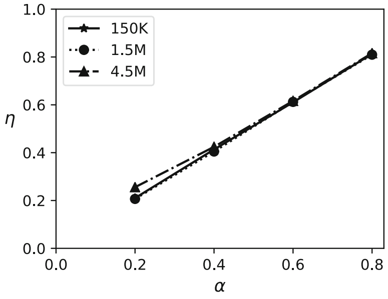


Fig. 14. Dataset size.

To explore the effectiveness of QB under different DB sizes, we tested QB for 3 DB sizes: 150K, 1.5M, and 4.5M tuples. Fig. 14 plots η values for the three sizes while varying α. The figure shows that $\eta < 1$, irrespective of the DB sizes, confirming that QB scales to larger DB sizes.

References

1. http://www.computerworld.com/article/2834193/cloud-computing/5-tips-for-building-a-successful-hybrid-cloud.html
2. https://digitalguardian.com/blog/expert-guide-securing-sensitive-data-34-experts-reveal-biggest-mistakes-companies-make-data
3. Agrawal, R., Kiernan, J., Srikant, R., Xu, Y.: Order-preserving encryption for numeric data. In: SIGMOD Conference, pp. 563-574. ACM (2004)
4. Arasu, A., et al.: Orthogonal security with cipherbase. In: CIDR. www.cidrdb.org (2013)
5. Arasu, A., Kaushik, R.: Oblivious query processing. In: ICDT, pp. 26–37. OpenProceedings.org (2014)
6. Bajaj, S., Sion, R.: Correctdb: SQL engine with practical query authentication. PVLDB **6**(7), 529–540 (2013)
7. Boyle, E., Gilboa, N., Ishai, Y.: Function secret sharing. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015. LNCS, vol. 9057, pp. 337–367. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46803-6_12
8. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
9. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Fragmentation and encryption to enforce privacy in data storage. In: Biskup, J., López, J. (eds.) ESORICS 2007. LNCS, vol. 4734, pp. 171–186. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-74835-9_12
10. Ciriani, V., De Capitani di Vimercati, S., Foresti, S., Jajodia, S., Paraboschi, S., Samarati, P.: Keep a few: outsourcing data while maintaining confidentiality. In: Backes, M., Ning, P. (eds.) ESORICS 2009. LNCS, vol. 5789, pp. 440–455. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04444-1_27
11. Ciriani, V., De Capitani, S., di Vimercati, S., Foresti, S., Jajodia, S.P., Samarati, P.: Combining fragmentation and encryption to protect privacy in data storage. ACM Trans. Inf. Syst. Secur. **13**(3), 22:1–22:33 (2010)

12. Costan, V., Devadas, S.: Intel SGX explained. IACR Cryptology ePrint Archive 2016:86 (2016)
13. Curtmola, R., Garay, J.A., Kamara, S., Ostrovsky, R.: Searchable symmetric encryption: improved definitions and efficient constructions. J. Comput. Secur. **19**(5), 895–934 (2011)
14. Derbeko, P., Dolev, S., Gudes, E., Sharma, S.: Security and privacy aspects in mapreduce on clouds: a survey. Comput. Sci. Rev. **20**, 1–28 (2016)
15. De Capitani di Vimercati, S., Erbacher, R.F., Foresti, S., Jajodia, S., Livraga, G., Samarati, P.: Encryption and fragmentation for data confidentiality in the cloud. In: Aldini, A., Lopez, J., Martinelli, F. (eds.) FOSAD 2012-2013. LNCS, vol. 8604, pp. 212–243. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-10082-1_8
16. De Capitani, S., di Vimercati, S., Foresti, S., Jajodia, G., Livraga, S.P., Samarati, P.: Fragmentation in presence of data dependencies. IEEE Trans. Dependable Sec. Comput. **11**(6), 510–523 (2014)
17. De Capitani, S., di Vimercati, S., Foresti, S., Jajodia, S.P., Samarati, P.: Fragments and loose associations: respecting privacy in data publishing. PVLDB **3**(1), 1370–1381 (2010)
18. Dolev, S., Gilboa, N., Li, X.: Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation: extended abstract. In: SCC@ASIACCS, pp. 21–29. ACM (2015)
19. Dolev, S., Li, Y., Sharma, S.: Private and secure secret shared MapReduce - (extended abstract). In: DBSec, pp. 151–160 (2016)
20. Emekçi, F., Metwally, A., Agrawal, D., El Abbadi, A.: Dividing secrets to secure data outsourcing. Inf. Sci. **263**, 198–210 (2014)
21. Gentry, C.: A fully homomorphic encryption scheme. Ph.D. thesis, Stanford University (2009)
22. Gilboa, N., Ishai, Y.: Distributed point functions and their applications. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 640–658. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-642-55220-5_35
23. Goldreich, O.: Towards a theory of software protection and simulation by oblivious RAMs. In: STOC, pp. 182–194. ACM (1987)
24. Goldreich, O.: The Foundations of Cryptography - Volume 2, Basic Applications. Cambridge University Press, Cambridge (2004)
25. Hacigümüs, H., Iyer, B.R., Li, C., Mehrotra, S.: Executing SQL over encrypted data in the database-service-provider model. In: SIGMOD Conference, pp. 216–227. ACM (2002)
26. Hacigümüs, H., Mehrotra, S., Iyer, B.R.: Providing database as a service. In: ICDE, pp. 29–38. IEEE Computer Society (2002)
27. Ishai, Y., Kushilevitz, E., Lu, S., Ostrovsky, R.: Private large-scale databases with distributed searchable symmetric encryption. In: Sako, K. (ed.) CT-RSA 2016. LNCS, vol. 9610, pp. 90–107. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-29485-8_6
28. Kellaris, G., Kollios, G., Nissim, K., O'Neill, A.: Generic attacks on secure outsourced databases. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, 24–28 October 2016, pp. 1329–1340 (2016)
29. Ko, S.Y., Jeon, K., Morales, R.: The HybrEx model for confidentiality and privacy in cloud computing. In: 3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud 2011, Portland, OR, USA, 14–15 June 2011 (2011)
30. Komargodski, I., Zhandry, M.: Cutting-edge cryptography through the lens of secret sharing. In: TCC, pp. 449–479 (2016)

