ON THE SECURITY OF NOSQL CLOUD DATABASE SERVICES

by

MOHAMMAD AHMADIAN
M.S. University of Central Florida, 2014
M.S. Amirkabir University of Technology, 2009

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

Major Professor: Dan C. Marinescu

ABSTRACT

Processing a vast volume of data generated by web, mobile and Internet-enabled devices, necessitates a scalable and flexible data management system. Database-as-a-Service (DBaaS) is a new cloud computing paradigm, promising a cost-effective and scalable, fully-managed database functionality meeting the requirements of online data processing. Although DBaaS offers many benefits it also introduces new threats and vulnerabilities. While many traditional data processing threats remain, DBaaS introduces new challenges such as *confidentiality violation* and *information leakage* in the presence of privileged malicious insiders and adds new dimension to the data security. We address the problem of building a secure DBaaS for a public cloud infrastructure where, the Cloud Service Provider (CSP) is not completely trusted by the data owner. We present a high level description of several architectures combining modern cryptographic primitives for achieving this goal. A novel *searchable security scheme* is proposed to leverage secure query processing in presence of a malicious cloud insider without disclosing sensitive information. A holistic database security scheme comprised of data confidentiality and information leakage prevention is proposed in this dissertation. The main contributions of our work are:

(i) A searchable security scheme for non-relational databases of the cloud DBaaS;

(ii) Leakage minimization in the untrusted cloud.

The analysis of experiments that employ a set of established cryptographic techniques to protect databases and minimize information leakage, proves that the performance of the proposed solution is bounded by communication cost rather than by the cryptographic computational effort.

To Ghazal, Ryan & my dear parents.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Dan C.Marinescu for the continuous support of my Ph.D. study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this dissertation. Moreover, I would like to thank the rest of my thesis committee: Prof. Joseph Brennan, Dr. Mark Heinrich, and Dr. Pawel Wocjan, for their encouragement and insightful comments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Cloud computing is an appealing alternative for data processing, however it raises serious concerns regarding the security of sensitive data. Is it feasible to outsource computation without revealing any private information? This dissertation tries to find a comprehensive and practical answer to this challenging question through the experimental design and theoretical analysis. One of the most popular service of cloud computing is Database as a Service (DBaaS), a transformative technology and ubiquitous in web and mobile applications. Any improvement in data security of DBaaS could have significant impact on the large set of applications from different area.

The idea of outsourcing storage and processing of private data to a third party is a high-risk decision that makes applications vulnerable to unauthorized access by an external or by malicious insiders (MIs). Security and privacy in the cloud environment are critical concerns for cloud users. Most of the cloud service providers (CSPs) support features that allow system administrators to deploy a basic level of security controls for hosted datasets. Nevertheless, there is no full-proof accepted solution to prevent unauthorized access by MIs who have unlimited access to the entire system. The security and privacy threats associated with cloud computing negatively affect all cloud services and act as an inhibitor for potential users. Many cloud users have sensitive data related to their enterprise, so any unauthorized data access will devastate their businesses.

In any cloud based applications, there are three interested parties: the data owners, a cloud service provider and the users' applications. In fact, these parties are interacting in the public cloud environment as a communication framework. Since this work is concentrated on security of DBaaS therefore, primarily we present the most accepted definition of DBaaS.

Definition 1 (DBaaS)

DBaaS is a cloud-based service that provides users with database functionality without deployment of on-premise physical hardware or software.

DBaaS provides two predominant database services in the DBaaS portfolio; relational (RDBMS) and non-relational (NoSQL).

Cloud relational database service. RDBMS is widely used as a key building block of online systems in order to support data management for many applications such as On-Line Transaction Processing (OLTP). Relational databases have been dominant database architecture for decades. RDBMS is popular because of many benefits such as persistence, integration, SQL, and concurrent transactions. Moreover, many IT professionals are trained to implement application using RDBMS. In the cloud DBaaS, an application developer plays a more important role than the traditional on-premise computation, due to the fact that cloud eliminates the need for database administrators. That is another reason for popularity of DBaaS. For those reasons, cloud computing adopted RDBMS and equipped it with more features and delivers it as a fully managed and integrated service. Therefore, the cloud RDBMS is ideally suited for complex query-intensive analytical workloads. Major CSPs such as Amazon Web Services, Microsoft Azure and Google Cloud Platform offer a broad range of cloud storage and data management that help organizations move faster from on-premise computing to cloud computing. For instance, Relational Database Service (RDS) offered by Amazon Web Services (AWS) is a cost-efficient database functionality. AWS RDS provides six popular database engines to choose from, including Amazon Aurora, Oracle, Microsoft SQL, PostgreSQL, MySQL and MariaDB.

Cloud NoSQL database service. The name *NoSQL* given to the storage model is misleading, NoSQL does not follow the database model of RDBMS and refers as "not only SQL". Michael Stonebreaker notes that blinding performance depends on removing overhead. Such overhead

has nothing to do with SQL, but instead revolves around traditional implementations of ACID[1] transactions, multi-threading, and disk management" [1]. The "soft-state" approach in the design of *NoSQL* databases allows data to be inconsistent and transfers the task of implementing only the subset of the ACID properties required by a specific application to the application developer. The *NoSQL* ensures that data will be "eventually consistent" at some future point in time, instead of enforcing consistency at the time when a transaction is "committed". Data partitioning among multiple storage servers and data replication are also principles of *NoSQL* philosophy; that increase availability, reduce the response time, and enhance scalability.

Scalability and availability are critical requirements for E-commerce, social networks and other applications dealing with very large data sets. Companies which are heavily involved in cloud computing and discovered early on the traditional RDBMS, cannot handle the massive amount of data and the real-time demands of online applications, critical for their business model. RDBMS schema is of little use for such applications and conversion to NoSQL databases seems to be a much better approach. Nowadays, NoSQL databases are widely supported by cloud service providers. Their advantages over traditional databases are critical for big data application. Big data and mobile applications are the two most important growth area of cloud computing. Big data growth can be viewed as a three-dimensional phenomenon; it implies the increasing volume of data, requires an increased processing speed to produce more results, and at the same time, it involves a diversity of data sources and data types [2]. A delicate balance between data security and privacy and efficiency of database access is critical for such applications. Many cloud services used by these applications operate under tight latency constraints. Moreover, these applications have to deal with extremely high data volumes and are expected to provide reliable services for very large communities of users.

[1] ACID (Atomicity, Consistency, Isolation, Durability) properties guarantee that transactions are processed reliably.

As an example of NoSQL service supported by cloud: AWS offers DynamoDB, a fully managed fast and flexible NoSQL database service that provides fast performance with consistent scalability. DynamoDB supports both document and key-value store models, which are very flexible data models. This feature makes DynamoDB an acceptable choice for mobile, web, gaming and Internet Of Things (IOT) applications. AWS Management Console or the Amazon DynamoDB Application Program Interface (API), can be used for scales up or down without downtime or performance degradation.

1.1 Motivation

Now that the advantages of processing data in cloud is discussed, the principal research challenge is to answer this question, "Is it possible to delegate processing of your data without getting your private information revealed?" In other words, the goal of this research is to resolve the conflict between the availability of data on a public-access cloud and providing the required protection with an acceptable cost factor. By using traditional cryptosystems, the cloud server needs to decrypt the data with secret decryption key before being able to process the data; however, this process reveals users' private information to adversary or MI. Resolving this issue requires a multidisciplinary approach that ties computer science and mathematics with application specific knowledge such as finite field. Therefore, the main objective of current dissertation is to design a secured solution for cloud-based on-line applications in order to address the corresponding security requirements. The key contributions and impact of this research are cloud-based large-scale database systems, online query processing and web applications.

Technology research analysis indicates that a large number of enterprises are using cloud DBaaS from major CSP. The number of websites hosted on AWS has increased from $6.8M$ in September 2012 to $11.6M$ in May 2013, a 71% upsurge [3] [4]. Furthermore, a 67% annual growth rate

is predicted for DBaaS by 2019. Undoubtedly, an efficient security scheme is required for high volume of data stored and processed in the cloud considering cloud threat model. Threats of cloud computing can be analyzed from multiple viewpoints; this work investigates it from the adversarial prospective which is a holistic multifaceted procedure considering the whole systems' security end-to-end. The adversarial threat analysis starts with thinking like a hacker and continues to prepare a corresponding countermeasure.

While many schemes have been proposed to search encrypted relational databases, less attention has been paid to NoSQL databases. In this work, we report on the design and the implementation of a security scheme, called "SecureNoSQL" for searching encrypted cloud NoSQL databases. To the best of our knowledge, our solution is one of the first efforts covering not only data confidentiality, but also the integrity verification of the datasets residing on cloud servers. In our system, a secure proxy carries out the required transformations and the cloud server is not modified. The construction is applicable to all NoSQL data models. In our experiments, we present its application to a *Document-store* data model.

The aggregation of large number of databases in cloud, increases the risk of sensitive information to be compromised even for the encrypted data. The concept of *cloud information leakage*, resulted from any cross-referencing attacks in the pool of cloud-hosted databases, is defined and a secure *selective disinformation document padding* method is proposed for NoSQL databases to leverage leakage-resilient data management in the presence of cloud internal and external adversaries.

We propose a solution that satisfies security requirements for applications using *non-relational* functionality of cloud DBaaS. In the second part of this work, the problem of minimizing information leakage is investigated. The proposed scheme is easily adaptable for a hybrid or a community cloud environment, where the security risk is lower than the public cloud.

1.2 Objectives

The primary research interests of this work are at the intersection of cloud computing and information security, in an area known as secure computation outsourcing on the third party. We seek to understand the needs for security and privacy of both individual users, as well as any large organization in a public cloud environment. The security of users' data in the cloud computing, as a large scale distributed computational platform, is a demanding challenge that influences all users. The evidence shows that the importance of information security in cloud is increasing as more online systems are moving into to the cloud. This vision motivates us to design security schemes that enable cloud users to securely receive the productivity and computational benefits of the cloud DBaaS without compromising security and privacy.

1.3 Problem statement

The data owner has a database containing sensitive information and intends to encrypt and upload it to a cloud DBaaS and give search permission to a large group of online users. The data owner wishes to keep the data and users queries private from the CSP. Users should be able to retrieve all the documents that satisfy specific condition posed by their queries from the encrypted database. An additional privacy requirement, critical for some applications such as stock market data, is considered to hide any information about the access pattern from a cloud insider.

The proposed solution requires only one interaction per query with a minimum communication between users application and DBaaS server. The performance of DBaaS server for processing a requested query over encrypted database still remains linear in the size of database. We address both confidentiality and leakage prevention requirement.

1.4 Dissertation organization

We discuss all of our approaches and solutions to address the problem stated above in the rest of this dissertation which has been organized as follows: the key required building blocks for proposed approach is presented in Chapter 2. The design and implementation of Order Preserving Encryption (OPE) and its application for protection of sensitive data is demonstrated in Chapter 3. The detail design of SecureNoSQL as key aspect of secure query processing and the structure of security plan and the notation of descriptive language for generation of security plan are discussed in Chapter 4. Afterwards, the mechanism for information leakage prevention is discussed in Chapter 5. Finally this dissertation is concluded in Chapter 6 with conclusion and ideas for the future work.

CHAPTER 2: BACKGROUND

2.1 Architectural challenges of relational database

Object-relational impedance mismatch. An object-relational impedance mismatch[1] refers to set of conceptual problems that can occur when a logical object in the application layer is mapped into multiple tables in the relational database and vice versa. The referred mismatch is a result of the differences between object model and relational model. For instance, information of a customer as a logical object in the object model could be broken down into smaller components such as address, order detail, personal information and billing details. In normalized relational database, each of those components have to be stored in different tables.

Mapping logical objects to tables and reverse creates a performance degradation. Most of the times, a users' application needs to store/retrieve a logically atomic object in the relational database. However, the database schema dictates to map a single logical object to multiple tables and vice versa. Object-relational impedance mismatch imposes significant performance cost especially for online systems which need interactive speed. The object-relational impedance mismatch is illustrated in Figure 2.1.

[1] This term comes from an analogy with electrical engineering where input and output impedance match results in maximum power flow.

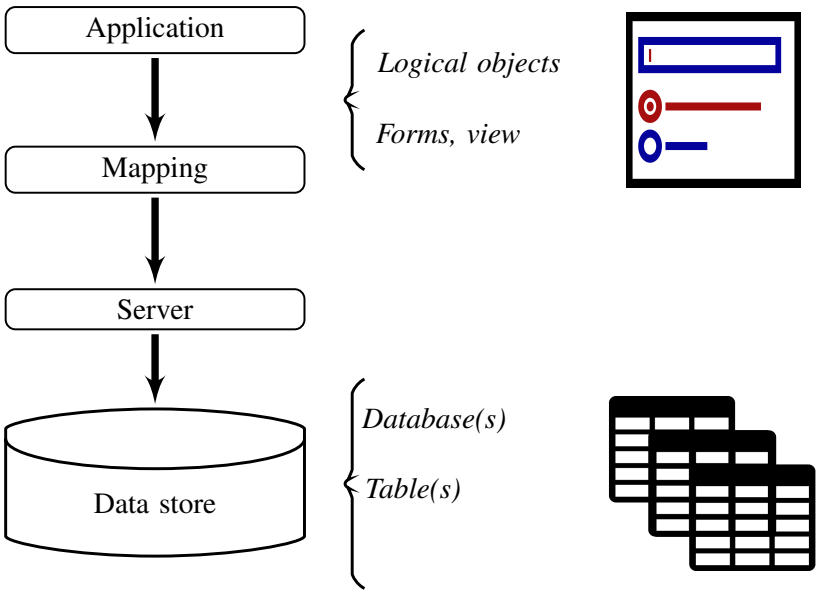


Figure 2.1: Object-relational impedance mismatch occurs because of differences between object model and relational model in accessing data.

Scalability of database. Database technology has two fundamental options to address rapid growth of database workload: (i) Scale-up (vertical scaling) - assign a higher share of local resources; (ii) Scale-out (horizontal scaling) assign more servers to carry out the workload.

Relational databases were started in the era when the volume of data was small and organized enough to run on a single server. Intrinsically, relational database can only be scaled on single server; thus, an increase in the data volume necessitates scale-up in hardware by adding more resources to the server. However, alongside the high cost, scale-up approach has serious limitations in terms of performance and server density. Instead, scale-out solution, in form of clusters of smaller commodity machines is optimized for data analytics workloads.

Business requirements necessitates to store heterogeneous data of each form such as email, voice, image, chat transcript, profile, etc. This sort of transformation causes huge scalability challenge for RDMS. To mitigate this challenge, an assortment of improvement is adopted, including using multi-layer and more complex master-slave architectures, data sharding, and replication. These innovative improvements certainly made RDBMS more scalable, but, there is always a single point of failure in the system. Another prohibitive reason for scalability of RDMS is its high setup cost.

On the other hand, NoSQL is designed for scale-out on a large number of inexpensive commodity machines using distributed file system. In addition, the hardware failure is considered and eliminated from the design phase, so that when a node fails other nodes handles its task (see Figure 2.2).

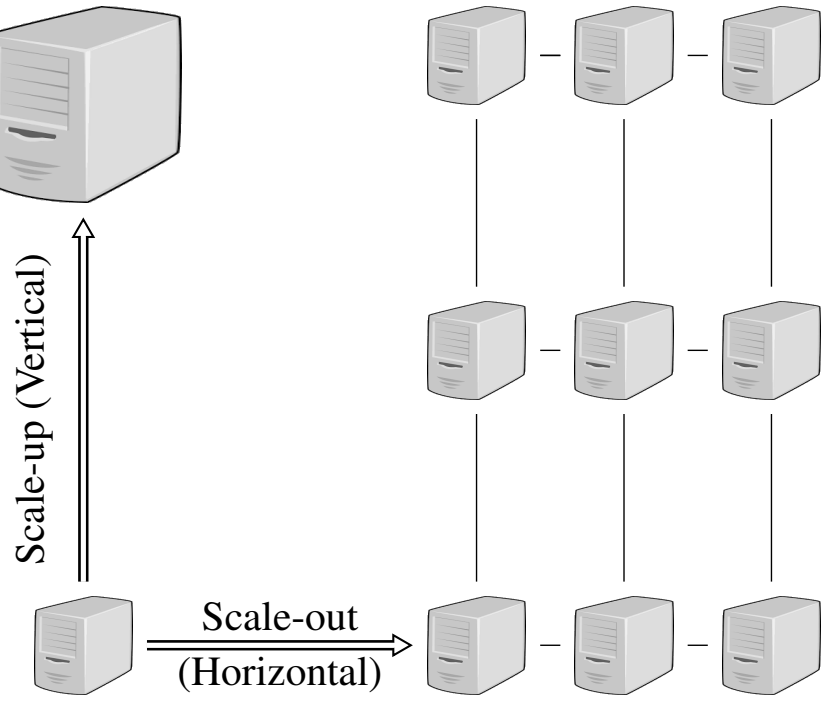


Figure 2.2: Scale-up and scale-out approaches to scale database with workload growth.

