Partitioned Data Security on Outsourced Sensitive and Non-sensitive Data (Extended Abstract)

Sharad Mehrotra,[1] Shantanu Sharma,[1] Jeffrey D. Ullman,[2] and Anurag Mishra*[1]
[1]University of California, Irvine, USA. [2]Stanford University, USA.
sharad@ics.uci.edu, shantanu.sharma@uci.edu, ullman@gmail.com

ABSTRACT

Despite extensive research on cryptography, secure and efficient query processing over outsourced data remains an open challenge. This paper continues along the emerging trend in secure data processing that recognizes that the 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. Taking the cue from recent papers on hybrid clouds, this paper explores a new direction of work that attempts to exploit database techniques to secure selection queries. We propose a new secure selection query approach, entitled query binning (QB) that allows non-sensitive parts of the data to be outsourced in clear-text while guaranteeing that no information is leaked by the joint processing of non-sensitive data (in clear-text) and sensitive data (in encrypted form). Interestingly, besides improving performance, we show that QB actually strengthens the security of the underlying cryptographic technique by preventing size, frequency-count, and workload-skew attacks.

1. INTRODUCTION

The last two decades have witnessed the development of secure and privacy-preserving encryption-based [6, 13, 29, 34, 57, 21, 16, 31, 53, 35, 46, 9, 19, 36] or secret-sharing-based [55, 30, 14, 41, 23, 62, 27, 44] techniques to realize the database as a service model [34]. Despite significant progress, a cryptographic approach that is both *secure* (*i.e.*, no leakage of sensitive data to the adversary) and *efficient* (in terms of time) simultaneously has proved to be very challenging. Broadly, work on cryptography to support for secure outsourcing has taken the following directions:

1. Techniques that support strong security guarantees may not be efficient to be of a practical value. The leading example is fully homomorphic encryption [29] that mixed with oblivious-RAM (ORAM) [31] (ORAM) offers possibly amongst the most secure mechanisms. However, such mechanisms incur high overhead in terms of computation time.

Techniques	Time	Preventing attacks		
		Size	Workload-Skew	Access-patterns
Deterministic encryption	$1.43x$			
Non-deterministic encryption [51]	$2.1x$			
DSSE [36]	$3281x$			✓
SGX [18]	$6724x$			✓
Full-retrieval	$11135x$	✓	✓	✓
Homomorphic Encryption with ORAM	¿ $11135x$			✓

Table 1: Comparing different cryptographic technique in terms of time (for selection queries over TPC-H data) and attacks. Details of attacks are presented in §2. x is the time to search a predicate in cleartext. ✓is showing a technique is not prone to a given attack.

2. Techniques that do not depend on the data encryption may provide strong security, especially, information-theoretical security, by distributing a value of the form of the secret-shares to non-colluding clouds. Shamir's secret-sharing [55], distributed point functions [30], function secret-sharing [14], accumulating-automata [23] are a few examples of such techniques. Such methods often limit the type of operations one can do while imposing high overhead in terms of communication.
3. Techniques that try to support a wide range of operations including index-based retrieval or joins, such as CryptDB [52], Arx [51], searchable encryption [57], and [26, 56] do not provide strong security due to either dependence on deterministic and order-preserving encryptions [6], traversal of the index by the cloud, or leakage of the searching token. Further, the papers [47, 38] 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 data in clear-text by doing a frequency analysis on the encrypted data.
4. Techniques/systems that exploit secure hardware (Intel Software Guard Extensions (SGX) [18]), *e.g.*, M2R [22], VC3 [54], and Opaque [69], also leak information during a query execution [48] and are inefficient (in terms of time) due to a limited available memory of SGX.

Table 1 summarizes different techniques based on efficiency and security. Note that none of the above-mentioned techniques are completely secure against each attack mentioned in the table, except the full retrieval of the database from the public cloud to the trusted private side.

Given the state of the research, this paper explores a radically different approach to secure outsourcing that scales cryptographic mechanisms using database techniques while providing strong security guarantees. Our work is motivated by recent works on the hybrid cloud that has exploited the fact that for a large class of

*A. Mishra contributed only in implementing the proposed algorithm.

Proceedings of the VLDB Endowment, Vol. 11, No. 5

DOI: https://doi.org/TBD

application contexts, data can be partitioned into sensitive and non-sensitive components. Such a classification was exploited to build hybrid cloud solutions [40, 68, 67, 50, 49] 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). While these techniques provide an effective and secure solution, they are, however, based on a hybrid cloud, requiring data owners to maintain potentially unbounded storage and also suffer from significant inter-cloud communication overheads.

Our goal, in this paper, is to explore how sensitive and non-sensitive classification can be exploited by secure data processing techniques that store data in the public cloud to bring new efficiencies to secure data processing. In particular, in the envisioned model, data is stored in a partitioned way – sensitive data is secured using any existing cryptographic technique and non-sensitive data resides in plaintext, and the query processing is also split into encrypted and plaintext query processing. We refer to this as *partitioned computing*. Unlike the case of the hybrid cloud, when implementing partitioned computing in the public cloud, data processing performed on the sensitive and non-sensitive parts of the data reveals exact encrypted tuples (depending on an underlying cryptographic technique) and cleartext tuples that satisfy the query to the adversary. Consequently, this leads to inferences about sensitive data, which will be explained in detail in §3.

We first define a security model (§4) that formally states what it means to be secure in partitioned computing. We then develop a query binning (QB) approach that realizes secure partitioned computing for selection queries. Selection queries are the most important part for us, since many cryptographic techniques [13, 29, 34, 57, 21, 16, 31, 53, 35, 46, 30] are developed for selection queries and there is no cryptographic secure technique for join. While our technique is designed to improve the performance of strong cryptographic techniques that offer strong security property over a large-scale dataset, we show there is an interesting side effect of using QB with a weak cryptographic technique, such as cloud-side indexable techniques (referred to as C-Ind in this paper) [56, 26, 51]. In particular, *QB provides an enhanced security by preventing several attacks such as output size, frequency-count, and workload-skew attacks, even when the underlying cryptographic technique is susceptible to such attacks*. Note that QB is not designed to prevent any attack on a cryptographic technique and involved leakage [37, 47, 48, 15, 38, 43, 32, 12, 33].

The primary contributions of this paper are listed below:

1. A formal definition of *partitioned data security* in the presence of a joint processing of sensitive and non-sensitive data (in §4).
2. An efficient QB approach (in §5) that guarantees partitioned security (Theorems 1 and 2), supporting insert operation (§A), cloud-side-indexes (§6), and can be built on top of any cryptographic technique.
3. An analytical formal model to compare QB with a pure cryptographic technique under different conditions and different security levels such as preventing size, frequency-count, and workload-skew attacks (§6).
4. C-Ind-based QB approach that is both efficient and secure (§6.4).
5. Experiments to validate an integration of QB with existing cryptographic techniques on top of secure databases.[1]

2. RELATED WORK

[1]The QB's performance is evaluated on two well-known systems A and B; due to legal restrictions, the real names of the systems A and B are omitted.

Broadly, existing research on secure selection query execution techniques can be classified into four categories:

Encryption-based techniques examples of which include order-preserving encryption [6], deterministic encryption [13], homomorphic encryption [29], bucketization [34], searchable encryption [57, 21], private information retrieval (PIR) [16], practical-PIR [63], oblivious-RAM (ORAM) [31], oblivious transfers [53, 35], oblivious polynomial evaluation [46], oblivious query processing [9], searchable symmetric encryption [19], and distributed searchable symmetric encryption (DSSE) [36].

Secret-sharing [55] based techniques that include distributed point function [30], function secret sharing [14], functional secret sharing [41], accumulating-automata [23], Splinter [62], and others [27, 44].

Trusted-hardware-based techniques that include [8, 7, 10, 54, 22, 60, 69].

Sensitivity-based techniques. The papers [40, 68, 67, 50] have explored secure MapReduce (MR) system implementations while [49] have explored secure SQL data processing. Both the secure MR and secure SQL execution solutions work on the principle of sensitivity-based data partitioning over the hybrid cloud. In [69], secure hardware, specifically Intel SGX [18], at the cloud is used to partition the computation.

Each of the above strategies has resulted in corresponding systems that support secure data processing. For instance, CryptDB [52], Monomi [61], TrustedDB [11], CorrectDB [10], SDB [64], ZeroDB [25], L-EncDB [42], MrCrypt [59], Crypsis [58], Arx [51], and Opaque [69] are some novel encryption-based systems. Likewise, Cypherbase [7], Microsoft Always Encrypted, Oracle 12c, Amazon Aurora [1], and MariaDB [2] are industrial secure encrypted databases. DSSE-based SDB [3] is a secret-sharing and encryption-based system while Arx [51] and Opaque [69] work on the data sensitivity principle.

Moreover, these systems/techniques are unable to prevent one or more of the following attacks: (*i*) size attack, *i.e.*, an adversary having some background knowledge can deduce the full/partial outputs by simply observing the output sizes [69]; (*ii*) frequency attack, *i.e.*, how many tuples have an identical value [47]; (*iii*) workload-skew attack, *i.e.*, an adversary, having the knowledge of frequent query selection predicates by observing many queries, can estimate which encrypted tuples potentially satisfy the frequent selection predicates; (*iv*) access-pattern attack, *i.e.*, encrypted tuples that *exactly* satisfy the query [16]. Note that computationally expensive and access-pattern hiding cryptographic techniques (*e.g.*, PIR, ORAM, DSSE, oblivious transfer, and secret-sharing) can prevent the size, frequency-count, and workload-skew attacks *only* on *non-skewed and non-deterministically encrypted* datasets. In contrast, to the best of our knowledge, there is no cryptographic technique that prevents all the four attacks on a *skewed dataset*. It is important to mention that QB mixed with a weak cryptographic technique [56, 26, 51] is efficient and secure against size and workload-skew attacks. This fact will be clear by performance analysis in §7 (Figure 7f).

In addition, we assume the data is partitioned, which is a common practice in organizations [4, 5], appropriately using existing techniques [28, 24] such that sensitive data is not revealed from non-sensitive data. Nevertheless, QB can prevent inference attacks on two databases while one of them is encrypted, mentioned in [20]. For hiding an exact selection predicate over an encrypted relation regardless of data sensitivity, an approach to create a set of selection predicates including the exact predicate is presented in [43], which, however, cannot be used to search over sensitive and non-sensitive relations or multiple relations [20], due to not dealing

Table 2: Comparison of different algorithms with our algorithms.

Algorithms	Storage		Computational cost for search		Insert cost	# rounds	Size attack	Workload-skew and frequency attacks	Data sensitivity
	Cloud	DB Owner	Cloud	DB Owner					
Encryption-based *indexable* solutions									
CryptDB* [52]	D	0	$\mathcal{O}(1)$	$\mathcal{O}(r)$	$\mathcal{O}(1)$	1	Y	Y	N
Searchable encryption [21]	$2D$ or $4D$	0	$D^{1/3}\log D$ or $D^{1/5}\log D$	$\mathcal{O}(r)$	NA	1	Y	Y	N
Arx* [51]	D	V_D	$\mathcal{O}(r\cdot\log D)$	$\mathcal{O}(r)$	$\mathcal{O}(1)$	1	Y	Y	N
Hybrid-Secure** [49]	D	αD	$\mathcal{O}(r+\log((1-\alpha)D))$	$\mathcal{O}(\log(\alpha D)+r)$	$\mathcal{O}(1)$	1	N	N	Y
Our solution with indexes**	**D**	$\mathbf{V_D}$	$\mathcal{O}(\sqrt{\lvert NS\rvert}r\log(\alpha D)+\sqrt{\lvert NS\rvert}(r+\log(1-\alpha)D)$	$\mathcal{O}(r\sqrt{\lvert S\rvert})$	$\mathcal{O}(1)$	**1**	**N**	**N**	**Y**
Encryption-based *non-indexable* solutions									
Searchable Encryption [57]	D	0	D	$\mathcal{O}(1)$	$\mathcal{O}(1)$	1	Y	Y	Y
Our solution without indexes**	**D**	$\mathbf{V_D}$	$\mathcal{O}(\sqrt{\lvert NS\rvert}\alpha D)+\sqrt{\lvert NS\rvert}(r+\log(1-\alpha)D)$	$\mathcal{O}(r\sqrt{\lvert S\rvert})$	$\mathcal{O}(1)$	**1**	**N**	**N**	**Y**

Notations. $*$: Non-secure systems, due to dependence on deterministic and order-preserving encryption (CryptDB) and at the time of selection and join operations (Arx). $**$: fully secure systems. Y: Yes. N: No. D: a database (containing only a single attribute, for simplicity). $V_D \ll D$: # unique values in the attribute that is equal to $|S|+|NS|$ in our case, where $|S|$: unique sensitive values, $|NS|$: unique non-sensitive values, and $|S|<|NS|$. r: # occurrences of a searching value. $\alpha<1$: be a ratio between the sizes of sensitive data and the entire dataset.

with the inference attacks. The paper [17] deals with column-level sensitivity by encrypting only sensitive columns; however, they are susceptible to reveal sensitive information based on background knowledge (an attack like presented in [45]), since the relation is not partitioned into two parts based on sensitivity. However, QB can also prevent such a type of attack by partitioning a relation.

