UNIVERSITY OF CALIFORNIA,
IRVINE

Exploiting Partial Sensitivity of Data Using Client Side Buckets

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Science

by

Anurag Mishra

2017

DEDICATION

To mom and dad

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank Professor Sharad Mehrotra for giving me an opportunity to work with him. I am thankful to him for believing in me and guiding me whenever I saw no light.
I would also like to thank Shantanu Sharma for his invaluable guidance.

ABSTRACT OF THE THESIS

Exploiting Partial Sensitivity of Data Using Client Side Buckets

By

Anurag Mishra

MASTER OF SCIENCE in Computer Science

University of California, Irvine, 2017

Professor Sharad Mehrotra, Chair

The world today runs on data and lots of private data, whose security is of paramount importance. We have many secured cryptographic techniques which are practical but not completely secure. On the other hand we have completely secure techniques which are not practical to be run as far as time complexity is concerned. In this scenario, we would like to exploit the fact that there is partial sensitivity of data. We partition the data into sensitive and non sensitive parts with the former being expensive to retrieve while the later is comparatively inexpensive. We propose that combining both the costs using our scheme gives better result than considering the entire data as sensitive. We will prove this using mathematical models as well as by experiments.

Chapter 1

Introduction

The traditional model of computing dictated all computing resources necessary for the running of the software system be in-house. So, for years we had the stacks of disks for storing data and all the horsepower for processing them within the physical (or in certain case the logical) premises. The primary reason was the prohibitively expensive cost of the above. But things started to change with the growth of the Internet and with the gradual decrease in the cost of hardware, there was a push to move computing infrastructure outside or in other words the gradual move to cloud.

1.1 Cloud Computing

Although the notion of cloud computing is not new, the term was used for the first time by Eric Schmidt, the CEO of Google. Cloud computing is an amalgamation of myriad technologies like Virtualization, Grid Computing, Utility Computing and so on. Before we delve further into the topic we would like to have a formal definition of cloud computing. We will adopt the definition provided The National Institute of

Standards and Technology (NIST) [43].

Definition of Cloud Computing as per NIST: *Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction.*

We also need to distinguish between infrastructure providers, who own and manage the necessary infrastructure and the services providers who lease computing resources from the infrastructure providers to solve some problem for the end clients.

With the formal definition of cloud computing out of our way, we can go over some of the merit associated with cloud computing.

One advantage of cloud computing is that the service provider does not have to bear the hefty cost of infrastructure up-front. Most of the infrastructure providers have a pay-as-you-go model. The service provider pays the infrastructure provider for every hour of service consumed. Another hidden advantage of this model is the elastic nature of scaling. Businesses generally have peak and non-peak periods of activity. The cloud model of computing is perfectly adaptable to this changing nature of demand. Upscaling and downscaling of resources is a breeze in the cloud, at least as compared to the traditional model of computing. As we have already mentioned, the general revenue model is that of pay-as-you-use, the service provider can requisition more resources when there is a surge in computing need and then easily scale back when the demand ebbs.

In the current era, most service providers are expected to have as low a downtime as possible. Cloud computing again comes to the rescue. The onus of having a high up-time is now on the infrastructure provider, who generally practice very stringent Service Level Agreement(SLA). Effectively, part of the risk of any service provider is now outsourced to the infrastructure provider. Cloud computing profits from economies of scale. A multi-tenant architecture provides the pooling of common resources and the cost associated with them is shared, thus lowering the overall cost for all. An infrastructure provider concentrates primarily on providing best infrastructure. Thus, they can invest in better personnel and better resources as well.

Physical View of Cloud Architecture We can view the cloud architecture as consisting of many loosely coupled layers stacked on top of one another. The bottom most layer is hardware. Storage devices like disks, computing devices like CPU (and/or GPU), networking devices like routers form the bulk of this layer. The layer above will have some virtualization support like XEN, KVM, or Hyper-V. The virtualization layer is responsible for compartmentalization of underlying resources for layers above. On top of this virtualization layer lies the API friendly layer which exposes standard low-level services like database service, transaction processing services. On top of this providers can also provide service specific layer.

Logical View of Cloud Architecture The loose coupling of the physical layers allows a logical grouping as well popularly known as SPI model. The lowest two layers coupled together forms Infrastructure as a Service (IaaS). Amazon EC2 is a good example. Platform as a Service (PaaS) consist of the bottom three layers, and thus low-level services like databases are exposed. Microsoft Windows Azure is prime candidate for PaaS. Going even further, there is Software as a Service which provides client specific services like Microsoft Office 365. Earlier we had distinguished between infrastructure providers and service providers. The infrastructure provider maps to IaaS and PaaS taken together and service provider maps to SaaS as per the logical view of cloud architecture.

In the above discussion, we have not mentioned the ownership of the cloud. Based on the ownership, the cloud can be public, private or in some cases hybrid.

We will take up Microsoft Azure Platform as an example to look at the various components. The stack consists of Windows Azure which is a general-purpose environment which supports .NET framework based software like those running inside Windows operating system. SQL Azure consists of SQL Azure database which is reckoned to be built atop of Microsoft SQL Server. Then there is .NET Services which exposes underlying features without explicitly requisitioning them.

1.2 Security in Cloud Computing

We have seen so many wonderful advantages of cloud computing. But like everything else, it has its own share of limitations. The most prominent ones are security followed by privacy. The control that one can have over the service decreases as one goes higher up the stack. Thus, if we use SaaS, we have the least amount of control while we have the highest level of control if we use IaaS. Here control means the control as a consumer of the service. For illustration purposes, consider SaaS with Microsoft Office 365 as an example. In this setup, the end user might not be aware where the underlying datacenter exists or what storage is being used. The security and uptime of the service is assured by the provider.

In a traditional setup security was propped up by the pillars of authentication and authorization. But they are no longer sufficient in case of cloud [39].There is the risk associated with public internet, multi-tenancy, data storage. To mitigate these concerns the cloud service providers must provide the required level of compliance for auditing purposes [51].

We will analyze various threats that each logical model suffers from.

- **SaaS** The service that is being consumed can be shared among the tenants in multiple ways. At one end of the spectrum, each consumer will have its own customized version of the software running; while at the other extreme, there can be a single instance of the program running for all users. The later allows more efficient use of resources, but has higher vulnerability. Very strict and efficient security policies are required to separate the data of each tenant [8]. In the SaaS model, the consumer of the service has no control over the underlying storage system. The provider is responsible for secured storage and processing of data. Further the backups taken most be stored securely and with proper authorization. The service offered by the provider are accessed over the public internet which has inherent security threats like man in the middle attack. Secured channels can be established between the client and service providers to mitigate this concern.

- **PaaS** The security of PaaS is very tightly coupled with the underlying IaaS. It is assumed that any breach in one of the guest environment will not affect another. But if the hypervisor which is responsible for hosting all the guest is compromised, then all the guests can leak sensitive information.

- **IaaS** Cloud computing is relatively cheap because it can optimize the use of resources among many users. By the clever use of software, the underlying hardware is shared among consumer of services. This means the same disks and memory is shared by many applications. Thus, any data in memory can be snooped upon by another process which has higher privileges. In many cases, there are public repository of Virtual Machines which can be copied and spun up. They are generally free of malicious content. But if the base is compromised, then the data of an unsuspecting user is under threat.

1.3 Existing Techniques

Over the years, information retrieval has gained significant research attention, leading to a fundamental database operation, *i.e.*, keyword-based search (*e.g.*, DISCOVER [31] and [9]) without knowing the schema of the database, and secure keyword query search [10, 21, 61, 25, 11], which helps in secure data outsourcing to the cloud. Broadly, existing research on secure query keyword search and tuple retrieval can be classified into three categories as follows:

- **Encryption based techniques** examples of which include order-preserving encryption [2], deterministic encryption [7], homomorphic encryption [22], bucketization [30], searchable encryption [56], private informational retrieval (PIR) [16], practical-PIR [60], oblivious-RAM (ORAM) [27], oblivious transfers [50, 32], oblivious polynomial evaluation [44], oblivious query processing [5], searchable symmetric encryption [17], and distributed searchable symmetric encryption [33].
- **Secret-sharing (SS) [53] based techniques** that include distributed point function [25], function secret sharing [11], functional secret sharing [37], Splinter [59], and others [20, 41].
- **Trusted hardware based techniques** that include [4, 3, 6, 52, 18, 58, 68].

While a large body of research and corresponding systems both within the research and open source community and also the commercial database vendors exist, existing techniques and systems exhibit several significant shortcomings. To appreciate shortcomings of existing technology, we first need to better appreciate features we desire from the secure solution. In our view, these include: (i) *efficiency*: the secure solution must not be orders of magnitude worse than the traditional database solution.

In particular, the overheads must not be so high so as to mitigate the advantages of the cloud pushing database owners to instead revert back to traditional on-site data management; (*ii*) *security*: the secure cloud solution should not leak sensitive data to the adversary (*i.e.*, the cloud).

Existing techniques, unfortunately, compromise on one or both of the above-mentioned desirable features/requirements of secure data outsourcing solutions. For instance, cryptographic approaches such as fully homomorphic encryption (FHE) coupled with ORAM that provide strong security guarantees and prevent information leakage incur very high overhead and do not scale to (real) large datasets and complex queries. The computational time for such techniques even for answering simple query [24] is unacceptable. Mechanisms based on secret-sharing are potentially more scalable than FHE. However, such techniques split data among multiple non-colluding clouds resulting in a significant communication overhead and can only support a limited set of selection and aggregation queries efficiently. In contrast, several partially homomorphic techniques that are significantly more scalable compared to FHE have been developed. Such techniques, however, limit the computations that can be performed on encrypted data and, furthermore, may also compromise security. For instance, Gentry and Halevi [23] has argued a fundamental tradeoff between preserving data privacy and utility, highlighting that a system that supports a more secure technique, (*e.g.*, ORAM and nondeterministic encryption) does not allow different types of operations to be applied on the data. Further, the papers [45, 35] show that when order-preserving and deterministic encryption techniques are used together, on a dataset in which the entropy of the values is not high enough, an attacker might be able to construct the entire clear-text by performing a frequency analysis on the encrypted data. Moreover, these systems are unable in handling the size attack, where an adversary having some background knowledge can deduce the outputs (by simply observing output sizes).

While the body of knowledge in secure data outsourcing is improving, and the security guarantee supported by new systems is improving with time (albeit at higher costs), the race to develop cryptography-based solutions that (*i*) are efficient, and (*ii*) offer provable security from the application's perspective is still ongoing.

1.4 Motivation

Recently, given the difficulty in developing a cryptography-based solution to support efficient and secure data outsourcing, several papers [36, 67, 66, 47, 46, 68] have proposed a new class of solution based on the observation that an entire dataset is not typically sensitive. For instance, the discussion in popular media on secure cloud adoption [1] clearly identifies classifying organizational data into sensitive (for which public cloud cannot be trusted) and non-sensitive (the loss of which is considered acceptable risk for the advantages of the cloud model) as one of the key strategies for adopting the cloud model. The data classification is not too difficult to see if we consider specific data outsourcing cases. For instance, as discussed in [46] in a university scenario, data such as class schedule, a list of faculties, faculty offices, faculty salary in public schools, a catalogue of courses, etc. are all non-sensitive whereas data about disabilities of students, their course enrollment, and grades would be sensitive. Existing research has explored exploiting the sensitive vs non-sensitive data classification in several data outsourcing contexts.

The papers [36, 67, 66, 47] have explored secure MapReduce (MR) system implementations while [46, 68] have explored secure SQL data processing. The secure MR solutions as well as the secure SQL execution have been explored in the context of hybrid clouds wherein data and computations are partitioned between the public and private clouds in such a way that sensitive data or any information about it is not leaked to the public cloud. In [18, 52, 68], secure hardware on the cloud in the form of Intel SGX is used to partition the computation.

Our thesis builds upon the above trend towards pragmatic security in cloud-based data outsourcing systems for keyword search queries with provable security guarantees. Similar to previous approaches, we exploit the fact that not all data might be sensitive but unlike existing hybrid cloud technique, we develop a cloud-only solution wherein data is entirely stored on the public cloud. Furthermore, unlike [18, 52, 68], our approach does not exploit any specialized *secure hardware that is also not secure under the size attack*, which will be discussed later in Section 4.3.

Naturally, to store sensitive data on the public cloud, we need to encrypt such data while non-sensitive data may be stored on the cloud in the clear-text. In our proposed approach, a query executes on both encrypted sensitive data as well as plaintext non-sensitive data, and the results of the *matches* returned to the data owner/querier, where they are appropriately decrypted, possibly filtered, and merged. It is important to mention that *we are not changing/modifying any existing secure/non-secure databases, hence, our approach can be used on top of any database.*

Specifically, our approach treats encrypted data processing of the sensitive data as an oracle or a blackbox for which (above-mentioned) any known partially homomorphic, searchable encryption technique, FHE, PIR, SS approaches, or SGX-based solutions can be used. In developing our approach, we will assume that the underlying cryptographic solution is based on non-deterministic encryption, though, such an assumption is not necessary. In particular, the proposed approach can be easily extended to other encryption techniques, SS techniques, or SGX-based solutions. *Our focus is on joint processing of non-sensitive data in the clear-text along with the blackbox processing of encrypted sensitive data in such a way that it prevents any information leakage as a result of the joint processing.* However, such a joint processing could result in leakage, which is also possible in current SGX-based cloud models [18, 52, 68], illustrated through an example below.

Tuple id	EId	First name	Last name	SSN	Salary	Department
t_1	E101	Adam	Smith	111	1000	Defense
t_2	E102	John	Williams	222	2000	Design
t_3	E103	Eve	Smith	333	500	Design
t_4	E102	John	Williams	222	5000	Defense

Figure 1.1: A relation: *Employee*.

Example 1 (Employee relation).: *Consider an* Employee *relation, which is given in Figure 1.1. In this relation, the attribute* SSN *might be deemed sensitive, and furthermore, all tuples of employees for the* Department = *"*`Defense`*" might be considered sensitive. In such a case, the* Employee *relation may be stored as three relations, as follows: (i)* Employee1 *with attributes* EId *and* SSN*; (ii)* Employee2 *with attributes* EId*,* First Name*,* Last Name*,* Salary*, and* Department*, where* Department = *"*`Defense`*"; and (iii)* Employee3 *with attributes* EId*,* First Name*,* Last Name*,* Salary*, and* Department*, where* Department <> *"*`Defense`*". In these relations,* Employee1 *and* Employee2 *contain only sensitive data, while* Employee3 *contains only non-sensitive data.*

Information leakage through non-sensitive data. Now, we show how query execution on the partitioned data can lead to the information leakage through non-sensitive data using the example. We consider an *honest-but-curious* adversarial cloud that will return the correct answers to the query but wishes to know information about the encrypted sensitive tables, *Employee1* and *Employee2*. Consider three queries on the *Employee2* and *Employee3* relations, as follows: (*i*) find all the details of `E102`, (*ii*) find all the details of `E101`, and (*iii*) find all the details of `E103`.

When answering a query, the adversary knows the tuple ids of retrieved encrypted tuples[1] 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*; shows in Table 1.1. Note that $E(t_i)$ shows encrypted t_i tuple.

Query value	Returned tuples/Adversarial view		
	Employee1	Employee2	Employee3
E102	$E(t_2)$, $E(t_4)$	$E(t_4)$	t_2
E101	$E(t_1)$	$E(t_1)$	null
E103	$E(t_3)$	null	t_3

Table 1.1: Queries and returned tuples/adversarial view.