2.2 New approaches for data processing

Data analytics, enterprise, and multimedia applications, as well as applications in many areas of science, engineering, and economics, including genomics, structural biology, high energy physics, astronomy, meteorology, and the study of the environment, take advantage of cloud computing for processing very large datasets. Companies heavily involved in cloud computing such as Google and Amazon, e-commerce companies such as *eBay*, and social media networks such as Facebook, Twitter, or LinkedIn discovered early on that traditional relational databases cannot handle the massive amount of data and the real-time demands of online applications critical for their business model. The relational schema is of little use for such applications and conversion to NoSQL databases seems a much better approach.

NoSQL describes a fairly large number of *NoSQL* database technologies, more than 120 by our count, have been created in recent years[2]. NoSQL databases are non-relational, distributed, horizontally scalable, and schema-free. They are classified based on their data models. Choosing a proper data model has an extremely important influence on the performance and scalability of the data stores. Since, our work has a tight connection to NoSQL data models, we provide brief definitions for several data models.

Key-value stores. This simple data model resembles an associative map or a dictionary where a key uniquely identifies the value. The data can be either a primitive data type such as a string, an integer, an array, or it can be an object. This model is effective for storing distributed data, thus, it is highly scalable which makes it ideal data model to cloud data management systems. Systems

[2]For compete list refer *http://www.nosql-database.org/*

such as Bigtable [5], Megastore [6], CouchDB [3], DynamoDB [7], *MemcacheDB* [4] and *Redis* [5] use this model. However, key-value store is not appropriate data model for applications demanding relations or structures.

Column-family stores. In this model, the data is stored in a column-oriented style and the dataset comprise several rows, and each row is indexed by a unique key, called *primary key*. Each row is composed of a set of column families, and different rows can have different column families. Similarly, to key-value stores, the row key resembles the key, and the set of column families resembles the value represented by the row key. However, each column family further acts as a key for the one or more columns that it holds, whereas each column consists of a key-value pair. *Hadoop HBase* directly implements the Google *Bigtable* concepts, whereas Amazon *SimpleDB* and *DynamoDB* contain only a set of column name-value pairs in each row, without having column families. Sometimes, *SimpleDB* and *DynamoDB* are classified as key-value stores. Typically, the data belonging to a row is stored together on the same server node. *Cassandra* provides the additional functionality of super-columns, which are formed by grouping various columns together. *Cassandra* can store a single row across multiple server nodes using composite partition keys. In column-family stores, the configuration of column families is typically performed during start-up. A column family in each row can contain different columns. A prior definition of columns is not required and any data type can be stored in this data model. In general, column-family provide more powerful indexing and querying than key-value stores because they are based on column families and columns in addition to row keys. Similarly, to key-value stores, any logic requiring relations must be implemented in the client application.

Document stores. In this model data are stored inside the internal structure, while in the key-

[3] *http://couchdb.apache.org*

[4] *http://www.Memcached.org*

[5] *http://redis.io*

value store the data are opaque to database. Now the database engine applies meta-data to create a higher level of granularity and delivers a richer experience for modern programming techniques. Document-oriented databases are using a key to locate the document inside data store. Most document stores use *JSON* or *BSON* (Binary JSON). *Document stores* are suited for applications where the input data can be represented in a document format. A document can contain complex data structures such as nested objects. Document store allows document grouping into collections. A document in a collection should have a unique key. Unlike an RDBMS, where every row in a table follows the same schema, a document in document stores may have a different structure. Document stores provide the capability of indexing documents based on the primary key as well as on the contents of the documents. Like key-value stores, they are inefficient in multiple-key transactions involving cross-document operations.

Graph databases. This data model based on graphs can be used to represent complex structures and highly connected data often encountered in real-world applications. In graph databases, the nodes and edges have individual properties consisting of key-value pairs. Graph databases are a good alternative for social networking applications, pattern recognition, dependency analysis and recommendation systems. Some graph databases such as *Neo4J* [6] support ACID properties. Graph data stores are not as efficient as other NoSQL data stores and do not scale well horizontally when related nodes are distributed to different servers.

Dynamic schema. Schema presents the structure of database in formal language by set of rules known as integrity constraints used to govern a database. Schema expresses the organization of data and relations among entities, tables, views and other components of a database.

Relational database is fixed-schema, so the schema need to be defined before entering data. The structure and type of data are predefined ;however, NoSQL databases are built to work with dy-

[6] *http://neo4j.com*

namic schema that enables it to store all type of data. Unlike the solid schema of RDBMS, in NoSQL data can be read, and schema is forming as reading is in progress. In this way, the data does not force to fit in the predefined fixed schema like the usual procedure in RDBMS. Now the data can be stored and processed in variety of formats, and it projects into the schema on the fly. This feature makes it possible to process complex analytics through having more data granularity by simpler algorithms.

2.3 Related work

Data encryption is an obvious path to database security but it should by done in a manner that does not hinder access to data. The first SQL-aware query processing for an encrypted database was *CryptDB* [8]. CryptDB satisfies data confidentiality for an SQL relational database. However, *CryptDB* cannot perform queries over data encrypted with different keys. One important application of searching encrypted data [9, 10, 11, 12, 13] is in cloud computing where the clients outsource their storage and computation. In [9] a practical searchable security scheme is introduced which can search on encrypted data sets in sub-linear time complexity by using different types of indices; however, it is not practical on NoSQL data sets which are designed to scale to millions of users doing updates simultaneously [14].

Order-preserving symmetric encryption (OPE) is a deterministic encryption scheme which maps integers in the range $[1, M]$ into a much larger range $[1, N]$ and preserves numerical ordering of plaintexts [15, 16]. OPE is attractive because fundamental database operations such as sorting, simple matching (i.e., finding m items in a database), range queries (i.e., finding all m items within a given range), and search operations can be carried out efficiently over encrypted data. Moreover, OPE allows query processing to be done as efficiently as for plain data; the database server locates the desired encrypted data in logarithmic-time via standard tree-based indexing data structures.

An investigation of OPE security against a *known plaintext attack* with known N plaintexts is reported in [17] and [18]; the last research concluded that the ideal OPE module accomplishes one-wayness security.[7] The Shannon entropy[8] achieved by an ideal OPE is maximal when the mapping of integers in the range $[1, M]$ to a much larger range $[1, N]$ results in a uniform distribution. The risk of disclosure caused by main memory attack is quantified by [19] and [20]. An application of OPE in cloud environment is reported in [21] and [22]. Furthermore, application of classic cryptography on relational database system for embedded devices was studied in [23].

NoSQL databases are suffering from lack of proper data protection mechanism because these databases have been designed to support high performance and scalability requirement. In order to protect personal and sensitive information, a privacy and security preserving mechanism is required in Big data platforms. Integration of privacy aware access control features into existing Big data is discussed in the [24] and [25]. In [26] and [27] the evolution of Big data Systems from the perspective of an information security application is studied. As a matter of fact, the proxy is very important element in the designed structure and from Information Technology prospect view there should be especial consideration for its protection. A cloud based monitoring and threat detection system proposed by [12] and [28] for critical component to make infrastructure systems secure.

Sometimes, data is encrypted before a database is stored on a cloud; database queries to the cloud database are also encrypted. The two systems discussed next do not require modifications of database services, encrypted data is processed identically as the plaintext data. Database optimizations such as multi-layer indexing and cache and file management are applied to encrypted databases without any modifications. CryptDB [29] can be used for encrypted SQL cloud

[7] One-way functions are easy to compute, but computationally hard to invert.

[8] The entropy measures the degree of uncertainty; the Shannon entropy of a discrete random variable X with n realizations $x_1, x_2, \ldots, x_n$ with probabilities $p_1, p_2, \ldots, p_n$, respectively, is: $H(X) = -\sum_{i=1}^{n} p_i \log p_i$.

databases. The procedure to search encrypted NoSQL databases [30] involving a secure proxy guarantees that an insider attacker would never obtain the decryption keys. The proxy encrypts the queries from the clients and decrypts the query responses from the server. The process is completely transparent to the clients which are not involved in encryption/decryption operations.

The proxy in [30, 22] ensures that an insider attacker could not access sensitive information. However, an insider attacker can exploit leaked information from multiple databases. Inference attacks against CryptDB are discussed in [31]. Deterministic and OPE cryptosystems leak critical information such as frequency and order of the original data and enable attackers to extract sensitive information.

Searchable encryption methods such as Oblivious RAM (ORAM) [32, 33] provide an acceptable level of security. However, the efficiency and high computational cost, as well as the excessive communication costs between the clients and the server make this method impractical [34].

The *information leakage* problem raised with advent of cloud database services and as large number of databases are being collected in DBaaS warehouse. We address this problem in the second part of this research. Two cloud information leakage prevention methods have been proposed: (i) restrict the sequence of queries or the query processing time; and (ii) insert disinformation documents. In the first approach, a quantitative characterization of correlation-based information leakage, formulated through the capacity of a $(n, q) - leakage$ channel is studied using restriction on the number of query or query processing time [35]. The n-channel capacity is defined as the maximum possible depth of query-able information for any user.

The capacity of a n-leakage channel, is defined as the probability of accessing a specific sensitive document in n trials. In this approach, a group of documents form a chain with track of key-value pairs, and attacker can locate the head of the chain. By following the footprints, the rest of sensitive

information can be accessed. The attacker is not only constrained by the number of trials, but also by the time necessary to achieve his objectives. Thus, query dependency graph is established and a proxy prevents query execution when the accumulated leaked information reaches the maximum level of disclosure. This method is vulnerable to collaborative attacks and it is not effective against insider an attacker [35].

The second approach propose insertion of disinformation in the database. The disinformation information will provide multiple values to an attribute and mislead an attacker. This solution drastically increases the number of total documents, the search time, and the communication costs[36]. However, multi-level indexing could reduce the query processing time in this case.

An exact leakage analysis in a cloud warehouse with hundreds of thousands of datasets with millions of documents each, is computationally unfeasible. Besides, exact answers for the aggregation queries for classification and cross-correlation analysis are not always required and a specific level of error is tolerable, so often fast approximation are preferable to a slow and accurate answer. The sample-based AQP method which offers answers coupled with error bars is investigated in [37, 38]

An analysis of cross-correlation among multiple databases is a process that has an exponential time and space complexity; therefore, AQP can be applied to obtain approximation in orders of magnitude faster than exact answers. However, uniform sampling cannot provide accurate response for correlated databases. Thus different methods of sampling known as biased sampling are proposed for providing better approximation [39, 40].

2.4 A cloud computing threat model

A threat model describes the threats against a system. The threat model of cloud computing can be analyzed from multiple viewpoints and we investigate it from an *adversarial* prospective. The

adversarial threat model for the Database as a Service (DBaaS) is a holistic process based on end-to-end security. The model identifies two classes of threats, as external and internal attackers.

External attacker. An attacker from the outside of cloud environment might obtain unauthorized access to the hosted databases by applying techniques or tools to monitor the communication between clients and the cloud servers. External attackers have to bypass firewalls, intrusion detection systems and other defensive tools without any authorization.

Malicious insiders. An insider attacker has different level of access to cloud resources. Unauthorized access by malicious insiders who can bypass most or all data protection mechanisms is a major source of concern for cloud users. Encrypted data and a secure proxy construction such as *SecureNoSQL*, guarantees that malicious insiders cannot access user data. The proxy encrypts/decrypts data and query/response between clients and cloud. There is still the residual risk of information leakage from encrypted datasets. A malicious insider could exploit the leaked information to organize more extensive attacks and amplify the information leakage.

Digital data can be in three states: at-rest (storage), in-process, or in-transit. Data at-rest refers to data stored in cloud persistent storage while data in-process refers to data being processed by CPU or in the memory, and data in-transit is denoted as data that flows over network equipment and cables. A comprehensive security protection strategy must be able to provide strong protection in all of the three states. In the past decades, research has been conducted on protection of data at-rest and in-transit. Cloud computing has emerged in the past few years as a new paradigm that provides data processing as a service. Recently security of data in-process has become a critical issue in cloud-based applications. The aim of our research is to present a new approach to securely outsource data processing to the cloud NoSQL database service.

2.5 Cloud data encryption schemes

Cloud data can be in three states at-rest, transit, or process and any comprehensive data security mechanism must protect data in any of these three states. The communication channels can be secured using standard HTTP over the SSL (Secure Socket Layer) communication protocol. Most CSPs provide an API for the web service that enables developers to use both the standard HTTP and the secure version of the HTTPS protocol. The security requirements of data in transit state can be fully satisfied using HTTPS for communication with cloud. The endpoint authentication feature SSL makes it possible to ensure that the clients are communicating with an authentic cloud server. The basic idea to maintain data confidentiality of data in at-rest and process is using cryptosystems, however, for processing the decryption key should disclose to the server which is a maximum confidentiality violation. Therefore, in this model, a new set of cryptography is required to secure data; meanwhile, exercising all features of cloud computing.

Random (RND). In a RND type encryption scheme, a message is coupled with a key k and a random Initial Vector (IV). This scheme is non-deterministic, encryption of the same message with the same key yields different ciphertext. This randomness provides the highest level of security.

The randomness property is achievable with different cryptosystems, Advanced Encryption Standard (AES) at Cipher Block Chaining (CBC) mode is one the most secure RND encryption. AES is a symmetric block cipher algorithm with a key size of 128,192 or 256 bits and with a block size of 128 bits. RND type schemes are semantically secure against chosen plaintext attacks and hides all kinds of information about ciphertext. Thus, RND scheme does not allow any efficient computation on the ciphertext. Equation 2.1 describes the encryption and decryption of a block

cipher in CBC mode.

$$
\begin{aligned}
&for\ j = 2 \ldots n \\
&\begin{cases}
C_1 = E_k(P_1 \oplus IV), \;\; P_1 = IV \oplus D_k(C_1) \\
C_j = E_k(P_j \oplus C_{j-1}), \;\; P_j = C_{j-1} \oplus D_k(C_j)
\end{cases}
\end{aligned}
\tag{2.1}
$$

Deterministic (DET). A DET encryption scheme produces the same ciphertext for an identical pair of plaintext and key. Block ciphers in Electronic Code Book (ECB) mode, with a constant initialization vector are deterministic (DET). Deterministic encryption scheme preserves equality; therefore, the frequencies information of the searched keywords leaks to the third party.

AES scheme in ECB mode is used for DET encryption over document-oriented NoSQL databases. DET scheme enables server to process pipeline aggregation stages such as group, count, retrieving distinct values and equality match [9] on the fields within an embedded document. The embedded document can maintain the link with the primary document through application of DET encryption. Equation 2.2 describes the encryption and decryption operation in a DET.

$$
for \;\; j = 1, .., n; \;\; C_j = E_k(P_j); \; P_j = D_k(C_j) \tag{2.2}
$$

Where: E_k is the encryption algorithm, D_k is the Decryption algorithm, k is the secret key, P is a block of plaintext data and C is a block of ciphered data.

Order-Preserving Encryption (OPE). OPE is a deterministic cryptosystem where ciphertext preserves the ordering of the plaintext data. Aggregate queries such as comparison, min, and max

[9] Equality matches over specific common fields in an embedded document will select documents in the collection where the embedded document contains the specified fields with the specified values.

can be executed on OPE-encrypted datasets. OPE offers less protection than Full Homomorphic Encryption (FHE) and leaks critical information about the plaintext data. Even Modular Order-Preserving Encryption (MOPE) [16] which is an extension to the basic OPE for security improvement, there is information leakage. An efficient inequality comparisons on the encrypted data elements can be performed by applying OPE which supports range queries, comparison, min and max on the ciphertext. We use the algorithm introduced in [15] and implemented in [21] for cloud database service. Equation 2.3 shows the preservation of order relation of plaintext in the ciphertext.

$$\begin{aligned} &\forall x, y \,|x, y \in Data\ Domain: \\ &x < y \implies OPE_k(x) < OPE_k(y) \end{aligned} \tag{2.3}$$

Where: OPE_k is key-based OPE.

Full Homomorphic Encryption (FHE). FHE enables any kind of general computation to be done on encrypted data with high semantic security which means there is no leakage about encrypted data [41]. However, it is inefficient and after lots of improvement still FHE is orders of magnitude slower than plaintext computation. Another reason for impracticality of FHE is that the user has to express any single query in form of circuit over entire dataset. Due to practical issues of FHE we applied *Additive Homomorphic Encryption* (AHOM). AHOM is partially homomorphic cryptosystem that allows the server to conduct homomorphic addition and multiplication computations on ciphertext with the result that is decrypted at the proxy. We select Paillier AHOME cryptosystem [42]. The homomorphic addition is formulated in Equation 2.4, where $m_1, m_2 \in \mathbb{Z}_n$ are plaintext messages, $r_1, r_2 \in \mathbb{Z}_n^*$ are randomly selected, and n is product of two large primes.

$$D_k(E_k(m_1, r_1).E_k(m_2, r_2)\, mod\, n^2) = m_1 + m_2\, (mod\ n) \tag{2.4}$$

Data granularity indicates the level of detail, the more detail, the higher granularity level. Encryption can be applied for datasets at various granularity level from high granularity data, like atomic level data element to low level granularity in aggregated atomic data items. For a data store encryption, it is conceivable to consider different encryption granularity levels per the corresponding data granularity. The higher level of encryption granularity, the higher the information leakage. For example, encryption of a single attribute leaks frequency information, while encryption of the entire document and collection as a single unit leaks less information. In the work reported in this study we use different encryption granularity per the data granularity level.