Table 2 presents a comparison of different techniques. Note that both QB and the most efficient but insecure indexable technique, (Arx [51]) store an identical amount of metadata at the DB owner. However, QB makes Arx completely secure against the size, frequency-count, and workload-skew attacks. However, the computational cost of QB is $\sqrt{|NS|}$ times higher than Arx, because of searching $\sqrt{|NS|}$ more predicates on encrypted data. Nevertheless, QB performs better than a recent non-indexable technique (for example, searchable encryption [30]) in terms of the number of rounds to retrieve all the occurrences of a predicate, time, and security levels. Table 2, also, clarifies that QB is better than a recent searchable encryption [21] by allowing dynamic operations, *e.g.*, insert, and stronger security guarantees.

3. PARTITION COMPUTATION

In this section, we first define more precisely what we mean by partitioned computing, 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. Table 3 enlists notations used in this paper.

The Partition Computation Model

We assume the following two entities in our model:

1. *A trusted database (DB) owner*: who divides a relation R having attributes, say $A_1, A_2, \ldots, A_n$, into the following two relations based on row-level data sensitivity: R_s and R_{ns} containing all sensitive and non-sensitive tuples, respectively.[2] The DB owner outsources the relation R_{ns} to a public cloud. The tuples of the relation R_s are encrypted using any existing mechanism before 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*.
   In our setting, the DB owner has to store 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 smaller than the size of the original data.
2. *The untrusted public cloud*: that stores the databases, executes queries, and provides answers.

[2]QB can also deal with column-level sensitivity, where sensitive and non-sensitive relations have different attributes. However, both the relations have an identical searchable attribute.

Notations	Meaning
$\lvert S\rvert$	Number of sensitive data values
$\lvert NS\rvert$	Number of non-sensitive data values
R_s	Sensitive parts of a relation R
R_{ns}	Non-sensitive parts of a relation R
s_i and ns_j	i^{th} sensitive and j^{th} non-sensitive values
SB	The number of sensitive bins
SB_i	i^{th} sensitive bin
$\lvert SB\rvert = y$	Sensitive values in a sensitive bin or the size of a sensitive bin
NSB	The number of non-sensitive bins
NSB_i	i^{th} non-sensitive bin
$\lvert NSB\rvert = x$	Non-sensitive values in a non-sensitive bin or the size of a non-sensitive bin
$q(w)$	A query, q, for a predicate w
$q(W_{ns})(R_{ns})$	A query, q, for a set, W_{ns}, of predicates in clear-text over R_{ns}
$q(W_s)(R_s)$	A query, q, for a set, W_s, of predicates in encrypted form over R_s
$q(W)(R_s, R_{ns})[A]$	A query, q, for a set, W, of values, searching on the attribute, A, of the relations R_s and R_{ns}, where $W = W_s \cup W_{ns}$
$E(t_i)$	i^{th} encrypted tuple

Table 3: Notations used in the paper.

Let us consider a query q over the relation R, denoted by $q(R)$. A partition computation strategy splits the execution of q into two independent sub-queries: $q(R_s)$: a query to be executed on the encrypted sensitive relation R_s, and $q(R_{ns})$: a query to be executed on the non-sensitive relation R_{ns}. The final results are computed (using a query q_{merge}) by appropriately merging the results of the two sub-queries at the DB owner side. In particular, the query q on a relation R is partitioned, as follows:

$$q(R) = q_{merge}\Big(q(R_s), q(R_{ns})\Big)$$

Let us illustrate partitioned computations through an example.

	EId	First name	Last name	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

Figure 1: A relation: *Employee*.

Example 1. Consider an *Employee* relation, see Figure 1. 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 following three relations: (*i*) *Employee1* with attributes *EId* and *SSN*; (*ii*) *Employee2* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* = "`Defense`"; and (*iii*) *Employee3* with attributes *EId*, *FirstName*, *LastName*, *Office#*, and *Department*, where *Department* <> "`Defense`". Since the relations *Employee1* and *Employee2* contain only sensitive data, these two relations are encrypted before outsourcing, while *Employee3*, 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* is achieved. Thus, the two occurrences of `E152` have two different ciphertexts.

Consider a query `q: SELECT FirstName, LastName, Office#, Department from Employee where FirstName = John`. In the 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 operator. Note that the execution of the query `q` will also retrieve the same tuples.

However, such a partitioned computation, if performed naively, leads to inferences about sensitive data from non-sensitive data. Before discussing inference attacks on the sensitive data from non-sensitive data, first, we present the adversarial model.

Adversarial Model

In our setting, recall that sensitive data is encrypted while non-sensitive data resides in clear-text. We assume an honest-but-curious adversary, which is considered in the standard setting for security in the public cloud [39, 65, 66] 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. The honest-but-curious adversary cannot launch any attack against the DB owner.[3] Furthermore, the honest-but-curious 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:

1. The full information of the non-sensitive data. For example, for the Employee relation in Example 1, an adversary knows the complete *Employee3* relation.
2. *Auxiliary* information of the sensitive data. The auxiliary information may contain metadata, schema of the relation, and the number of tuples in the relation.
   For example, the adversary knows that there are two sensitive relations, one of them containing four tuples and the other one containing two tuples, in the *Employee1* and the *Employee2* relations; refer to Example 1. The adversary is not aware of the following information before the query execution that how many people work in a specific sensitive department, is a specific person working only in a sensitive department, only in a non-sensitive department, or both.
3. Adversarial view. The adversarial view contains which encrypted sensitive tuples and cleartext non-sensitive tuples are sent in response to a query. Example 2 will illustrate the adversarial view.
4. Some frequent query values. The adversary observes query predicates on the non-sensitive data, and hence, can deduce the most frequent query predicates by observing many queries.

Inference Attacks in Partitioned Computations

To see the inference attack on the sensitive data while jointly processing sensitive and non-sensitive data, consider following three queries on the *Employee2* and *Employee3* relations.

Example 2. (*i*) retrieve tuples corresponding to employee `E259`, (*ii*) retrieve tuples corresponding to employee `E101`, and (*iii*) retrieve tuples corresponding to employee `E199`.[4] When answering a query, the adversary knows the tuple ids of retrieved encrypted tuples and the full information of the returned non-sensitive tuples. We refer to this information gain at the adversary by observing query execution as the *adversarial view*, shown in Table 4, where $E(t_i)$ denotes an encrypted tuple t_i.

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

Table 4: Queries and returned tuples/adversarial view.

Outputs of the above three queries will reveal enough information to learn something about sensitive data. In the first query, the adversary learns that `E259` works in both sensitive and non-sensitive departments, because the answers obtained from the two relations contribute to the final answer. Moreover, the adversary may learn that which sensitive tuple has an *Eid* equals to `E259`. In the second query, the adversary learns that `E101` works only in a sensitive department, because the query will not return any answer from the Employee3 relation. In the third query, the adversary learns that `E199` works only in a non-sensitive department.

The Query Binning (QB) Approach

In order to prevent such an attack, we need a new security definition. Before we discuss the formal definition of partitioned data security (in §4), we first provide a possible solution for the inference attack prevention and then intuition for the security definition.

The query binning (QB) stores a non-sensitive relation, say R_{ns}, in clear-text while a sensitive relation, say R_s, using a cryptographically secure approach. QB prevents leakage such as in Example 2 by appropriately mapping a query for a predicate, say $q(w)$, to corresponding queries both 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

[3] We do not consider cyber-attacks that can exfiltrate data from the DB owner directly, since defending against generic cyber-attacks is beyond the scope of this paper.

[4] We used random *Eids*, which is also common in a real employee relation. In contrast, in sequential ids, the absence of an id from the non-sensitive relation directly informs to the adversary that the given id exists in the sensitive relation.

set of predicates that are executed over the relation R_{ns} in plaintext and, respectively, over the sensitive relation R_s, using the underlying cryptographic method. The set of predicates in $q(W_{ns})(R_{ns})$ ($q(W_S)(R_s)$) correspond to the non-sensitive (sensitive) *bins* including the predicate w, denoted by NSB (SB). The predicates in $q(W_s)(R_s)$ are encrypted before transmitting to the cloud.

The bins 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 the predicate w are retrieved, and, (*ii*) joint execution of the queries $q(W_{ns})(R_{ns})$ and $q(W_s)(R_s)$ (hereafter, denoted by $q(W)(R_s, R_{ns})$, where $W = W_s \cup W_{ns}$) does not leak the predicate w. 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. Note that *bins are created only once for all the values of a searching attribute before any query is executed*. The details of the bin formation will be discussed in §5.

Thus, by following QB, Table 5 shows that the adversary cannot know the query value w or 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 W_s, and, furthermore, the query value, w, is obscured in many requested non-sensitive values of W_{ns}, which are in clear-text. Consequently, *the adversary is unable to find an intersection of the two sets, which is the exact value*.

Query value	Returned tuples/Adversarial view	
	Employee2	Employee3
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

Table 5: Queries and returned tuples/adversarial view, following QB.

For answering the above-mentioned three queries, QB creates two bins on sensitive parts: {E101, E259}, {E152, E159}, and two sets on non-sensitive parts: {E259, E254}, {E199, E152}. Note that here while answering a query, the adversary cannot learn which employee works only in defense, design, or in both.

4. PARTITIONED DATA SECURITY

In our model, *the goal of the adversary is to find as much sensitive information as possible (using the adversarial view or background knowledge)*, and *our goal is to prevent the information leakage through non-sensitive data, the size, frequency-count, and workload-skew attacks on the sensitive data*. We first assume the following two conditions: (*i*) the sensitive data is strongly encrypted such that the property of ciphertext indistinguishability is achieved; (*ii*) uniform queries, *i.e.*, the DB owner asks queries for each sensitive and non-sensitive value. We assume a uniform query distribution for simplicity, and later in §A, this assumption will be relaxed. Now, to define a notion of *partitioned data security*, we first define the following three terms: *associated values*, *associated tuples*, and *relationship between counts of sensitive values*.[5]

Notations used in the definitions. Let $t_1, t_2, \ldots, t_m$ be tuples of a sensitive relation, say R_s. Thus, the relation R_s stores the encrypted tuples $E(t_1), E(t_2), \ldots, E(t_m)$. Let $s_1, s_2, \ldots, s_{m'}$ be values of an attribute, say A, that appears in one of the sensitive tuples of R_s. Note that $m' \leq m$ (since several tuples may have an identical value) and, furthermore, $s_i \in Domain(A)$, $i = 1, 2, \ldots, m'$, where $Domain(A)$ represents the domain of values the attribute A can take. By $\#_s(s_i)$, we refer to the number of sensitive tuples that have the value of the attribute A to be s_i. We further define $\#_s(v_i) = 0, \forall v_i \in Domain(A), i \notin s_1, s_2, \ldots, s_{m'}$. Let $t_1, t_2, \ldots, t_n$ be tuples of a non-sensitive relation, say R_{ns}. Let $ns_1, ns_2, \ldots, ns_{n'}$ be values of the attribute A that appears in one of the non-sensitive tuples of R_{ns}. As before, $n' \leq n$, and $ns_i \in Domain(A)$, $i = 1, 2, \ldots, n'$.

Associated values. Let $e_i = E(t_i)[A]$ be the encrypted representation of an attribute value of A in a sensitive tuple of the relation R_s, and ns_j be a value of the attribute A for some tuple of the relation R_{ns}. We say that e_i is *associated* with ns_j, (denoted by $\stackrel{a}{=}$), if the plaintext value of e_i is identical to the value ns_j. In Example 1, the value of the attribute Eid in tuple t_4 (of *Employee2*) is associated with the value of the attribute Eid in tuple t_2 (of *Employee3*), since both values correspond to E259.

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

Relationship between counts of sensitive values. Let v_i and v_j be two distinct values in $Domain(A)$. We denote the relationship between the counts of sensitive tuples with these A values (*i.e.*, $\#_s(v_i)$ (or $\#_s(v_j)$)) by $v_i \stackrel{r}{\sim} v_j$. Note that $\stackrel{r}{\sim}$ can be one of $<, =$, or $>$ relationships. For instance, in Example 1, the E101 $\stackrel{r}{\sim}$ E259 corresponds to $=$, since both values have exactly one sensitive tuple, while E101 $\stackrel{r}{\sim}$ 199 is $>$, since there is one sensitive tuple with value E101 while there are zero records with E199.

Given the above definitions, we can now formally state the security requirement that ensures that simultaneous execution of queries over sensitive (encrypted) and non-sensitive (plaintext) data does not leak any information.

Definition: Partitioned Data Security. 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 $q(W)(R_s, R_{ns})[A]$ be a query, q, for an attribute value $w \in W = W_s \cup W_{ns}$ (where W_s (W_{ns}) contains some encrypted sensitive (clear-text non-sensitive) values) in the 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:

(01) $Pr_{Adv}[e_i \stackrel{a}{=} ns_j|X] = Pr_{Adv}[e_i \stackrel{a}{=} ns_j|X, q(W)(R_s, R_{ns})[A]]$, 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}.

(02) $Pr_{Adv}[v_i \stackrel{r}{\sim} v_j|X] = Pr_{Adv}[v_i \stackrel{r}{\sim} v_j|X, q(W)(R_s, R_{ns})[A]]$, for all $v_i, v_i \in Domain(A)$.