Note that outputs of the three queries will reveal enough information in learning about sensitive data. In the first query, the adversary learns that `E102` works in both sensitive and non-sensitive departments, because the answers obtained from the three relations contribute in the final answer. 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 `E103` works only in a non-sensitive department.

1.5 Problem Statement and Contribution

We assume the existence of both sensitive and non-sensitive data. Our goal is to increase the efficiency of query answering by taking advantages of the fact that part of the data is non-sensitive, while preserving the security of the sensitive part (unlike in Example 1). Specifically, we provide a technique, entitled *query bucketization* (QB), to prevent information leakage (such as that shown in Example 1) that may result from the joint processing of sensitive and non-sensitive data on the cloud. We note that our technique does not prevent leakage that may result solely from encrypted

[1]The adversary may not learn the tuple ids of encrypted sensitive tuple when using techniques such as PIR, ORAM, or SS.

data processing of sensitive data. For instance, if we use a weak encryption strategy (*e.g.*, deterministic encryption) to encrypt sensitive data, then information about which sensitive records contain the same value would leak.

Our goal in this thesis is not to develop a new cryptographic solution to search, but to explore how existing cryptographic techniques can be safely composed with the query bucketization approach that allows for non-sensitive data to be stored in the clear-text without any new additional leakage. To this effect, we define a concept of leakage through non-sensitive data and prove that the proposed query bucketization approach prevents any such leakage. Note that a direct corollary is that if the encrypted data processing approach (used in conjunction with the query bucketization) ensures the security of sensitive data, the combined approach will offer perfect security.

The query bucketization approach can, thus, be viewed as a pragmatic optimization of existing (and future) cryptographic approaches that offer at least an identical level of security offered by those techniques(our technique can prohibit size attack, so in cases it will provide a superior level of security), but improves efficiency by allowing non-sensitive data to be processed in the clear-text form. We note that we primarily develop the thesis in the context of keyword query though we offer extensions to the approach to handle range and other more complex selection queries. Extending the approach to joins and more complex SQL queries remains future work.

Chapter 2

Related Work

2.1 Encryption Based Techniques

Order Preserving Encryption

Order preserving Encryption(OPE) was first introduced in 2004 [2]. This encryption scheme guarantees that if , X $\geq$ Y, then Enc(X) $\geq$ Enc(Y). Thus the order among keywords is preserved even after encryption and so it supports range queries and sorting operations very effectively. But this scheme leaks information and to rectify this problem a modular OPE technique [42] has been proposed. This technique adds a secret offset before encrypting using standard OPE. This reduces the likelihood of knowing the location of the data item in the encrypted storage.

Deterministic Encryption

Deterministic Encryption guarantees that if X = Y , then Enc(X) = Enc(Y). There is a single key which is used to encrypt all the values. This allows the server to run point queries effectively. But this is prone to statistical attacks.

Non-Deterministic Encryption

Contrary to deterministic encryption discussed above, the non-deterministic encryption makes sure that for the same plain text, different cypher text will be generated. One can get this result by using different encryption key for each keyword. This setup is very secure, but the server is unable to process even basic point queries natively.

Homomorphic Solution

Fully Homomorphic Encryption(FHE) was introduced by [22]. This scheme allows addition and multiplication over the encrypted text and with recent advancements, this scheme allows any function to be evaluated over the encrypted text. FHE guarantees very strong encryption but on the downside, the cost of operation is prohibitively high. Thus, FHE is not used in practical encryption techniques. If a constraint is added on the type of function to be applied, then one can use Partial Homomorphic Encryption(PHE). The security of PHE is same as that of FHE, while the cost of operation is significantly less than FHE. Some of the popular PHE systems are, Paillar [48] scheme which supports Addition and ElGamal [19] scheme which supports multiplication.

Oblivious RAM

The server can see the access pattern and form frequency and co-occurrence information. To hide such access pattern, oblivious RAM has been proposed. ORAM is a construction which translates the logical address of the application to a physical address on the server. The ORAM setup guarantees that all the sequence of physical address generated are of identical probability for the input sequence of physical addresses. In other word the adversary cannot distinguish between two physical address sequences generated for two input logical sequence of addresses.

In the most trivial solution, ORAM will read all the items for each operation. So, adversary cannot mark the items of interest. This will lead to an overhead of $O(n)$ for every operation. The first reasonable implementation of ORAM was first put forward by Goldreich and Ostrovsky [28] as Square Root ORAM. The fundamental building block in this scheme is an oblivious sort. Two popular sorting network based algorithm are AKS $O(n\log^2 n)$ and that of Batcher $O(n\log n))$. Another oblivious sort algorithm was proposed by Goodrich [29] which uses a randomized modified version of shellsort. The amortized overhead cost in Square Root ORAM is $O(\sqrt{n}\log^2 n)$ which is significantly less than $O(n)$, the trivial solution. There was a further reduction of the overhead by using a Hierarchical Solution, which reduced the cost to an amortized $O(\log^3 n)$. The worst case was still $O(n\log^2 n)$.There were many gradual improvement of the above approach reducing the amortized cost. A novel construction for ORAM, Tree based ORAM, was proposed by Shi et al. [55] which had the worst case overhead of $O(\log n^3)$. A slight modification with a greedy eviction strategy was proposed in Path ORAM [57].

Private Information Retrieval

Consider a database which has n items, numbered i_1 to i_n. A client would like to retrieve a particular item i_j from the server without revealing the value of i. This can be achieved using private information retrieval(PIR). PIR comes in primarily two flavors.One of them is having replicated database which do not collude among themselves. The other option is to have a single database and employ cryptographic techniques. The latter is called single database computationally private information Retrieval(cPIR).

A trivial solution would be to download the entire database to the client, which will not reveal any information. But this will have a linear communication cost with respect to the size of the database. Chor et al. [15] introduced a sub-linear communication scheme which involved replicated databases. Single database PIR solution was proposed by [38] Kushilevitz and Ostrovsky, with a communication cost of $(n-O(\frac{n}{k}-k^2))$ and more than one round of interaction between client and server. As far as accessing the elements of the database is concerned, the PIR scheme must touch all n elements. If it does not, then the adversary can infer that some elements do not satisfy by query of the client. To solve the above problem, the problem was reduced to retrieve k elements from the database, which allows to partition the data into k bins and using explicit batch codes [34]. One of the ways to develop a PIR system is to use a homomorphic encryption system in the background. Kushilevitz and Ostrovsky [38] used the homomorphic system of Goldwasser and Micali for constructing PIR. Similarly the homomorphic system of Pailler was used by Chang [13] for PIR. Another school of thought has been to use $\Phi-$Hiding Assumptions. The PIR scheme of Lipmaa [40] has reduced the communication cost to $\theta(k \log^2 n + l \log n)$, where k is a non-constant security parameter.

Searchable Symmetric Encryption

ORAM and Fully Homomorphic encryption are considered to be very expensive. This motivated the search for other searchable encryption techniques with provable security. Song, Wagner and Perrig were the first to introduce Searchable Symmetric Encryption(SSE) [56]. Further work on the security definitions of SSE was done by Eu-Jin Goh [26]. The assumption, till then, was that the adversary was non-adaptive. This limitation was removed from the security definition by Curtmola et al. [17]. They allowed the adversary to be adaptive and still there should be no leakage of access and search pattern. The construction proposed for the searchable encryption was to use an inverted index solution as described in [17] having a search complexity $O(|DB[W]|)$. Initially the scheme had a few limitations which were overcome gradually. First the scheme was made adaptively secure by Chase and Kamara [14] and further refinement to allow the scheme to be dynamic was done by Cash et al. [12].

2.2 Secret Sharing Techniques

Secure multi-party computation

In a secure multi-party computation setup; there are n participants say $P_1, P_2, \ldots P_n$ and each of the participant has some value $V_1, V_2, \ldots V_n$. A target function must be calculated whose input requires all the values from $V_1, V_2, \ldots V_n$ but none of the participant should know any value other than the value they possess. There is an added constraint that they cannot use any secured participant who is not part of the system to calculate the target function. Historically, the problem was put forward as a secure two party computation by Yao [62]. The solution is famously known as Yao's Garbled Circuit [63]. The system can be extended to multi-party protocols where there are sophisticated solution like Shamir Secret Sharing and Replicated Secret Sharing.

Shamir Secret Sharing

To uniquely determine a polynomial in K-1 degree, a minimum of K points are required. K-1 points cannot uniquely determine the polynomial. This concept is exploited in Shamir Secret Sharing setup [54]. The data is divided into n parts and K is the threshold for the minimum number of parts required to reconstruct the original data. If K = n, then all the parts are required. Ideally n is greater than K. To find the coefficients, from the k parts, Lagranges interpolation is applied on the k data items. The scheme is mathematically proven to be secure as any less than k parts cannot recreate the original data.

Chapter 3

System Setting

The model. We assume two entities in our model, as follows:

1. *A database (db) owner*: who divides a relation R having attributes, say $A_1, \ldots, A_n$, into two relations based on data sensitivity, as follows: R_s and R_{ns} containing all sensitive and non-sensitive tuples, respectively. 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, as we mentioned earlier, we emphasize to use nondeterministic encryption of R_s for the purpose of simplicity.

2. *The untrusted public cloud*: that stores the databases, executes queries, and provides answers.

Query execution. In the model, the DB owner wishes to search an attribute value (or a keyword), say $Q = w$, on R_s and R_{ns}. Assume that the value w exist in R_s and R_{ns}. The user creates two lists, say L_s and L_{ns}, of values, L_s for search on R_s and L_{ns} for search on R_{ns}, and sends them to the cloud. The list L_s contains encrypted representation of some values including w, and the list L_{ns} contains some values including w in the clear-text.

After that, the sensitive and non-sensitive tuples containing all the required attribute values are fetched using an existing (secure) technique on the relation R_s and a direct SQL query execution on the relation R_{ns}, respectively. Here, we are not dealing with how the values of the list L_s and L_{ns} are converted into a secure search and a SQL queries, respectively. The cloud sends output tuples to the DB owner that filters extra tuples. Notations used in this thesis are given in Table 3.1.

Notations	Meaning
$\|S\|$	Number of sensitive data values
$\|NS\|$	Number of non-sensitive data values
R_s	Sensitive parts of a relation R
R_{ns}	Non-sensitive parts of a relation R
B_s	The number of sensitive buckets
B_{si}	i^{th} sensitive buckets
$\|B_s\| = y$	Sensitive values in a sensitive bucket
B_{ns}	The number of non-sensitive buckets
B_{nsi}	i^{th} sensitive buckets
$\|B_{ns}\| = x$	Non-sensitive values in a non-sensitive bucket
$Q(w)(S, NS)$	A query Q for a value w searching on S and NS
t_w	Tuples having a value w

Table 3.1: Notations used in the thesis.

3.1 Adverserial Model

In our setting, recall that sensitive data is encrypted while non-sensitive data resides in the clear-text. We assume an honest-but-curious adversary, which is considered in the standard setting for security in the public cloud [64, 65] that is *not trustworthy or secure.*

An adversarial public cloud, thus, correctly computes assigned tasks; however, it may exploit side knowledge (*e.g.*, query execution, background knowledge, output size) to gain as much information as possible about sensitive data. The adversary cannot launch any attack against the DB owner.[1] Furthermore, an adversary can eavesdrop on the communication channels between the cloud and the DB owner, and that may help in gaining knowledge about sensitive data, the query, or the results. The adversary has full access to the following information:

1. The full information of the non-sensitive data. For example, for the given Employee relation in Example 1, an adversary knows the complete *Employee3* relation.

2. *Auxiliary* information of the sensitive data. Here, the auxiliary information may contain the structure of the relation, number of tuples in the relation, and that some departments are sensitive by finding that all the departments are not in the non-sensitive database.

   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 *Employee1* and *Employee2* relations; refer to Example 1. However, the adversary is not aware of the following information that how many people work in a specified sensitive department, is a specified person working only in a sensitive department, only in a non-sensitive department, or both, or what is the SSN of an employee.

3. Adversarial view. As shown in Example 1. Recall that the adversarial view contains the values in L_s and L_{ns}, and returned tuples satisfying L_s and L_{ns}.

4. Some frequent query values.

[1]We do not consider cyber-attacks that can exfiltrate data from the DB owner directly since defending against generic cyber-attacks are beyond the scope of this thesis.

3.2 Security Definition and Correctness

In the above-mentioned adversarial setting, and inputs and outputs of the algorithm, we consider the sensitive data as an oracle. Here, the goal of the adversary is to find as much as possible sensitive information (using the adversarial view or background knowledge), and our goal is to prevent the information leakage through non-sensitive data. To define a notion of *perfect data security*, we first define two terms *associated values and tuples* and *related sensitive tuples*, as follows:

Notations for the definitions. Consider two attribute values v and $v\prime$ in the domain of an attribute A_j of the relation R. Let $E(v)$ and $E(v')$ be encrypted representations of v and v', respectively, in the relation R_s. Let $E(t_v)$ and $E(t_{v'})$ be encrypted tuples containing the value v and v', respectively, in the relation R_s. Later, we will indicate the value v by a notation s_i that denotes that s_i is the i^{th} sensitive value in the domain of the attribute A_i in the relation R_s. Let ns_i be a non-sensitive value at i^{th} position in the domain of the attribute A_i of the relation R_{ns}.

Associated values and associated tuples. We say the value $E(v)$ is *associated* (denoted by $\overset{\text{a}}{=}$) with a non-sensitive value, say ns_i, of the relation R_{ns} in the attribute A_i, if the clear-text representation of the value $E(v)$ is identical to ns_i. In Example 1, the value `E102` is an associated value.

We say a sensitive tuple $(E(t_v))$ is *associated* with a non-sensitive tuple, if the two tuples share a common value in the attribute A_i. In Example 1, the sensitive tuple t_2 is *associated* with the non-sensitive tuple t_4, because these two tuples have an identical value of the `EId` attribute.

Related sensitive values. We say the value $E(v)$ is *related* (denoted by $\overset{\text{r}}{=}$) with the valve $E(v')$ in the attribute A_i, if there exist any relationship between the clear-text representation of the encrypted values $E(v)$ and $E(v')$, *i.e.*, $v < v'$, $v > v'$, $v = v'$, or $v \neq v'$. In Example 1, the values of the `Salary` attribute has a relationship with each other.

We say a sensitive tuple, $E(t_v)$, is *related* with the sensitive tuple $E(t_{v'})$, if the adversary can find a relation between the two tuples based on the encrypted representation of the attribute A_i. In Example 1, if we encrypt `Salary` attribute of the *Employee* relation using order-preserving encryption, then the adversary can learn an order of the values (however, such a situation can be eliminated using nondeterministic encryption).

Definition: Perfect Data Security. *Let X be auxiliary information about the sensitive data, S be the sensitive data, NS be the non-sensitive data, s_i is the i^{th} sensitive value, and ns_i is the i^{th} non-sensitive value. The perfect data security states that the adversary having the X finds the following two conditions after executing a set of queries, Q, on S and NS at a specified attribute, A_i of a relation, denoted as $Q(S, NS)[A_i]$:*

(01) $Pr[s_i \overset{\text{a}}{=} ns_j | X] = Pr[s_i \overset{\text{a}}{=} ns_j | X, Q(S, NS)[A_i]]$,

where $i \in 1, 2 \ldots, |S|$ and $j \in 1, 2, \ldots, |NS|$

(02) $Pr[s_i \overset{\text{r}}{=} s_j | X, Q(S)] = Pr[s_i \overset{\text{r}}{=} s_j | X, Q(S, NS)[A_i]]$,

where $i \neq j$ and $i, j \in 1, 2, \ldots |S|$

The first equation (01) captures the fact that an initial probability of *associating* a sensitive value with a non-sensitive value will be identical after executing several queries on the database. The second equation (02) captures the fact that all the (encrypted) sensitive values preserve an identical relation, which is leaked by an underlying encrypted technique, after executing a query on the sensitive and the non-sensitive data, to the relation after executing a query only on the sensitive data. In addition to the data security, we define algorithm correctness next.