Protection of attributes as the lowest level of data granularity necessitates application of cryptosystems. RND cryptosystems leak minimum information from the encrypted attributes. The RND encryption function can be constructed from a DET one, simply by concatenation of a fixed length random number r to each input. Equation 2.5 outlines how a RND encryption function $E_k(x)$ is built from a DET cryptosystem.

$$E_k(x) = E'_k(x \,||\, r) \tag{2.5}$$

Where: k is the encryption key; E' is a DET encryption; r is a random number.

Data security and integrity are important factors when choosing a database for cloud applications. They are particularly critical for applications running on public clouds where multiple virtual machines (VMs) often share the same physical platform [43, 44, 45]. The importance of database security and its impact on a large number of individuals are illustrated by the consequences of three major security breaches: [46]; and [47]. In November 2013 approximately 40 million records were stolen from an unencrypted database used by Target stores. The compromised information included *personally identifiable information* (PII) and credit card data. According to a SEC (Se-

curities and Exchange Commission) report, two months later a cyber-attack on JP Morgan Chase, compromised PII records of some 76 million households and 7 million small businesses. More recently, in July 2017, massive data breach was discovered in *Equifax* and private information of 143 million people was exposed to the attackers. The leaked information includes social security numbers, birthdays, address and credit card numbers.

Traditional cryptography primitives can protect data while at-rest (storage), but plaintext data is vulnerable to insider interference during the processing time. This is particularly troubling when searching databases containing personal information such as healthcare or financial records [48], as the entire database is exposed to such attacks. These limitations motivate us to investigate methods for searching encrypted NoSQL databases. Though general computations with encrypted data are theoretically feasible using FHE, this is by no means a practical solution at this time. Existing FHE increase the processing time of encrypted data by many orders of magnitude compared with the processing of plaintext data. A recent implementation of FHE [49, 50] requires about six minutes per batch; the processing time for a simple operation on encrypted data dropped to almost one second after improvements [51]. Related areas of research are: Learning With Errors (LWE) [52] and Lattice-based encryption [53, 54] and Attribute-based Encryption [55].

CHAPTER 3: ORDER PRESERVING ENCRYPTION IN A HYBRID CLOUD

In this work, we argue that hybrid clouds provide an ideal environment for applications which require cooperation of multiple organizations; each organization has to coordinate some aspects of its activity and share some data with several other organizations, yet has strict security requirements for its own private data. Such applications could benefit from a hybrid cloud environment. Ideally, "big data" should be encrypted and stored on the public cloud. Private data should be migrated and processed on the private cloud while non-confidential data should be processed on the public cloud. We apply Order Preserving Encryption (OPE) for the sensitive information in hybrid cloud environment. To illustrate our approach, we discuss an implementation of the OPE and present a hybrid cloud solution for the control of a smart power grid. This work was reported in International Parallel and Distributed Processing Symposium (IPDPS 2014).

3.1 Motivation

The obvious approach to ensure security is to encrypt all data stored on the public cloud. This solution incurs a significant overhead as we consider vast amounts of data, in the petabyte range; moreover, data has to be decrypted before processing thus, it is exposed even for short time to insider attacks. In this study we discuss order preserving encryption for the sensitive fields of a data record and argue that this solution balances security concerns with efficiency for many applications [56, 15, 29]. It can be applied to many cases when most of the time only a relatively small range of encrypted data has to shipped back to a trusted system, in our case the private cloud, for processing.

Typically, very large data sets are generated by vast arrays of sensors monitoring the environment, sensors monitoring vital signs of outpatients, or by services offered to large groups of individuals when information is partially collected by humans and partially generated automatically. In most cases only some elements of the data records must be kept confidential. This is true for healthcare systems, financial systems, or systems used by utility companies where only personal data must be kept confidential. Some operations on partially encrypted data require access only to the plaintext elements of individual records. The most frequent operation is in fact searching the database to identify the record or records that satisfy a query. So a major challenge for the applications we consider for hybrid clouds is an efficient encryption scheme which allows searching encrypted data. Order preserving encryption allows efficient range queries on ciphertexts when the data is stored on an untrusted server, in our case the public cloud. The data is stored in encrypted form, but indexing and query processing is done exactly as for plaintext data.

3.2 Order Preserving Encryption

In this section we overview symmetric order-preserving encryption, give its intuitive justification, reflect on its relationship with the negative hypergeometric distribution, and discuss its security limitations. A symmetric encryption scheme $\mathcal{S}$ with the plaintext space $\mathcal{P}$ and the ciphertext $\mathcal{C}$ consists of three algorithms:

(a) The key generation algorithm $\mathcal{K}$ used to generate the encryption key K; this is a randomized algorithm.

(b) The encryption algorithm $\mathcal{E}$ which produces a ciphertext c given a plaintext message m

$c = \mathcal{E}(K, \mathcal{P}, \mathcal{C}, m)$.

(c) The deterministic decryption algorithm $\mathcal{D}$ which given a ciphertext c produces either the message m or a symbol $\perp$ indicating that c is invalid

$m = \mathcal{D}(K, \mathcal{P}, \mathcal{C}, c)$.

The correctness condition of the algorithm requires that

$m = \mathcal{D}[(K, \mathcal{P}, \mathcal{C}, \mathcal{E}(K, \mathcal{P}, \mathcal{C}, m)] \ \forall K$ and $\forall m \in \mathcal{P}$.

The intuition for order preserving encryption which maps a range of integers say $[1, M]$ into a much larger range of integers $1, N$ is supported by the following proposition [15]:

Proposition: There is a one-to-one and onto mapping (a bijection) from the set of all order preserving functions from a domain $\mathcal{M}$ of size M to domain $\mathcal{N}$ of a much larger size, $N >> M$ to set of all possible combinations of M out of N ordered items.

The basic idea of the proof can be illustrated using two orthogonal axes; on the x-axis we depict the set of ordered integers $\mathcal{M} = [1, M]$ and on the y-axis the ordered set of integers $\mathcal{N} = [1, N]$. We can construct an order preserving mappings $f : \mathcal{N} \mapsto \mathcal{M}$ by picking up a subset $\mathcal{S} \subset \mathcal{N}$ and mapping the i-th smallest element of $\mathcal{M}$ to the smallest i-th smallest element of $\mathcal{S}$. The function f is an order-preserving function from $\mathcal{M}$ to $\mathcal{S}$; moreover an order-preserving function corresponds to a unique M out of N combinations, namely a particular choice of the set $\mathcal{S}$. Given $M, N \in \mathbb{N}$, $x, x+1 \in \mathcal{M}$, and $y \in \mathcal{N}$ the following equation defines a probability density function:

$$\begin{aligned} &\Pr\left[f(x) \leq y \leq f(x+1) : f \xleftarrow{\mathcal{S}} OPF_{\mathcal{M},\mathcal{N}}\right] \\ &= \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}}. \end{aligned} \tag{3.1}$$

Let us know consider a statistics experiment: we have N balls, y are white/marked and $(N - y)$

are black/unmaked. We draw balls without replacement and ask ourselves what is the probability that we get x white/marked balls in M trials. This probability, $f(x)$ is the ratio of total number of successes to the total number of ways we can conduct the M trials. A success in M trials requires us to get specific numbers of white balls and black balls. The number of ways we can choose x out of y white balls is $\binom{y}{x}$; the number of ways to choose $(M-x)$ out of $(N-y)$ black balls is $\binom{N-y}{M-x}$. There are $\binom{N}{M}$ ways to conduct M trials when there are a total of N balls. It follows that

$$p(x) = \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}}. \tag{3.2}$$

$p(x)$ is a probability density function

$$\sum_{x=0}^{N} p(x) = 1. \tag{3.3}$$

We use the Vandermonde identity

$$\sum_{x=0}^{N} \binom{p}{x}\binom{q}{r-x} = \binom{p+q}{r}. \tag{3.4}$$

In our case $p = y$, $q = N - y$, and $r = M$ thus

$$\sum_{x=0}^{N} \binom{y}{x} \cdot \binom{N-y}{M-x} = \binom{N}{M}. \tag{3.5}$$

It follows that

$$\sum_{x=0}^{N} \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}} = \frac{\binom{N}{M}}{\binom{N}{M}} = 1. \tag{3.6}$$

The statistics experiment presented here is characterized by a *hypergeometric distribution* when the probability density function $p(x)$ gives the number of successes in a sequence of M draws without replacement from a population of size N when the total number of successes is y.

Equations 3.1 and 3.2 justify our intuition that a random order-preserving function can be constructed using a hypergeometric distribution. An efficient way to do so is use the hypergeometric distribution to lazy sample a random order-preserving function as described later in the study.

The OPE scheme described here is deterministic and leaks the order relations among the plaintexts. OPE algorithms satisfy a condition called *pseudo-random order-preserving function against chosen-plaintext attack*. This means that no adversary can distinguish between oracle access to the encryption algorithm and a random order-preserving function with the same domain and range.

3.3 OPE Algorithms

Without loss of generality, we consider the plaintext space $\mathcal{M}$ as the set of integers $\{1, 2, ..., M\}$ and the ciphertext space $\mathcal{N}$ as $\{1, 2, ..., N\}$. Recall that an Order Preserving Encryption scheme, (OPE) can be viewed as a deterministic function $f : \mathcal{M} \to \mathcal{N}$ which preserves the numerical order of $\mathcal{M}$ when mapped into $\mathcal{N}$. The function f randomly selects M integers in the set $\mathcal{N}$ and orders them. Then, to encrypt $m \in \mathcal{M}$, f selects the m^{th} element of this list. The cardinality of $\mathcal{N}$ is many orders of magnitude larger than M thus, a direct realization of this idea is impractical.

A practical implementation OPE algorithm is based on a process called *lazy sampling* of a Hyper Geometric Distribution (HDG). The elements of range $\mathcal{N}$ of f can be classified as marked and unmarked; when an element of $\mathcal{N}$ is selected by f it is said to be marked, otherwise it is unmarked. This process resembles the statistical experiment with marked and unmarked balls discussed in Section 3.2. If we draw balls without replacement, the number x of marked balls drawn after y

samplings is described by HDG, see Equation 3.2. The correspondence between OPF and HGD enables us to determine how many points of our OPF mapping lie under a given point of the range; we treat the points in the range $\mathcal{N}$ as the number of samples in a hypergeometric experiment where the number of marked balls is $M = |\mathcal{M}|$ and number of unmarked balls is $N - M = |\mathcal{N}| - |\mathcal{M}|$.

The algorithms for encryption and decryption use the following notations:

$| x |$: the length in bits of the string x.

$x \mid y$: a unique encoding of the concatenation of strings x and y when the concatenated string is restorable.

$1^{\ell},\ \ell \in \mathbb{N}$: the string of ℓ 1 bits.

$x \xleftarrow{\$} S$: x is selected uniformly at random from set S.

$a \xleftarrow{\$} A(x, y, ...)$: the value assigned to a by the random algorithm A when the input is $x, y, ...$

$[a]$: the set {0,1,..,a} when $a \in \mathbb{N}$.

$\perp$: symbol indicating that the ciphertext was invalid so, there is not a corresponding plaintext m.

$w \xleftarrow{cc} S$: means that w is assigned a value sampled uniformly at random from set S using coins cc of length ℓ_S, where ℓ_S denotes the number coins needed.

$\ell_1 = \ell(D, R, y)$: the number of coins needed by HGD on inputs D, R, y

Given $y, \mathcal{M}, \mathcal{N}$ we could sample effectively the HGD distribution by drawing a random coin cc. Then let the random integer x represent the count of marked balls in a sample of size y. Given $m \in \mathcal{M}$ and the key $k \in \mathcal{K}$ we can sample OPF and produce a cipher $n \in \mathcal{N}$ by searching

the domain and the range of the function f in a recursive manner. Pseudo-random integers with a number of bits dictated by the length of the block are produced by a pseudo-random generating function PRF. In each iteration of binary search the parameters of the HGD sampling algorithm are changing thus However, we need a variable length PRF. The proposed variable length pseudo-random function(VL_PRF) is composed of a Variable Output Length Pseudo-random Function(VOL_PRF) with a consistent Variable Input Length Pseudo-random Function VIL_PRF as follows

$VL\_PRF = VOL\_PRF(1^{ell}, VIL\_PRF(\mathcal{K}, x))$.

The parameters of $VL\_PRF(k, 1^{\ell}, D, R, b||z)$ are: key $k \in \mathcal{K}$, size of output, Domain, Range, and b, a indicator for specifying the working set given by

$$b = \begin{cases} 0 & z \in R \\ 1 & z \in D \end{cases} \tag{3.7}$$

The algorithm described below is based on the one in [15]:

(1) Start with the entire domain $D = \mathcal{M}$ and range $R = \mathcal{N}$.

(2) Chose $y \leftarrow \frac{max(\mathcal{N})}{2}$ as the pivot in range.

(3) Use a key $k \in \mathcal{K}$ to produce a pseudo-random bit sequence.

(4) Pass the pseudo-random bit sequence to the HGD sampling routine along with y, $\mathcal{M}$, and $\mathcal{N}$.

(5) The sampling function HGD returns x such that $x \leq y$ and we name x as a pivot of domain. This x describes the number of points of order-preserving function that are less than y.

(6) The m^{th} point of our OPF is the ciphertext of m, so we compare x and m

- If $m < x$ then repeat the process for the points of the domain less than or equal to x and less than or equal to y.
- If $m > x$ then repeat the process for the points of the domain greater than x and y.

(7) The termination condition is to have one element in the domain; then we pick one of the points in range as a accompanying ciphertext.

Algorithm 1: The encryption algorithm

procedure ENC(D,R,m) $M \leftarrow |D|$;
$N \leftarrow |R|$;
$d \leftarrow min(D) - 1$;
$r \leftarrow min(R) - 1$;
$y \leftarrow r + \lceil \frac{N}{2} \rceil$;
if $|D| = 1$ **then**
$cc \xleftarrow{\$}$ VL-PRF(K,$1^{\ell R}$,$(D, R, 1 \| m)$);
$c \xleftarrow{cc} R$;
return c;
$cc \xleftarrow{\$}$ VL-PRF(K,$1^{\ell 1}$,$(D, R, 0 \| y)$);
$x \xleftarrow{\$}$ HGD(D,R,y;cc);
if $m \leq x$ **then**
$D \leftarrow d + 1, ..., x$;
$R \leftarrow r + 1, ..., y$;
else
$D \leftarrow x + 1, ..., d + M$;
$R \leftarrow y + 1, ..., r + N$;
return $Enc(D, R, m)$;
end procedure

Algorithm 2: The decryption algorithm

procedure DEC(D,R,K,c) $M \leftarrow |D|$;
  $N \leftarrow |R|$;
  $d \leftarrow min(D) - 1$;
  $r \leftarrow min(R) - 1$;
  $y \leftarrow r + \lceil \frac{N}{2} \rceil$;
  if $|D| = 1$ **then**
    $m \leftarrow min(D)$;
    $cc \xleftarrow{\$}$ VL-PRF(K,$1^{\ell R}$,$(D, R, 1 \| m)$);
    $w \xleftarrow{cc} R$ **if** $w = c$ **then**
      return m;
      else
        return $\perp$;
    $cc \xleftarrow{\$}$ VL-PRF(K,$1^{\ell 1}$,$(D, R, 0 \| y)$);
    $x \xleftarrow{\$}$ HGD(D,R,y;cc);
    if $c \leq y$ **then**
      $D \leftarrow d + 1, ..., x$;
      $R \leftarrow r + 1, ..., y$;
      else
        $D \leftarrow x + 1, ..., d + M$;
        $R \leftarrow y + 1, ..., r + N$;
        return $Dec(D, R, K, m)$;
end procedure

3.4 A case study, the application of OPE to data protection in a smart power grid

A *smart power grid (SmartPG)* is an infrastructure for the production and distribution of electric power where wired and wireless communication channels, sensors, and distributed networks of computer systems are used to improve the efficiency, reliability, economics, and sustainability of energy production and distribution [57, 58]. A smart power grid includes a network of energy producers from renewable sources, such as solar, wind, and geothermal which help reduce the demand for power from the power stations burning fossil fuels. A smart power grid will reduce the carbon emissions and, at the same time, lower the cost for energy generation and transport.

The control of a smart grid is data-intensive, involves fairly complex workflows, consists of tasks with different performance and timing requirements. Each utility company must collect a fair amount of data regarding both producers and consumers of energy, process the data and using the results control the power generation and distribution. At the same time, each electric power company has to trade energy and share some resources with its peers.

A SmartPG is a system which includes intelligent components to optimize electric power generation and distribution [57, 58]. A national SmartPG will integrate resources of several utility companies which supply electricity to different regions of the country. The main features of a smart grid are [59, 60]:

(i) Increased reliability due to fault detection, self-healing, and flexibility in network topology. The smart grid will support bidirectional energy flows allowing excess energy generated locally to be pumped back into the grid.

(ii) Increased efficiency and lower costs for energy production due to greater utilization of generators. The distributed generation will reduce the demand for power from the power stations, reduce the carbon emissions due to burning of fossil fuels, and the cost for energy generation and transport.

(iii) Efficient management; a smart grid infrastructure could warn larger consumers to reduce the load temporarily and give the power stations the time to startup new generators when the consumption is very high.