The first equation (01) captures the fact that an initial probability of *associating* a sensitive tuple with a non-sensitive tuple will be identical after executing several queries on the relations. The second equation (02) states that the probability of adversary gaining information about the relative frequency of sensitive values does not increase by the query execution. In Example 2, an execution of any three queries (for values 101, 199, or 259) without using QB does not satisfy the above first equation. For example, the query for 199 retrieves the only tuple from non-sensitive relation, and

[5]To develop the notation, defining security, and developing QB (in §5), we assume that search is performed on a specific attribute, A, over a relation, R. The approach trivially generalizes when several attributes are searchable – we need to maintain metadata required for QB not just for A, but for all searchable attributes in R.

that changes the probability of estimating whether 199 is sensitive or non-sensitive to 0 as compared to an initial probability of the same estimation, which was 1/2. Hence, an execution of the three queries violates partitioned data security. However, the query execution for 259 and 101 satisfies the second equation, since the count of returned tuples from *Employee2* is equal. Hence, the adversary cannot distinguish between the count of the values (259 and 101) in the domain of Eid of *Employee2* relation.

We also define algorithm correctness, as follows:

Correctness. Recall that as mentioned in §1, a query on an identical attribute of the relations R_s and R_{ns} for a value, say w, is correct if it returns all the sensitive and non-sensitive tuples containing w that are identical to the returned tuples in response to a query executed only on the relation R.

Definition: Algorithm correctness. Let $Result$[6] be a function for computing the final output. An algorithm is correct if it returns tuples containing the value $w \in W$ in the attribute A of the R_s and R_{ns} relations, where the output tuples are identical to an answer to the query, $q(w)(R)[A]$, execution only on the attribute A of the relation R:

$$q(w)(R)[A] \equiv Res[q(W)(R_s, R_{ns})[A]]$$

5. QUERY BINNING TECHNIQUE

We develop our strategy 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. We begin in §5.1 by developing the approach for the case when a sensitive tuple is associated with at most one non-sensitive tuple (Algorithm 1). We then (in §5.2) develop a simple extension of Algorithm 1 to deal with a situation where the number of non-sensitive (or sensitive) values is close to a square number.[7] Finally, we provide a general strategy to create bins when a sensitive tuple is associated with several non-sensitive tuples, in §5.3.

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. We will show, in proposed approach, 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.

[6]In our context, the function $Result$ will decrypt sensitive tuples, merge sensitive-non-sensitive tuples, and filter extra tuples for providing the final outputs.

[7]The rationale for explaining the strategy first for any number of non-sensitive values and then explaining it to a case of non-sensitive values close/equal to a square number will become clear shortly.

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

5.1 The Base Case

The QB consists of two steps. First, query bins are created (information about which will reside at the DB owner) using which queries will be rewritten. The second step consists of rewriting the query based on the binning.

Here, QB is explained for the base case, where a sensitive tuple, say t_s, is associated with at most a single non-sensitive tuple, say t_{ns}, and vice versa (*i.e.*, $\stackrel{a}{=}$ is a 1:1 relationship). Thus, if the value has two tuples, then one of them must be sensitive and the other one must be non-sensitive, but both the tuples cannot be sensitive or non-sensitive. A value can also have only one tuple that must be either sensitive or non-sensitive. Note that if $t_1, t_2, \ldots, t_l$ are sensitive tuples, with values of an attribute A being $s_1, s_2, \ldots s_n$, $s_i <> s_j$ if $i <> j$. Thus, in the remainder of the section, we will refer to association between encrypted value $E(t_i)[A]$ and a non-sensitive value ns_j simply as an association between values s_i and ns_j. That is, $s_i \stackrel{a}{=} ns_j$ represents $E(t_i)[A] \stackrel{a}{=} ns_j$.

The scenario depicted in Example 1 satisfies the base case. The *EId* attribute values corresponding to sensitive tuples include ⟨E101, E259⟩ and from the non-sensitive relation values are ⟨E199, E259⟩ for which $\stackrel{a}{=}$ is 1:1. We discuss QB under the above assumption, but relax the assumption in §5.3. Before describing QB, we first define the concept of *approximately square factors of a number*.

Approximately square factors. *We say two numbers, say x and y, as* approximately square factors of a number*, say $n > 0$, if they are equal or close to each other such that the difference between x and y is less than the difference between any two factors of n (and $x \times y = n$).*

Step 1: Bin-creation. The QB, described in Algorithm 1, finds two approximately square factors of $|NS|$, say x and y, where $x \geq y$. The QB creates $SB = x$ sensitive bins, where each sensitive bin contains at most y values. Thus, we assume $|S| \geq x$. The QB, further, creates $NSB = \lceil |NS|/x \rceil$ non-sensitive bins, where each non-sensitive bin contains at most $|NSB| = x$ values. Note that we are assuming that $|S| \leq |NS|$.[8]

[8]The QB can also handle the case of $|S| > |NS|$ by applying Algorithm 1 in a reverse way, *i.e.*, factorizing $|S|$.

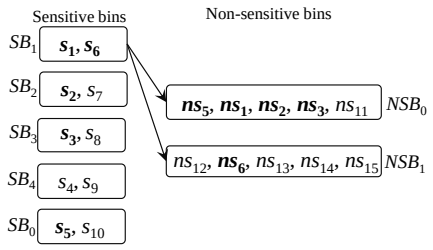


Figure 2: The QB for 10 sensitive and 10 non-sensitive values.

Assignment of sensitive values. We number the sensitive bins from 0 to $x-1$ and the values therein from 0 to $y-1$. To assign a value to sensitive bins, QB first permutes the set of sensitive values. Such a permutation is kept secret from the adversary by the DB owner.[9] In order to assign sensitive values to sensitive bins, QB takes the i^{th} sensitive value and assign it to the i *modulo* x sensitive bin (see Lines 2 and 5 of Algorithm 1).

Assignment of non-sensitive values. We number the non-sensitive bins from 0 to $\lceil |NS| \rceil / x - 1$ and values therein from 0 to $x-1$. In order to assign non-sensitive values, QB takes a sensitive bin, say j, and its i^{th} sensitive value. Assign the non-sensitive value associated with the i^{th} sensitive value to the j^{th} position of the i^{th} non-sensitive bin. Here, if each value of a sensitive bin has an associated non-sensitive value and $|S| = |NS|$, then QB has assigned all the non-sensitive values to their bins (Line 6 of Algorithm 1). Note that it may be the case that only a few sensitive values have their associated non-sensitive values and $|S| \leq |NS|$. In this case, we assign the sensitive and their associated non-sensitive values to bins like we did in the previous case. However, we need to assign the non-sensitive values that are not associated with a sensitive value, by filling all the non-sensitive bins to size x (Line 7 of Algorithm 1).

Example 3: (Query binning example). We show the bin-creation algorithm for 10 sensitive values and 10 non-sensitive values. We assume that only five sensitive values, say s_1, s_2, s_3, s_5, s_6, have their associated non-sensitive values, say $ns_1, ns_2, ns_3, ns_5, ns_6$, and the remaining 5 sensitive (say, $s_4, s_7, s_8, \ldots s_{10}$) and 5 non-sensitive values (say, $ns_{11}, ns_{12}, \ldots, ns_{15}$) are not associated. For simplicity, we use different indexes for non-associated values.

The QB creates 2 non-sensitive bins and 5 sensitive bins, and divides 10 sensitive values over the following 5 sensitive bins: SB_0 $\{s_5, s_{10}\}$, SB_1 $\{s_1, s_6\}$, SB_2 $\{s_2, s_7\}$, SB_3 $\{s_3, s_8\}$, SB_4 $\{s_4, s_9\}$; see Figure 2. Now, QB distributes non-sensitive values associated with the sensitive values over two non-sensitive bins, resulting in the bin NSB_0 $\{ns_5, ns_1, ns_2, ns_3, *\}$ and NSB_1 $\{*, ns_6, *, *, *\}$, where a $*$ shows an empty position in the bin. In the sequel, QB needs to fill the non-sensitive bins with the remaining 5 non-sensitive values; hence, ns_{11} is assigned to the last position of the bin NSB_0, and the bin NSB_1 contains the remaining 4 non-sensitive values such as $\{ns_{12}, ns_6, ns_{13}, ns_{14}, ns_{15}\}$.

Aside. Note that QB assigned at least as many values in a non-sensitive bin as it assigned to a sensitive bin. The QB may form the non-sensitive and sensitive bins in such a way that the number of values in sensitive bins is higher than the non-sensitive bins. We chose sensitive bins to be smaller since the processing time on encrypted data is expected to be higher than clear-text data processing; hence, by searching and retrieving fewer sensitive tuples, we decrease the encrypted data-processing time.

[9]We emphasize to first permute sensitive values to prevent the adversary to create bins at her end; *e.g.*, if the adversary is aware of a fact that employee ids are ordered, then she can also create bins by knowing the number of resultant tuples to a query. However, for simplicity, we do not show permuted sensitive values in any figure.

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**

```
1 Function retrieve_bins(q(w)) begin
2     for (i, j) ∈ (0, SB − 1), (0, |SB| − 1) do
          if w = SB_i[j] then
3             return SB_i and NSB_j; found ← true; break
4     if found ≠ true then
5         for (i, j) ∈ (0, NSB − 1), (0, |NSB| − 1) do
6             if w = NSB_i[j] then
                  return NSB_i and SB_j; break
7     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
```

Step 2: Bin-retrieval – answering queries. Algorithm 2 presents the pseudocode for the bin-retrieval algorithm. The algorithm, first, checks the existence of a query value in sensitive bins and/or non-sensitive bins (see Lines 2 and 4 of Algorithm 2). If the value exists in a sensitive bin and a non-sensitive bin, the DB owner retrieves the corresponding two bins (see Line 7). Note that here the adversarial view is not enough to leak the query value or to find a value that is shared between the two bins. The reason is that the desired query value is encrypted with a set of other encrypted values and, furthermore, the query value is obscured in many requested non-sensitive values, which are in clear-text. Consequently, the adversary is unable to find an intersection of the two bins, which is the exact value.

There are following three other cases to consider:

1. Some sensitive values of a bin are not associated with any non-sensitive value. For example, in Figure 2, the sensitive values s_4, s_7, s_8, s_9, and s_{10} are not associated with any non-sensitive value.
2. A sensitive bin does not hold any value that is associated with any non-sensitive value. For example, the sensitive bin SB_4 in Figure 2 satisfies this clause.
3. A non-sensitive bin containing no value that is associated with any sensitive value.

In all the three cases, if the DB owner retrieves only either a sensitive or non-sensitive bin containing the value, then it will lead to information leakage similar to Example 2. In order to prevent such leakage, Algorithm 2 follows two rules stated below (see Lines 3 and 6 of Algorithm 2):

Tuple retrieval rule R1. If the query value w is a sensitive value that is at the j^{th} position of the i^{th} sensitive bin (*i.e.*, $w = SB_i[j]$), then the DB owner will fetch the i^{th} sensitive and the j^{th} non-sensitive bins (see Line 3 of Algorithm 2). By Line 2 of Algorithm 2, the DB owner knows that the value w is either sensitive or non-sensitive.

Tuple retrieval rule R2. If the query value w is a non-sensitive value that is at the j^{th} position of the i^{th} non-sensitive bin, then the DB owner will fetch the i^{th} non-sensitive and the j^{th} sensitive bins (see Line 6 of Algorithm 2).

Note that if query value w is in both sensitive and non-sensitive bins, then both the rules are applicable, and they retrieve the *exact required* bins. In addition, if the value w is neither in a sensitive or a non-sensitive bin, then there is no need to retrieve any bin.

Aside. After knowing the bins, the DB owner sends all the sensitive values in the encrypted form and the non-sensitive values in clear-text. However, the tuple retrieval based on the encrypted values may reveal the tuple ids that satisfy the requested values. We can

also hide the access-patterns by using PIR, ORAM, or DSSE, on each required sensitive value.[10]

Associated bins. *We say a sensitive bin is associated with a non-sensitive bin, if the two bins are retrieved for answering at least one query.*

Our aim when answering queries for all the sensitive and non-sensitive values using Algorithm 2 is to associate each sensitive bin with each non-sensitive bin; resulting in the adversary being unable to predict which is the value shared between two bins.

Example 3: (Query binning example: continued.) Now, we show how to retrieve tuples. If a query is for a sensitive value, say s_2, refer to Figure 2, then the DB owner fetches two bins SB_2 and NSB_0. If a query is for a non-sensitive value, say ns_{14}, then the DB owner fetches two bins NSB_1 and SB_3. Thus, it is impossible for the adversary to find (by observing the adversarial view) which is an exact query value from the non-sensitive bin and which is the sensitive value associated with one of the non-sensitive values. This fact is also clear from Table 6, which shows that the adversarial view is not enough that can lead to information leakage from the joint processing of sensitive and non-sensitive data, unlike Example 2. In Table 6, $E(s_i)$ shows the encrypted value of s_i, and we are showing the adversarial view only for some of the possible queries. In this example, note that the bin SB_2 gets associated with the bin NSB_0, when answering the query for the value s_2. In a similar manner, the bin NSB_1 gets associated with the bin SB_3, when answering the query for the value ns_{14}.