Correctness. We are focusing on the search query for an attribute value. Let R be a relation that is partitioned into a sensitive relation R_s and a non-sensitive relation R_{ns}. In our context, a query on the R_s and R_{ns} relations for an attribute value 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 R be a relation containing sensitive and non-sensitive data. Let S and NS be the sensitive and non-sensitive datasets of R, respectively. Let $q(w)(S, NS)[A_i]$ be a query, q, for an attribute value w in the attribute A_i of S and NS datasets. Let Result*[2] *be a function for computing the final output. Then, an algorithm is correct if it returns tuples containing the value w in the attribute A_i of S and NS datasets, where the output tuples are identical to answer to the query execution only on the relation R:*

$$q(w)(R)[A_i] \equiv Res[q(w)(S, NS)[A_i]]$$

[2]In our context, the function *Result* will decrypt sensitive data and merge sensitive-non-sensitive data for providing the final outputs.

Chapter 4

Query Bucketization

In this section, we introduce the query bucketization technique (which is executed at the DB owner side) to ensure the perfect data security while processing search queries over non-sensitive data in the clear-text. We develop our strategy initially under the assumption that queries are only on a single attribute A_i and will generalize the approach to search over multiple attributes. The bucketization approach takes as input (*i*) the set of data values (of A_i) that are sensitive along with their count, and (*ii*) the set of data values (of A_j) that are non-sensitive, along with their count. The bucketization returns a partition of attribute values that form the query buckets for both the sensitive as well as for the non-sensitive parts of the query. We begin by developing the approach for the case when a sensitive tuple is *associated* with at most one non-sensitive tuple. We then develop a simple extension to deal with a situations where the number of non-sensitive (or sensitive) values are close to a square number. Finally, we will provide a general strategy to create buckets when a sensitive tuple is *associated* with several non-sensitive tuples.

Informally, the technique distributes attribute values in a rectangle, where rows are sensitive buckets, and columns are non-sensitive buckets. For example, suppose there are 16 values, which we 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 grid, as follows:

	B_{ns1}	B_{ns2}	B_{ns3}	B_{ns4}
B_{s1}	11	2	5	14
B_{s2}	10	3	8	7
B_{s3}	0	15	6	4
B_{s4}	13	1	12	9

In this example, we have four sensitive buckets B_{s1} {11,2,5,14}, B_{s2} {10,3,8,7}, B_{s3} {0,15,6,4}, B_{s4} {13,1,12,9} and four non-sensitive buckets B_{ns1} {11,10,0,13}, B_{ns2} {2,3,15,1}, B_{ns3} {5,8,6,12}, B_{ns4} {14,7,4,9}. When a query arrives for a value, say 1, the DB owner searches for the tuples containing either of 2,3,15,1 (viz. B_{ns2}) on the non-sensitive data and queries for tuples containing values in B_{s4} (viz., 13,1,12,9) over the sensitive data using the cryptographic mechanism integrated into the approach. Note that such a search is over the encrypted representation and that the DB owner does not expose the elements in the sensitive bucket in the clear-text form to the adversary. As we will show, in the query bucketization, while the adversary learns that the user's query corresponds to one of the four values in B_{ns2}, since query values in B_{s1} are encrypted, it does not learn any sensitive value or *association* between sensitive and non-sensitive data.

4.1 The Base Case

The query bucketization technique consists of two steps. First, query buckets 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 bucketization.

Here, we explain the technique for the base case, where a sensitive tuple, say t_s, is associated with at most a single non-sensitive tuple t_{ns} and vice versa (*i.e.*, $\overset{\text{a}}{=}$ 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.

The scenario depicted in Example 1 satisfied the base case. The *EId* attribute values corresponding to sensitive tuples include $\langle$`E101`, `E102`$\rangle$ and that from non-sensitive relation $\langle$`E103`, `E102`$\rangle$ for which $\overset{\text{a}}{=}$ is 1:1. We will further assume that the number of sensitive values $|S|$ is less than or equal to the number of non-sensitive values $|NS|$ (recall that $|NS|$ and $|S|$ denote the number of non-sensitive and sensitive values, respectively). We discuss the query bucketization solution under the above assumption, but relax the assumption subsequently. Before we describe the bucketization approach, 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 very 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: Bucket creation. The query bucketization technique finds two approximately square factors of $|NS|$, say x and y, where $x \geq y$. We create $B_s = x$ sensitive buckets, where each sensitive bucket contains at most y values. Thus, we are assuming $|S| \geq x$. We further create $B_{ns} = \lceil |NS|/x \rceil$ non-sensitive buckets, where each non-sensitive bucket contains at most $|B_{ns}| = x$ values. Note that we are assuming that $|S| \leq |NS|$.[1]

Assignment of sensitive values. We numbered the sensitive buckets from 0 to $x - 1$ and the values therein from 0 to $y - 1$. To assign a value to sensitive buckets, we first generate a permutation of the set of sensitive values — that is, values that are present in sensitive tuples. Such a permutation will be kept secret from the adversary by the DB owner.[2] In order to assign sensitive values to sensitive buckets, we take the i^{th} sensitive value and assign it to i *modulo* x sensitive bucket.

Assignment of non-sensitive values. We numbered the non-sensitive buckets from 0 to $\lceil |NS| \rceil / x - 1$ and values therein from 0 to $x - 1$. Take a sensitive bucket, 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 i^{th} non-sensitive bucket. Here, if each value of a sensitive bucket has an *associated* non-sensitive value and $|S| = |NS|$, then we have assigned all the non-sensitive values to their buckets. However, 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 buckets 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 buckets to size x.

Example 2: (Query bucketization example). *We show the bucket creation*

[1]We are assuming that the amount of sensitive data is smaller than the non-sensitive data. However, we can handle the case of $|S| > |NS|$ by applying the same procedure but in the reverse way, *i.e.*, factorizing $|S|$.

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

method for 10 sensitive values and 10 non-sensitive values. We assume that the only five sensitive values say s_1, s_2, s_3, s_5, s_6 *have their* associated *non-sensitive values* $ns_1, ns_2, ns_3, ns_5, ns_6$*, 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.*

We create 2 non-sensitive buckets and 5 sensitive buckets, and divide 10 sensitive values over 5 sensitive buckets, as follows: B_{s0} $\{s_5, s_{10}\}$, B_{s1} $\{s_1, s_6\}$, B_{s2} $\{s_2, s_7\}$, B_{s3} $\{s_3, s_8\}$, B_{s4} $\{s_4, s_9\}$*; see Figure 4.1. Now, we distribute non-sensitive values* associated *with the sensitive values over two non-sensitive buckets, resulting in the bucket* B_{ns0} $\{ns_5, ns_1, ns_2, ns_3\}$ *and* B_{ns1} $\{*, ns_6, *, *, *\}$*, where* $*$ *shows empty positions in the bucket. In sequel, we need to fill the non-sensitive buckets by the remaining 5 non-sensitive values; hence,* ns_{11} *is assigned to the last position of the bucket* B_{ns0}*, and the bucket* B_{ns1} *contains the remaining 4 non-sensitive values such as* $\{ns_{12}, ns_6, ns_{13}, ns_{14}, ns_{15}\}$.

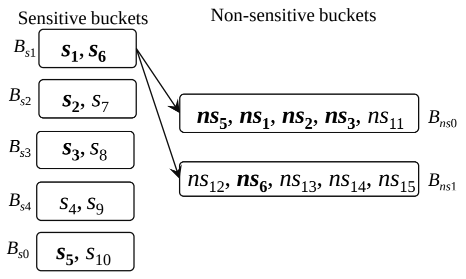


Figure 4.1: The query bucketization technique for 10 sensitive and 10 non-sensitive values.

Aside. Note that we assigned less or equal number of data values in a sensitive bucket than a non-sensitive bucket.

We could have formed the non-sensitive and sensitive buckets in such a way that the number of elements in sensitive buckets is higher than the non-sensitive buckets. We chose sensitive buckets to be smaller since (i) the processing time on encrypted data

Algorithm 1: Bucket 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: $B_s$: sensitive buckets; $B_{ns}$: non-sensitive buckets
1 Function $create\_buckets(S, NS)$ begin
2     Permute all sensitive values
3     $x, y \leftarrow approx\_sq\_factors(|NS|)$: $x \geq y$
4     $|B_{ns}| \leftarrow x$, $B_{ns} \leftarrow \lceil |NS|/x \rceil$, $B_s \leftarrow x$, $|B_s| \leftarrow y$
5     for $i \in (1, |S|)$ do $B_s[i$ modulo $x][*] \leftarrow S[i]$;
6     for $(i, j) \in (0, B_s - 1), (0, |B_s| - 1)$ do $B_{ns}[j][i] \leftarrow allocateNS(B_s[i][j])$ ;
7     for $i \in (0, B_{ns} - 1)$ do $B_{ns}[i][*] \leftarrow$ fill the bucket if empty with the size
        limit to $x$ ;
8     return $B_s$ and $B_{ns}$
9 Function $allocateNS(B_s[i][j])$ begin
      find a non-sensitive value *associated* with the $j^{th}$ sensitive value of the $i^{th}$
        sensitive bucket
```

is higher than the clear-text data processing; hence, by searching and retrieving less sensitive tuples, we decrease the encrypted data processing time; and (ii) it may be the case that the size of each sensitive tuple is significantly large; hence, retrieving fewer sensitive tuples is always beneficial. For example, in the case of hospital database that may hold cancer cells images as sensitive data while the non-sensitive holds information about fever and cold without any image.

Step 2: Bucket retrieval – answering queries. The bucket retrieval algorithm first checks the existence of a query value in sensitive buckets and/or non-sensitive buckets. If the value exists in a sensitive bucket and a non-sensitive bucket, the DB owner retrieves the two buckets. 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 buckets.

There may be three other cases, as follows: (i) some sensitive values of a bucket are not *associated* with any non-sensitive value, (ii) a sensitive bucket does no holds any value that is *associated* with any non-sensitive value, and (iii) many non-sensitive buckets containing no value that is *associated* with any sensitive value. In all the three cases, if the DB owner retrieves only bucket of one side containing the value, then it will lead to information leakage, like Example 1. In order to prevent information leakage, Algorithm 2 follows two rules given below :

$$\text{(R1) } q(w) \wedge w \in S, w = B_{si}[j] \mapsto B_{si} \wedge B_{nsj}$$

$$\text{(R2) } q(w) \wedge w \in NS, w = B_{nsi}[j] \mapsto B_{nsi} \wedge B_{sj}$$

The notations $q(w)$ denotes a query, q, for an attribute value w, and $\mapsto$ denotes what does the DB owner receive as an answer to the query. By Line 2 of Algorithm 2, the DB owner knows that the value w is either sensitive or non-sensitive.

The Rule (R1) says that the value w is a sensitive value that is at the j^{th} position of an i^{th} sensitive bucket, and the DB owner fetches the i^{th} sensitive and the j^{th} non-sensitive buckets (see Line 3 of Algorithm 2).

The Rule (R2) says that the value w is a non-sensitive value that is at the j^{th} position of an i^{th} non-sensitive bucket, and the DB owner fetches the j^{th} non-sensitive and the j^{th} sensitive buckets (see Line 6 of Algorithm 2).

Example 2: (Query bucketization example: continued.) *Now, we show how to retrieve tuples. If a query is for a sensitive value, say s_2, refer to Figure 4.1, then the DB owner fetches two buckets B_{s2} and B_{ns1}. If a query is for a non-sensitive value, say ns_{14}, then the DB owner fetches two buckets B_{ns1} and B_{s3}. Thus, it is impossible for the adversary to find that which is an exact query value from the non-sensitive bucket and which is the sensitive value* associated *with one of the non-sensitive values. This fact is also clear from Table 4.1, which is showing that the adversarial view is not enough to deduce any information about sensitive data, unlike Example 1. In Table 4.1, $E(s_i)$ shows the encrypted value of s_i, and for simplicity, we are not depicting the adversarial view for each value.*

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

Algorithm 2: Bucket retrieval algorithm.

```
  Inputs: $q(w)$: a query $q$ for an attribute value $w$.
  Outputs: $B_{sa}$ and $B_{nsb}$: one sensitive bucket and one non-sensitive bucket to
    be retrieved for answering $q(w)$.
  Variables: $found \leftarrow$ false
1 Function retrieve_buckets($q(w)$) begin
2     for $(i,j) \in (0, B_s - 1), (0, |B_s| - 1)$ do
          if $w = B_{si}[j]$ then
3             return $B_{si}$ and $B_{nsj}$; $found \leftarrow$ true; break
4     if $found \neq$ true then
5         for $(i,j) \in (0, B_{ns} - 1), (0, |B_{ns}| - 1)$ do
6             if $w = B_{nsi}[j]$ then
                  return $B_{nsi}$ and $B_{sj}$; break
```

4.2 A Simple Extension of the Base Case

Algorithm 1 creates buckets when the number of non-sensitive data values are not a prime number, by finding the two approximately square factors. However, Algorithm 1 may result in the higher *cost* (*i.e.*, the number of retrieve 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 we may have 2 non-sensitive buckets having 41 values in each and 41 sensitive buckets having exactly one value in each. In this case, answering a query results in retrieval of 42 tuples. We may also create two sensitive buckets and 41 non-sensitive buckets containing exactly two non-sensitive values in each. These query buckets will require us to fetch 23 tuples. However, we can reduce the cost significantly.

An extension to the query bucketization technique is provided in Algorithm 3 that handles the case when the number of NS values is close to a square number. For simplicity, in this extension, we considered the base case scenario for defining the number of tuples with each value. We first find two approximately square factors of non-sensitive values and the cost; lines 2 and 3. We also find a square number closest to the non-sensitive values and find the cost; line 4. Now, we create buckets using the method that results in fewer numbers of retrieved tuples (line 5). When we create buckets using the square number to the non-sensitive values (line 6), the *remaining* non-sensitive values (greater than the square number) that do not have any associative sensitive value can be handled by putting an equal number of the remaining values in the buckets. Note that the assignment of sensitive values and *associated* non-sensitive values is carried out as we did in Algorithm 1.

Example 4: (An example for the query bucketization extension). *Consider 41 sensitive data values and 82 non-sensitive data values, having one tuple each. In this case, 81 is the closest square number to 82. Here, we can create 9 non-sensitive buckets and 9 sensitive buckets. We follow Algorithm 1 for allocating sensitive values and* associated *non-sensitive values, resulting in that a sensitive bucket holds at most 5 values and a non-sensitive bucket holds at most 10 values. Here, we need to retrieve at most 15 tuples to answer a query.*

Algorithm 3: An extension to the query bucketization, the base case.