(iv) Economic benefits; the producers and the consumers could be informed about the energy costs in real time and can develop optimal strategies to minimize cost; for example, the consumers could pay the peak energy prices only for critical loads. *Peak curtailment or peak leveling* - the prices of electricity are increased during high demand periods, and decreased during low demand periods.

A SmartPG has a large number of components subject to intricate interactions among the components. In addition to large power stations a smart power grid will include distributed solar and wind power generators, fuel cells, and other sources of energy located at customer premises. A complex systems of sensors and intelligent meters will provide real-time information that has to be analyzed to optimize the generation and distribution of the electric power in a smart grid.

3.5 Workload partitioning

Two of the most challenging questions in the design of a hybrid cloud are how to: distribute the workload between the public and the private clouds and how to ensure privacy and security for the data on the public cloud. The two questions are related to one another; the public cloud should process and store the most data-intensive tasks, but the more data we store and process on the public cloud the more difficult it is to guarantee the data security and privacy, while supporting efficient algorithms for data analysis some carried out by the public cloud and others by the private clouds of individual utilities.

In case of a smart power grid the smart meters and the sensors scattered throughout the smart grid produce a vast amounts of data. The very large number of smart meters located at the customers premises combined with an equally large number of sensors monitoring the power stations, the transmission lines, and the power distribution centers generate data at different rates. Some of the sensors have to be monitored every few seconds, a few critical ones even more frequently, while smart meters will probably be monitored every few minutes. It seems reasonable to assume that the data collected every hour will be of the order of petabytes and that it should be collected by applications running on the public cloud.

The strategy for workload distribution discussed in this section has several advantages:

(1) Allows utility companies to share the resources of a public cloud while guaranteeing the security of private information.

(2) Reduces the probability of a denial of service attack on the private cloud of the utility companies as the high traffic is directed to the public cloud.

(3) Reduces the storage and processing requirements for the private cloud without compromising security.

(4) Enables a the rapid retrieval of the information for to a particular utility company, power distribution center, and customer as we shall discuss later.

(5) Supports a high utilization of the public cloud clusters dedicated to the smart grid operation; this is possible because the actual data acquisition rates are known and the data analysis done on the public cloud is predictable. The high server utilization has a positive effect on the energy efficiency and, at the same time reduces the costs for the cloud service provider and for the utility companies. At the same time, it allows an effective load balancing among the servers.

We assume that the data collected falls into several categories:

a. Data from smart meters - the measurement data is encrypted with the private key of the utility company and indexed by information reflecting the Id of the public utility, the Id of the power distribution center, and the Id of the customer. The timestamp and possibly other information is sent as plaintext. The data from smart senors is used by an utility company for:

(PM) - Power management. It is done periodically or when a special event triggers the need; it requires only data for a specific time interval and a range of power distribution centers. When such data is needed an application running on the public cloud extracts the data and sends it to the private cloud of the utility where it is decrypted and sent to the power management and analysis

application.

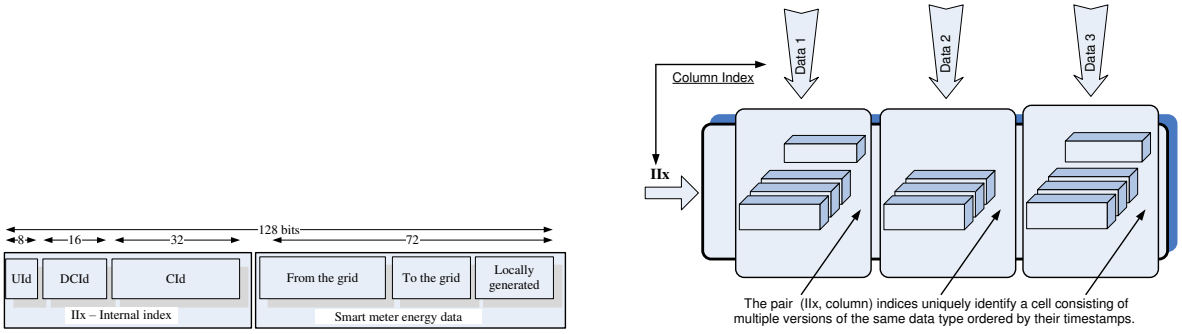
(B) - Billing. It is done at the end of a billing cycle and requires the readings of the smart meters at that time. An applications running on the public cloud extracts the data for all customers and identifies the record corresponding to the end of the billing cycle and sends the data to the private cloud of the utility. The private cloud decrypts the data and sends it to the billing application.

(CEC) - Customer Energy Consumption. It is done at customer request; it provides the profile of energy consumption. An application on the public cloud extracts the data for a given customer and for the limited time interval, e.g., one or more days, and sends it to the private cloud. The private cloud decrypts the data and sends it to the energy profiling application.

Figure 3.1(a) shows an encrypted record produced by a smart meter. The 128 bits include:

1. The UtilityId (UId) - an integer in the range $[0 - 255]$ (8 bits).
2. The DistributionCenterId (DCId)- an integer in the range $[0 - 32,767]$ (16 bits).
3. The CustomerId (CId) - an integer in the range $[0 - 4,294,967,295]$ (32 bits).
4. Smart meter data (72 bits)
   (a) Energy from the grid - an integer in the range $[0 - 4,294,967,295]$ (32 bits).
   (b) Energy supplied to the grid - an integer in the range $[0 - 1,048,576]$ (20 bits).
   (c) Energy generated locally - an integer in the range $[0 - 1,048,576]$ (20 bits).

IIx, is the encrypted Internal Index of a customer, a concatenation of UId, DCId, and CId. This construction guarantees the global uniqueness of the IIx and a strict ordering based on the UId, DCId, and CId. This strict ordering facilitates the data retrieval as we shall see next. When the



(a) (b)

Figure 3.1: (a) An encrypted record for the data provided by a smart sensor consists of 128 bits and includes the internal index and the data from the smart meter. The internal index, IIx, concatenates UId, DCId, and CId. (b) The storage for smart meter data; multiple data fields provide data regarding the energy consumed from the grid, the energy pumped into the grid, and energy generated locally.

public cloud smart meeter application receives a data record from a smart meter it uses the leftmost 56 bits as an index and stores the smart meter data it in a data structure similar to the one in Figure 3.1(b). When a (PM), (B) or (CDC) application running on the private cloud needs data from the public cloud for it provides a range of Internal Indexes (IIx) and a range of timestamps. The basic organization of the data collected and stored on the public cloud resembles the Bigtable [5] the distributed storage system from Google. For example, Figure 3.1(b) shows the organization of the data collected from the smart meters [2].

b. Data from utility sensors - provide utility-sensitive data regarding the energy produced by the power generators, the "green" energy producers scattered throughout the power distribution network of each utility, the backbone of the power grid, and the power transport lines connecting customers with distribution centers. Such data is periodically analyzed by the utility company to determine the cost of a KWh in different regions, to anticipate power shortage or power surplus and to trade power with companies. The measurement data is encrypted with the private key of

the utility company and indexed by information reflecting the Id of the public utility and the Id of the sensor. The data for a specific group of sensors and a given time interval is collected by an application running on the public cloud and it is send to the private cloud of the utility company.

c. Data required for grid management is reported by strategically placed sensors the measurement data and the identity of the sensor are sent as plaintext and is processed on the public cloud.

3.6 Experimental results

To evaluate the performance of OPE we created a benchmark running on a public cloud. We wish to compare the response time of the OP-encrypted database with the one when the database contains plaintext records. We also want to study the scalability of the OPE algorithm.

Our experiments are carried out on the Amazon Cloud and we use Amazon Web Services (AWS) [61]. An EC2 instance runs a `MySQL` database server which accesses two databases, one with plaintext data and one with OPE-encrypted data,both databases contain 5×10^5 records. The EC2 instance runs on a medium server with a 32-bit architecture, 5 ECU (1 ECU is the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor) and 1.7 GB of RAM.

We use MySQL 5.5.34 and add an OPE module to MySQL-proxy 0.8.3 to translate queries to encrypted query and to retrieve response in client side. `MySQL Proxy` is an open source software that can monitor, analyze, or transform the communication between the clients and the server(s); its flexibility allows for a wide variety of uses, including adding security modules, load balancing, failover, query analyze, and query filtering and modification.

The response time includes the communication time from client to server and back, the encryption time at the client side, the search time on the server, and the decryption time of the response at the

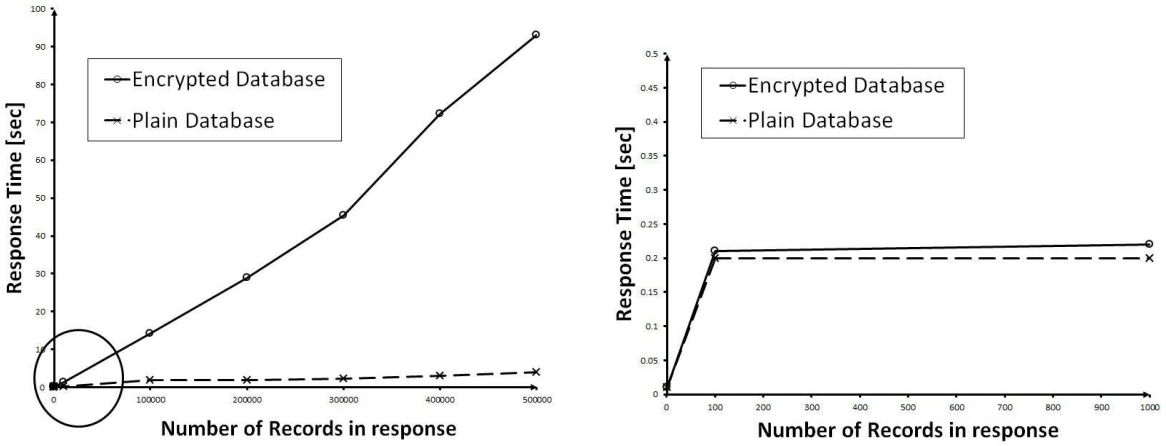
client side. We expect the response time for the OP-encrypted database to increase as the number of records in a response increases due to the increased decryption time. The communication time between the client running on a local PC and the AWS varies. To have a more accurate comparison between the two scenarios we repeat each query and compute the average response time.

To study the scalability of the OPE algorithm we compare the response time to queries to the two databases, the OP-encrypted and plaintext one. The number of records in a query response are: $1, 2, 3, 4$ and 5×10^5. Each query is sent $1,000$ times and the average response time is determined.

To accurately reflect the search time we disable the query cache of the `MySQL`; otherwise the database server will automatically save the result of a query and the next query will not involve a database search. We performed 40 different tests with different configuration with plain and encrypted databases. The response time for all 40 tests are shown in Figure 3.2 (a). As expected, we see a rather drastic increase in the response time as the number of records in the response increases due to the overhead of decrypting the response.

According to [62] the number of records in the response to a query is a critical parameter for production database systems. For example, in more than 92 % of queries the response contains $0 - 100$ records while less than 1% of queries response contains more than $10,000$ records. Figure 3.2 (b) shows a close-up of the region of interest, when the response to a query contains at most $1,000$ records.

Figure 3.3 shows the ratio of response time for plainttext versus an OP-encrypted database function of the number of records in the response to a query. It shows that OP-encrypted databases perform well in the most common cases.



(a) (b)

Figure 3.2: (a) A comparison of the response time in milliseconds for the OP-encrypted database with plain database. The number of records in the response to a query is: $1, 2, 3, 4$ and 5×10^5. (b) A close-up of the region of interest, when the response to a query contains at most $1,000$ records.

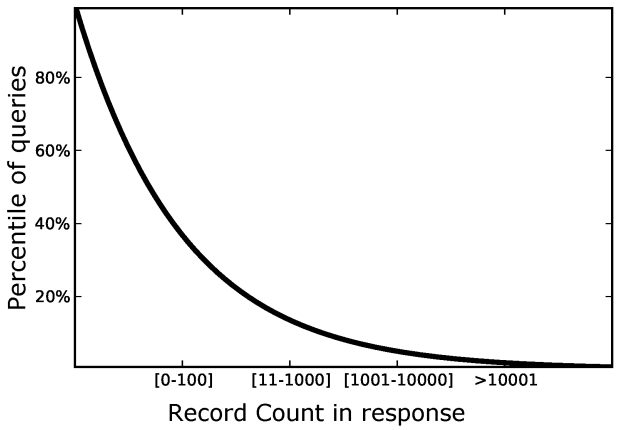


Figure 3.3: Percentages of queries function of the number of records in the response.

3.7 Conclusion

There is no general solution for secure processing of "big data" on a public cloud are feasible at this time, we can identify a class of applications when we can support secure data processing. These are applications requiring cooperation of multiple organizations; each organization shares some data with several other organizations, yet has strict security requirements for its own private data. Application in healthcare, transportation, finance, government, and other areas fit this profile. Such applications could benefit from a hybrid cloud environment; they could store and process massive amounts of data on the public cloud and process private information on a private cloud.

The method proposed in this work can be applied to cases when most of the time only a relatively small range of encrypted data has to shipped back to a trusted system, in our case the private cloud, for processing. Our evaluation carried out on a relatively large database proves that OPE can support a range of operations on the encrypted data such as: comparison, *MIN(), MAX(), Group by, Order by* and *Join*. The overhead for using a OP-encrypted database is only of $10-15\%$ higher than that of using a plaintext one for the common database usage. In our experiments we use a traditional data base, though most "big data" applications use NoSql databases.

We implemented an Order Preserving Encryption algorithm and applied OPE to an application involving a smart power. In this case a vast array of smart sensors generate a very large amount of data which is stored on the public cloud. Sensitive data from individual records are then sent and processed on the private cloud of each utility company. This scheme is only feasible when the amount of sensitive data is relatively small.

CHAPTER 4: SEARCHABLE SECURITY SCHEME FOR NoSQL CLOUD DATABASES

While many schemes have been proposed to search encrypted relational databases, less attention has been paid to NoSQL databases. In this research we report on the design and the implementation of a security scheme called "SecureNoSQL" for searching encrypted cloud NoSQL databases. Our solution is one of the first efforts covering not only data confidentiality, but also the integrity of the datasets residing on a cloud server. In our system a secure proxy carries out the required transformations and the cloud server is not modified. The construction is applicable to all NoSQL data models and, in our experiments, we present its application to a *Document-store* data model. The contributions of this study include: (1) a descriptive language based on a subset of *JSON* notations; (2) a tool to create and parse a *security plan* consisting of cryptographic modules, data elements, and mappings of cryptographic modules to the data fields; and (3) a query and data validation mechanism based on the *security plan*.

4.1 Motivation

The primary motivation of search on encrypted data is originated from the fact that in spit of growing popularity of cloud NoSQL database services. Few researchers have addressed the fundamental issue of data security. Previous work has mainly focused on secure search on encrypted relational databases. Our research provides considerable insight into security of NoSQL database services.

Data is the most valuable asset for the most data owners, even legal restriction apply to outsource customers data off-site without proper privacy protection mechanism. We consider this issue and try to build secure DBaaS service on top of a public cloud settings where, the cloud service itself

is not fully trusted. Our work is one step forward to protect processing of sensitive information against reasonably internal or external threats.

4.2 System organization

We restrict our discussion to query processing particularly over encrypted *NoSQL* databases. A secure proxy called "SecureNoSQL" for accessing cloud remote servers and applying efficient cryptographic primitives for query, response and data encryption/decryption is introduced. We also designed a descriptive language using *JSON*[1] notation which enables its users to generate a *security plan*. The security plan has four sections which elaborately introduce the data elements, cryptographic modules and the mappings between them. The main contributions of this research are:

- A JSON-based language for users to: (i) create a security plan for the database; (ii) describe the security parameters; and (iii) assign proper cryptographic primitives to the data elements.
- A multi-key, multi-level security mechanism for policy enforcement. This feature is essential because the encryption key is subject to more frequent changes than the crypto-module. Furthermore, keys are assigned for a single data element, while encryption algorithms could be applied for several data elements with several keys. This separation allows a more efficient enforcement of security policy and of key management.
- An effective validation process for the security plan. This validation process enables users to initially evaluate all requests locally, rather than forwarding large numbers of fallacious

[1] *JSON* (JavaScript Object Notation), is a lightweight text-based syntax for storing and exchanging data objects consisting of key-value pairs. It is used primarily to transmit data between a server and a web application. JSON's popularity is due to the fact that it is self-describing and easy to understand by human and machine. For more information, visit: http://www.json.org

key-value pairs to a cloud server. It also limits the cloud server workload and reduces the response time latency.

- Support for a comprehensive, flexible protection. The solution is open-ended; users can add new customized cryptographic modules simply by using the designed descriptive language.
- A balanced system with a security level-proportional overhead; the overhead is proportional to the desired level of security.
- A secure proxy which translates queries to run over encrypted data on the remote cloud server with respect to semantics of queries. The cloud database server is not modified and treats encrypted documents in the same way as a plaintext database. Properties of the distributed database such as replication hold for encrypted data.
- Support for cloud data integrity and protection against an insider attack.