Exact query value	Returned tuples/Adversarial view	
	Sensitive data	Non-sensitive data
ns_1 or s_1	$E(s_1)$,$E(s_6)$	ns_1,ns_2,ns_3,ns_4,ns_5
ns_2 or s_2	$E(s_2)$,$E(s_7)$	ns_1,ns_2,ns_3,ns_4,ns_5
ns_3 or s_3	$E(s_3)$,$E(s_8)$	ns_1,ns_2,ns_3,ns_4,ns_5
ns_5 or s_5	$E(s_5)$,$E(s_{10})$	ns_1,ns_2,ns_3,ns_4,ns_5
ns_6 or s_6	$E(s_1)$,$E(s_6)$	ns_{12},ns_6,ns_{13},ns_{14},ns_{15}
s_7	$E(s_2)$,$E(s_7)$	ns_{12},ns_6,ns_{13},ns_{14},ns_{15}
ns_{12}	$E(s_5)$,$E(s_{10})$	ns_{12},ns_6,ns_{13},ns_{14},ns_{15}
ns_{13}	$E(s_2)$,$E(s_7)$	ns_{12},ns_6,ns_{13},ns_{14},ns_{15}

Table 6: Queries and returned tuples/adversarial view after retrieving tuples according to Algorithm 2.

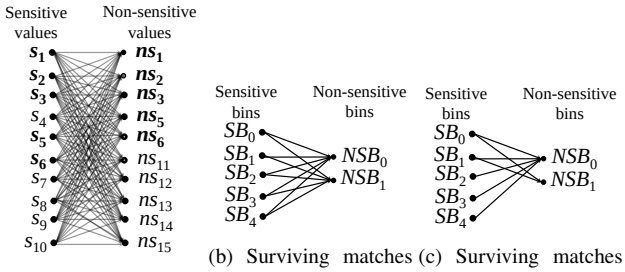
Algorithm Correctness

We will prove that QB does not lead to information leakage through the joint processing of sensitive and non-sensitive data. To prove correctness, we first define the concept of *surviving matches*. Informally, we show that QB maintains surviving matches among all sensitive and non-sensitive values, resulting in all sensitive bins being associated with all non-sensitive bins. Thus, an initial condition: a sensitive value is assumed to have an identical value to one of the non-sensitive value is preserved.

Surviving matches. We define surviving matches, which are classified as either *surviving matches of values* and *surviving matches of bins*, as follows:

Before query execution. Observe that before retrieving any tuple, under the assumption that the 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 associated with the value s_i. Thus, the adversary will consider that the value $E(s_i)$ is associated with 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. 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 10 sensitive and 10 non-sensitive values as shown in Figure 3a.

[10]QB is designed as a general mechanism that provides partitioned data security when coupled with any cryptographic technique. For special cryptographic techniques that hide access-patterns, it may be possible to design a different mechanism that may provide partitioned data security. QB, nonetheless, provides rational security against size and workload-skew attacks while using access-pattern hiding techniques.



(a) Initial matches. (b) Surviving matches surviving after the tuple retrieval following Algorithm 2. (c) Surviving matches without following Algorithm 2.

Figure 3: An example to show security of QB using surviving matches for 10 sensitive and 10 non-sensitive values.

After query execution. Recall that the query execution on the datasets creates an adversarial view that guides the adversary to create a (new) bipartite graph containing SB nodes on one side and NSB nodes on the other side. The edges in the new graph (obtained after the query execution) are called *surviving matches of the bins*. For example, after executing queries according to Algorithm 2, the adversary can create a bipartite graph having 5 nodes on one side and 2 nodes on the other side as shown in Figure 3b. Note that since bins contain values, the surviving matches of the bins can lead to the surviving matches of the values. Hence, from Figure 3b, the adversary can also create a bipartite graph as in Figure 3a.

We show that a technique for retrieving tuples that drops some surviving matches of the bins leading to drop of the surviving matches of the values is not secure, and hence, results in the information leakage through non-sensitive data.

Example 4: Dropping surviving matches. In Figure 2, for answering queries for associated values s_1, s_2, s_3, s_5, s_6, ns_1, ns_2, ns_3, ns_5, or ns_6, the DB owner must follow Line 3 or 6 of Algorithm 2 for retrieving the two bins holding corresponding sensitive and non-sensitive data; otherwise, the DB owner cannot retrieve two bins that share a common value. Now, retrieved tuples for these values create an adversarial view as shown in the first five lines of Table 6. However, for answering values s_4, s_7, s_8, s_9, s_{10}, ns_6, ns_{12}, ns_{13}, ns_{14}, or ns_{15} (recall that these values are not associated), if the DB owner does not follow Algorithm 2 and retrieves the bin containing the desired value with any randomly selected bin of the other side, then it could result in the following adversarial view; see Table 7.

Exact query value	Returned tuples/Adversarial view	
	Sensitive data	Non-sensitive data
s_4	$E(s_4)$,$E(s_9)$	ns_1,ns_2,ns_3,ns_5,ns_{11}
s_7	$E(s_2)$,$E(s_7)$	ns_1,ns_2,ns_3,ns_5,ns_{11}
s_8	$E(s_3)$,$E(s_8)$	ns_1,ns_2,ns_3,ns_5,ns_{11}
s_9	$E(s_4)$,$E(s_9)$	ns_1,ns_2,ns_3,ns_5,ns_{11}
s_{10}	$E(s_5)$,$E(s_{10})$	ns_1,ns_2,ns_3,ns_5,ns_{11}
ns_{11}	$E(s_1)$,$E(s_6)$	ns_1,ns_2,ns_3,ns_5,ns_{11}
ns_{12}	$E(s_1)$,$E(s_6)$	ns_6,ns_{12},ns_{13},ns_{14},ns_{15}
ns_{13}	$E(s_1)$,$E(s_6)$	ns_6,ns_{12},ns_{13},ns_{14},ns_{15}
ns_{14}	$E(s_1)$,$E(s_6)$	ns_6,ns_{12},ns_{13},ns_{14},ns_{15}
ns_{15}	$E(s_1)$,$E(s_6)$	ns_6,ns_{12},ns_{13},ns_{14},ns_{15}

Table 7: Queries and returned tuples/adversarial view without following Algorithm 2.

Algorithm 3: An extension to the bin-creation Algorithm 1 for the base case, $|S| < |NS|$.

Inputs: $|NS|$, $|S|$. **Outputs:** SB, NSB

1 **Function** $bin\_extension(S, NS)$ **begin**
2 Permute all sensitive values
3 $x, y \leftarrow approx\_sq\_factors(|NS|)$: $x \geq y$; $cost_d \leftarrow x + y$
4 $z \leftarrow closest\_SquareNum(|NS|)$, $cost_{sn} \leftarrow 2(z/\sqrt{z})$
5 **if** $(cost_{sn} + \lceil(|NS| - z)/\sqrt{z}\rceil < cost_d)$ **then**
6 Execute Algorithm 1(S, z) and add $(NS - z)/\sqrt{z}$ number of the *remaining* non-sensitive values in each non-sensitive bins
7 **else** Execute Algorithm 1(S, NS)

Having such an adversarial view (Table 7), the adversary can learn two facts that

1. Encrypted sensitive tuples of the bins SB_0, SB_2, SB_3, SB_4 have associated non-sensitive tuples in the bin NSB_0, not in NSB_1.
2. Non-sensitive tuples of the bin NSB_1 have their associated sensitive tuples only in the bin SB_1.

Based on this adversarial view (Table 7), the bipartite graph drops some surviving matches of the bins (see Figure 3c). (That fact leads to drop of the surviving matches of the values, specifically, surviving matches between sensitive values s_3, s_4, s_5, s_8, s_9, s_{10} and non-sensitive value ns_6, ns_{12}, ns_{13}, ns_{14}, ns_{15}.) Hence, a random retrieval of bins is not a secure technique to prevent the information leakage through non-sensitive data accessing.

In contrast, if the DB owner uses Line 3 or 6 of Algorithm 2 for retrieving values that are not associated, the above-mentioned facts (*i*) and (*ii*) no longer hold. Figure 3b shows the case when each sensitive bin is associated with each non-sensitive bin, if Algorithm 2 is followed. Thus, we can see that all the surviving matches of the bins and values are preserved after answering queries. Therefore, for the example of 10 sensitive and 10 non-sensitive values, QB (Algorithms 1 and 2) is secure, and under the given assumptions (§4), the adversary cannot find an exact association between a sensitive and a non-sensitive value.

Security and correctness proofs. are presented in §B.

5.2 A Simple Extension of the Base Case

Algorithm 1 creates bins when the number of non-sensitive data values[11] is not a prime number, by finding the two approximately square factors. However, Algorithm 1 may exhibit a relatively higher *cost* (*i.e.*, the number of the retrieved tuple) when the sum of the approximately square factors is high.

For example, if there are 41 sensitive data values and 82 non-sensitive data values, then Algorithm 1 creates 2 non-sensitive bins having 41 values in each and 41 sensitive bins having exactly one value in each (Line 4 of Algorithm 1). Consequently, answering a query results in retrieval of 42 tuples. (We may also create two sensitive bins and 41 non-sensitive bins containing exactly two non-sensitive values in each, resulting in retrieval of 23 tuples.) However, the cost can be further reduced by a significant amount.

An extension to the bin-creation Algorithm 1 is provided in Algorithm 3 that handles the case when the number of non-sensitive values ($|S| < |NS|$) is close to a square number.[12] Algorithm 3 first finds two approximately square factors of non-sensitive values and the cost; Line 3. Algorithm 3 also finds a square number, say z, closest to the non-sensitive values and the cost; Line 4. Now, Algorithm 3 creates bins using a method that results in fewer numbers of retrieved tuples (Line 5). When Algorithm 3 creates bins using the square number closest to the non-sensitive values (Line 6), the *remaining* non-sensitive values (*i.e.*, $|NS| - z^2$) can be handled by assigning an equal number of the remaining non-sensitive values in the bins. Note that the sensitive and associated non-sensitive values are assigned to bins in an identical manner as in Algorithm 1 (Lines 5-7).

Example 5: (An example of QB extension — Algorithm 3). Consider again the example of 41 sensitive and 82 non-sensitive values (presented at the beginning of this section). In this case, 81 is the closest square number to 82. Here, Algorithm 3 creates 9 non-sensitive bins and 9 sensitive bins. By Lines 5 and 6 of Algorithm 1, sensitive values and associated non-sensitive values are allocated, resulting in that a sensitive bin holds at most 5 values and a non-sensitive bin holds at most 10 values. Thus, at most 15 tuples are retrieved to answer a query.

Aside. In QB, the bin size impacts the overall performance, which will be validated by the experiments (§7). Hence, a careful selection of the bin size is mandatory.

5.3 General Case: Multiple Values with Multiple Tuples

In this section, we will generalize Algorithms 1-3 to consider a case when data values have a different number of tuples. First, we will show that sensitive values with a different number of tuples may provide enough information to the adversary leading to the size, frequency-count attacks, and may disclose some information about the sensitive data. Hence, in the case of multiple values with multiple tuples, Algorithms 1-3 cannot be directly implemented. We, thus, develop a strategy to overcome such a situation.

Size attack scenario in the base QB. Consider an assignment of 10 sensitive and 10 non-sensitive values to bins using Algorithm 1; see Figure 2. Assume that a sensitive value, say s_1, has 1000 sensitive tuples and an associated non-sensitive value, say ns_1, has 2000 tuples, while all the other values have only 1 tuples. Further, assume that each data value represents the salary of employees.

In this example, consider a query execution for a value, say ns_1. The DB owner retrieves tuples from two bins: SB_1 (containing encrypted tuples of values s_1 and s_6) and NSB_0 (containing tuples of values $ns_1, ns_2, ns_3, ns_5, ns_{11}$); see Figure 2. Obviously, the number of retrieved tuples satisfying the values of the bins SB_1 and NSB_0 will be highest (*i.e.*, 3005) as compared to the number of tuples retrieved based on any two other bins. Thus, the retrieval of the two bins SB_1 and NSB_0 provides enough information to the adversary to determine which one is the sensitive bin associated with the bin holding the value ns_1. Moreover, after observing many queries and having background knowledge, the adversary may estimate that 1000 people in the sensitive relation earn a salary equals to the value ns_1.

Thus, in the case of sensitive values with a different number of tuples, Algorithm 1 cannot satisfy the second condition of partitioned data security (*i.e.*, the adversary is able to distinguish two sensitive values based on the number of retrieved tuples, which was not possible before the query execution, and concludes that a sensitive value (s_1 in the above example) has more tuples than any other sensitive value) though preserving all surviving matches, and holding Theorems 1 and 2 to be true.

In order to hold the second condition of partitioned data security (and for the scheme to be resilient to the size and frequency-count attacks, as illustrated above), sensitive bins need to hold an identical number of tuples. A trivial way of doing this is to outsource some encrypted fake tuples in a way that the number of tuples in each sensitive bin will be identical. However, we need to be

[11] Recall that we considered the case of $|S| \leq |NS|$.

[12] The case of $|S| > |NS|$ can be handled by applying Algorithm 3 in a reverse way.

careful; otherwise, adding fake tuples in each sensitive bin may increase the *cost*, if all the heavy-hitter sensitive values are allocated to a single bin. This fact will be clear in the following example.

(a) The first way. (b) The second way.

Figure 4: An assignment of 9 sensitive values to 3 bins.