```
   Inputs: $|NS|$, $|S|$.
   Outputs: $B_s$, $B_{ns}$
1  Function $bucket\_extension(S, NS)$ begin
2      $x, y \leftarrow approx\_sq\_factors(|NS|)$: $x \geq y$
3      $cost_d \leftarrow x + y$
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 buckets
7      else Execute Algorithm $1(S, NS)$
```

4.3 General Case: Multiple Values with Multiple Tuples

In this section, we will consider a case when data values have different number of tuples. We will see that sensitive values with different number of tuples provide enough information to the adversary to disclose the sensitive data; hence, provide a way to overcome such a situation. We mentioned that SGX-bsed solutions are not enough secure for handing joint processing of skewed or non-skewed sensitive and non-sensitive data. This fact will also clear from the following *size attack scenario.*

Size attack scenario under SGX-based solutions. SGX-based solutions are capable to process sensitive data in the enclave. However, the joint processing of the sensitive and non-sensitive data may lead to the information leakage through non-sensitive data, as shown in Example 1. Moreover, these solutions are not secure when sensitive values have skewed tuples. Consider a query, say q_1, `find all employees earing $10K`, and assume that most of the employees earn \$10K. The joint processing of the query q_1 on sensitive and non-sensitive data in the current SGX-based solutions [18, 52, 68] will reveal the number of sensitive employees earning \$10K as well as the query. The current SGX-based solutions [18, 52, 68] provide padding to hide an exact number of resultant tuples, however, the padding is not enough to maintain security. For example, consider a query `find all employees earing $50K`, and assume that fewer employees earn \$50K, then addition of fake tuples will distinguish that most of the employees in sensitive data earn \$10K and fewer employees earn \$50K. In short, all SGX-based solutions lag behind in providing a quantitative amount of fake tuples to be added for hiding output sensitive data size.

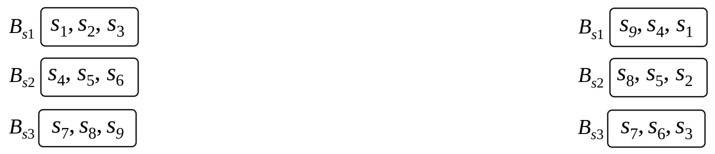
Size attack scenario in the naive query bucketization. The query bucketization technique can hide the exact query value unlike the previous example. However, in the case of different number of tuples with sensitive values, the size of sensitive buckets is non-identical, and different-sized sensitive buckets may leak sensitive data when accessing non-sensitive data.

For example in the case of 10 sensitive and 10 non-sensitive values, refer to Figure 4.1, consider a sensitive value, say s_1, has 1000 sensitive tuples and an *associated* non-sensitive value, say ns_1, has 2000 has tuples, while all the other values have only 100 sensitive tuples and the *associated* 200 non-sensitive tuples.

In this scenario, consider the query execution for a value, say ns_1. The DB owner retrieves tuples according to two buckets such as B_{s1} (containing encrypted tuples having s_1 and s_6) and B_{ns0} (containing tuples of $ns_1, ns_2, \dots, ns_5$). Obviously, the number of retrieved tuples satisfying the values of the buckets B_{s1} and B_{ns0} will be highest (*i.e.*, 3900) as compared to the number of tuples retrieved according to any two buckets. Thus, the retrieval of the two buckets B_{s1} and B_{ns0} provides enough information to the adversary that which is the sensitive bucket *associated* with the bucket holding the attribute value ns_1. Note that this type of size attack cannot be prevented by any SGX-based solutions [18, 52, 68] too, while we use these solution as a blackbox for sensitive data processing.

In order to hold the second equation of the perfect data security, which is not maintained in the above example, to be true, we need to make identical-sized sensitive buckets. A simple way of doing this is to outsource some encrypted fake tuples in a way that the size of each sensitive bucket will be identical. However, we need to be careful; otherwise, adding fake tuples in each sensitive bucket may increase the *cost* (*i.e.*, the number of retrieved tuples), if all the heavy-hitter sensitive values are allocated to a single bucket. This fact will be clear by the following example.

Example 5: (Illustrating ways to assign sensitive values to buckets). *We consider 9 sensitive values, say $s_1, s_2, \dots s_9$, having 10, 20, 30, 40, 50, 60, 70, 80, and 90 tuples, respectively. There are multiple ways for allocating these values to three buckets so that we need to add a minimum number of fake tuples to each bucket. In Figure 4.2, we show two different ways to assign these values to buckets. The best we can do to minimize the addition of fake encrypted tuples and hence minimizing the cost is to use three buckets as given in Figure 4.2b. Here, we need to add only 20 and 10 fake encrypted tuples to the bucket B_{s1} and B_{s2}, respectively. However, the first way, see Figure 4.2a, requires us to add 180 and 90 fake encrypted tuples to the buckets B_{s1} and B_{s2}, respectively for making identical-sized buckets.*



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

Figure 4.2: An example of assigning 9 sensitive values to 3 buckets.

It is important to mention that there is no need to add any fake tuple if the all the non-sensitive values have an identical number of tuples. In this case, the adversary cannot deduce which sensitive bucket contains sensitive parts corresponding to a non-sensitive value. However, it is obvious that we cannot add any fake non-sensitive tuple in the clear-text.

Before describing how to add fake encrypted tuples to buckets, we show that the partitioning of values to B_s buckets leading to identical-sized buckets, where a bucket is not adhered to hold y values, is not a communication efficient solution. For example, consider 9 sensitive values, where the first value has 100 tuples and all the other values have 25 tuples each. In this case, we will get buckets as shown in Figure 4.3. Note that the buckets B_{s2} and B_{s_3} are *associated* with all the three non-sensitive buckets while the bucket B_{s1} is *associated* with only B_{ns1}. Thus, the given bucketization does not prevent the survival matching edges. In order to prevent all the survival matching edges, we need to ask fake queries for buckets $\langle B_{s1}, B_{ns2}\rangle$ and $\langle B_{s1}, B_{ns3}\rangle$.

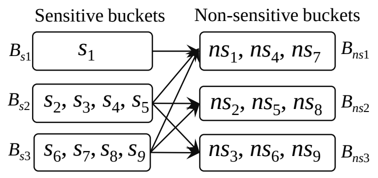


Figure 4.3: An assignment of a heavy hitter value but losing survival matching edges.

Adding fake encrypted tuples. As an assumption, we know the number of sensitive buckets and use the following strategy for allocating sensitive values to the buckets:

1. Sort all the values in a decreasing order of the number of tuples.
2. Select B_s largest values and allocate one in each bucket.
3. Select the next value and find a bucket that is containing the fewest number of tuples. If the bucket is holding y values, then add the value to the bucket; otherwise, select other buckets with the fewest number of tuples. Repeat this step, until all the values are not allocated to the bucket.

4. Add fake tuples to the buckets so that each bucket contains identical number of tuples

5. Allocate non-sensitive values according to Algorithm 1, lines 6 and 7.

Chapter 5

Effective Cost Model

5.1 Motivation

The previous chapter dealt with the implementation of bucketization technique and its security details. In this chapter we will develop an empirical cost model to estimate the effectiveness of this scheme. Every secured database has some overhead which it has to pay for security when compared to database storing data in plain text. We capture this overhead by a term β. Imagine a database D with d distinct domain values $V = v_1, \ldots, v_d$ and r copies per v_i on average, Thus $D = d * r$. Consider retrieving a set of records for a given domain value v_i in a given encrypted strategy(E) and a cost of retrieving it in plain text. We define β as below.