The organization of the system is presented in Section 4.2 and structure of the security plan and the notation of the descriptive language for generation of security plan is discussed in Section 4.3. Then the mechanism of query processing is investigated in Section 4.4. Finally, in Section 4.6 we report on, measurements of the database response time to different types of queries and on the encryption and decryption time for OPE encryptions with output lengths of $64, 128, 256, 512$ and 1024-bit.

This section introduces a framework to incorporate data confidentiality and information leakage prevention algorithms. SecureNoSQL leverages secure query processing for web and mobile applications using DBaaS. Two different system organizations can address our design objectives. The first is suitable when all database users belong to the same organization. Then the proxy runs on a trusted server behind a firewall and the communication between clients and the proxy is secure.

When the clients access the cloud using the Internet the second organization is advisable. In this case, either the client software includes a copy of the proxy and only encrypted data is transmitted over public communication lines, or the *Secure Sockets Layer(SSL)* protocol is used to establish a secure connection to the proxy. Figure 4.1 illustrates the high-level architecture of *SecureNoSQL* as a secure proxy between user's applications and cloud NoSQL database server. The system we report on was designed with several objectives in mind:

- Support multi-user access to an encrypted *NoSQL* database. Enforce confidentiality, privacy of transactions and data integrity.
- Hide from the end-users the complexity of the security mechanisms; the database access should be transparent and the user's access should be the same as for an unencrypted database.
- Avoid transmission of unencrypted data over public communication lines.
- Do not require any modification of the *NoSQL* database management system.
- Create an open-ended system; allow the inclusion of cryptographic modules best suited for an application.

These objectives led us to design a system where a proxy mitigates the client access to the cloud remote server running an unmodified *NoSQL* database processing system. In this system the processing of a query involves three phases:

1. Client-side query encoding in *JSON* format carried out by the client software;
2. Query encryption and decryption done by a trusted proxy; and
3. Server-side query processing performed by an unmodified *NoSQL* database server.

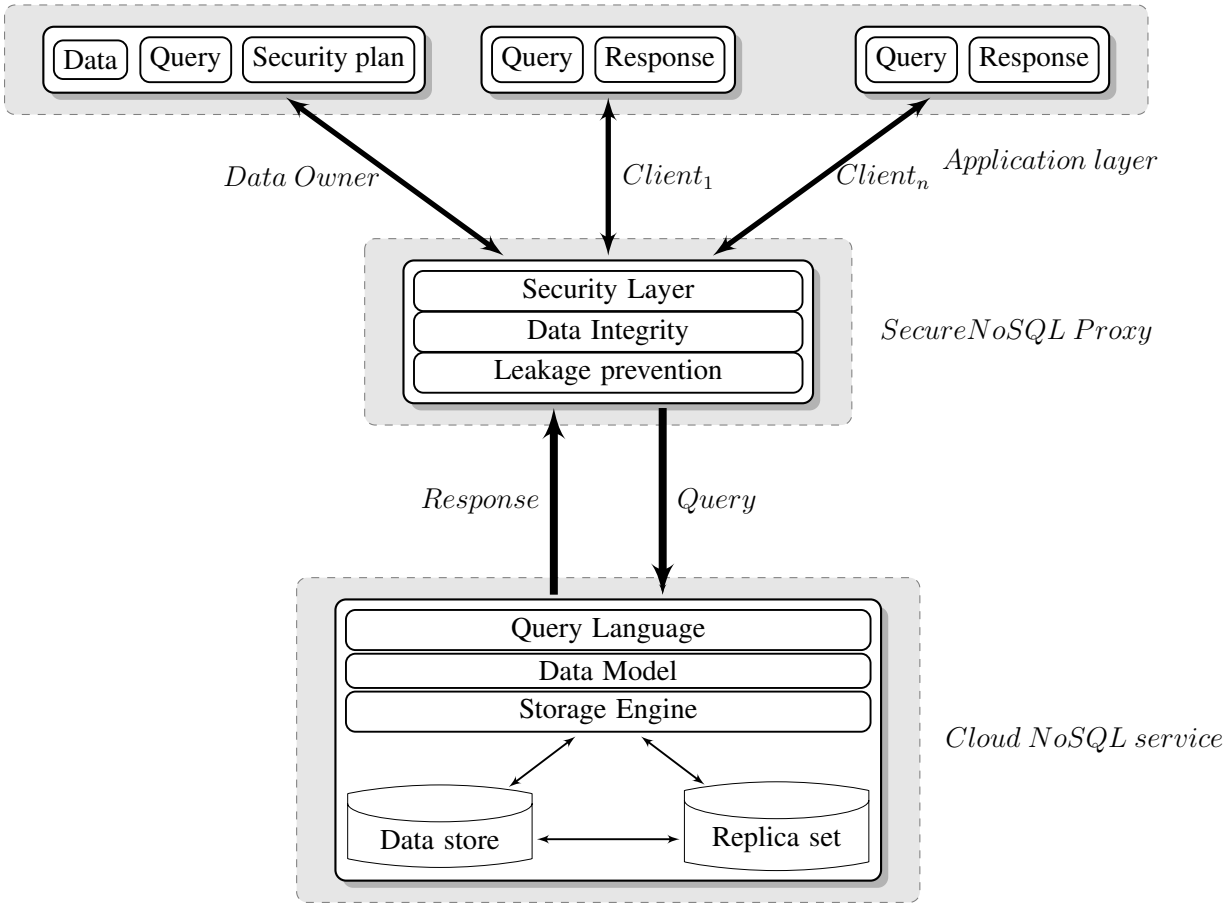


Figure 4.1: The organization of the *SecureNoSQL*.

SecureNoSQL is based on general principles of NoSQL database products. We introduce a new concept, the *security plan*, materialized as a JSON description of data elements, metadata and parameter configuration of cryptosystems. A descriptive language is introduced to generate and parse the security plan automatically. JSON, a dominant format in NoSQL databases, is selected to express the designed security plan. We used a subset of JSON notation readable by human and machine.

Document databases, such as MongoDB, store documents inside the collection by JSON repre-

sentation in a similar way as tables and records in relational database systems. A query and the corresponding response are also represented in the JSON format; therefore, the governing format in document database is JSON. BSON, a binary extension of JSON, is used by document-oriented databases for efficient encoding/decoding.

JSON query model is a functional, declarative notation, designed especially for working with large volumes of structured, semi-structured and unstructured JSON documents. The data owner develops the security plan that outlines and maps out the determined crypto-primitive with specific parameters to a particular data element.

4.3 Descriptive language for security plan

We introduce a new construction henceforth named *Security plan* to bind appropriate cryptosystems to the data elements of a NoSQL database. The notations of JavaScript are employed to create a descriptive language for expressing parameters of cryptosystems, defining data elements and mapping between these two. More details on security plan and descriptive language will be given in this section.

Database security plan. The security plan identifies the mechanism to maintain the security of the data elements in a database. It also determines how to interpret queries issued by a specific application. The security plan has four sections, see Figure 4.2, describing the security rules for the data elements and for meta-data such as the field-name (Key) and the collection name. These sections are the building blocks of the security plan showing how the rules are enforced. The sections and their roles are:

1. *Collection:* includes the name of a collection and a reference to the encryption module used to encrypt the name of the collection and the name of fields (metadata).

2. *Cryptographic modules:* lists the cryptographic modules for encrypting the fields of the database entries in the query.

3. *Data elements:* lists the properties of each data field including the data type; the data type determines the cryptographic modules to be applied to each field.

4. *Mapping cryptographic modules to the fields:* assigns the cryptographic modules to data fields; proxy uses this information to encrypt and decrypt the data elements.

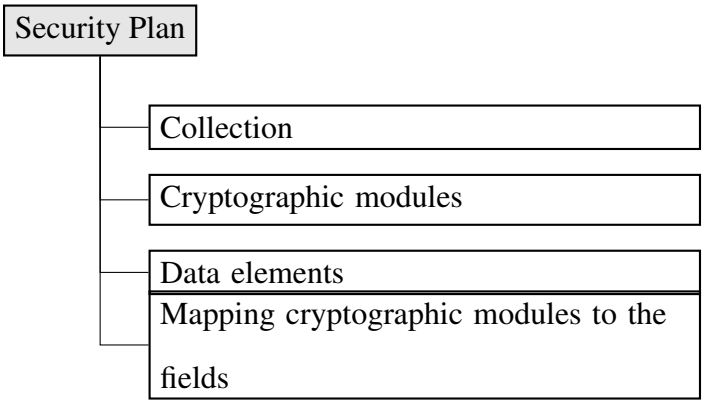
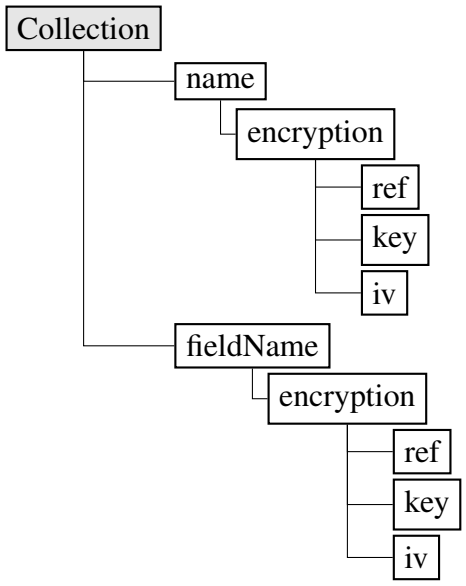


Figure 4.2: The high level structure of the security plan.

Collection. A collection is defined as a group of *NoSQL* documents, the equivalent of relational database table, see Figure 4.3. The name of the collection must be encrypted. The listing 4.3b illustrates how to secure a sample collection using the description language.



(a)

```
{
"$collection" : {
"name" : "Personnel",
"$encryption":{"$ref":"/AES-DET",
"key":"02468acebdf135790369cf258be147ad",
"iv":"2468ace0" }
},
"$fieldName":   {
"$encryption": {
"$ref":"/AES-DET",
"key":"0123456789abcdef0123456789abcdef",
"iv":"ffeeddcc"}
}
}
```

(b)

Figure 4.3: The structure of a collection: (a) The chart outlines the structure of a collection containing the name of collection and name of all attributes which are considered as a meta-data, and should be protected with proper cryptographic module. (b) The description of a collection and security parameters in designed JSON based language. In this specific case the Advanced Encryption Standard in deterministic (AES-DET) mode with a 128-bit key and an *initialization vector* (IV) is assigned to encrypt the name of the collection and the fields name.

The key-value pairs (KVP) are the primary data model for a *NoSQL* database. The *key* is used as an index to access the associated value of the data pointed by the reference *ref*. The *initialization vector* (IV) is a fixed-size, random input to the cryptographic module *encryption*. Additionally, a collection exists within a single database. Documents within a collection can have different fields. Typically, all documents in a collection are related with one another.

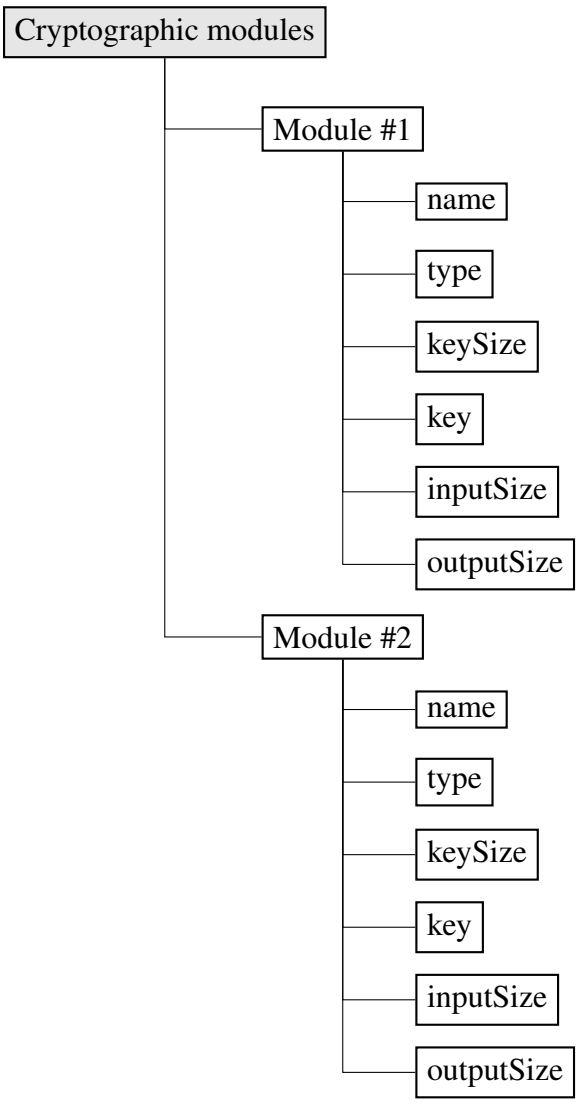
Cryptographic modules. The choice of a particular cryptosystem depends on the security policy of application. Multiple criteria for algorithm selection include: (i) the security against theoretical attacks; (ii) the cost of implementation; (iii) the performance; and (iv) whether the encryption and decryption can be parallelized. Other factors involved in the selection of an algorithm are the memory requirements and the integration in the overall system design.

The *Cryptographic modules* introduce all encryption modules and their parameters such as key, key-size, initialization vector and output-size. The structure of this section is shown in Figure 4.4a complemented by the listing in Figure 4.4b presenting the second section of security plan for the previous example.

Our proof of concept uses the parametric Order Preserving Encryption (OPE) and the Advanced Encryption Standard (AES) modules. The system is open-ended; users can add the cryptosystems best suited to the security requirements of their application. In our design the definitions of the cryptographic modules and of the pairs, encryption key and initialization value, are separated following the so-called *key separation principle* [63]. This security practice is based on the observations that users have long- and short-term security policies. The cryptographic modules are less likely to change while the key and the initialization value change frequently.

The data elements. The third section of security plan, the data elements and their properties are covered. Figure 4.5 presents the structure and description of *Data element* section of *Security plan*. The listing displayed in Figure 4.5b displays data elements and its *JSON* description for previous example. To ensure the desired level of security the security plan should provide the description of all sensitive data elements of database in third section of security plan.

Mapping cryptographic modules to the fields. The last section of security plan is for binding cryptographic modules to the sensitive data fields. Figure 4.6 and the listing presented in Figure 4.6b presents a mapping for a sample document in our designed descriptive JSON-based language.

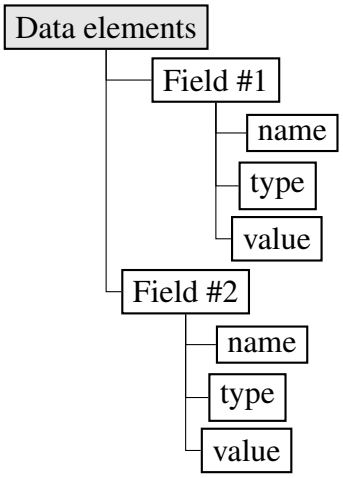


(a)

```
{
"OPE" : {
"properties" : {
"encryptionMethod" : {
"type" : "string",
"enum" : [ "OPE" ]  },
"keySize" : {
"type" : "integer",
"minimum" : 64,
"maximum" : 4096,
"default" : 128 },
"key" : {
"type" : "string",
"pattern" : "^([0-9a-fA-F]{2})+$" },
"inputSize" :  {
"type" : "integer",
"minimum" : 8,
"maximum" : 128,
"default" : 32 },
"outputSize" :   {
"type" : "integer",
"minimum" : 64,
"default" : 128 } },
"required" : [  "key", "
   encryptionMethod" ],
"additionalProperties" : false
}
}
```

(b)

Figure 4.4: The structure and function of Cryptographic modules: (a) The *Security Plan* with the second section, the cryptographic module, expanded. The attributes included for each module are: name, type, key size, key, input and output size. (b) The OPE encryption including the cryptosystems and their attributes. The proxy applies these modules using the key-value pairs (KVP).



(a)

```
{
"id":{ "type":"integer" },
"name":{ "type":"string"  },
"salary"{ "type":"integer" },
"balance"{ "type":"integer" },
"ccn"{ "type":"integer" },
"ssn"{ "type":"integer" },
"email":{ "type":"string"  }
"required":["id","name","email","
   salary"]
}
```

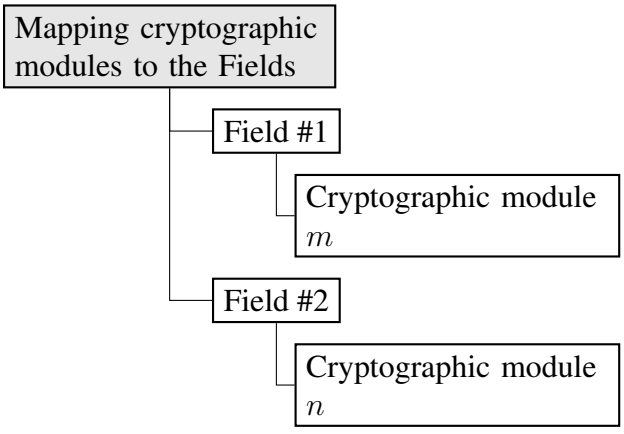
(b)

Figure 4.5: Structure and description of *Data element*: (a) The chart outlines the structure of *Data elements* containing attributes of data elements such as name, type and value for of collection and name and then introduces security parameters for each data element. (b) The data element section of a sample database which is represented in designed notation. A data item has 7 fields: id, name, salary, balance, ccn, ssn, and email. The id, name, email, and salary are required fields.