Example 6: (Illustrating ways to assign *sensitive* values to bins to minimize the addition of fake tuples). Consider 9 sensitive values, say $s_1, s_2, \ldots, s_9$, having 10, 20, 30, 40, 50, 60, 70, 80, and 90 tuples, respectively.[13] There are multiple ways of assigning these values to three bins so that we need to add a minimum number of fake tuples to each bin. Figure 4 shows two different ways to assign these values to bins. Figure 4b shows the best way – to minimize the addition of fake encrypted tuples; hence minimizing the cost. Here, 20 and 10 fake encrypted tuples are added to the bins SB_0 and SB_1, respectively, to have an identical number of tuples in each bin. However, bins in Figure 4a require us to add 180 and 90 fake encrypted tuples to the bins SB_0 and SB_1, respectively.

Note that there is no need to add any fake tuple if all the non-sensitive values have an identical number of tuples. In that case, the adversary cannot deduce which sensitive bin contains sensitive tuples associated with a non-sensitive value. However, it is obvious that any fake non-sensitive tuple cannot be added in clear-text.

Before describing how to add fake encrypted tuples to bins, we show that a partitioning of sensitive values over SB bins may lead to an identical number of tuples in each bin, where a bin is not required to hold at most y values, is not a communication-efficient solution. For example, consider 9 sensitive values, where a value, say s_1, has 100 tuples and all the other values, say $s_2, s_3, \ldots, s_9$, have 25 tuples each. In this case, we may get bins as shown in Figure 5. Note that the bins SB_1 and SB_2 are associated with all the three non-sensitive bins while the bin SB_0 is associated with only NSB_0 (thus, the given bins do not prevent the surviving matches). In order to associate each sensitive bin with each non-sensitive bin (and hence, preventing all the surviving matches), we need to ask fake queries for bins $\langle SB_0, NSB_1 \rangle$ and $\langle SB_0, NSB_2 \rangle$.

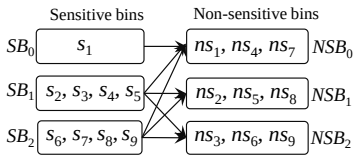


Figure 5: An assignment of a heavy-hitter value but dropping surviving matches.

Adding fake encrypted tuples. As an assumption, we know the number of sensitive bins, say SB, using Algorithm 1 or 3. Here, our objective is to assign sensitive values to bins such that each bin holds an identical number of tuples while minimizing the number of fake tuples in each bin. To do this, the strategy is given below:

1. Sort all the values in a decreasing order of the number of tuples.
2. Select SB largest values and allocate one in each bin.
3. Select the next value and find a bin that is containing the fewest number of tuples. If the bin is holding less than y values, then add the value to the bin; otherwise, select another bin with the fewest number of tuples. Repeat this step, for allocating all the values to sensitive bins.
4. Add fake tuples' values to the bins so that each bin contains an identical number of tuples.
5. Allocate non-sensitive values as per Algorithm 1 (Lines 6 and 7).

In Appendix C, we will provide a lemma that states how many fake tuples' values are required to add when using the above strategy and a proof outline for partitioned data security for the above strategy.

[13]We assume that there are 9 non-sensitive values, and computed that we need 3 sensitive and 3 non-sensitive bins.

6. COMPARING QB TO FULLY CRYPTOGRAPHIC SOLUTIONS

In §5, we described QB, which eliminates the necessity of encrypted data processing over non-sensitive data, while still ensuring security guarantees. In this section, we compare QB to simply storing the entire data (both sensitive and non-sensitive) in an encrypted form and processing queries over encrypted representation. We compare QB to a fully cryptographic approach from both the perspective of performance and security.

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 clear-text non-sensitive data, and, a set of encrypted predicates selection queries albeit over a smaller database consisting only of sensitive data. §6.2 and §6.3 discuss how QB performance can be compared to the pure cryptographic approaches for non-indexable and indexable cryptographic mechanisms.

From the security perspective, we have already shown that QB offers no less security than the underlying cryptographic technique it is implemented on. In particular, we showed that QB offers partitioned data security in the sense that the adversary learns nothing from the joint processing of sensitive and non-sensitive data. What is more interesting is that QB offers additional security properties compared to the host cryptographic technique in that it ensures provable security from the size, frequency-count, and workload-skew attacks even if the host technique does not guarantee such properties. In §6.4, we explore an alternative way to compare QB with a cryptographic mechanism, while both the techniques provide an identical security guarantee.

In particular, we incorporate QB with an efficient but a weak cryptographic mechanism. Coupled with QB, the mechanism offers similar security guarantees as much stronger cryptographic techniques, albeit at lower overheads.

6.1 Preliminaries

Two parameters dictate the overhead of QB, as follows:

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 for a *fixed dataset* on a specific *database system* (in both cases).

Clearly, the parameter β captures the overhead of a cryptographic technique and is dominated by the type of underlying encryption and the cryptographic technique used for searching a predicate. For example, the β value will be higher for search operation using a fully homomorphic encryption technique (because of a higher value of the cryptographic matching operation time) as compared

to deterministic encrypted search. Obviously, for the most efficient cryptographic technique, β should be 1.

We first note a β^* for a fully secure solution, where the DB owner retrieves the entire dataset and performs computations (*i.e.*, data decryption and query execution) at her end, and hence, the adversary cannot launch any attack. A cryptographic technique that possesses a higher value of β than β^* is not useful in our context. Table 8 shows β^* values computed on systems A and B.

Given the above two parameters α and β, we can compare QB with cryptographic techniques and use the following notations: $C[|SB|, E(S)]$: the cost to search $|SB|$ selection predicates on encrypted sensitive data. $C[|NSB|, NS]$: the cost to search $|NSB|$ selection predicates on clear-text non-sensitive data. $C[E(S+NS)]$: cost to search the desired predicate on the encrypted sensitive as well as encrypted non-sensitive data. $C[E(S)]$: cost to search the desired predicate on the encrypted sensitive data.

We define a parameter η to be the ratio between the cost of performing a search using QB and the cost of performing the search when the entire data (viz. sensitive and non-sensitive data) is fully encrypted using the cryptographic mechanism. In particular,

$$\eta = \frac{C[|SB|, E(S)]}{C[E(S+NS)]} + \frac{C[|NSB|, NS]}{C[E(S+NS)]}$$

Note that if the value of η is less than 1, then QB performs better than a fully cryptographic approach, else it performs worse. Estimating this ratio requires us to differentiate between cryptographic technique being indexable or non-indexable.

6.2 Comparing QB at Different Levels of Security

6.2.1 *Non-indexable cryptographic technique on encrypted data*

Several cryptographic search techniques that do not exploit indices have been proposed in the literature [57, 30]. Such techniques typically perform a linear scan over encrypted tuples to determine the set of tuple-ids that satisfy the query.

Note that in QB mixed with a non-indexed scheme on the sensitive data, search for all the $|SB|$ predicates can be performed using a single linear scan over encrypted data, using techniques such as [57, 30], and then tuple retrieval is performed possibly using direct addresses (or PIR/ORAM techniques, depending upon the desired security). Thus, the cost of searching a single predicate over encrypted data using a linear scan absorbs the cost of searching $|SB|$ predicates using a linear scan on the same data.

Thus, the first term of η becomes $\frac{C[E(S)]}{C[E(S+NS)]}$. Since the cost of a linear scan is proportional to the size of the database, the first term reduces to simply α. The second term can be upper bounded by $|NSB|/\beta$, since the cost of clear-text processing of $(1-\alpha)D$ tuples can be upper bounded by the cost of processing D tuples in clear-text, and furthermore, the cost of processing $|NSB|$ such queries is bounded by $|NSB|$ times that cost, leading to the second term of $|NSB|/\beta$.

Therefore, $\eta < \alpha + \frac{|NSB|}{\beta}$. Since QB is better compared to the fully cryptographic approach only when $\eta < 1$, we can state that QB would result in improved performance if $\alpha + |NSB|/\beta < 1$, that is, $\beta > |NSB|/(1-\alpha)$.

Figure 6 plots a graph of η as a function of β, for varying sensitivity. Table 8 shows our experimental values of η at different β and α values. In the experiments,[14] we have $|NSB| \approx 400$. Thus, even for a very large value of $\alpha = 0.6$, QB is efficient if $\beta > 1000$.

[14] §7 provides details of experimental setup.

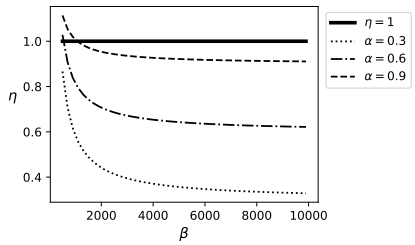


Figure 6: Efficiency graph using equation $\eta = \alpha + |NSB|/\beta$: comparing QB with non-indexable techniques.

α	System A		System B	
	Tech 1 (β_1=5764.99)	Tech 2 (β^*=9500.26)	Tech 1 (β_1=868.97)	Tech 2 (β^*=1117.73)
0.3	0.3108	0.1973	0.5649	0.59693
0.6	0.6106	0.3874	0.768	0.80906
0.9	0.9116	0.5567	1.0163	1.07006

Table 8: η values for different β and α values.

Note that the real η values are very close to the predicted eta values, shown in Figure 6. For example, for the technique 1 implemented on the system A and at $\alpha = 0.6$, we have $\eta = 0.6106$ while the predicted η value is 0.6693, shown in Figure 6.

Discussion on different techniques used to capture β values. Table 8 shows β values for two techniques: Technique 1 retrieves the searching attribute of a sensitive relation at the DB owner side that searches $|SB|$ predicates over the attribute after decryption and then retrieves full tuples corresponding to $|SB|$ predicates' addresses. Technique 2 retrieves the entire sensitive relation at the DB owner side that searches the desired predicate over decrypted sensitive relation, and hence, we have the highest β value, *i.e.*, β^*.

6.3 Indexable Techniques on Encrypted Data

Several secure and non-secure indexable cryptographic techniques for avoiding the overhead of the linear scan operation have been proposed in the literature [56, 26, 51, 19, 36]. For instance, secure indexable techniques such as [36, 19] require us to first create an index (particularly, a B-tree) on a key attribute at the trusted side before outsourcing the index to the cloud. Access-patterns during retrieval are hidden using PIR/ORAM execution over the index. We collectively refer to such techniques as *U-Ind*.

While U-Ind can have significant overhead, more efficient approaches that, however, compromise security have been described in [56, 26, 51]. For instance, [51] describes a non-deterministic encryption mechanism wherein, for each predicate k, the technique keeps track of the number of its occurrences. The technique encrypts the i^{th} occurrence of k as a concatenated string $\langle k, i\rangle$ thereby ensuring that no two occurrences of k result in an identical ciphertext. Such a ciphertext representation can then be indexed on the cloud-side. During retrieval, the user keeps track of the histogram of occurrences for each predicate and generates appropriate ciphertexts that can be used to query the index on the cloud. Let us refer to the above technique as *C-Ind*.

It is not difficult to see that C-Ind, by itself is susceptible to the size, frequency-count, and workload-skew attacks. However, it is significantly more efficient (almost as efficient as the plaintext version). For instance, for the above-mentioned C-Ind technique, $\beta = 1.4$ on the system A and $\beta = 2.5$ on the system B, while β values on the systems A and B for a U-Ind approach are very high, 1200 and 2200, respectively.

Let us next explore under what conditions QB improves the performance of an index-based cryptographic technique. As before $\eta = \frac{C[|SB|,E(S)]}{C[E(S+NS)]} + \frac{C[|NSB|,NS]}{C[E(S+NS)]}$ must be below 1 for QB to provide any benefit over using a cryptographic approach. Unlike the case

of non-indexable approaches, the effect of combining QB with indexable approaches may not be as significant since, (based on the first term of η) in QB, we search for $|SB|$ predicates compared to a single predicate search in the cryptographic approach. Notice that unlike the cost of several searches being absorbed in a single linear scan in the case of non-indexable search, here, conservatively, we will need to pay the cost of index retrieval over sensitive data $|SB|$ times. Also, in index retrieval, the cost of searching the predicate in a (smaller) database of size S or over the larger database of size $NS+S$ will likely be similar. Thus, the first term of η approximates to $|SB|$. The second term, however, as before remains $|NSB|/\beta$.

Therefore, $\eta > |SB| + |NSB|/\beta$. As before, we will benefit from QB if $\eta < 1$, which is impossible to achieve in the case of indexable technique, since $|SB| > 1$. Thus, QB by itself is not expected to improve the performance of an index-based approach. Such a comparison, however, is not fully fair to QB, since QB, as we mentioned before, may offer a much higher level of security compared to the underlying cryptographic approach on which QB is applied. We discuss a more fair approach to compare QB with pure cryptographic approaches under identical security in the following subsection.

It is, however, important to note QB provides additional security from the size, frequency-count, and workload-skew attacks even if the underlying cryptographic technique does not. This can be of a great value for index-based approaches, since index-based approaches (given they only traverse a subset of the database using the index) are vulnerable to such attacks.