$$\beta(D, E) = \frac{\sum_{v_i \in V} retrieve(v_i, D, E)}{\sum_{v_i \in V} retrieve(v_i, D, plaintxt)} \tag{5.1}$$

Depending on the type of encryption supported by the database, we will have different values of β.

To compare the savings that result due to using bucketization, we have to introduce another parameter η. Let the domain value v_i be associated with bucket B_m, having v_s domain values, on the sensitive side and B_n, having v_{ns} domain values, for the non sensitive side. Time taken for retrieving each individual bucket is T_S (from sensitive side) and T_{NS} (from non sensitive side). T_{FE} is time taken to retrieve all records for v_i, when the entire data set is encrypted and we do not use any bucketization technique.

$$T_S = \sum_{i=1}^{v_s} \sum_{v_i \in V} retrieve(v_i, D, E) \tag{5.2}$$

$$T_{NS} = \sum_{i=1}^{v_{ns}} \sum_{v_i \in V} retrieve(v_i, D, plaintext) \tag{5.3}$$

$$T_{FE} = \sum_{v_i \in V} retrieve(v_i, D, E) \tag{5.4}$$

We now define η as follows:

$$\eta = \frac{T_S + T_{NS}}{T_{FE}} \tag{5.5}$$

Note that both the definition of β and η depend upon not just the underlying cryptographic technique, but in general, will depend upon the database being stored and the distribution of values we will see. The main objective of the section is to gain insight about conditions when bucketization is beneficial and not to exactly characterize cost model for different techniques. As a result, we will make simplifying assumptions about the data and value distributions to make analysis possible.In particular, we will assume uniform distribution of the values, that is, r_i(defined in table 5.1) for each value v_i is identical and secondly our entire data is in memory.

If $\eta < 1$, then our bucketization technique is effective.

To construct the model, we would need various notations which are introduced below

Notations	Meaning
B_S	Size of sensitive bucket
B_{NS}	Size of non-sensitive bucket
α	Sensitivity factor
D	Total number of keys
d	Total number of unique keys
r	Replication Factor($D = r * d$)
$E(D)$	Cost of retrieving a single key over encrypted data of size D
$NE(D)$	Cost of retrieving a single key over non-encrypted data of size D
$\alpha_{(r,\alpha)}$	Replication reduction factor when sensitivity $= \alpha$
t_n^r	Time to fetch 1 record over network
t_d^r	Time to decrypt 1 record
t_n^f	Time to filter 1 record
H_X	Height of B^+ Tree when it contains X elements
t_i^e	Time to process each node of an encrypted B^+ Tree

Table 5.1: Notations for cost model.

Lemma 1 *Let D be a database and let I be a B^+ index over attribute A_i. Let us assume that height of tree is H_i. Now consider the database contains only $\alpha * D$ number of tuples. The B^+ index is still on the same attribute A_i. The height of the B^+ tree does not change if $\alpha > \frac{1}{F}$, where F is the fan-out of the B^+ tree.*

Proof sketch. Consider two separate B^+ tree indices B_1 and B_2 for D_1 and D_2 elements respectively such that $D_1 > D_2$. Let $D_2 = \alpha * D_1$ and hence $\alpha < 1$. The height of a B^+ tree can be approximated to be $\lceil \log_F D \rceil$ where D is the number of elements in the B^+ tree. The height of the B^+ tree will change if $D1 > F * D2$, if the base of the *log* function is F; or conversely the height will not change if $D1 < F * D2$.

$$\begin{aligned}
& D_1 < F * D_2 \\
\Rightarrow \quad & \frac{D_1}{D_2} < F \\
\Rightarrow \quad & \frac{D_1}{\alpha * D_1} < F \\
\Rightarrow \quad & \frac{1}{\alpha} < F \\
\Rightarrow \quad & \alpha > \frac{1}{F} \qquad (5.6)
\end{aligned}$$

Typically the fan-out of a B^+ tree in a database is in the order of hundreds. Considering a minimalistic value of $F = 100$, we would get a lower bound of α to be 0.01.On the other end of the spectrum if $\alpha \to 1$, it is trivial to show that the value of D remains almost same and thus the height as well.

5.2 Non-Indexable Technique

This is the most trivial solution, where the database does not allow search over encrypted data in sub-linear time. We cannot use a database which uses deterministic encryption, as this leaks information like correlation.So we use non-deterministic encryption. The most trivial solution is to download the entire database and filter out the rows which are of interest to us. Later we will discuss more sophisticated techniques. β for this scenario is defined as the overhead of fetching a record in encrypted setting to that of non-encrypted setting. $E(D)$ refers to the cost of retrieving a given domain value from the encrypted database, while $NE(D)$ refers to the cost of fetching the same domain value from a plain text setting. We have to keep in mind that the domain value may be mapped to more that one tuple depending on the value of replication factor.

$$\beta_{NI} = \frac{E(D)}{NE(D)} \tag{5.7}$$

Setup The sensitive data is in non-deterministic fashion while the non-sensitive data is stored in plain text.To study the effectiveness of our bucketization technique, we will use compare with a database where the entire data set is stored using non-deterministic encryption.

Lemma 2 *The time required to find an element in database storing clear text data is approximately independent of the data set size, provided* $\alpha > \frac{1}{F}$.

$$NE(\alpha * D) \approx NE(D) \tag{5.8}$$

Proof sketch

$$\begin{aligned} NE(\alpha * D) &= t_i^p * H_{\alpha * D} + t_a^r * \alpha_{(r,\alpha)} * r \\ &= t_i^p * H_D + t_a^r * \alpha_{(r,\alpha)} * r \quad \text{(By Lemma 1)} \qquad (5.9) \\ NE(D) &= t_i^p * H_D + t_a^r * r \qquad (5.10) \\ NE(D) - NE(\alpha * D) &= t_a^r * r(1 - \alpha_{(r,\alpha)}) \qquad (5.11) \end{aligned}$$

The value of $\alpha_{(r,\alpha)}$ can range between α and 1. For our scheme the value of 1 is the worst setting possible, which means the time to process $NE(\alpha * D)$ is same as $NE(D)$. Any value of less than 1, will lower the value of η as we will pay a lower cost of processing a smaller fraction of non-encrypted data. But we will assume the worst case possible for our technique and try to see if our scheme is still effective. Thus we will consider $NE(\alpha * D) \implies NE(D)$, but never the converse.

Lemma 3 *The time required to find an element in an encrypted database,having no index, of size $\alpha * D$ is α times the processing time required for a data set of size D under the same conditions.*

$$E(\alpha * D) = \alpha * E(D) \tag{5.12}$$

Proof sketch

$$E(D) = t_n^r * D + t_d^r * D + t_f^r * D \tag{5.13}$$

$$\begin{aligned} E(\alpha * D) &= t_n^r * \alpha * D + t_d^r * \alpha * D + t_f^r * \alpha * D \\ &= \alpha * (t_n^r * D + t_d^r * D + t_f^r * D) \\ &= \alpha * E(D) \end{aligned} \tag{5.14}$$

To estimate the value of η, we will fetch the entire data from Sensitive dataset and filter the required values. At the Non-Sensitive side, we can use index to fetch the bucket of required elements. The total time is then compared with the case when the entire dataset was Sensitive.

$$\begin{aligned} \eta &= \frac{E(\alpha * D) + B_{NS} * NE((1-\alpha) * D)}{E(D)} \\ &= \frac{\alpha * E(D)}{E(D)} + \frac{B_{NS} * NE((1-\alpha) * D)}{E(D)} \quad \text{(as per Lemma 3)} \\ &= \alpha + \frac{B_{NS} * NE(D)}{E(D)} \quad \left(\text{if } \alpha < 1 - \frac{1}{F} \text{ and Lemma 2}\right) \\ &= \alpha + B_{NS} * \frac{1}{\frac{E(D)}{NE(D)}} \\ &= \alpha + B_{NS} * \frac{1}{\beta_{NI}} \quad \text{(as per the definition of } \beta \text{ in 5.7)} \end{aligned} \tag{5.15}$$