The method presented in this research can be easily extended to the other *NoSQL* data models discussed in Section 2.3. Figure 4.7 shows how this extension from the key-value pair to the document store model can be carried out.

Query and data validation The proxy validates the data and the query as a JSON-formatted input with the reference security plan. Then the proxy enforces the crypto-primitives, and generates new query following the NoSQL query semantics. During this process the proxy applies to each field the cryptographic modules. Finally, the proxy forwards the newly encrypted query/data to the *NoSQL* database server. Figure 4.8 depicts the schema validation process.

For better illustration, consider listings depicted in Figure 4.9a as an input data; after running validation process the output is generated (see Figure 4.9b). The output of validation process is a single file which contains descriptive information for data and meta-data in designed format and



(a)

```
{
"id":{ "$ref": "#/ope128" },
"name":{ "$ref": "#/AES-DET" },
"email":{ "$ref": "#/AES-DET" },
"salary":{ "$ref": "#/ope256" },
"ssn":{ "$ref": "#/ope256" },
"ccn":{ "$ref": "#/ope256" },
"balance":{ "$ref": "#/ope256" }
}
```

(b)

Figure 4.6: The structure and description of *Mapping cryptographic modules to the Data element*: (a) *Security plan* with the fourth section expanded. This section establishes a correspondence between the data fields and the cryptographic modules used to encrypt and decrypt the data fields. (b) The mapping section of the schema for a sample database with 7 fields. For example, the *id* and the *name* will be encrypted with *OPE* 128 bit and *AES-DET*, respectively.

ready to execute on the *SecureNoSQL*.

The output of validation process is a single file containing descriptive information for data and meta-data expressed in the required format and ready to execute. The output of validation process for the example is illustrated in Figure 4.9b. As it was noted earlier, the schema reflects the desired security level expressed by the security plan for the database. Table 4.1 shows the overhead for several parameters and crypto-primitives.

Table 4.1: The overhead of encryption for several encryption schemes.

Database	**Plain**	**OPE64**	**OPE128**	**OPE256**	**OPE512**
Size (MB)	170	430	508	662	1000

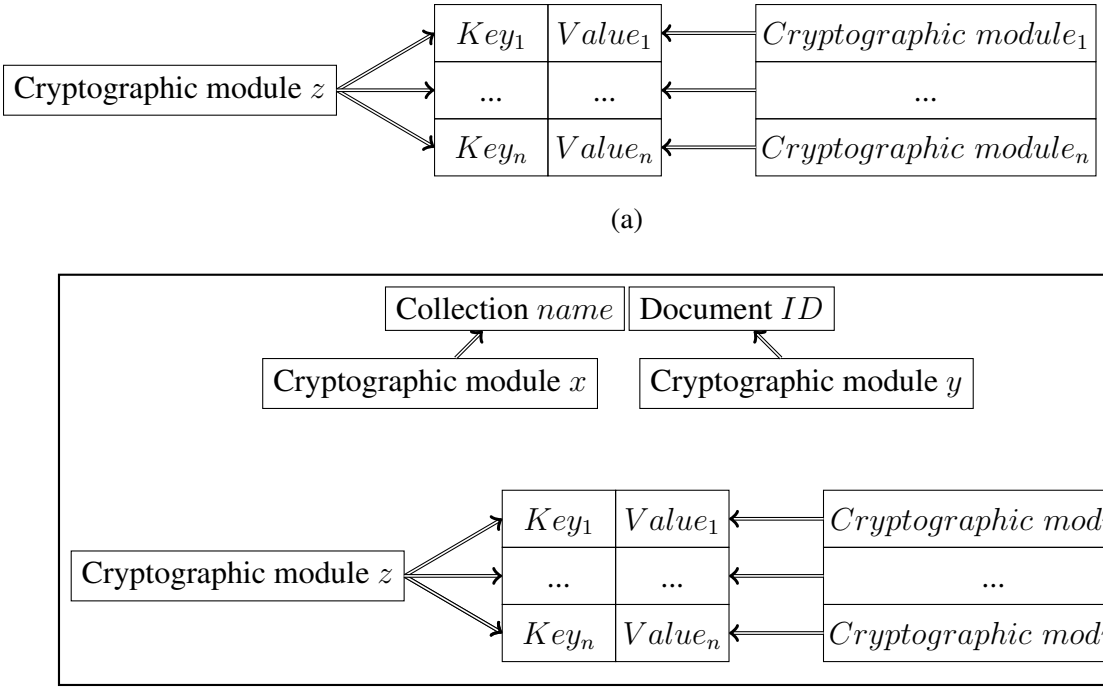


Figure 4.7: *SecureNoSQL* applied to: (a) The key-value data model; $Key_1, \ldots, Key_n$ are all encrypted using the cryptographic module z while the corresponding values, $Value_1, \ldots, Value_n$ are encrypted with cryptographic modules $1, 2, \ldots, n$, respectively. (b) The document store data model; the meta-data such as collection name encrypted as well as attributes with assigned cryptographic modules.

4.4 Processing queries on encrypted data

According to the proposed scheme, in order to process queries over encrypted data the queries should transfer to the encrypted version with respect to *security plan*, and this task is designed to be conduct in the proxy. The *security plan* discussed in Section 4.3, supplies the parameters of the cryptographic modules to be applied for the data elements involved in the query. Figure 4.10 displays the processing and rewriting of a sample query.

For better understanding the query encryption, in table4.2 you can find some sample encrypted

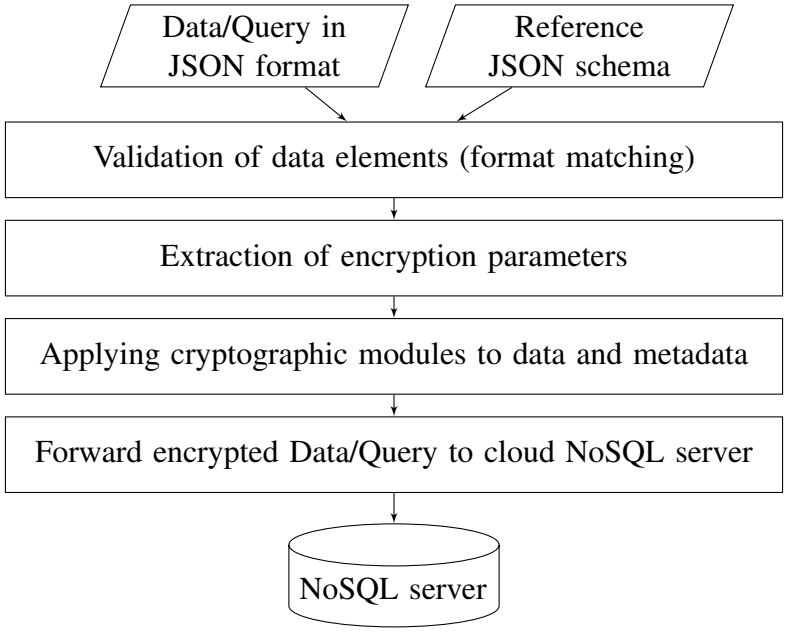


Figure 4.8: The validation process of input data against security plan in the client side.

queries after enforcing *security plan*. As it can be seen, data elements and immediate values are encrypted; however, the output is consistent with NoSQL semantics.

4.5 Integrity verification of data/query/response

Integrity and confidentiality are two critical components of data security. Integrity refers to the consistency of the outsourced data. The proposed integrity verification algorithm in *SecureNoSQL* guarantees the integrity of data/queries (see Algorithm 3 and Figure 4.11). Data owner first applies encryption scheme on the documents, and then calculates Hashed Message Authentication Code (HMAC) for each one of encrypted documents. A hash value of any given document is a fixed-length of 512 bit and data owner concatenates a unique document identifier (ID) with hash value and stores the results in efficient structure like HashTable which has constant looks-up time

Figure 4.9: The security plan for the sample database: (a) The data element section of sample security plan. (b) The output of the *JSON* data validation for the sample database.

$O(\log n)$. Next, data owner transfers the encrypted dataset to the cloud and sends HashTable containing hash values to the proxy. Once the proxy receives the query response from the server, it initiates the verification process to check the authenticity of the documents by recalculating the

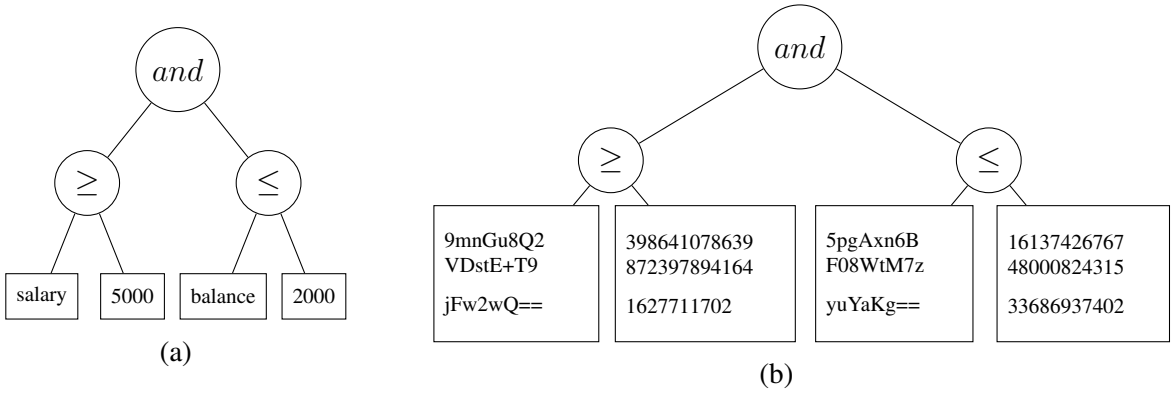


Figure 4.10: The query *db.customers.find({salary:{$gt:5000}, balance:{$lt:2000}})* received from an application. (a) The parsing tree of the query. (b) The defined cryptographic modules are applied to the data elements and the encrypted equivalent is created.

hash values. This process is illustrated in Figure 4.11.

Algorithm 3: Document Integrity Verification Algorithm in the Proxy

Input: Plaintext query q from client application c_i

Output: Are the documents in the response authentic? Yes/No

1 $q_{enc} = Encrypt(q)$;

2 $q_{enc} \xrightarrow{forward}$ *to cloud database server*;

3 $R_{enc} \xleftarrow{receive}$ *from cloud database server*;

4 **repeat**

5 $\quad H_d = HMAC(R_{enc}[i], key)$;

6 $\quad$ **if** *(*$HashTable[user_i] \neq H_d$*)* **then**

7 $\quad\quad$ **return** $false$

8 **until** *(There is a document in* R_{enc}*)*;

9 **return** $true$;

In this configuration the data owners just trust the proxy (SecureNoSQL) and cloud servers are not

Table 4.2: Sample queries and their corresponding encrypted version

	Query	**Encrypted equivalent query**
1	db.customers.find({ssn:936136916})	db[”k/IevnbanDMQHNkb9cRgUg==”].find({”5pgAxn6BF08WtM7zyu YaKg==”:741724054784419080417111188338621437778})
2	db.customers.find({balance:{$gte: 5084610},balance:{$lte:9911843}})	db[”k/IevnbanDMQHNkb9cRgUg==”].find({”3iXpo2l8xZpW7J7TezFdeA==”:{$gte:402982988013604629517872370128473753},”3iXpo21 8xZpW7J7TezFdeA==”{$lte:7855963556987175927802686333694542 31}})
3	db.customers.aggregate([{$group:{_id :null,minBalance:{$min:”$balance”}}}])	db[”k/IevnbanDMQHNkb9cRgUg==”].aggregate([{$group:{_id: null,EncMinBalance:{$min:”$3iXpo2l8xZpW7J7TezFdeA==”}}}])
4	db.customers.aggregate([{$group:{_id: null,maxBalance:{$max:”$balance”}}}])	db[”k/IevnbanDMQHNkb9cRgUg==”].aggregate([{$group:{_id:null ,EncmaxBalance:{$max:”$3iXpo2l8xZpW7J7TezFdeA==” }}}])
5	db.customers.find({$or:[{Salary:{$gt: 516046}},{balance:{$lt:285462}}]})	db[”k/IevnbanDMQHNkb9cRgUg==”].find({ $or: [{ ”9mnGu8Q2V DstE+T9jFw2wQ==”: { $gt: 4099418621678574661319324412988584 9}},{”3iXpo2l8xZpW7J7TezFdeA==”:{$lt:226574304531446346797 91167652174833}}]})

trustworthy. Thus, a result of data integrity verification, all active attacks conducted by internal or external attacker will be detected by the proposed approach. The Message Authentication Code (MAC) is created by using the keyed Hash Message Authentication Code (HMAC) as rephrased in Eq. 4.1.

$$HMAC(K, document) = H((K \oplus okeyPad) \| H((K \oplus ikeyPad) \| document) \tag{4.1}$$

Where:

H represents the hash function

$\oplus$ is the XOR operator

okeyPad is one-block-long outer pad

ikeyPad is one-block-long inner key pad

Algorithm 4 presents the pseudo-code of the HMAC function for a block size of 64 bytes. The computed hash values with correspondent document's unique identifier can be stored in the form of key-value pair in a hash-table thus, allowing the proxy to carry the lookup in constant time during the verification process.

Algorithm 4: Keyed Hash Message Authentication Code (HMAC) generation.

Input: Document: d, user key: k
Output: hash value

if $(length(key) > blocksize)$ **then**
  $key = hash(key)$;
if $(length(key) < blocksize)$ **then**
  $key = key \,||\, [0x00 * (blocksize - length(key))]$;
$okeyPad = [0x5c * (blocksize)] \oplus k$;
$ikeyPad = [0x36 * (blocksize)] \oplus k$;
return $hash(okeyPad||hash(ikeyPad \,||\, d))$;

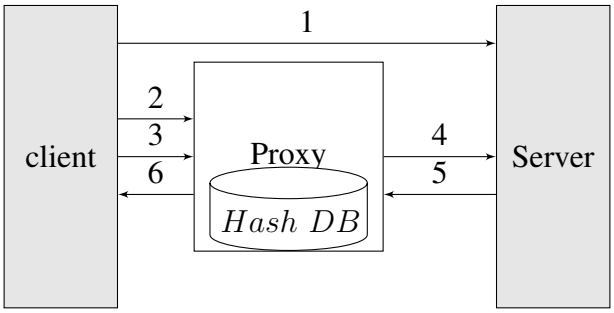


Figure 4.11: Integrity verification of data/query/response: (1) Data owner transfers the encrypted database to the cloud server. (2) Data owner sends the Hash database to proxy. (3) Clients send plain queries to the proxy. (4) The proxy translates queries to the encrypted version, and forwards them to the cloud server. (5) The cloud server returns the query response set. (6) The proxy runs a hash verification process on the query response set, and then based on the result either forwards to the decrypted response or reports integrity violation to the client.

4.6 Results and discussion

The response time of a query to an encrypted *NoSQL* database has several components:

1. the time to encode the query in *JSON* format;
2. the time to encrypt and decrypt the data;
3. the communication time to/from the server;
4. the database response time.

For our experiments we first created a sample database with one million records and then determined the overhead of searching an encrypted database. To do so we measured the database response time for queries when the records were unencrypted versus when records were encrypted. Then, we measured the encryption and the decryption time for different sizes of the ciphertext. We wanted to isolate the different components of the response time dominated by the communication time.

The environment used for testing was set up on the Linux operating system. We chose *MongoDB* [64], classified as a *NoSQL* document store database 3.0.2. The random data generator in *JS, PHP,* and *MySQL* format was generated by using a tools [65] to generate a one million record plaintext data set. Each record had seven different data fields including *name, email, salary*, as shown in Listing 4.9b.

We applied OPE $64, 128, 256$ and 512 bit to numeric data type, and the *AES-DET 128* bit for the string data type of the plaintext data set and generated four encrypted data sets of one million records each. Finally, we uploaded the five datasets and created five *MongoDB* databases, one with the unencrypted data, and four with the encrypted data. Once the *MongoDB* databases were

created we run several types of queries including equality, greater than, less than, greater than or equal to, less than or equal to, and OR logical operations.

The experiments to measure the query time must be carefully designed. To construct average query processing time each experiment has to be carried out repeatedly. We noticed a significant reduction of database management response time after the first execution of a query, a sign that *MongoDB* is optimized and caches the results of the most recent queries. A solution is to disable the cache, or if this is not feasible, to clear the cache before repeating the query. Another important observation is that modern processors have a 64-bit architecture and are optimized for operations on 64-bit integers. For three of the five types of queries, $Q2$ (Range query), $Q3$ (equality), and $Q4$ (logical), database response time is slightly shorter for the encrypted database than for the unencrypted one when the keys are 32-bit integers. A plausible explanation for this is most likely related to the cache management.

The results reported in Table 4.3 and in Figure 4.12 show the database response time for the five *MongoDB* experiments. Each query was carried out 100 times with disabled query cache and the average query response in milliseconds was calculated. We also measured the encryption and the decryption time and the results are reported in Figure 4.13. The measurement process was automated, and it was running under the control of a script which generated the data and reported the processing time.