6.4 Comparing QB at an Identical Security Level

Our goal in this section is to explore conditions (*i.e.*, β values) for which a direct cryptographic approach may outperform QB while offering the same level of security. In particular, consider a cryptographic approach C_2 that offers security against the frequency-count, size, and workload-skew attacks, and let its corresponding β value be β_2. Note that QB in conjunction with weaker techniques such as *C-Ind* offers a similar security property. Let us assume that the corresponding β value for C-Ind be β_1. We can now compare the performance of a QB approach based on *C-Ind* with a fully secure cryptographic approach C_1 with similar security properties as follows:

$$\eta = \frac{C[|SB|, E_{c1}(S)]}{C[E_{c2}(S+NS)]} + \frac{C[|NSB|, NS]}{C[E_{c2}(S+NS)]}$$

In the above equation, we used the notation $C[E_{cx}(D)]$: the cost to search a predicate on an encrypted dataset D with the help of an underlying cryptographic technique C_x. The second term of the above equation is reduced to $|NSB|/\beta_2$ (by following the analysis presented in §6.2.1). We can multiply the first term by $C[E_{c1}(S+NS)]$ and obtain the following equation.

$$\eta = \frac{C[|SB|, E_{c1}(S)]}{C[E_{c2}(S+NS)]} \times \frac{C[E_{c1}(S+NS)]}{C[E_{c1}(S+NS)]} + \frac{|NSB|}{\beta_2}$$

The first term of the above equation will be reduced to $|SB| \times \frac{C[E_{c1}(S+NS)]}{C[E_{c2}(S+NS)]}$ (by following the analysis presented in §6.3). Note that the cost of searching a single predicate on the entire dataset can be captured by the β value of the underlying technique.

Therefore, $\eta = |SB| \times \beta_1/\beta_2 + |NSB|/\beta_2$. Notice that if $\eta > 1$, then the cryptographic technique C_2 is better. Solving equation for β_2 means $\beta_2 < \beta_1 \times |SB| + |NSB|$. Since the β value of the C-Ind technique is $\beta_1 \approx 1.4 \times |SB| + |NSB|$ (experimentally as shown in §7), it must be the case that $\beta_2 < 1.4$ for the approach C_2 to be better than QB based on C-Ind. Given approximate values of $|NSB|$ and $|SB|$ be 400, all the cryptographic techniques can only win against QB of the same level of security, only if β is below 960, approximately.

To the best of our knowledge, we note that no such technique that offers the above security exists at this level of β value. Thus, we believe that QB in conjunction with a simple C-Ind technique is the best solution. Notice that QB with C-Ind techniques is only secure for the size, frequency-count, and workload-skew attacks not for access-pattern hiding. We could also design a QB-based approach that is secure to all the four attacks by using DSSE [36] or Curtmola et al. [19] as an underlying cryptographic technique. We believe that these modified QB techniques are the best for preventing all the four attacks, and it will remain same as long as the β values of the new technique C_2 is higher than the β value of DSSE, *i.e.*, $\beta_2 > \beta_1$, where $\beta_1 = 1200$ is computed for DSSE on the system A.

7. EXPERIMENTS

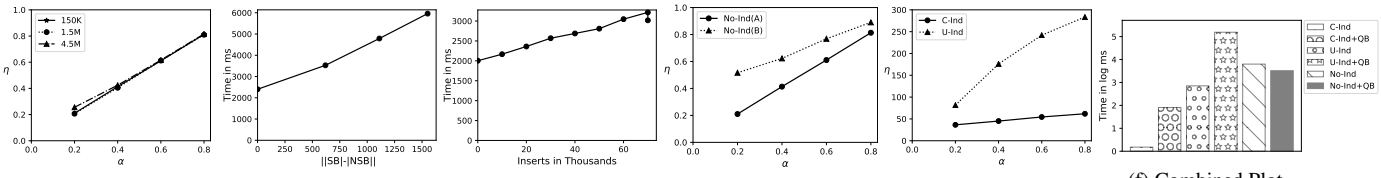
In this section, we: (*i*) study QB under different parameter settings (*e.g.*, DB size and bin size) and overhead of insertion, (*ii*) validate the analytical model developed in §6 to identify when QB performs better than a purely cryptographic approach, (*iii*) compare different cryptographic approaches, with and without QB, in terms of security and performance with the goal to identify relative overheads needed to achieve higher levels of security.

Experimental setup. We used virtual machines (VM), each with 2.6 GHz, 4 core processor, 16 GB RAM, and 1TB physical disk in our in-house cloud. Various cryptographic approaches used in experiments (with and without QB) include: (*i*) non-deterministic encryption supported by two popular commercial systems A and B[15] combined with search implemented by retrieving the column in the query, decrypting to determine the relevant rows which are then retrieved subsequently. We name these techniques No-Ind(A) and No-Ind(B), respectively. We also used two indexable approaches: (*i*) U-Ind: a commercial implementation that uses DSSE preventing access-pattern attacks, and (*ii*) C-Ind: a cloud-side index [51]. For each of the 4 approaches, we also developed a QB approach on top.

Dataset and sensitivity. We used TPC-H benchmark to generate the dataset for our experiments. The sensitivity factor (α) was varied from 0.1 to 0.9 for experiments. For QB based approaches, the sensitive part was stored in the underlying cryptographic system while the non-sensitive data stored in plain text over which a non-clustered B^+ tree index was created. Furthermore, sensitive and non-sensitive bins were stored at the DB owner side with other metadata to formulate appropriate partitioned queries. Such metadata storage is propositional to the domain size of the searchable attributes and it is independent of the database size. For TPC-H LINEITEM table, metadata for attributes L_PARTKEY and L_SUPPKEY were 13.6MB and 0.65MB, respectively. The metadata storage can further be reduced using compression. The total execution time was computed by retrieving the tuples associated with the predicate in both the bins. Each experimental setup was repeated 50 times to minimize the effect of outliers.

Exp 1: Robustness of QB. To explore the effectiveness of QB under different DB sizes, we tested QB for 3 DB sizes: 150K, 1.5M, and 4.5M tuples using No-Ind(A) and No-Ind(B) as underlying cryptographic mechanisms. Figure 7a plots η values for the three sizes for No-Ind(A) while varying α. The figure shows that $\eta < 1$, irrespective of the DB sizes, confirming that QB scales to larger DB sizes (results over No-Ind(B) are similar). Figure 7b plots an

[15]We refer to the systems as A and B to hide their true identity.



(a) DataSet Size (b) Bin Size (c) Inserts (d) Non Indexable (e) Indexable (f) Combined Plot

Figure 7: Experiments.

average time for a selection query using QB with a different bin size, which is in turn governed by the values of $|SB|$ and $|NSB|$, respectively. We plot the effect of $||SB| - |NSB||$ on retrieval time and find that the minimum time is achieved when $|SB| = |NSB|$. Thus, the optimal choice is $|SB| = |SB| = \sqrt{|NS|}$ (Line 5, Algorithm 3). Finally, Figure 7c plots an average cost of search after a number of insertions. In the experiment, insertions are processed (as per the method, given in §A) in batches of 10K and after each batch, selection queries are executed to determine overhead due to insertion. Finally, after 7 batches of insertion, Algorithm 3 is re-executed to recreate bins. The figure confirms that the query cost increases but only marginally in the presence of insertion and (as shown by the last plotted point) reduces by re-binning.

Exp 2. Validation of analytical model. To validate the model for indexable and non-indexable approaches (§6), we implemented fully cryptographic approaches, e.g., C-Ind [51], No-Ind(A), and No-Ind(B), and compared with the corresponding QB by varying α. Figure 7d shows the effect of α on η for non-indexable systems. The results are as per the analytical model, where QB should be effective even at 0.8 sensitivity. Figure 7e presents the results of QB on C-Ind and U-Ind, which are indexable for which, as expected, QB does not improve performance.

Exp 3: Overheads for security. Figure 7f shows average execution times for different schemes with/without QB at a given sensitivity level $\alpha = 0.5$. C-Ind+QB, No-Ind(A)+QB, No-Ind(B)+QB offer the same level of security (viz., security against size, workload-skew, and frequency attacks), which is higher security compared with C-Ind, No-Ind(A) and No-Ind(B). The figure clearly shows C-Ind+QB as the most compelling scheme from both security and performance perspective; which requires significantly less time as compared to a direct implementation of U-Ind without using QB. C-Ind+QB, however, cannot be compared with U-Ind in terms of security, which prevents access-pattern attacks but not preventing workload-skew, frequency, or size attacks. U-Ind+QB is the most secure technique that prevents workload, frequency, and size attacks while also preventing access-pattern attacks. It, however, incurs a significant overhead compared to C-Ind+QB. A more efficient approach compared to U-Ind+QB that offers security against all four attacks is an interesting direction of future work.

8. CONCLUSION

This paper proposes query binning (QB) technique that serves as a meta approach on top of existing cryptographic techniques to support secure selection queries when a relation is partitioned into cryptographically secure sensitive and clear-text non-sensitive sub-relations. Further, we develop (*i*) a new notion of partitioned data security that restricts exposing sensitive information due to the joint processing of the sensitive and non-sensitive relations, and (*ii*) an analytical model to investigate the efficiency of QB against pure cryptographic techniques. Besides improving efficiency, while supporting partitioned security, interestingly, QB enhances the security of the underlying cryptographic technique by preventing size, frequency-count, and workload-skew attacks. As a result, combining QB with efficient but non-secure cloud-side indexable cryptographic approaches can result in an efficient and significantly more secure search. Furthermore, existing indexable/non-indexable cryptographic techniques that prevent access-patterns can also benefit from the added security that QB offers.

9. REFERENCES

[1] Amazon Aurora, available at:`https://aws.amazon.com/rds/aurora/`.

[2] MariaDB, available at:`https://mariadb.com/`.

[3] Stealth SDB, available at:`http://www.stealthsoftwareinc.com/`.

[4] `http://www.computerworld.com/article/2834193/cloud-computing/5-tips-for-building-a-successful-hybrid-cloud.html`.

[5] `https://www.getfilecloud.com/blog/2015/07/5-tips-on-optimizing-your-hybrid-cloud/`.

[6] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order-preserving encryption for numeric data. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, France, June 13-18, 2004*, pages 563–574, 2004.

[7] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal security with cipherbase. In *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[8] A. Arasu, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. A secure coprocessor for database applications. In *23rd International Conference on Field programmable Logic and Applications, FPL 2013, Porto, Portugal, September 2-4, 2013*, pages 1–8, 2013.

[9] A. Arasu and R. Kaushik. Oblivious query processing. In *Proc. 17th International Conference on Database Theory (ICDT), Athens, Greece, March 24-28, 2014.*, pages 26–37, 2014.

[10] S. Bajaj and R. Sion. Correctdb: SQL engine with practical query authentication. *PVLDB*, 6(7):529–540, 2013.

[11] S. Bajaj and R. Sion. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Trans. Knowl. Data Eng.*, 26(3):752–765, 2014.

[12] C. Bao and A. Srivastava. Exploring timing side-channel attacks on path-ORAMs. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2017, McLean, VA, USA, May 1-5, 2017*, pages 68–73, 2017.

[13] M. Bellare, A. Boldyreva, and A. O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology - CRYPTO 2007, 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007, Proceedings*, pages 535–552, 2007.

[14] E. Boyle, N. Gilboa, and Y. Ishai. Function secret sharing. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, pages 337–367, 2015.

[15] D. Cash, P. Grubbs, J. Perry, and T. Ristenpart. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 668–679, 2015.

[16] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *J. ACM*, 45(6):965–981, 1998.

[17] V. Ciriani, S. D. C. di Vimercati, S. Foresti, S. Jajodia, S. Paraboschi, and P. Samarati. Fragmentation and encryption to enforce privacy in data storage. In *Computer Security - ESORICS 2007, 12th European Symposium On Research In Computer Security, Dresden, Germany, September 24-26, 2007, Proceedings*, pages 171–186, 2007.

[18] V. Costan and S. Devadas. Intel SGX explained. *IACR Cryptology ePrint Archive*, 2016:86, 2016.

[19] R. Curtmola, J. A. Garay, S. Kamara, and R. Ostrovsky. Searchable symmetric encryption: Improved definitions and efficient constructions. *Journal of Computer Security*, 19(5):895–934, 2011.

[20] H. S. Delugach and T. H. Hinke. Wizard: A database inference analysis and detection system. *IEEE Trans. Knowl. Data Eng.*, 8(1):56–66, 1996.

[21] I. Demertzis and C. Papamanthou. Fast searchable encryption with tunable locality. In *SIGMOD*, 2017.

[22] T. T. A. Dinh, P. Saxena, E. Chang, B. C. Ooi, and C. Zhang. M2R: enabling stronger privacy in mapreduce computation. In *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, pages 447–462, 2015.

[23] S. Dolev, N. Gilboa, and X. Li. Accumulating automata and cascaded equations automata for communicationless information theoretically secure multi-party computation: Extended abstract. In *SCC@ASIACCS*, pages 21–29. ACM, 2015.

[24] S. Doudalis, I. Kotsogiannis, S. Haney, A. Machanavajjhala, and S. Mehrotra. One-sided differential privacy. *CoRR*, abs/1712.05888, 2017.

[25] M. Egorov and M. Wilkison. ZeroDB white paper. *CoRR*, abs/1602.07168, 2016.

[26] Y. Elovici, R. Waisenberg, E. Shmueli, and E. Gudes. A structure preserving database encryption scheme. In *Secure Data Management, VLDB 2004 Workshop, SDM 2004, Toronto, Canada, August 30, 2004, Proceedings*, pages 28–40, 2004.