Analysis B_{NS} is the size of a single bucket for non-sensitive data. The value of β_{NI}, which is the β value considering the database supports only non-indexable search, is very high as there is a huge overhead of downloading the entire database and is in the range of 5000 to 10000. Thus the second term in the above equation is very small and hence the value of η is driven by the sensitivity factor α. We have plotted the graph in figure 5.1 using equation 5.15.

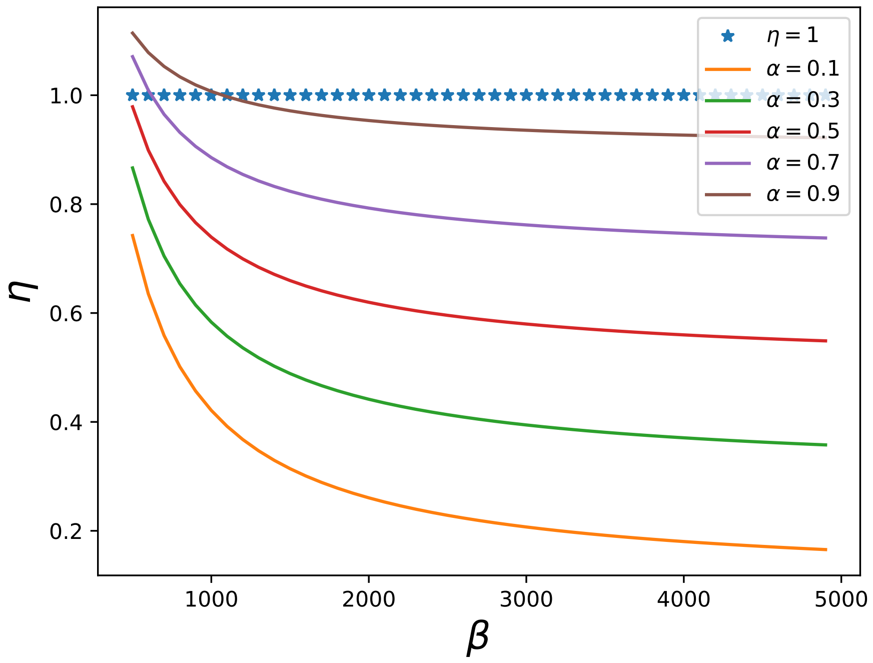


Figure 5.1: Non-Indexable Search.

5.3 Indexable Technique

Searching over encrypted data in logarithmic time is the gold standard of secured databases. Although current research has not yet achieved any effective technique which allows all operations like point query, range queries, sorting over encrypted data without leaking information in logarithmic time, there has been a lot of work which allows a subset of operation to be performed in logarithmic scale. We propose a model that will enable us to evaluate the effective cost when using the bucketization approach on top of a database that supports logarithmic time searchable encryption.As usual we define β_I as the overhead of fetching a record in encrypted setting to that in plain text. Here the E in $E(D)$ refers to a database which support indexable encryption unlike the non-indexable encryption in the earlier section.

$$\beta_I = \frac{E(D)}{NE(D)} \tag{5.16}$$

Setup The Sensitive dataset is stored in the database which supports indexable search, whose β value is β_I. The Non-Sensitive part is stored as plaintext. To calculate the effective cost, we store the entire dataset in another instance of the same database which was used to store the sensitive part.

Lemma 4 *In an indexed encrypted database setting, the processing time of* $(\alpha * D)$ *elements is same as the processing time of* D *elements times* $\alpha_{(r,\alpha)}$.

$$E(\alpha * D) = \alpha_{(r,\alpha)} * E(D) \tag{5.17}$$

Proof sketch

$$\begin{aligned} E(\alpha * D) &= [(t_i^e * H_{\alpha * D}) + t_d^r] * (\alpha_{(r,\alpha)} * r) \\ &= [(t_i^e * H_D) + t_d^r] * (\alpha_{(r,\alpha)} * r) \quad \text{(height invariance by Lemma 1)} \\ &= \{[(t_i^e * H_D) + t_d^r] * r\} * \alpha_{(r,\alpha)} \\ &= E(D) * \alpha_{(r,\alpha)} \end{aligned}$$

Now we can estimate the effective cost η. Recall that η is the ratio of the cost of processing a query in a bucketization setting to that of processing without any bucketization technique.

$$\begin{aligned}
\eta &= \frac{B_S * E(\alpha * D) + B_{NS} * NE((1-\alpha) * D)}{E(D)} \\
&= \frac{B_S * E(\alpha * D)}{E(D)} + \frac{B_{NS} * NE((1-\alpha) * D)}{E(D)} \\
&= \frac{B_S * \alpha_{(r,\alpha)} * E(D)}{E(D)} + \frac{B_{NS} * NE((1-\alpha) * D)}{E(D)} \quad \text{(as per Lemma 4)} \\
&= B_S * \alpha_{(r,\alpha)} + \frac{B_{NS} * NE(D)}{E(D)} \quad (\text{if } \alpha < 1 - \frac{1}{F} \text{ and Lemma 2}) \\
&= B_S * \alpha_{(r,\alpha)} + B_{NS} * \frac{1}{\frac{E(D)}{NE(D)}} \\
&= B_S * \alpha_{(r,\alpha)} + B_{NS} * \frac{1}{\beta_I} \quad (\text{as per the definition of } \beta \text{ in 5.16}) \qquad (5.18)
\end{aligned}$$

In most of our setup the value of $\alpha_{(r,\alpha)}$ is approximately 1.Thus the above equation can be reduced to

$$\eta = B_S + B_{NS} * \frac{1}{\beta_I} \qquad (5.19)$$

If we ignore this assumption, we will get a slightly lower value of η.To make our technique more robust we are approximating to a higher value of η, or in other words we have assuming scenarios where data distribution is antagonistic to our approach.

Analysis The value of η in the above equation is dominated by the first term, B_S, which is the size of a single bucket for sensitive data, whose value is more than 1 in the bucketization technique.Hence the value of $\eta > 1$, which means the bucketization technique is computationally ineffective. On the other hand, this technique is immune to size attack, as the number of elements returned by our technique always returns the same which is not the case otherwise. The graph is drawn in the figure 5.2.

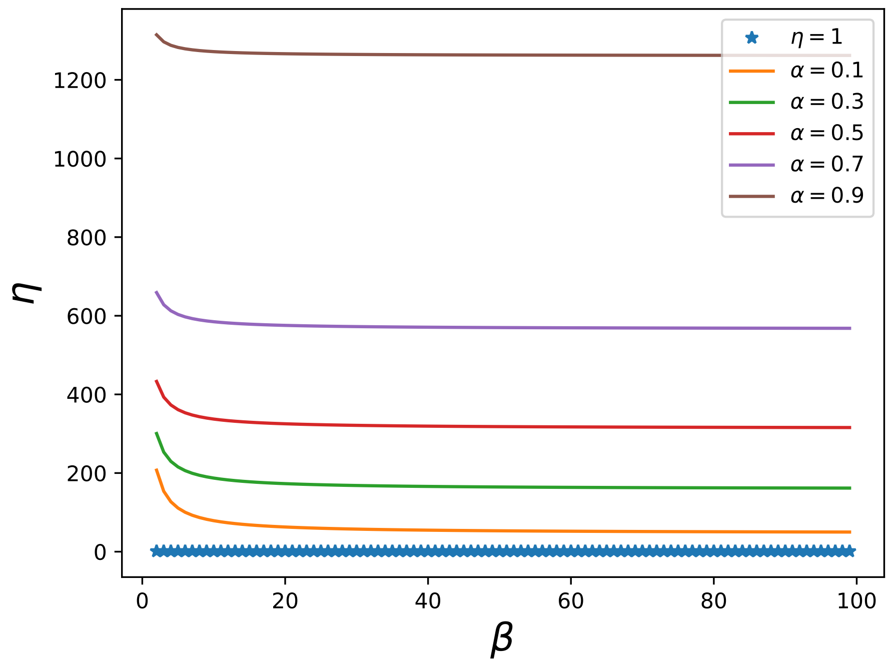


Figure 5.2: Indexable Search.

5.4 Hybrid Technique

In the indexable technique discussed above, we found out that the effective cost(η) was always more than one but we had more security. To get the best of both world, we will use a hybrid approach. The storage and retrieval of the sensitive dataset is as per the technique described in [49]. Let the value of β associated with this technique be β_P and system be designated as $System_P$. Non-Sensitive data is stored in plain-text. To measure the effectiveness of this setup we will use any other system which supports indexable search over encrypted data.Let the value of β associated with this system be β_S and the system be designated as $System_S$. We have already defined the notion of $E(D)$. To distinguish between the two system, we have added a subscript to denote the system for which we are mapping the $E(D)$ to. The two values of β

are defined below.

$$\beta_P = \frac{E_P(D)}{NE(D)} \tag{5.20}$$

$$\beta_S = \frac{E_S(D)}{NE(D)} \tag{5.21}$$

Effective Cost (η) can be estimated as below:

$$\begin{aligned}
\eta &= \frac{B_S * E_P(\alpha * D) + B_{NS} * NE((1-\alpha) * D)}{E_S(D)} \\
&= \frac{B_S * E_P(\alpha * D)}{E_S(D)} + \frac{B_{NS} * NE((1-\alpha) * D)}{E_S(D)} \\
&= \frac{B_S * \alpha_{(r,\alpha)} * E_P(D)}{E_s(D)} + \frac{B_{NS} * NE((1-\alpha) * D)}{E_S(D)} \quad \text{(as per Lemma 4)} \\
&= B_S * \alpha_{(r,\alpha)} * \frac{\frac{E_P(D}{NE(D)})}{\frac{E_S(D)}{NE(D)}} + \frac{B_{NS} * NE(D)}{E_s(D)} \quad (\text{if } \alpha < 1 - \frac{1}{F} \text{ and Lemma 2}) \\
&= B_S * \alpha_{(r,\alpha)} * \frac{\beta_P}{\beta_S} + B_{NS} * \frac{1}{\frac{E_S(D)}{NE(D)}} \quad \text{(as per equation 5.20 and 5.21)} \\
&= B_S * \alpha_{(r,\alpha)} * \frac{\beta_P}{\beta_S} + B_{NS} * \frac{1}{\beta_S} \quad (\text{as per the definition of } \beta \text{ in 5.21}) \qquad (5.22)
\end{aligned}$$

In most of our setup the value of $\alpha_{(r,\alpha)}$ is approximately 1. We have discussed the significance of this assumption in the previous two sections. Thus the above equation can be reduced to

$$\eta = B_S * \frac{\beta_P}{\beta_S} + B_{NS} * \frac{1}{\beta_S} \tag{5.23}$$

Analysis The notation B_{NS} and B_{NS} are the same as used in the previous section. To get an insight into the equation, we will use some of the β values we found. By implementing the strategy given in [49], we found $\beta_P = 1.3647$. β_S was found to be 3290.944423. Due to such a high numeric value of β_S, η was found to be less than one.

It should be taken into consideration that $System_P$ by itself leaks correlation and size information. So, without bucketization technique, $System_P$ cannot be completely secure. On the the other hand $System_S$ has higher notion of security, albeit at a higher value of β_S.By using such a hybrid approach, we can get security as well as acceptable time complexity. The graph is drawn in the figure 5.3.

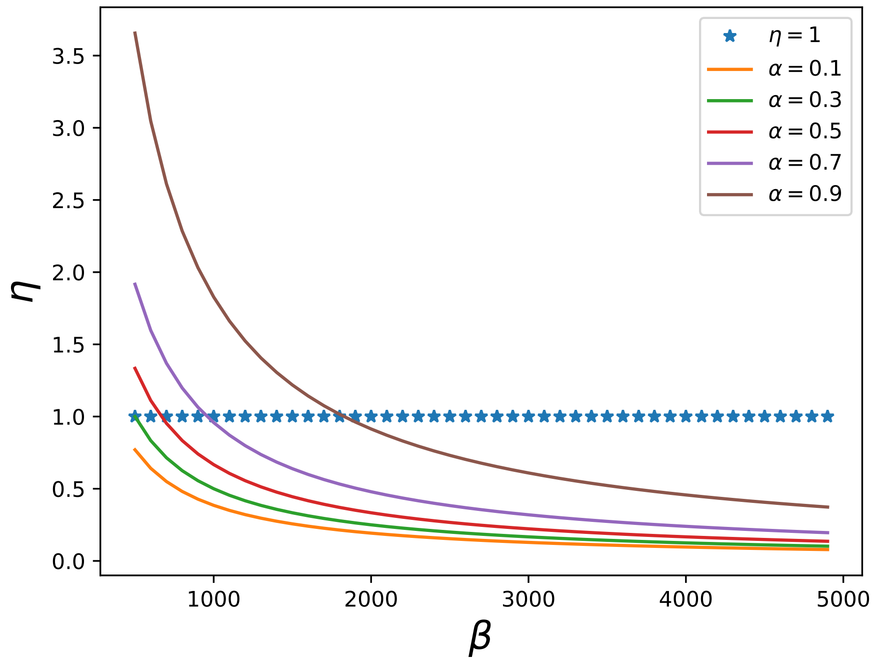


Figure 5.3: Hybrid Model.

Chapter 6

Experiments

6.1 Setup

In this section, we compare the performance of our proposed solution against the standard techniques for retrieving tuples from existing systems. We explicitly do not want to reveal the name of the system due to legal issues, and hence we name these systems as A, B, C and so on. We will give more details about the system, as we introduce them for each case.

We conducted our experiments on a in-home cloud. For setting up the system, we spin up a Virtual Machine(VM) and install the required guest operating system on top of the VM. We have used the recommended operating system for each database. The VM has a 2.6 GHz and 4 core processor. The memory has been configured to be 16 GB, while the physical disk has a capacity of 1 TB.