Our measurements show that the response time of the *NoSQL* database to encrypted data depends on the type of the query. The shortest and longest database response time occur for $Q1$ (comparison) and $Q5$ (aggregated queries), respectively; for these two extremes the time for the unencrypted database was almost double, but the time for encrypted databases increases only by $70 - 80\%$. As expected, the query processing type for a given type of query increases, but only slightly, less than 5% when the key length increases from 64, to 128, 256, and 512 bit.

Table 4.3: The query processing time in milliseconds (ms) for the plaintext and for the ciphertext. 32-bit plaintext integers are encrypted as $64, 128, 256$ and 512-bit integers. The record count gives the number of records retrieved by each one of the five types of queries, $Q1 - Q5$.

Query type	Number of matching record(s)	32-bit plaintext	64-bit ciphertext	128-bit ciphertext	256-bit ciphertext	512-bit ciphertext
Q1: Comparison	461,688	340	310	355	370	380
Q2: Equality	1	340	380	390	400	410
Q3: Range	991,225	370	350	360	380	400
Q4: Logical	551380	500	540	550	555	560
Q5: Aggregation	1	600	660	670	680	690

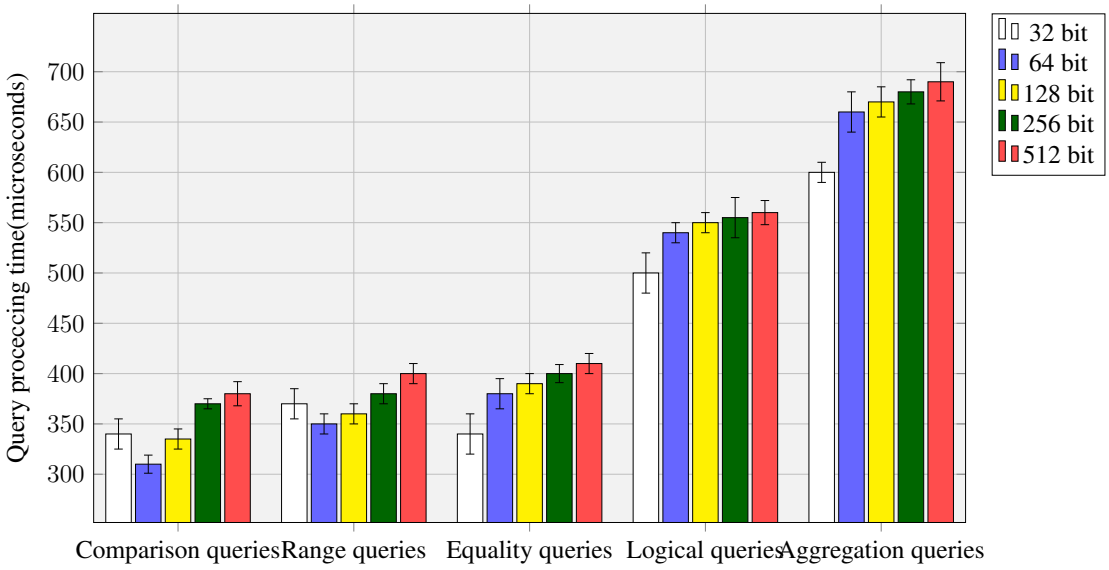


Figure 4.12: The query processing time in milliseconds (ms) for the unencrypted database and for the encrypted databases when the 32-bit keys are encrypted as $64, 128, 256$ and 512-bit integers.

The OPE encryption time increases significantly with the size of the encryption space; it increases almost tenfold when the size of the encrypted output increases from 64-bit to 1024-bit and it is about 10 ms for 256-bit. The decryption time is considerably smaller; it increases only slightly from 0.11 ms to 0.17 when the size of the encrypted key increases from 64-bit to 1024 bit.

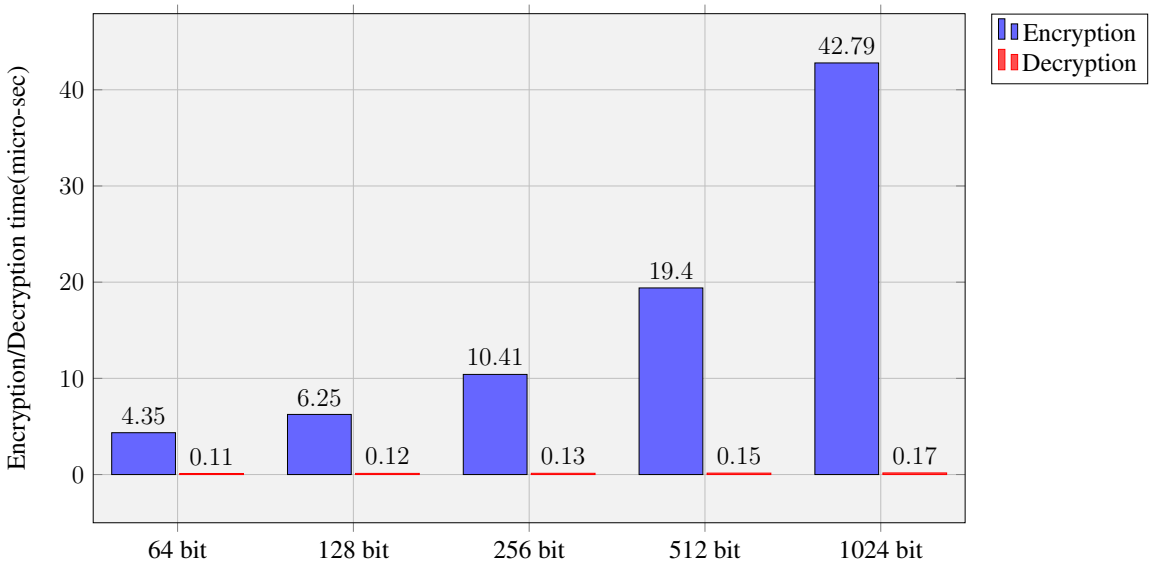


Figure 4.13: Execution time of the OPE module when the key is encrypted as 64, 256, 512, and 1024 bit

Secure proxy is an important element for the proposed architecture; therefore, the potential attacks that could affect the proxy, also should be taken into consideration. In general, two major possible attacks on proxy are Denial of Service (DoS) and unauthorized access. In DoS attack, the attacker sends so many network traffic to the proxy, that the system is not capable of process within the expected time frame. Successful DoS attacks can turn the proxy to a bottleneck of the system. In unauthorized access attacks, attackers use a proxy to mask their connections while attacking the different targets.

Several solutions exist for improving the security of proxy against DoS attacks and reducing the consecutive impacts, including blocking the undesired packets or using multiple proxies with load balancers. Moreover, for prevention of unauthorized access attacks, it is required to use best fit authorization to access the proxy. User authentication based on group membership with different authorizations are the best practical solutions.

4.7 Conclusions and future work

Though the OPE encryption scheme has known security vulnerabilities it can be very useful for *NoSQL* database query processing for the data models discussed in Section 2.3. While the *key* is encrypted using OPE, the other fields of a record can be encrypted using strong encryption, thus reducing the vulnerability of the data attacks. Strong encryption of the value fields could increase the encryption time but will have little effect on the decryption time.

An important observation is an increase in size of co-domain of the OPE mapping function from 2^{64} to $2^{128}, 2^{256}$, and to 2^{512} results in an increase of database response time up to 5%, except for $Q3$-type queries when the increase is significant. The penalty for using encrypted, rather than plain NoSQL databases such as *MongoDB* is less than 5% for $Q2, Q4$, and $Q5$ which is relatively small. Moreover, the overall query response time is dominated by the communication time.

The secure proxy is a critical component of the system. The proxy is multi-threaded and its cache management is non-trivial. The management of the security attributes is rather involved. On the other hand, a proxy integrated in the client-side software can be light-weight and considerably simpler. We are currently implementing the two versions of proxy. Experimental results for multiple large datasets with up to one million documents show that SecureNoSQL is rather efficient. Our approach can be extended to a multi-proxy structure for Big Data applications. We are now implementing a sophisticated mechanism for maintaining consistency of hash values database in the proxies datasets based on the PAXOS algorithm [66, 2].

CHAPTER 5: LEAKAGE PREVENTION

Web and mobile applications take advantage of the Database as a Service (DBaaS) to access cloud databases. The popularity of DBaaS amplifies the security risks for cloud stored data and exposes an information leakage channel that can be exploited by cross-referencing cloud-hosted databases, even if encrypted. Cloud databases include data with various degrees of sensitivity. The sensitivity analysis introduced in this research identifies the most valuable information that must be protected for limiting the effects of information leakage. Then the proposed method is extended to leakage analysis in the DBaaS warehouse by leveraging the Approximate Query Processing (AQP) technique. This approach, compromises between orders of magnitude processing time improvement to the limited inaccuracy in response. We report on experiments conducted to assess the effectiveness of AQP for preventing information leakage.

5.1 Motivation

Information leakage is the inadvertent disclosure of sensitive information. Information leakage in a cloud environment enables an attacker to infer sensitive information either through multiple database searches, or cross-correlations among databases. The threat posed by information leakage is amplified as public cloud data warehouses maintain numerous databases from many organizations.

The ability to link individual items of information from different sources poses a new type of threat to cloud users. An attacker could link low-risk items of information to extract sensitive information. For example, cross-correlation of information from a metro card connected to user's debit card for auto-refill could reveal sensitive financial information. Moreover, personal information

from the financial record could be linked to the health record of an individual.

Nowadays, many enterprises use DBaaS offered by major Cloud Service Providers (CSPs) to access an increasingly larger number of cloud hosted databases [2]. A 67% annual growth rate is predicted for DBaaS by 2019. CSPs guarantee availability and scalability, but the data confidentiality poses significant challenges in the face of new threats. Some of the threats emanate from insider attackers who have the ability to correlate information from multiple cloud databases.

Data encryption provides data and query privacy but, contrary to the common belief, encrypted cloud data and encrypted queries are vulnerable to information leakage. Encryption does not protect all information about the encrypted data. The leakage can be exploited by external, as well as insider agents. A malicious insider can infer sensitive information through cross-referencing databases in the data warehouse. Moreover, the collection name, the attribute name (or table, field name in RDBMS), the number of attributes involved in a query, and the query length often reveal sensitive information about the encrypted data.

A motivating sequence of events illustrate the effects of data correlation and, implicitly of information leakage. In August 2006, AOL, a global on-line mass media corporation, released search logs of over 650 000 users for research purposes. The data included searches conducted over a period of three months with user names changed to random ID numbers. An analysis of the searches conducted by a user, made him/her uniquely identifiable. Correlating data released by AOL with publicly available datasets revealed additional private information about AOL users.

The discussion in this report is restricted to NoSQL databases with a flexible schema. A NoSQL database is a collection of documents $D = d_1, \ldots, d_n$. A document is a set of key-value pairs $key_i, value_i$, each representing an attribute of an object. Enforcing partial security mechanism which covers only a subset of attributes, may not provide comprehensive protection as protected information could be inferred using low-risk datasets hosted by the same cloud.

The number of documents in cloud databases limits the ability to analyze in real-time the dangers posed by information leakage. The alternative we propose is based on random sampling and error estimation regarding information leakage. This solution dramatically cuts the analysis time, especially, whenever the sample size is small enough to fit in the main memory of the database servers. As expected, approximate measurements based on data sampling exhibit different levels of errors. Sampling-based Approximate Query Processing (AQP) provides bounds on the error caused by sampling [37, 67].

We propose insertion of *disinformation documents* into the collection. A secure proxy like the one in [30] mediates the interaction between clients and the DBaaS server. The proxy intercepts the client queries, transfers them to the encrypted queries and passes them to the cloud DBaaS server which responds with a combination of valid and forged documents.

Eventually, the proxy decrypts the query response and filters out the disinformation documents and forwards the desired document to the user's application. The Selective Disinformation Document Padding (SDDP) is proposed to avoid the overhead of disinformation document padding in the dataset. We also investigate an Encrypted Data Indexing features for minimizing the query processing time of augmented ciphertext datasets. The contributions of this research are:

1. A method to quantify exact volume of information leakage because of explicit and implicit attribute cross-correlations in a cloud data warehouse.

2. A selective disinformation document padding method to reduce information leakage with limited overhead.

3. A scalable leakage assessment and parameter extraction algorithm for very large databases based on approximate query processing.

5.2 System Models and Assumptions

The threat model. This study is focused on system end-to-end security based on an adversarial perspective. An adversarial threat analysis starts with thinking like an attacker and continues to prepare the corresponding countermeasure. Two classes of threats, external and internal attacks are identified by the model. External attacks can be conducted after obtaining unauthorized access to data or by using tools to monitor the communication between the clients and the cloud servers. External attackers face a more complex task since they need to bypass firewalls, intrusion detection systems, and other protection mechanisms.

Insider attacks can be conducted by the employees and the contractors of large data centers with access to the software, the hardware, and the data. A malicious insider could leak highly sensitive documents or use it for nefarious activities. There is also the risk of an intruder gaining the same level of access using the credentials of a legitimate employee.

The cloud database service cross-correlation model. Consider W_{DBaaS} the set of collections managed by a DBaaS as: $\mathcal{W} = \{\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_m\}$. Each collection $\mathcal{C}_i, \forall i \in [1, m]$ consists of an arbitrary number q of documents: $\mathcal{C}_i : \{d_1, \ldots, d_q\}$. A document $d_j = \{A_1, \ldots, A_l\}$ includes an arbitrary number l of $\langle key, value \rangle$ pairs known as attributes.

Consider the scenario where a target entity $\mathbb{T}$ has several documents in the multiple data collections hosted by the same DBaaS. Assume a cloud malicious insider knows only one document d_1 about $\mathbb{T}$. In this setting, the *reference document* $\mathbb{R}$ is defined as a document that contains all attributes of entity $\mathbb{T}$. Any cross-correlation relationships can be exploited to extract more information about target $\mathbb{T}$ by cloud malicious insider. Set $\mathcal{L}$ contains all attributes that are extracted as a result of cross-correlation function call. Although computationally it is very challenging to obtain complete information of $\mathbb{T}$, the attacker tries to get as close as possible to the reference document. The

cross-correlation function, $\Psi_{\mathbb{T}}(\mathcal{C}_i, \mathcal{C}_j)$ and the leaked attributes set $\mathcal{L}$ about an entity $\mathbb{T}$ are defined in Equation 5.1, also in Section 5.5 the concept of cross-correlation function is implemented by NoSQL query language.

$$\begin{aligned} &\Psi_{\mathbb{T}}(\mathcal{C}_i, \mathcal{C}_j) : \\ &\forall\, d \in \mathcal{C}_i \ \wedge\ \forall\, d' \in \mathcal{C}_j \\ &if(\mu(d, d') == True) \implies \mathcal{L} = \{Att \mid \forall Att \in d' \wedge Att \notin d\} \end{aligned} \tag{5.1}$$

The *feasibility* function μ in Equation 5.1 determines if a given pair of documents are belongs to the same entity, and can be merged according the type of shared attribute. The μ function is defined in Equation 5.2. Note that Att_i, Att_j are identifier type attributes.

$$\mu(d, d') : \begin{cases} True \ \ iff \ \exists Att_i \in d \wedge \exists Att_j \in d' \mid \\ [(Att_i.key == Att_j.key) \wedge (Att_i.value == Att_j.value)] \\ False \qquad Otherwise \end{cases} \tag{5.2}$$

An insider can bypass internal protection mechanism and pose serious risks to data *confidentiality*, *integrity* and *inference* violations. The list of possible insider attacker actions are denoted as $A = \{C, I, \Psi\}$ respectively.

The risk factors and the corresponding solutions as well as the advantages and disadvantages of the proposed solution are summarized in Figure5.1.

An insider has read/write access in the data warehouse and activity log files and could target sensitive information about entity $\mathbb{T}$ stored as a set of $\langle key, value \rangle$ pair(s). The attacker has one initial

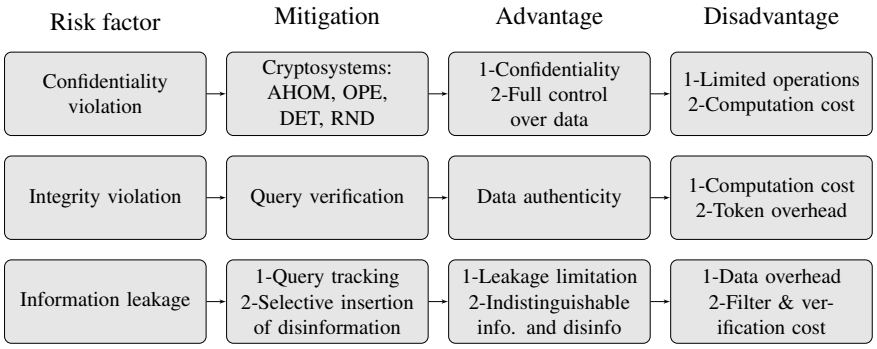


Figure 5.1: Risk factors posed by malicious insiders, and the corresponding solutions with advantages and disadvantages.

document stored in collection $\mathcal{C}_i$ and his goal is to obtain sensitive attributes of $\mathbb{T}$ through the cross-correlation among other collections.