[27] F. Emekçi, A. Metwally, D. Agrawal, and A. El Abbadi. Dividing secrets to secure data outsourcing. *Inf. Sci.*, 263:198–210, 2014.

[28] C. Farkas and S. Jajodia. The inference problem: A survey. *SIGKDD Explorations*, 4(2):6–11, 2002.

[29] C. Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford University, 2009.

[30] N. Gilboa and Y. Ishai. Distributed point functions and their applications. In *Advances in Cryptology - EUROCRYPT 2014 - 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings*, pages 640–658, 2014.

[31] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, pages 182–194, 1987.

[32] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. Cache attacks on Intel SGX. In *Proceedings of the 10th European Workshop on Systems Security, EUROSEC 2017, Belgrade, Serbia, April 23, 2017*, pages 2:1–2:6, 2017.

[33] P. Grubbs, K. Sekniqi, V. Bindschaedler, M. Naveed, and T. Ristenpart. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017*, pages 655–672, 2017.

[34] H. Hacigümüs, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data, Madison, Wisconsin, June 3-6, 2002*, pages 216–227, 2002.

[35] Y. Ishai and E. Kushilevitz. Private simultaneous messages protocols with applications. In *Fifth Israel Symposium on Theory of Computing and Systems, ISTCS 1997, Ramat-Gan, Israel, June 17-19, 1997, Proceedings*, pages 174–184, 1997.

[36] Y. Ishai, E. Kushilevitz, S. Lu, and R. Ostrovsky. Private large-scale databases with distributed searchable symmetric encryption. In *Topics in Cryptology - CT-RSA 2016 - The Cryptographers' Track at the RSA Conference 2016, San Francisco, CA, USA, February 29 - March 4, 2016, Proceedings*, pages 90–107, 2016.

[37] M. S. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*, 2012.

[38] G. Kellaris, G. Kollios, K. Nissim, and A. O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 1329–1340, 2016.

[39] L. Kissner and D. X. Song. Privacy-preserving set operations. In *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, pages 241–257, 2005.

[40] S. Y. Ko, K. Jeon, and R. Morales. The HybrEx model for confidentiality and privacy in cloud computing. In *3rd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'11, Portland, OR, USA, June 14-15, 2011*, 2011.

[41] I. Komargodski and M. Zhandry. Cutting-edge cryptography through the lens of secret sharing. In *Theory of Cryptography - 13th International Conference, TCC 2016-A, Tel Aviv, Israel, January 10-13, 2016, Proceedings, Part II*, pages 449–479, 2016.

[42] J. Li, Z. Liu, X. Chen, F. Xhafa, X. Tan, and D. S. Wong. L-encdb: A lightweight framework for privacy-preserving data queries in cloud computing. *Knowl.-Based Syst.*, 79:18–26, 2015.

[43] C. Liu, L. Zhu, M. Wang, and Y. Tan. Search pattern leakage in searchable encryption: Attacks and new construction. *Inf. Sci.*, 265:176–188, 2014.

[44] W. Lueks and I. Goldberg. Sublinear scaling for multi-client private information retrieval. In *Financial Cryptography and Data Security - 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers*, pages 168–186, 2015.

[45] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. *L*-diversity: Privacy beyond *k*-anonymity. *TKDD*, 1(1):3, 2007.

[46] M. Naor and B. Pinkas. Oblivious polynomial evaluation. *SIAM J. Comput.*, 35(5):1254–1281, 2006.

[47] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 644–655, 2015.

[48] O. Ohrimenko, M. Costa, C. Fournet, C. Gkantsidis, M. Kohlweiss, and D. Sharma. Observing and preventing leakage in mapreduce. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, pages 1570–1581, 2015.

[49] K. Y. Oktay, M. Kantarcioglu, and S. Mehrotra. Secure and efficient query processing over hybrid clouds. In *33rd IEEE International Conference on Data Engineering, ICDE 2017, San Diego, CA, USA, April 19-22, 2017*, pages 733–744, 2017.

[50] K. Y. Oktay, S. Mehrotra, V. Khadilkar, and M. Kantarcioglu. SEMROD: secure and efficient MapReduce over hybrid clouds. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data, Melbourne, Victoria, Australia, May 31 - June 4, 2015*, pages 153–166, 2015.

[51] R. Poddar, T. Boelter, and R. A. Popa. Arx: A strongly encrypted database system. *IACR Cryptology ePrint Archive*, 2016:591, 2016.

[52] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB: processing queries on an encrypted database. *Commun. ACM*, 55(9):103–111, 2012.

[53] M. O. Rabin. How to exchange secrets with oblivious transfer. *IACR Cryptology ePrint Archive*, 2005:187, 2005.

[54] F. Schuster, M. Costa, C. Fournet, C. Gkantsidis, M. Peinado, G. Mainar-Ruiz, and M. Russinovich. VC3: trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 38–54, 2015.

[55] A. Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[56] E. Shmueli, R. Waisenberg, Y. Elovici, and E. Gudes. Designing secure indexes for encrypted databases. In *Data and Applications Security XIX, 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Storrs, CT, USA, August 7-10, 2005, Proceedings*, pages 54–68, 2005.

[57] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *2000 IEEE Symposium on Security and Privacy, Berkeley, California, USA, May 14-17, 2000*, pages 44–55, 2000.

[58] J. J. Stephen, S. Savvides, R. Seidel, and P. Eugster. Practical confidentiality preserving big data analysis. In *6th USENIX Workshop on Hot Topics in Cloud Computing, HotCloud '14, Philadelphia, PA, USA, June 17-18, 2014.*, 2014.

[59] S. D. Tetali, M. Lesani, R. Majumdar, and T. D. Millstein. MrCrypt: static analysis for secure cloud computations. In *Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013*, pages 271–286, 2013.

[60] Q.-C. To, B. Nguyen, and P. Pucheral. Private and scalable execution of SQL aggregates on a secure decentralized architecture. *ACM Trans. Database Syst.*, 41(3):16:1–16:43, Aug. 2016.

[61] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *Proc. VLDB Endow.*, 6(5):289–300, Mar. 2013.

[62] F. Wang, C. Yun, S. Goldwasser, V. Vaikuntanathan, and M. Zaharia. Splinter: Practical private queries on public data. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 299–313, 2017.

[63] S. Wang, X. Ding, R. H. Deng, and F. Bao. Private information retrieval using trusted hardware. *IACR Cryptology ePrint Archive*, 2006:208, 2006.

[64] W. K. Wong, B. Kao, D. W. Cheung, R. Li, and S. Yiu. Secure query processing with data interoperability in a cloud database environment. In *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, pages 1395–1406, 2014.

[65] S. Yu, C. Wang, K. Ren, and W. Lou. Achieving secure, scalable, and fine-grained data access control in cloud computing. In *INFOCOM*, pages 534–542, 2010.

[66] S. Yu, C. Wang, K. Ren, and W. Lou. Attribute based data sharing with attribute revocation. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security, ASIACCS 2010, Beijing, China, April 13-16, 2010*, pages 261–270, 2010.

[67] C. Zhang, E. Chang, and R. H. C. Yap. 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, May 26-29, 2014*, pages 31–40, 2014.

[68] K. Zhang, X. Zhou, Y. Chen, X. Wang, and Y. Ruan. 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, October 17-21, 2011*, pages 515–526, 2011.

[69] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2017, Boston, MA, USA, March 27-29, 2017*, pages 283–298, 2017.

APPENDIX

A. DESIDERATA

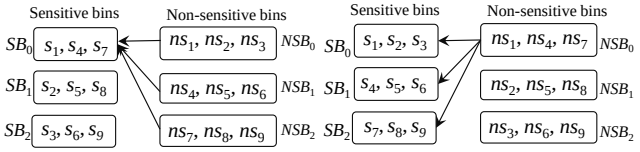
Handling workload-skew attack. In the workload-skew attack, as mentioned in §2, the adversary may estimate which encrypted tuples potentially satisfy the frequent selection predicates, while knowing the frequent selection predicates by observing many queries in the absence of an access-pattern hiding scheme. Note that the workload-skew attack is entirely different from the workload attack, where an *active* adversary having the knowledge of partial workload tries the workload to break a secure scheme, and, in this paper, we are not dealing with workload attack, since we assumed an honest and curious adversary, which cannot launch any attack.

In QB, we have so far assumed that the adversary is unaware of the exact workload-skew (more than what is visible via the adversarial view of the queries) due to uniform query distribution. However, the knowledge of the workload-skew can be exploited by the adversary to learn associations between encrypted and plaintext values, breaking the security of the scheme. We illustrate the workload-skew based attack (see Figure 8a) and also our approach for addressing it for the base case of QB. The case of multiple tuples with multiple values can be generalized trivially.

Figure 8a shows bins created by Algorithm 1 for 9 sensitive values and their associated 9 non-sensitive values. Consider the values ns_1, ns_4, and ns_7 occur most frequently in the query workload. Hence, in this example, the adversary can trivially figure out by observing the sensitive tuple retrieval that only the bin SB_0 has the associated sensitive values with ns_1, ns_4, and ns_7. The reason is that these four bins are retrieved more frequently compared to any other bin. Thus, the adversary can determine that the encrypted values s_1, s_4, and s_7 are associated with either ns_1, ns_4, or ns_7. This is more information than what the adversary had prior to the query execution since each sensitive value, *e.g.*, s_1, could be any of the 9 non-sensitive values. However, it is hard for the adversary to find out which sensitive value out of the three sensitive values of the bin SB_0 is exactly associated with ns_1, ns_4, or ns_7.[16] In order to prevent the workload-skew attack, we need to allocate sensitive values carefully so that the sensitive values associated with frequent selection predicates are distributed over all the bins. The strategy for handling the workload-skew attack in QB under the assumption of having $k \cdot |SB|$, where $k > 0$, frequent selection predicates is as follows:

1. *Create bins*. Find two largest divisors, say $x \geq y$, of $|NS|$, create $NSB = \lceil |NS|/x \rceil$ non-sensitive bins, and x sensitive bins (Lines 3 of Algorithm 1 or Line 6 of Algorithm 3).
2. *Assign non-sensitive values*. Create groups, each of size x, of the frequent predicates, resulting in $u \leq NSB$ groups. Assign one group to one non-sensitive bin. Now, assign all the remaining non-sensitive values, as follows: if any existing non-sensitive bin has less than x values, then assign the remaining values to the non-sensitive bin such that bin size is x, and then, assign further remaining values to other (empty) non-sensitive bins.
3. *Assign sensitive values*. Assign the sensitive values associated with a non-sensitive value, say $ns_j = NSB_z[j]$, where $0 \leq j \leq x-1$, to the j^{th} sensitive bin at the z^{th} position.

[16] We are not assuming that a sensitive bin is not associated with each non-sensitive bin. But, because of non-uniform query distribution, the other bins are retrieved less frequently than the bins having frequent selection predicates.



(a) The workload-skew attack. (b) A solution.

Figure 8: The workload-skew attack and solution under QB, where ns_1, ns_4, and ns_7 are frequent predicates.

By following the above steps, Figure 8b shows 3 sensitive bins in the case of ns_1, ns_4, and ns_7 as the frequent query predicates. Note that the execution of QB using this strategy insists on retrieving all the sensitive bins for answering frequent predicates. Thus, the adversary cannot determine which bucket has a sensitive value associated with the values ns_1, ns_4, or ns_7.[17]

Insert operation and re-binning. For outsourcing new tuples, the DB owner has to wait for new tuples until she collects new sensitive and non-sensitive values equals to the size of an existing sensitive bucket. The QB randomly inserts a new sensitive (or non-sensitive) value in an existing sensitive (or non-sensitive) bin by increasing the size of the bin, if the bin is already full. However, if the newly inserted tuples have a value that already exists in the outsourced sensitive or non-sensitive data, then QB inserts the value in an appropriate bin such that both the bins are retrieved together, if a query asks for the value. However, an insertion of more values in existing bins incur the overhead when retrieving a bin. Hence, Algorithm 1 is re-executed when the overhead crosses a user-defined threshold.

When a query arrives for a sensitive value having an associated non-sensitive value, QB checks the value in all the sensitive and non-sensitive bins and retrieves the two bins. However, if a sensitive (or non-sensitive) value does not hold any associated non-sensitive (or sensitive) value, the DB owner retrieves the bin containing the value with any random non-sensitive (or sensitive) bin. Since the new values are inserted into existing associated sensitive and non-sensitive bins, the retrieval of any value does not remove any surviving matching. Hence, the query execution after an insert operation does not violate partitioned data security.

B. SECURITY PROOFS OF QB

We prove that QB is secure and satisfies the definition of partitioned data security (Theorem 2) by first proving that all the sensitive bins are associated with all the non-sensitive bins (Theorem 1), which is intuitively clear by Example 4. Recall that the only way a surviving match could be removed is if there is no sensitive value in a sensitive bin, say SB_j that does not have an associated non-sensitive value. In this case for answering a value belonging to SB_j, we retrieve either only the bin SB_j or the bin SB_j with any randomly selected non-sensitive bin. Note that the adversary cannot learn anything from the encrypted data, since the keys are only known to the DB owner.

Theorem 1 *Let $|S|$ and $|NS|$ be the number of sensitive and non-sensitive values, respectively. By following Algorithm 1, $|S|$ and*

[17] If there are less then y frequent predicates in a non-sensitive bin, then we need to send fake queries as frequent as frequent predicates, leading to retrieval of each sensitive bin.