We used TPC-H benchmark to generate the dataset for our experiments. We randomly mark some tuples as sensitive and the rest as non sensitive. The sensitivity factor(α) is varied from 0.1 to 0.9 for our experiments. Once we have partitioned the data into two sets, we store the sensitive data in the system under consideration, which should support encryption and the non sensitive data in plain text. Please note, that we create non-clustered B^+ tree index on the plain text data. Buckets are created as per the data present in sensitive and non sensitive data. This marks the end of the initialization phase. Based on the keyword to be searched, we select a bucket created in the previous state, one for sensitive and one for sensitive as per the Algorithm 2. We note the time taken to retrieve both the buckets. To compare the effectiveness of our technique, we store the entire dataset as encrypted in the system under consideration and find out the time taken to retrieve records based on the same keyword. We repeat the experiment fifty times, to minimize the effect of outliers.

6.2 Non Indexable Technique

We consider two popular databases, which support non deterministic encryption. Lets us call them System A and B. The Beta(β) values for the systems were found to be 1117.73 and 9500.268 respectively. As both the systems do not support any indexable search on encrypted data and further as the data is encrypted in non deterministic fashion, we have to download the entire data from the encrypted table and filter out the required tuples. While the plain text data can be retrieved using the index created. We compare the time taken to fetch a bucket from the sensitive side and the non sensitive side in figure 6.1 and 6.2 for System A and B respectively for different levels of sensitivity. We denote the time taken as S and NS. As the time taken by non sensitive is orders of magnitude less than the sensitive side,we have used *log* to show the time taken.

We then sum up the time taken to retrieve the two buckets and compare that with

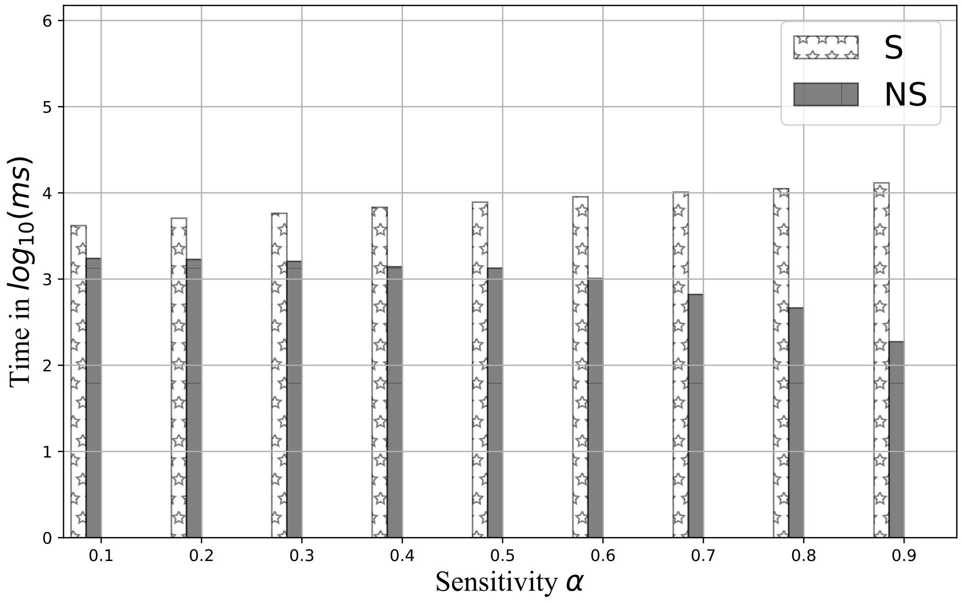


Figure 6.1: Comparison of Sensitive vs Non-Sensitive Time in System A.

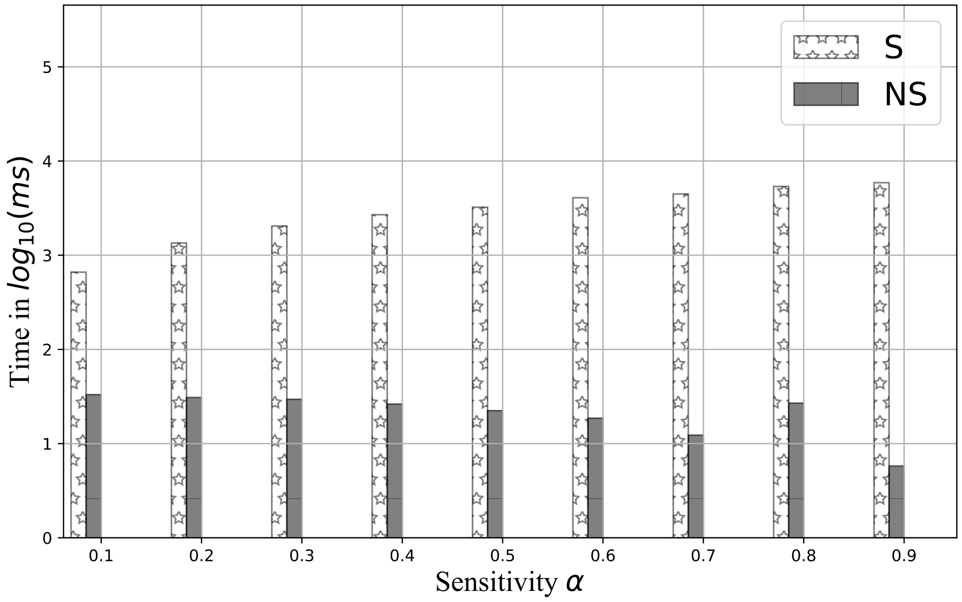


Figure 6.2: Comparison of Sensitive vs Non-Sensitive Time in System B.

the time taken when the entire data was sensitive. This fact is shown in figure 6.3 for System A and in figure 6.4 for System B. As we see that as the level sensitivity increase the time taken to retrieve data from the sensitive side increase. The reason being that we need to download the entire dataset and filter our required tuples which is linear in time to the size of data set. We then calculate effective cost as per the definition given in equation 5.5. The effective cost increase as the sensitivity increases primarily due to the increase in cost of processing a larger encrypted data set. The η values are plotted in fig 6.5 and 6.6 for System A and B respectively, which is below the value of 1. The results show that our technique is efficient till $\alpha = 0.8$ for System A and $\alpha = 0.9$ for System B.

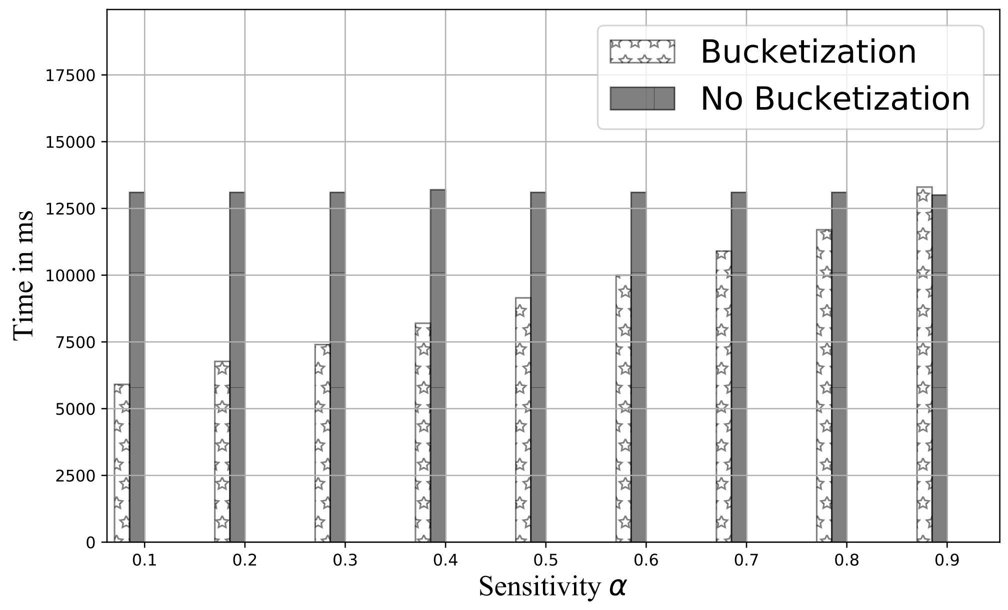


Figure 6.3: Bucketization vs No Bucketization in System A.

6.3 Indexable Technique

To exploit indexing we use the technique given in [49] on System A and System B. As the scheme of processing encrypted data is now modified, we call this system as System C and System D. Further we test another system which supports indexable

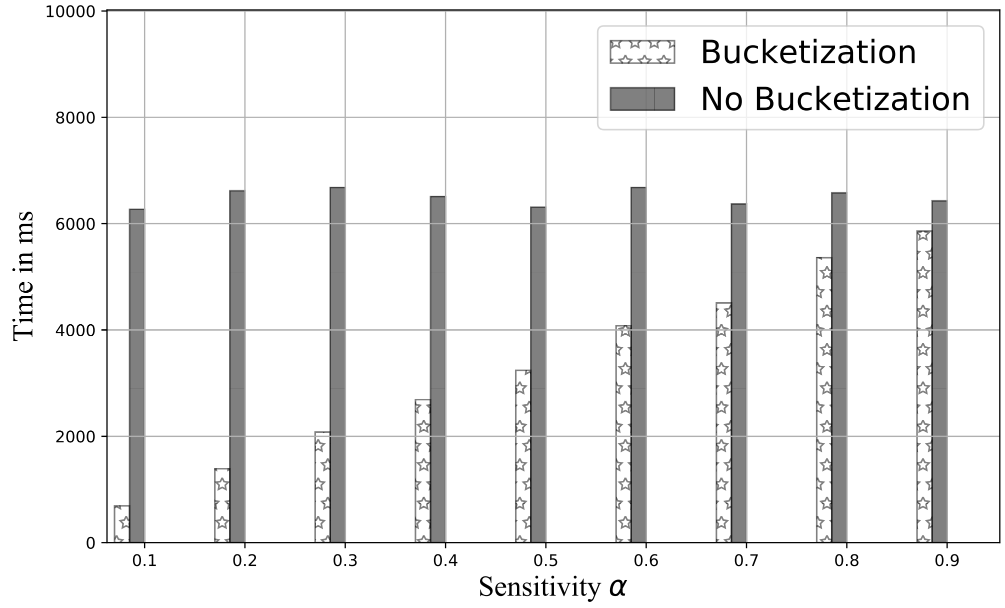


Figure 6.4: Bucketization vs No Bucketization in System B.

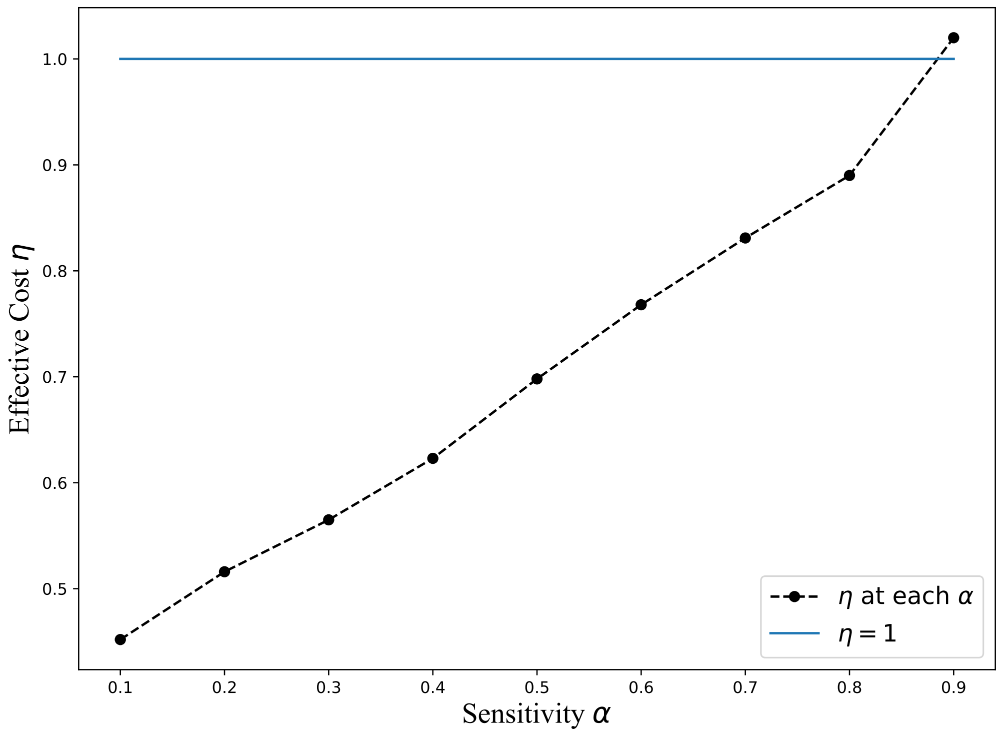


Figure 6.5: Effective Cost in System A.

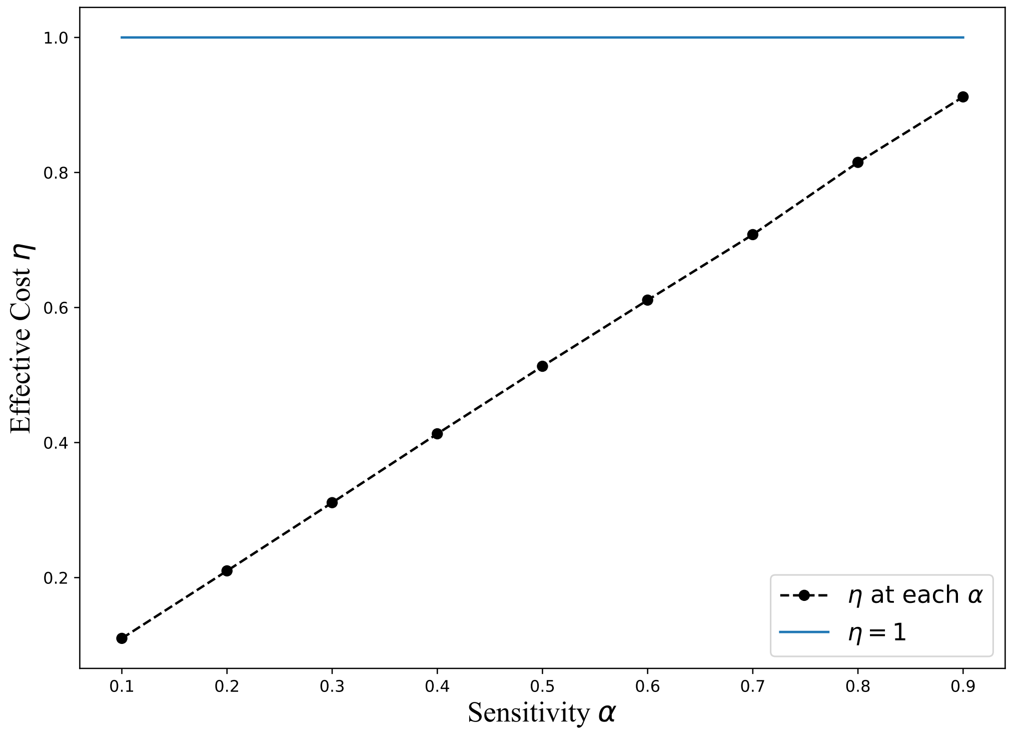


Figure 6.6: Effective Cost in System B.

search over encrypted data. We have named this as System E. The value of $\beta(s)$ for the three system are 12.58, 1.364 and 3290.94 for System C, D, E respectively. We follow the same methodology, as we did for non-indexable search. The difference being that the sensitive data can now be processed in time which is sub-linear. We have compared the time taken to fetch sensitive and non sensitive bucket in figure 6.7 , 6.8 and 6.9 for System C, D and E respectively. Then we sum up the time taken to fetch both the buckets and compare the time taken to fetch the required tuples when the entire data set is encrypted. This fact is represented in figures 6.10, 6.11 and 6.12. We see that the time take by our Bucketization technique is very high as compared to straight forward retrieval. This is also represented by the graphs for η in 6.13, 6.14 and 6.15. The reason being in these system cannot fetch a bucket in which the processing cost is amortized among the individual cost of fetching each tuple. So, we end up paying the security overhead of retrieving each tuple B_S number of times, where B_S is the size of each bucket for sensitive data. As a note, we must mention that, although we lose out in efficiency, we gain additional security as our model is immune to size attack while the base model is not.

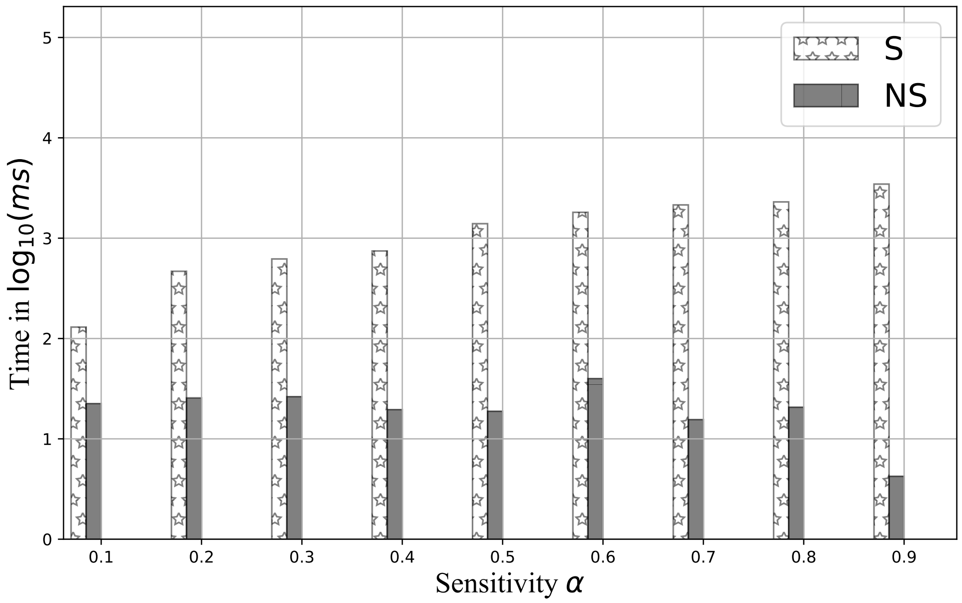


Figure 6.7: Comparison of Sensitive vs Non-Sensitive Time in System C.

6.4 Hybrid Approach

In the hybrid approach, we store and process the sensitive data using the approach given in [49] along with our bucketization scheme, while the non sensitive data is stored in plain text. We consider another highly secure indexable search solution, in which we store the entire data set. Let this overall setup be known as System E. We have a highly efficient setup which can retrieve a bucket very fast due to index and secure as well due to and secured due to bucketization. We follow our normal strategy to compare the system and plot the values in figure 6.16 and 6.17. As we can see that we are efficient by a fair margin, with η remaining way less than 1, even

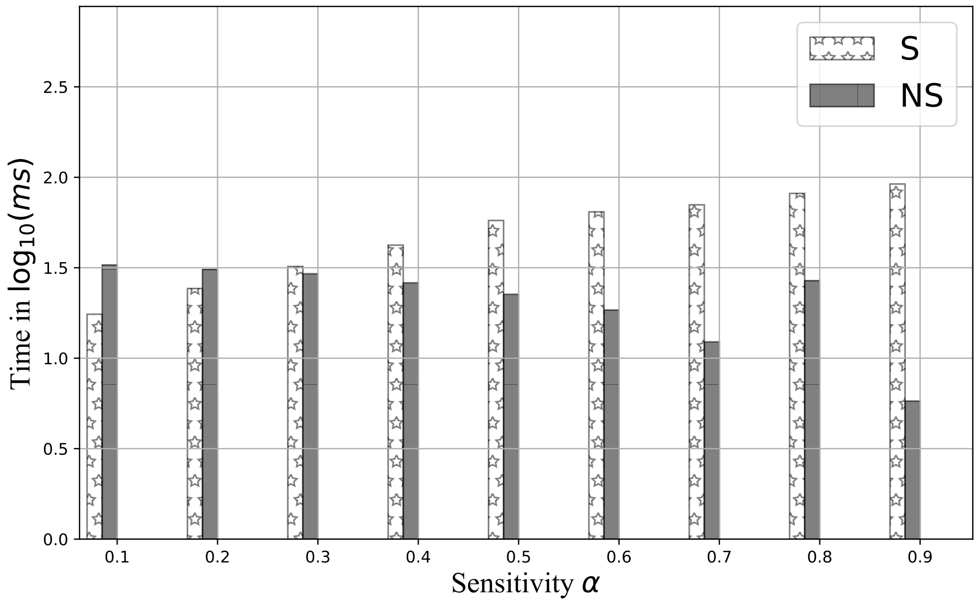


Figure 6.8: Comparison of Sensitive vs Non-Sensitive Time in System D.

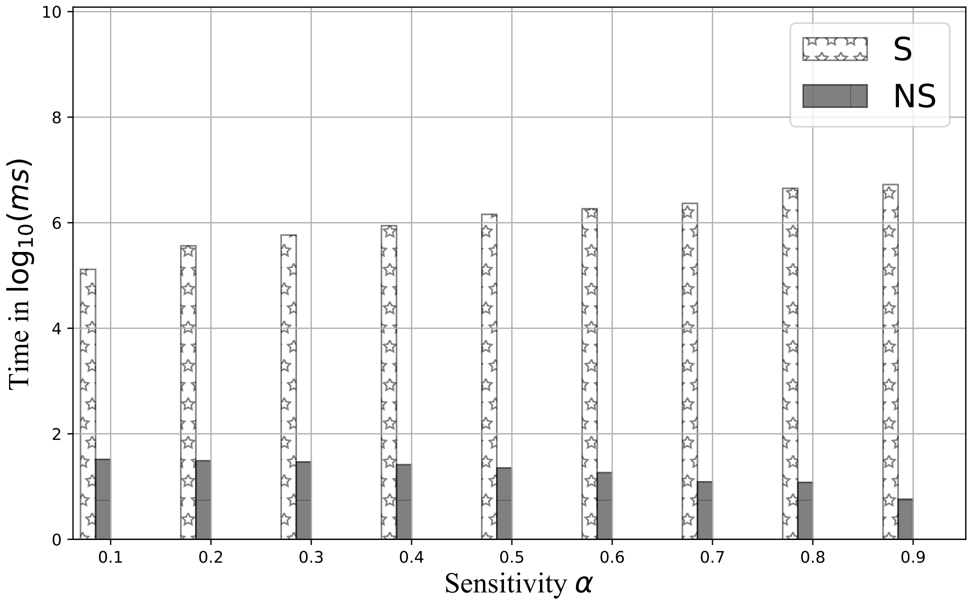


Figure 6.9: Comparison of Sensitive vs Non-Sensitive Time in System E.

if the sensitivity is increased. The reason behind this lies in equation 5.23.

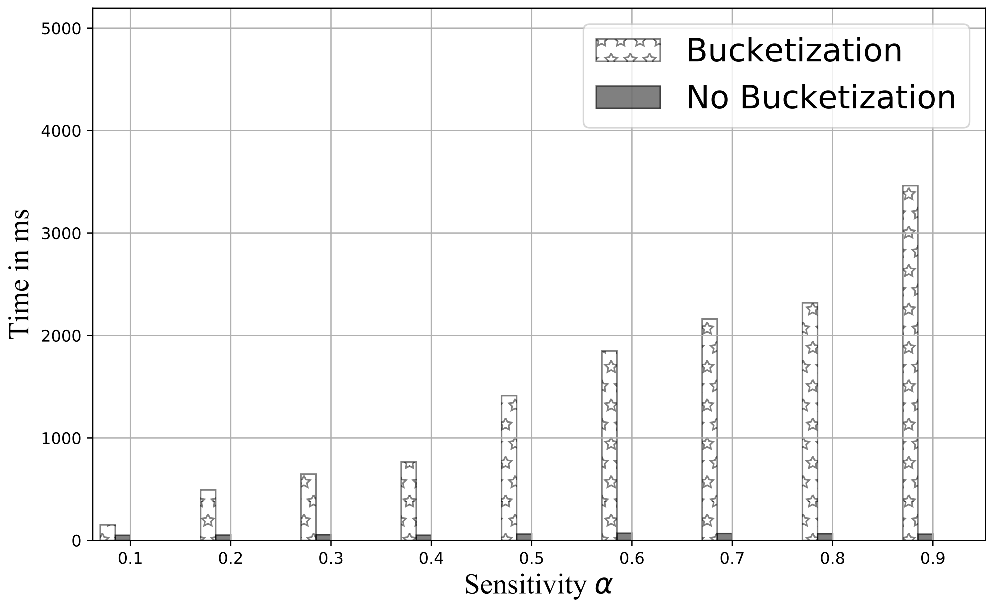


Figure 6.10: Bucketization vs No Bucketization in System C.

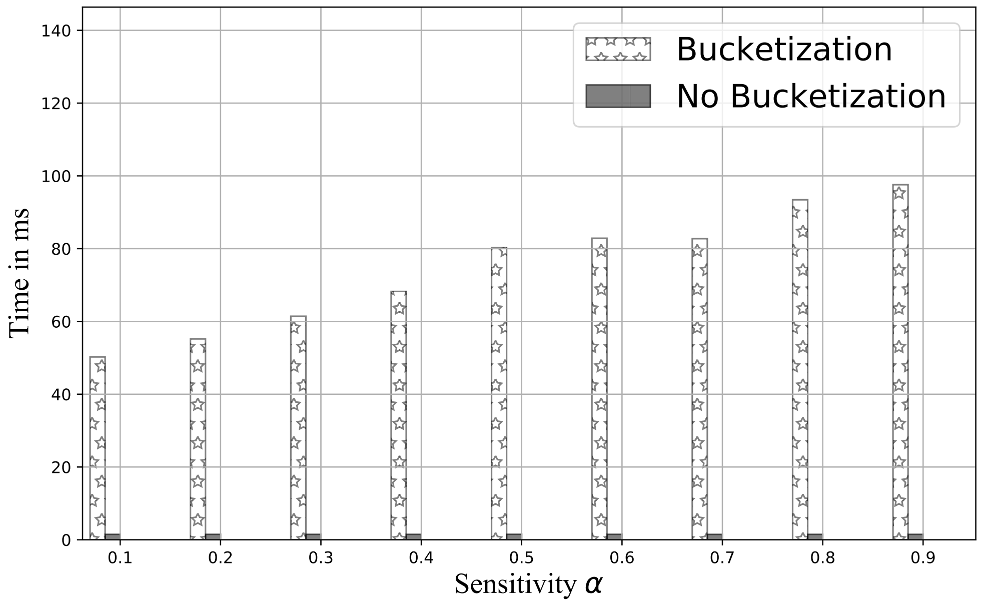


Figure 6.11: Bucketization vs No Bucketization in System D.

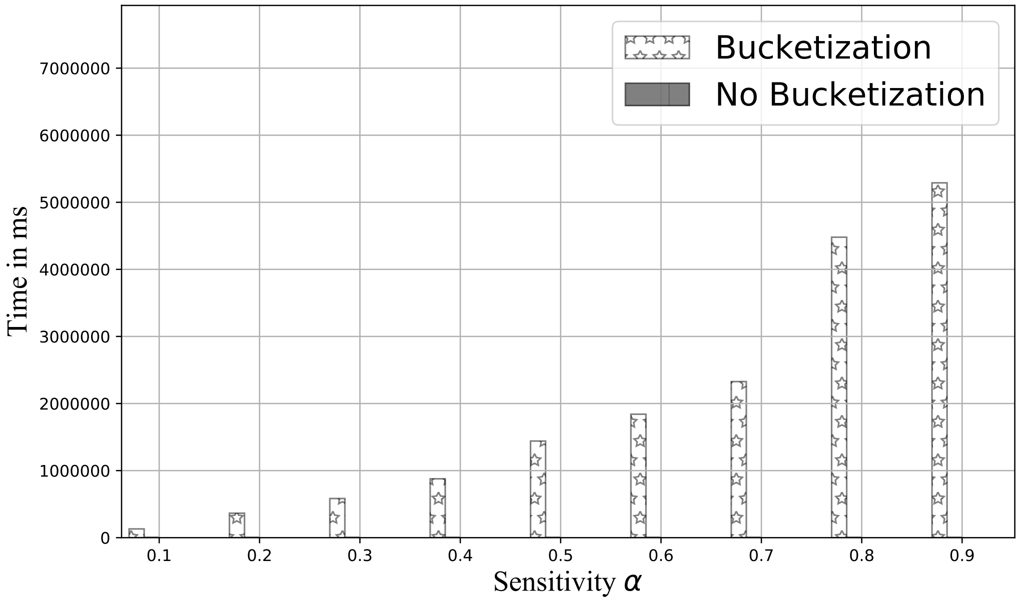


Figure 6.12: Bucketization vs No Bucketization in System E.

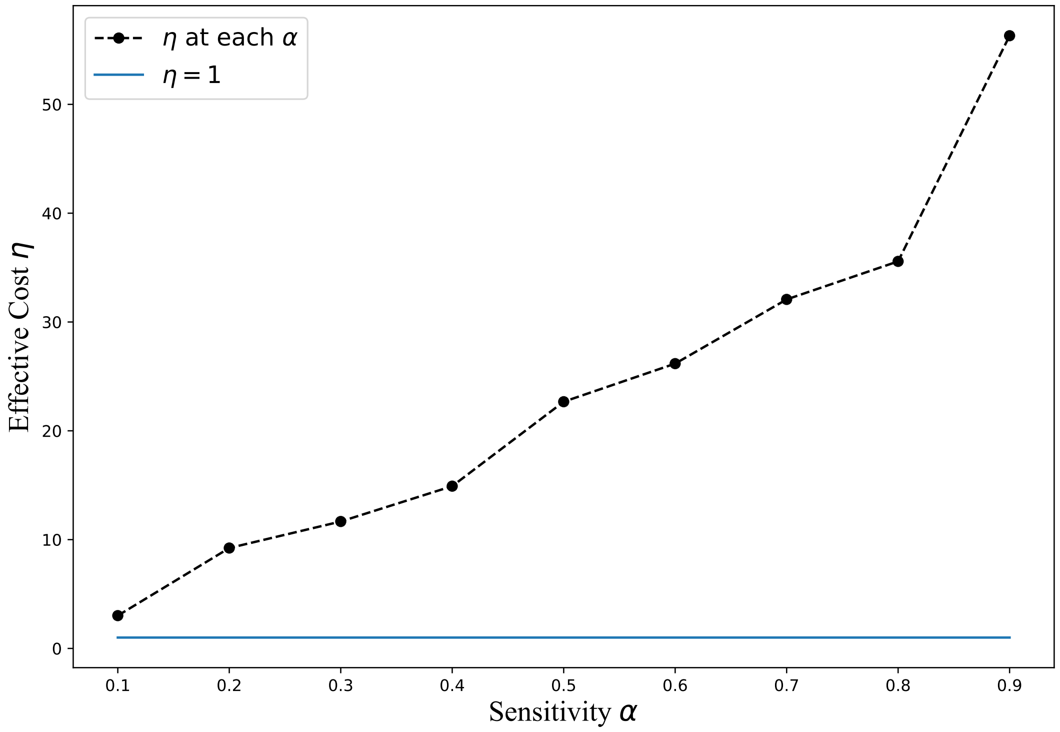


Figure 6.13: Effective Cost in System C.

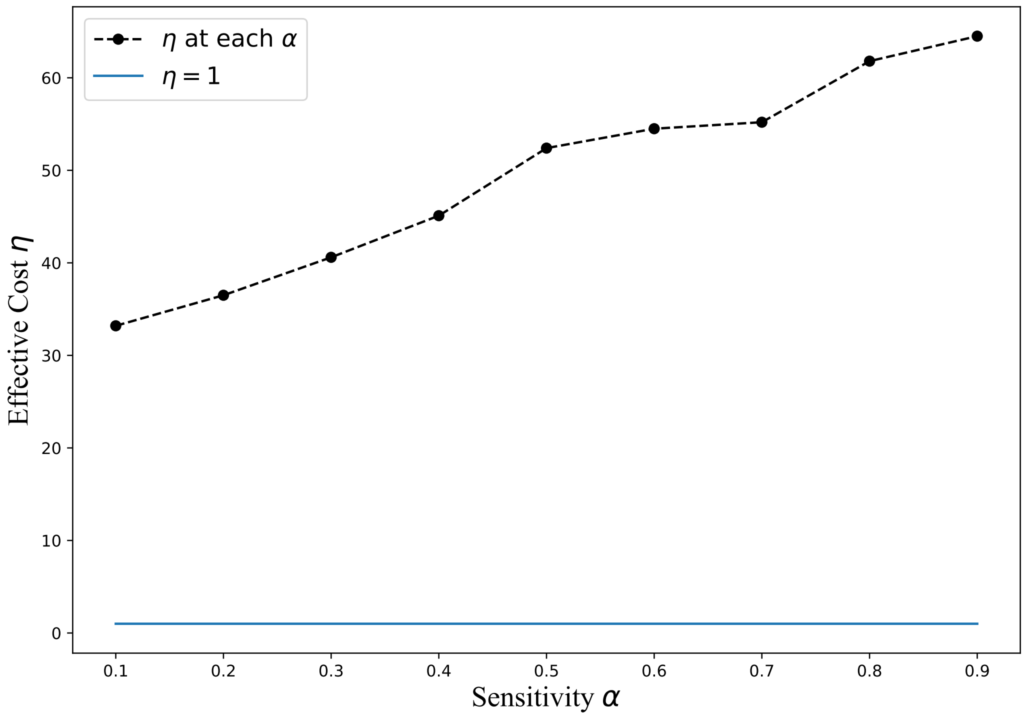


Figure 6.14: Effective Cost in System D.

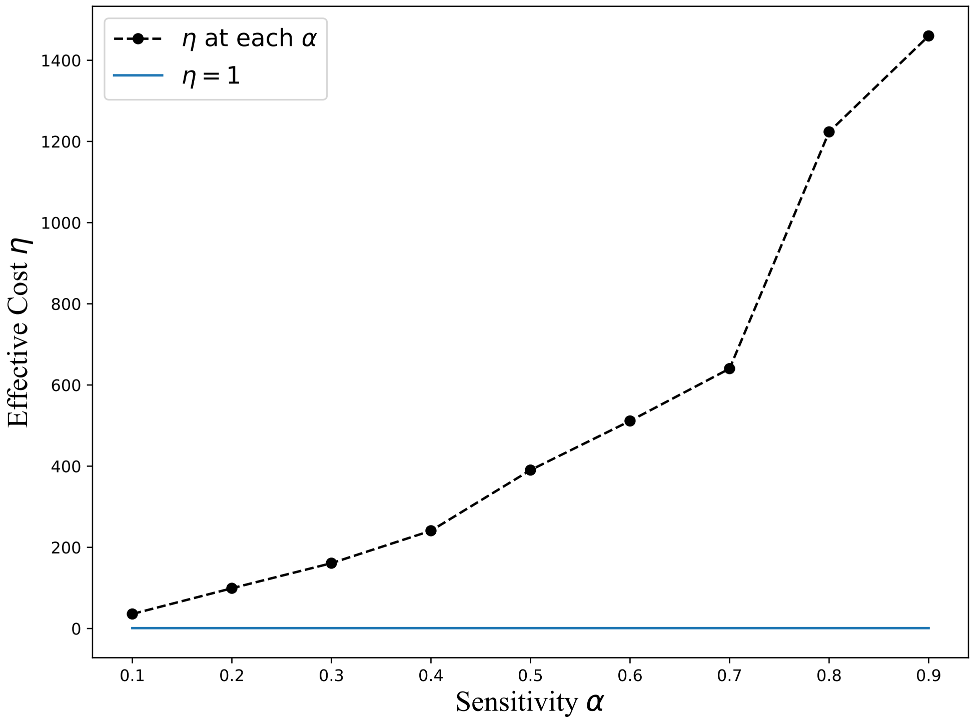


Figure 6.15: Effective Cost in System E.

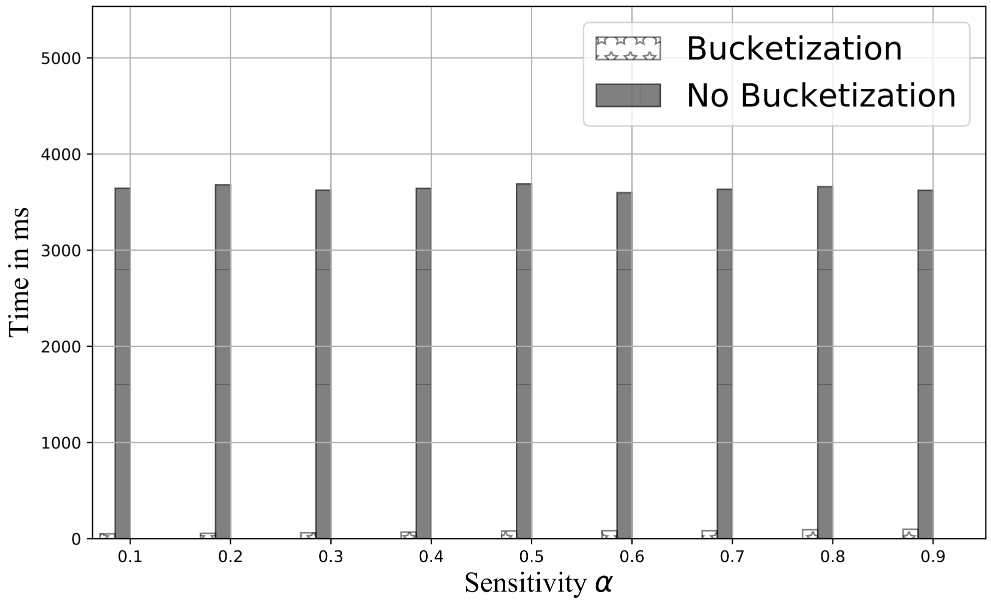


Figure 6.16: Bucketization vs No Bucketization in System F.

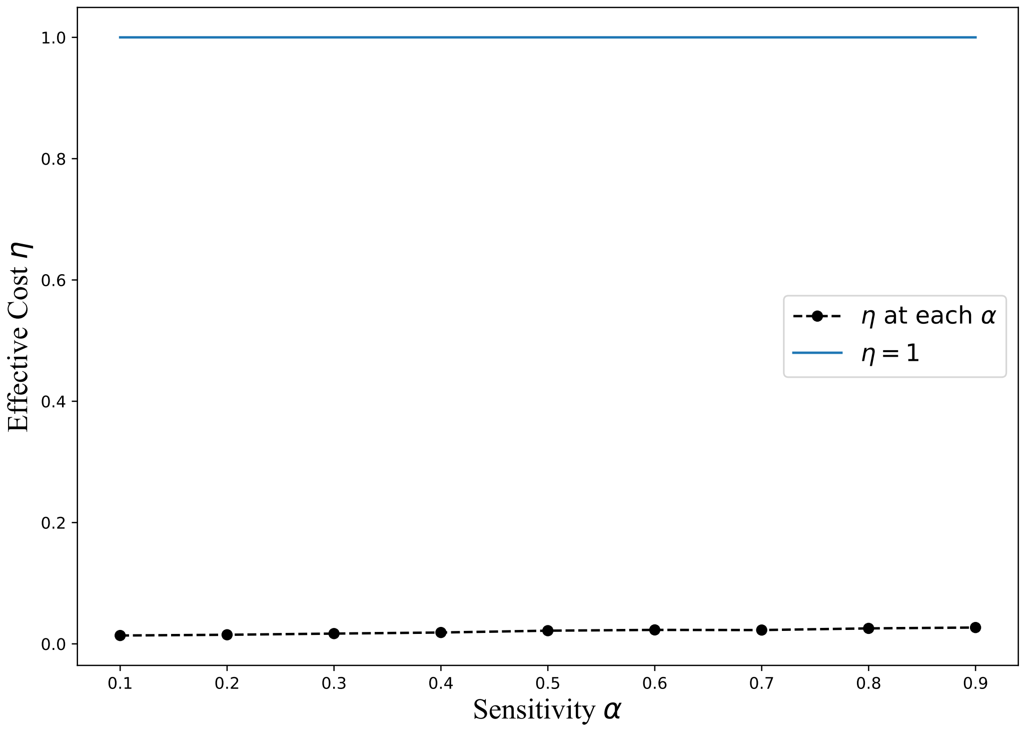


Figure 6.17: Effective Cost in System F.

6.5 Effect of Size of Data

To show that our methodology is immune to size of data set, we conducted our experiment on data set of size one hundred thousand, one and half million and finally on four and half million tuples. Each of them was tested for various values of sensitivity(α). The results show that the effective cost is less that 1, which signifies that the effectiveness of our scheme on the face huge data set.

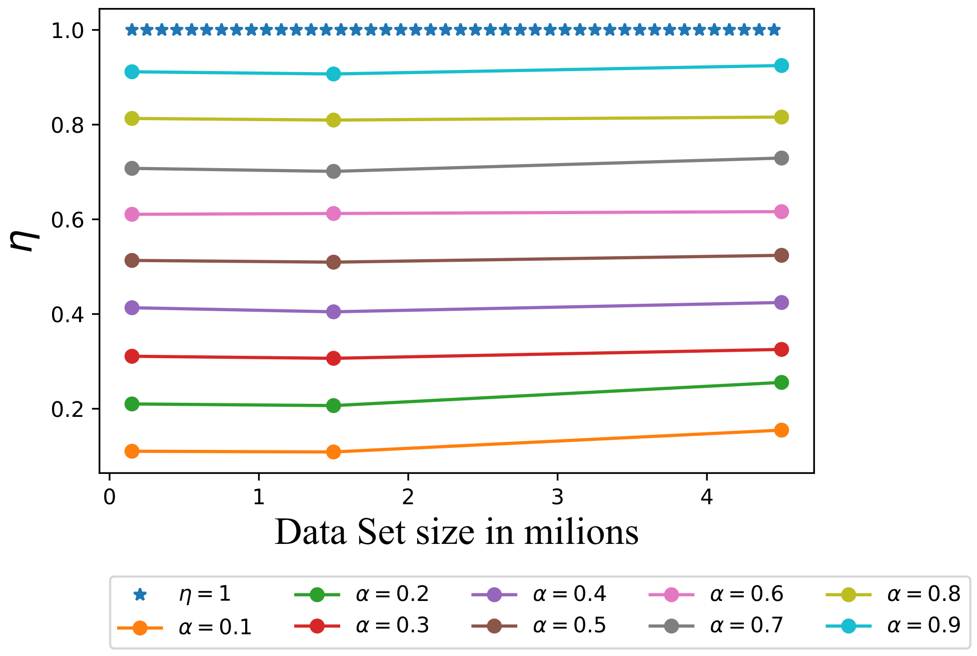


Figure 6.18: Effective Cost vs Size of data set.

Chapter 7

Conclusion and Future Work