31. Li, L., Militzer, M., Datta, A.: rPIR: ramp secret sharing based communication efficient private information retrieval. IACR Cryptology ePrint Archive 2014:44 (2014)
32. Lueks, W., Goldberg, I.: Sublinear scaling for multi-client private information retrieval. In: Böhme, R., Okamoto, T. (eds.) FC 2015. LNCS, vol. 8975, pp. 168–186. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47854-7_10
33. Mehrotra, S., Sharma, S., Ullman, J.D., Mishra, A.: Partitioned data security on outsourced sensitive and non-sensitive data. In: 34th IEEE International Conference on Data Engineering, ICDE 2019, Macau, China, April 08-12, 2019. Technical report, Department of Computer Science, University of California, Irvine (2018). http://isg.ics.uci.edu/pubs/tr/partitioned.pdf
34. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM J. Comput. **35**(5), 1254–1281 (2006)
35. Naveed, M., Kamara, S., Wright, C.V.: Inference attacks on property-preserving encrypted databases. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12-16 October 2015, pp. 644–655 (2015)
36. Oktay, K.Y., Kantarcioglu, M., Mehrotra, S.: Secure and efficient query processing over hybrid clouds. In: ICDE, pp. 733–744. IEEE Computer Society (2017)
37. Oktay, K.Y., Mehrotra, S., Khadilkar, V., Kantarcioglu, M.: SEMROD: secure and efficient MapReduce over hybrid clouds. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, 31 May–4 June 2015, pp. 153–166 (2015)
38. Popa, R.A., Redfield, C.M.S., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: SOSP, pp. 85–100. ACM (2011)
39. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptology ePrint Archive, 2005:187 (2005)
40. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979)
41. Song, D.X., Wagner, D.A., Perrig, A.: Practical techniques for searches on encrypted data. In: IEEE Symposium on Security and Privacy, pp. 44–55. IEEE Computer Society (2000)
42. Wang, S., Ding, X., Deng, R.H., Bao, F.: Private information retrieval using trusted hardware. IACR Cryptology ePrint Archive, 2006:208 (2006)
43. Zhang, C., Chang, E., Yap, R.H.C.: Tagged-MapReduce: a general framework for secure computing with mixed-sensitivity data on hybrid clouds. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing, CCGrid 2014, Chicago, IL, USA, 26–29 May 2014, pp. 31–40 (2014)
44. Zhang, K., Zhou, X., Chen, Y., Wang, X., Ruan, Y.: Sedic: privacy-aware data intensive computing on hybrid clouds. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, 17–21 October 2011, pp. 515–526 (2011)
45. Zheng, W., Dave, A., Beekman, J.G., Popa, R.A., Gonzalez, J.E., Stoica, I.: Opaque: an oblivious and encrypted distributed analytics platform. In: NSDI, pp. 283–298. USENIX Association (2017)
46. Li, Y., Mehrotra, S., Panwar, N., Sharma, S., Almanee, S.: Obscure: information-theoretic oblivious and verifiable aggregation queries. Technical report. Department of Computer Science, University of California, Irvine (2018). http://isg.ics.uci.edu/pubs/tr/Obscure.pdf