$|NS|$ values are distributed over SB sensitive and NSB non-sensitive bins, respectively. Answering a set of queries using QB (Algorithm 2) will not remove any surviving matches of the bins and that leads to preserve all the surviving matches of the values.

PROOF. We show that QB will not remove any surviving matches of the bins by showing that a sensitive bin, say SB_j, must be associated with all the non-sensitive bins. A similar argument can be proved for any non-sensitive bin. Let y be the number of sensitive values in the bin SB_j, and let $p \geq y$, ($p = NSB$) be the number non-sensitive bins. We will prove the following three arguments:

1. If a sensitive value, say $s_i \in SB_j$, is associated with a non-sensitive value (*i.e.*, $\exists ns_z \in R_{ns} : ns_z \stackrel{a}{=} s_i$), then two bins, SB_j, and one non-sensitive bin, holding the value ns_z, are retrieved.
2. If a sensitive value, say $s_i \in SB_j$, is not associated with any non-sensitive value (*i.e.*, $\forall ns_j \in R_{ns} : s_i \stackrel{a}{\neq} ns_j$), then the bin SB_j and one of the non-sensitive bins are retrieved. Following that, if all the sensitive values of the bins SB_j are not associated with any non-sensitive value (*i.e.*, $\forall ns_j \in R_{ns}, \forall s_i \in SB_j : s_i \stackrel{a}{\neq} ns_j$), then the bin SB_j and y different non-sensitive bins are retrieved.
   By proving the first and second arguments, we will show that if there are *only* y non-sensitive bins, then a sensitive bin must be associated with all the y non-sensitive bins. The following third argument will consider more than y non-sensitive bins.
3. If there are more than y non-sensitive bins (say, $NSB_y, NSB_{y+1}, \ldots, NSB_p$) having x values that are not associated with any sensitive value (*i.e.*, $\forall ns_j \in NSB_y \vee NSB_{y+1} \vee \ldots \vee NSB_p, ns_j \stackrel{a}{\neq} s_i, i = 1, 2, \ldots, |S|$), then each of these non-sensitive bins must be associated with the bin SB_j.

By satisfying the above three arguments, we prove that, thus, the bin SB_j is associated with all non-sensitive bins, and hence, all surviving matches of the bins and, eventually, values are preserved.
First case. The value s_i is allocated to i *modulo* x sensitive bin at an index, say z, where $z = 0, 1, \ldots y-1$, and its associated non-sensitive value is allocated to the i *modulo* x position of the z^{th} non-sensitive bin. When answering a query for s_i according to the rule R1, the bin SB_j with the bin NSB_z are retrieved. Consequently, the desired tuples containing s_i and its associated non-sensitive value are retrieved, and that are correct answers to the query.
Second case. When answering a query for the value $s_i = SB_j[u]$ ($u \in 0, 1, y-1$) that does not have any associated non-sensitive value, by following the rule R1, the bin SB_j with one of the non-sensitive bin NSB_u are retrieved. Moreover, answering queries for all the y values (0, 1, $y-1$) of the bin SB_j, by following rule R1, requires us to retrieve the SB_j with all the $y-1$ (0, 1, $y-1$) non-sensitive bins.
Third case. Since the non-sensitive bin, say NSB_z, where $z = y, y+1, \ldots, p$, must hold a value at the j^{th} position, by following the rule R2, the bin NSB_z and the sensitive bin SB_j are fetched for answering a query for ns_j.

Therefore, the bin SB_j is associated with all the non-sensitive bins, and hence, all the surviving matches between the values of the bin SB_j and all the non-sensitive bins are also maintained. □

Since we proved all sensitive bins are associated with all the non-sensitive bins, based on this fact, we will show that the first condition of partitioned data security holds to be true for any query. Here, we do not show the second equation of partitioned data security definition (*i.e.*, $Pr_{adv}[s_i \stackrel{r}{\sim} s_j|X] = Pr_{adv}[s_i \stackrel{r}{\sim} s_j|X, q(W)(R_s, R_{ns})[A]]$); recall that here in the base case, we assumed that a value has only a single sensitive tuple; hence, the condition holds true.

Theorem 2 (Preserve partitioned data security) *Let X be auxiliary information about the sensitive data, R_s and R_{ns} be the sensitive and non-sensitive relations, respectively. s_i is the i^{th} sensitive value in an attribute, say A, of the relation R_s and ns_i is the i^{th} non-sensitive value in the attribute A of the relation R_{ns}. An execution of a set of queries on the attribute A on the relations using QB leads to the following equation to be true: $Pr_{adv}[s_i \stackrel{a}{=} ns_j|X] = Pr_{adv}[s_i \stackrel{a}{=} ns_j|X, q(W)(R_s, R_{ns})[A]]$, where $i \in 1, 2, \ldots, |S|$ and $j \in 1, 2, \ldots, |NS|$.*

Proof sketch. We provide an example of four values to show the correctness of the above theorem. Let v_1, v_2, v_3, and v_4 be values containing only one sensitive and one non-sensitive tuples. Let E_1, E_2, E_3, and E_4 be encrypted representation of these values in an arbitrary order, *i.e.*, it is not mandatory that E_1 is the encrypted representation of v_1. In this example, the cloud stores an encrypted relation, say R_s, containing four encrypted tuples holding encrypted representations E_1, E_2, E_3, E_4 and a clear-text relation, say R_{ns}, containing four clear-text tuples containing values v_1, v_2, v_3, v_4. The objective of the adversary is to deduce a clear-text value corresponding to an encrypted value. Note that before executing a query, the probability of an encrypted value, say E_i, to have the clear-text value, say v_i, $1 \leq i \leq 4$ is 1/4, which QB maintains at the end of a query.

Assume that the user wishes to retrieve the tuple containing v_1. By following QB, the user asks a query, say $q(E_1, E_3)(R_s)$, on the encrypted relation R_s for E_1, E_3, and a query, say $q(v_1, v_2)(R_{ns})$, on the clear-text relation R_{ns} for v_1, v_2. After executing the queries, the adversary holds an adversarial view given in Table 9.

Exact query value (hidden from adversary)	Returned tuples/Adversarial view	
	Sensitive data	Non-sensitive data
v_1	E_1,E_3	v_1,v_2

Table 9: Queries and returned tuples/adversarial view after executing a query for v_1, by following Algorithm 2.

In this example, we show that the probability of finding the clear-text value of an encrypted representation, say E_i, $1 \leq i \leq 4$, remains identical before and after a query. In order to show that when a query comes for $2 \times \sqrt{n}$ values by following QB, where n is the number of values in the non-sensitive relation, $\sqrt{n}$ values are asked for the sensitive relation and $\sqrt{n}$ values are asked for the non-sensitive relation, we need to figure out:

1. All possible allocations of the non-sensitive $\sqrt{n}$ values, say $v_1, v_2, \ldots, v_{\sqrt{n}}$, to $\sqrt{n}$ encrypted sensitive values, say $E_1, E_2, \ldots, E_{\sqrt{n}}$. Here, we use the term *allocation* to show the fact that the encrypted representation of E_i has the clear-text value v_i.
   In our example of four values, we find allocations of four non-sensitive values v_1, v_2, v_3, v_4 to encrypted representation E_1, E_2, E_3, E_4.
2. All possible allocations of $\sqrt{n}$ non-sensitive values, except one non-sensitive value, say v_i, that is allocated to an encrypted sensitive values, say E_i, to the remaining encrypted sensitive values.
   In the case of four values and above-mentioned queries, we find allocations of the non-sensitive values v_2, v_3, v_4 to the encrypted

sensitive values E_2, E_3, E_4 while assuming that the encrypted representation of v_1 is E_1.

The ratio of the above two provides the probability of finding a clear-text value corresponding to its encrypted value after the query execution.

When the query arrives for $\langle E_1, E_3, v_1, v_2\rangle$, the adversary gets the fact that the clear-text representation of E_1 and E_3 cannot be v_1 and v_2 or v_3 and v_4. If this will happen, then there is no way to associate a sensitive bin 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

$$\langle v_1, v_2, v_3, v_4\rangle, \langle v_1, v_2, v_4, v_3\rangle,$$
$$\langle v_1, v_3, v_4, v_2\rangle, \langle v_1, v_4, v_3, v_2\rangle.$$

However, the allocations $\langle v_1, v_3, v_2, v_4\rangle$ and $\langle v_1, v_4, v_2, v_3\rangle$ to E_1, E_2, E_3, and E_4 cannot exist. 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. This results in four more possible allocations of the values to E_1, E_2, E_3, and E_4, such as

$$\langle v_2, v_1, v_3, v_4\rangle, \langle v_2, v_1, v_4, v_3\rangle,$$
$$\langle v_2, v_3, v_4, v_1\rangle, \langle v_2, v_4, v_3, v_1\rangle.$$

However, $\langle v_2, v_3, v_1, v_4\rangle$ and $\langle v_2, v_4, v_1, v_3\rangle$ cannot exist. Similarly, assuming the clear-text representation of E_1 is v_3 or v_4, we get 8 more possible allocations of the values to E_1, E_2, E_3, and E_4, such as:

$$\langle v_3, v_1, v_2, v_4\rangle, \langle v_3, v_2, v_1, v_4\rangle,$$
$$\langle v_3, v_4, v_1, v_2\rangle, \langle v_3, v_4, v_2, v_1\rangle,$$
$$\langle v_4, v_1, v_2, v_3\rangle, \langle v_4, v_2, v_1, v_3\rangle,$$
$$\langle v_4, v_3, v_1, v_2\rangle, \langle v_4, v_3, v_2, v_1\rangle.$$

Here, the following four allocations of the values to encrypted representation cannot exist:

$$\langle v_3, v_1, v_4, v_2\rangle, \langle v_3, v_2, v_4, v_1\rangle,$$
$$\langle v_4, v_1, v_3, v_2\rangle, \langle v_4, v_2, v_3, v_1\rangle.$$

Thus, the retrieval of the four tuples containing one of the following: $\langle E_1, E_3, v_1, v_2\rangle$, 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. A similar argument also hold for other encrypted values. Hence, an initial probability of associating a sensitive value with a non-sensitive value remains identical after executing a query.

Thus, we can conclude the following:

1. All possible allocations of $\sqrt{n}$ non-sensitive values, except one non-sensitive value, say v_1, that we allocate to an encrypted sensitive values, say E_1, to the remaining encrypted sensitive values is $(n-1)! - x$, where n is the number of values in the non-sensitive relation and x is the number of allocations of values $v_2, v_3, \ldots, v_{\sqrt{n}}$ to $E_2, E_3, \ldots, E_{\sqrt{n}}$ that cannot exist.
2. All possible allocations of the non-sensitive $\sqrt{n}$ values, say $v_1, v_2, \ldots, v_{\sqrt{n}}$, to $\sqrt{n}$ encrypted sensitive values, say $E_1, E_2, \ldots, E_{\sqrt{n}}$, is $n \times ((n-1)! - x)$. This is true because we cannot allocate any combination of the values asked in the query to any encrypted representations that are asked by the query.

Thus, the retrieval of $2\times\sqrt{n}$ values results in $n\times((n-1)!-x)$ possible allocations of $\sqrt{n}$ non-sensitive values to $\sqrt{n}$ encrypted sensitive values, while $(n-1)!-x$ allocations exist when a queried non-sensitive value is assumed to be the clear-text of a queried encrypted representation. Therefore, the probability of finding the exact allocation of the non-sensitive values to encrypted sensitive value while considering a non-sensitive value is the clear-text of an encrypted value is $\frac{(n-1)!-x}{n\times((n-1)!-x)} = \frac{1}{n}$.

C. PROOF OUTLINE FOR THE GENERAL CASE

The following lemma shows how many fake tuples are required to be added to a sensitive bin, resulting in an identical number of tuples in each sensitive bin.

Lemma 1 *Let t_l and t_s be the largest and smallest number of tuples have an identical value, respectively. By following the strategy presented in Section 5.3, the strategy adds at most $max(t_l, t_l - t_s)$ fake tuples in a sensitive bin to have an identical number of tuples in each bin.*

Now, we borrow notations from Theorem 2 that also holds for the general case of QB and give a proof outline of the second condition of partitioned data security.

Theorem 3 (Preserve partitioned data security) *An execution of a set of queries on an attribute, say A, on the relations R_s and R_{ns}, where values of the attribute A have a different number of tuples, using QB leads to the following equations to be true: $Pr_{adv}[v_i \overset{r}{\sim} v_j|X] = Pr_{adv}[v_i \overset{r}{\sim} v_j|X, q(W)(R_s, R_{ns})[A]]$, where $v_i, v_j \in Domain(A)$ of the relation R_s.*

Proof sketch. Recall that we are not dealing with how does an encryption mechanism reveal an order of sensitive values. In our context, the probability of relating any two sensitive values will not be identical when the number of output tuples to a query to one of the values is more than the number of tuples to a query to another value. Since we add fake tuples to sensitive bins, each sensitive bin has an identical number of tuples. Hence, having a heavy-hitter non-sensitive value that is associated with a heavy-hitter sensitive value, the adversary cannot distinguish between any two sensitive bins and deduce which sensitive bin holds the heavy-hitter sensitive value. Thus, the probability of relating two sensitive values remains a constant after answering queries for any two values.