We have tried to present a scheme which achieves a two-fold benefit. On the one hand the technique can enhance the security of databases which have a fairly small time for retrieval, but leak information. On the other hand, our scheme reduces the time for databases which are completely secure but have a high cost of processing. We have also developed a model to examine the efficiency of a database in terms of β; the order of β needed to have a technique which is more effective than our scheme.

Currently, our system supports key-word search only. The next logical course of action is to support range queries and joins. Another aspect which we would like to explore, is given a data set which is completely sensitive, to come up with an effective amount of non-sensitive data which can be added to the data set, so that we can apply our technique.

Bibliography

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

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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.

[8] C.-P. Bezemer and A. Zaidman. Multi-tenant saas applications: maintenance dream or nightmare? In *Proceedings of the Joint ERCIM Workshop on Software Evolution (EVOL) and International Workshop on Principles of Software Evolution (IWPSE)*, pages 88–92. ACM, 2010.

[9] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan. Keyword searching and browsing in databases using BANKS. In *Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, February 26 - March 1, 2002*, pages 431–440, 2002.

[10] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano. Public key encryption with keyword search. In *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, pages 506–522, 2004.

[11] 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.

[12] D. Cash, J. Jaeger, S. Jarecki, C. S. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner. Dynamic searchable encryption in very-large databases: Data structures and implementation. Citeseer, 2014.

[13] Y.-C. Chang. Single database private information retrieval with logarithmic communication. In *Australasian Conference on Information Security and Privacy*, pages 50–61. Springer, 2004.

[14] M. Chase and S. Kamara. Structured encryption and controlled disclosure. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 577–594. Springer, 2010.

[15] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *Foundations of Computer Science, 1995. Proceedings., 36th Annual Symposium on*, pages 41–50. IEEE, 1995.

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

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

[18] 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.

[19] T. ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

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

[21] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold. Keyword search and oblivious pseudorandom functions. In *Theory of Cryptography, Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005, Proceedings*, pages 303–324, 2005.

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

[23] C. Gentry and S. Halevi. Implementing Gentry's fully-homomorphic encryption scheme. In *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, pages 129–148. 2011.

[24] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. E. Lauter, M. Naehrig, and J. Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pages 201–210, 2016.

[25] 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.

[26] E.-J. Goh et al. Secure indexes.

[27] 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.

[28] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[29] M. T. Goodrich. Randomized shellsort: A simple oblivious sorting algorithm. In *Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms*, pages 1262–1277. Society for Industrial and Applied Mathematics, 2010.

[30] 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.

[31] V. Hristidis and Y. Papakonstantinou. DISCOVER: keyword search in relational databases. In *VLDB 2002, Proceedings of 28th International Conference on Very Large Data Bases, August 20-23, 2002, Hong Kong, China*, pages 670–681, 2002.

[32] 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.

[33] 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.

[34] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Batch codes and their applications. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 262–271. ACM, 2004.

[35] 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.

[36] 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.

[37] 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.

[38] E. Kushilevitz and R. Ostrovsky. Replication is not needed: Single database, computationally-private information retrieval. In *Foundations of Computer Science, 1997. Proceedings., 38th Annual Symposium on*, pages 364–373. IEEE, 1997.

[39] W. Li and L. Ping. Trust model to enhance security and interoperability of cloud environment. In *IEEE International Conference on Cloud Computing*, pages 69–79. Springer, 2009.

[40] H. Lipmaa. An oblivious transfer protocol with log-squared communication. In *International Conference on Information Security*, pages 314–328. Springer, 2005.

[41] 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.

[42] C. Mavroforakis, N. Chenette, A. O'Neill, G. Kollios, and R. Canetti. Modular order-preserving encryption, revisited. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 763–777. ACM, 2015.

[43] P. Mell. The nist definition of cloud computing.

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

[45] 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.

[46] 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.

[47] 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.

[48] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 223–238. Springer, 1999.

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

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

[51] J. Rittinghouse and J. Ransome. Cloud computing: Implementation, management, and security. 2009.

[52] 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.

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

[54] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[55] E. Shi, T.-H. H. Chan, E. Stefanov, and M. Li. Oblivious ram with o ((logn) 3) worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*, pages 197–214. Springer, 2011.

[56] D. X. Song, D. Wagner, and A. Perrig. Practical techniques for searches on encrypted data. In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[57] E. Stefanov, M. Van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path oram: an extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 299–310. ACM, 2013.

[58] 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.

[59] 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.

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

[61] Z. Xia, X. Wang, X. Sun, and Q. Wang. A secure and dynamic multi-keyword ranked search scheme over encrypted cloud data. *IEEE Trans. Parallel Distrib. Syst.*, 27(2):340–352, 2016.

[62] A. C. Yao. Protocols for secure computations. In *Foundations of Computer Science, 1982. SFCS'08. 23rd Annual Symposium on*, pages 160–164. IEEE, 1982.

[63] A. C.-C. Yao. How to generate and exchange secrets. In *Foundations of Computer Science, 1986., 27th Annual Symposium on*, pages 162–167. IEEE, 1986.

[64] 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.

[65] 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.

[66] 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.

[67] 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.

[68] W. Zheng, A. Dave, J. Beekman, R. A. Popa, J. Gonzalez, and I. Stoica. Opaque: An oblivious and encrypted distributed analytics platform. In *NSDI*, 2017.