Confidentiality violation risk. The attacker misuses her existing privileges to gain further access to sensitive information without any trace of intrusion. Such attacks are very hard to detect because the attacker is authorized to carry out the operation used for intrusion.

Mitigation: Cryptographic schemes can be used to encrypt data before cloud outsourcing. Processing encrypted data without decryption restrict the selection of cryptographic schemes selection.

An insider attacker can extract an encrypted sensitive attribute $\hat{A} = \langle E_k(key), E_k(value) \rangle$ related to an entity of interest $\mathbb{T}$ stored in collection $\mathbb{C}_i$ by iteratively calling the cross-correlation function $\Psi(\mathcal{C}_i, \mathcal{C}_j), \forall j \in [1, n]$. Two collections $\mathcal{C}_i$ and $\mathcal{C}_j$ may initially not share any documents, but they may be linked by cross-correlation that each one of them has with other collections.

Integrity violation (active attack) risk. Integrity verification ensures that data is only modified by an authorized user and identifies integrity violation done by an intruder. Query integrity verification is a tamper-resistant algorithm built on Message Authentication Codes (MAC). We append a new

attribute named *eTag* to each document containing a keyed hash value of the entire document. The data owner generates the augmented attribute $\langle eTag, E_k(d_i)\rangle$ for any document d_i.

As stated in the semantic security principle, all documents should be indistinguishable. Thus, the same procedure for generation of eTag conducted for the disinformation documents. In particular, a sensitive attribute can be seen as an equivalent class for $\mathcal{V}$ attributes that have diverse values for attributes to prevent disclosure. Finally, the attributes of real and disinformation documents are encrypted as stated in the security plan and forwarded to the DBaaS in the cloud. Figure 5.2 illustrates the process of eTag attribute production for documents.

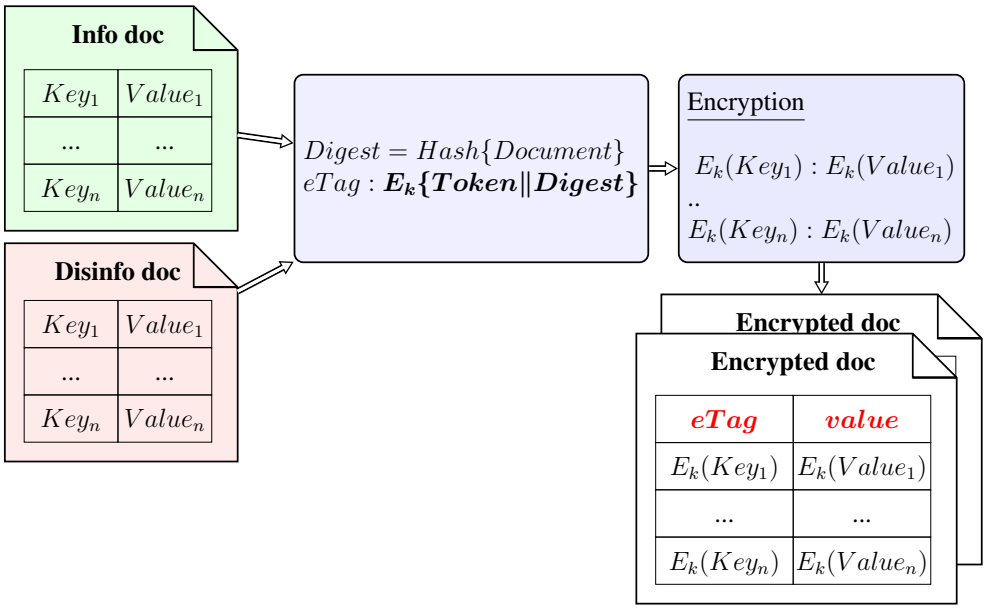


Figure 5.2: The high level description for *eTag* attribute creation.

Query integrity verification. The augmented eTag attribute is useful in two ways, including data integrity verification, and distinguishing disinformation documents from query response. The proxy decrypts eTag value with the decryption key and verifies the tag of original documents from the query response and filters out the disinformation documents. The proxy verifies the authenticity of whole document with recalculation and verification with eTag value. Having eTag attribute im-

poses a small overhead to the documents, however it guarantees the document never gets modified by cloud insiders. Intuitively, eTag is not involved in a query processing and this attribute will be used for the internal structure for providing integrity of data store. The *verification* process grantees the integrity of any given document d as presented in Equation 5.3.

$$eTag^{'} = MAC_k(d); \begin{cases} if(eTag == eTag^{'}) : d\ valid \\ else:\ d\ invalid. \end{cases} \tag{5.3}$$

The crypto-hash functions are building blocks of the introduced query integrity verification algorithm, and therefore, having an efficient hash function leads to low latency integrity verification. We examined the performance of four popular hash functions with respect to the variety of document size. The result is displayed in Figure 5.3. Considering the performance and security metrics, we select *SHA1* over other hash functions including *MD5, RIPEMD* and *SHA256* to be used in eTag algorithm. As it can be observed from Figure 5.3, the reason of selection of SHA1 is its high performance (speed) at different input documents sizes.

The dynamic nature of a dataset as a persistent storage necessitate to have basic operations to create, read, update and delete (known as CRUD). The augmented dataset (with disinformation documents), naturally are subjected to CRUD operations. The create and read operations act on the augmented dataset in a similar way to a normal dataset, while the update/delete operations on the original documents should be projected on the corresponding disinformation documents. For the performance purpose, the update/delete operations on the related disinformation documents, can be processed immediately or in lazy fashion, postponed to the next period of the data analysis. Furthermore, the garbage collector service, run by the proxy, can be developed to delete any unrelated disinformation from the dataset. Design and development of such a service is beyond the scope of this work.

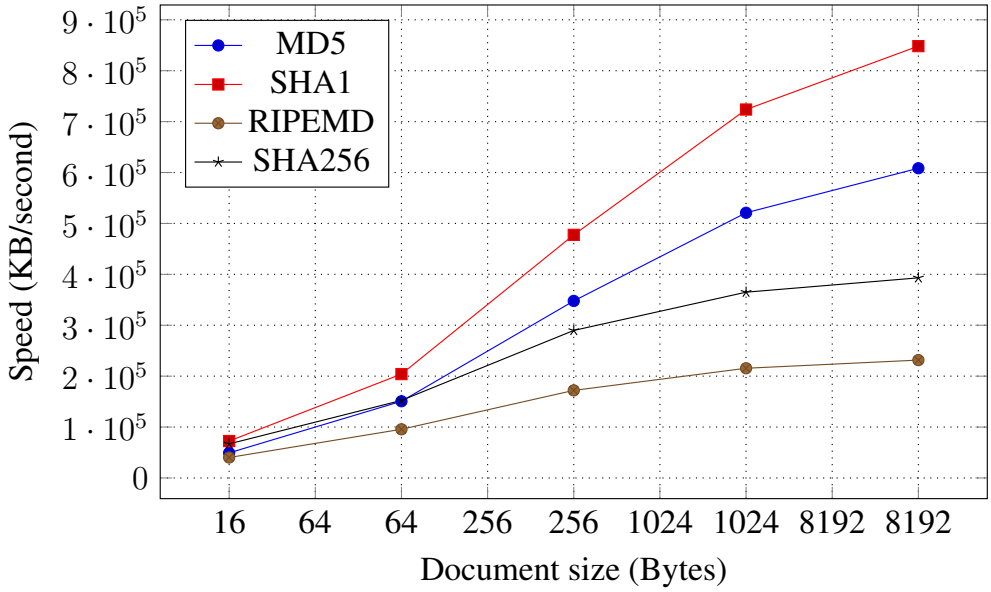


Figure 5.3: Performance of four popular cryptographic hash functions in respect to document size.

5.3 DBaaS Leakage Management

Two basic methods for quantifying information leakage due to cross-correlation among the databases and collection are discussed in this section. These methods require searching the entire databases for exact matching of attributes, a time-consuming proposition, only feasible for small size databases.

The attributes of a document are categorized in three classes: (1) Identifier attribute, uniquely identifies a single document in the database, e.g. social security number, phone number, or email address. (2) Semi-identifier attributes which do not uniquely identify a document, but collectively can distinguish a document, e.g. combination of name, department name, gender and age may identify an individual. (3) Feature attribute expresses a characteristic of an object by giving sensitive information. For example, account balance in a finance database, or the name of a person's disease in a health record.

a. Explicit correlation. A shared identifier attribute among the possible databases perform join between two databases using an explicit equality match of shared identifier attribute. This circumstance is described as *explicit correlation*. A correlation function in Equation 5.1 assigns a non-negative real number value as correlation score to a each involved datasets.

A fundamental requirement to manage the amount of information leakage, is to introduce measurement metrics. Two concepts, *precision* and *recall* adopted from data retrieval are used to measure the information leakage. Precision is denoted by P_{d_i} as the ratio of the weighted value of true information in d_i to value of all attributes in d_i. Similarly, the recall value for any document represented by R_{d_i} which reflects the ratio of true information in document to value of information in reference document. In this study weights of attributes are a function of the attribute class, while in [36] all attributes have an equal weight. Equation 5.4 will be used to quantify the value of information for a given document d consisting of n attributes, where ω_i is the assigned weight for i-th attribute.

$$\begin{aligned} &\Omega(d) = \sum_{1}^{n} \omega_i; \qquad Such\, that\, \omega_i \in [0,1] \\ &P_d = \frac{\Omega(d \cap \mathbb{R})}{\Omega(d)}; R_d = \frac{\Omega(d \cap \mathbb{R})}{\Omega(\mathbb{R})}; F_1 = \frac{2 \times P_d \times R_d}{P_d + R_d} \end{aligned} \tag{5.4}$$

To manage the cross-correlation leakage, we intentionally insert documents containing false or misleading values for sensitive attributes denoted as *disinformation document* which includes common attributes shared with the original document. Because of disinformation document insertion, the extraction of new attributes value from correlation will be more expensive for an attacker by factor of the number of disinformation per original document.

EXAMPLE 1. Consider five documents from different databases selected from the DBaaS warehouse, belonging to two target individuals named "Kate Jones", and "Mike Smith". The goal is to extract leaked attribute using Equations 5.1, 5.2, and 5.4. The attribute classification and their

assigned weights are given in table 5.1.

Table 5.1: Weights of attributes

Attribute name	**Type**	ω_i
Zip	Semi-identifier	0.3
Address	Semi-identifier	0.5
Phone	Identifier	1.0
Account	Identifier	1.0
Name	Semi-identifier	0.6
Age	Feature	0.1
Income	Feature	0.1
SSN	Identifier	1.0
Email	Identifier	1.0

$$d_1 = \{\mathbf{zip} : 456, \mathbf{address} : \text{“}2512\, Uni.\, NY\text{”}, \mathbf{phone} : \underline{111}\}$$

$$d_2 = \{\mathbf{ssn} : 123,\ \mathbf{age} : 33,\ \mathbf{account} : 222\}$$

$$d_3 = \{\mathbf{name} : \text{“}Kate\ Jones\text{”}, \mathbf{age} : 30, \mathbf{address} : \text{“}abc\text{”}$$

$$, \mathbf{email} : \text{“}kj@a.com\text{”}\}$$

$$d_4 = \{\mathbf{name} : \text{“}Mike\, Smith\text{”}, \mathbf{income} : 70k, \mathbf{ssn} : 123$$

$$, \mathbf{phone} : \underline{111}\}$$

$$d_5 = \{\mathbf{name} : \text{“}Kate\ Jones\text{”}, \mathbf{email} : \text{“}kj@a.com\text{”},\ \mathbf{ssn} : 777\}$$

In the above example, $\mathbb{R}_{Mike\, Smith},\ \mathbb{R}_{Kate\, Jones}$ are reference documents for two target entities. The extractable information through the correlation are as follows:

$\mu(d_1, d_2) = FALSE; \mathcal{L} = \{zip, address, phone\}$

$\mu(d_1, d_3) = FALSE; \mathcal{L} = \{zip, address, phone\}$

$\mu(d_1, d_4) = TRUE;$

$\mathcal{L} = \{zip, address, phone, name, income, ssn\}$

$Back\,Track$

$\mu(d_1, d_2) = TRUE;$

$\mathcal{L} = \{zip, address, phone, name, income, ssn, age, account\}$

$\mu(d_1, d_3) = FALSE;$

$\mathcal{L} = \{zip, address, phone, name, income, ssn, age, account\}$

$\mu(d_1, d_5) = FALSE;$

$\mathcal{L} = \{zip, address, phone, name, income, ssn, age, account\}$

$\mathbb{R}_{Mike\,Smith} = \{zip, address, phone, name, income, ssn, age$

$, account\}$

$\mathbb{R}_{Kate\,Jones} = \{name, age, address, email, ssn\}$

The recall value for these documents can be calculated by using Equation 5.4:

$$R_{d_1} = \frac{1.7}{5.4} \approx 0.31, R_{d_2} = \frac{2.1}{5.4} \approx 0.39, R_{d_3} = \frac{2.2}{3.2} = 0.69,$$
$$R_{d_4} = \frac{2.6}{5.4} = 0.48 \; and \; R_{d_5} = \frac{2.5}{3.2} = 0.78$$

In the example above, the disinformation documents ($\rho_1, .., \rho_6$,) with low precision are created as bellow. After inserting them into the original collections, the cloud insider has double values for each sensitive attributes. Therefore, the real value for attributes cannot be extracted with high confidence.

$$\rho_1 = \{\mathbf{zip} : 654, \mathbf{address} : \text{“}1500PlaceAZ\text{”}, \mathbf{income} : 60k$$
$$, \mathbf{ssn} : 321, \mathbf{phone} : 111\}$$
$$\rho_2 = \{\mathbf{ssn} : 321, \mathbf{age} : 43, \mathbf{address} : \text{“}abcAZ\text{”}, \mathbf{phone} : 876\}$$
$$\rho_3 = \{\mathbf{ssn} : 321, \mathbf{account} : 444\}$$

Similarly for “Kate Jones” we have :

$$\rho_4 = \{\mathbf{age} : 20,\ \mathbf{address} : \text{“}efd\text{”}, \mathbf{email} : \text{“}kj@a.com\text{”}\}$$
$$\rho_5 = \{\mathbf{name} : \text{“}ClaireShepard\text{”},\ \mathbf{ssn} : 543,\ \mathbf{email} : \text{“}kj@a.com\text{”}\}$$
$$\rho_6 = \{\mathbf{ssn} : 543,\ \mathbf{email} : \text{“}xy@b.com\text{”}\}$$
$$P_{\rho_1} = \frac{1}{2.9} = 0.34; P_{\rho_2} = \frac{0}{2.6} = 0; P_{\rho_3} = \frac{0}{2} = 0$$
$$P_{\rho_4} = \frac{1}{1.6} \approx 0.62; P_{\rho_5} = \frac{1}{2.5} \approx 0.4; P_{\rho_6} = 0$$

In short, it is more desirable to have low recall and precision values reflecting more uncertainty, and consequently less information leakage. Furthermore, other probabilistic means are required to measure the information leakage due to statistical properties of attributes in very large databases. Under those circumstances, we introduce our second method to measure leaked information due to statistical correlations.

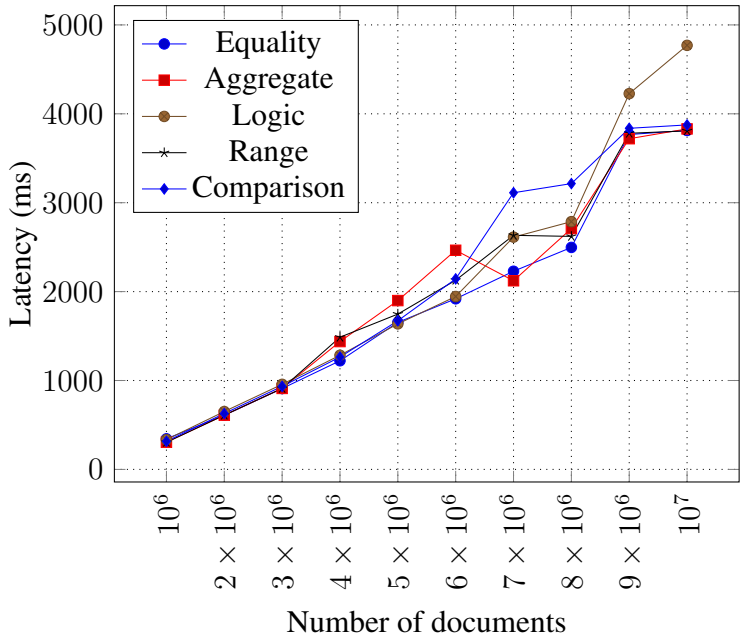
Implicit correlation. Sometimes there is a hidden mutual dependency between two data elements and then observation of one data item could result in inferring meaningful information about the other. The mutual dependency leaks out sensitive information about secret data which was supposed to be confidential. Identifying implicit and semantically correlated subset of attributes with different data types is a challenging work. The information theoretical methods are used to quantify implicit correlations. To facilitate discussion, an example of statistical property correlation of attributes is given below.

EXAMPLE 2. An on-line stock exchange website stores the information of share in the cloud DBaaS. One simple analysis indicates there are correlation between number of buyers, the quantity of buy orders and quantity of sell orders[1]. In the scenario of this example, the price trend is derived from the statistical properties of two different attributes.

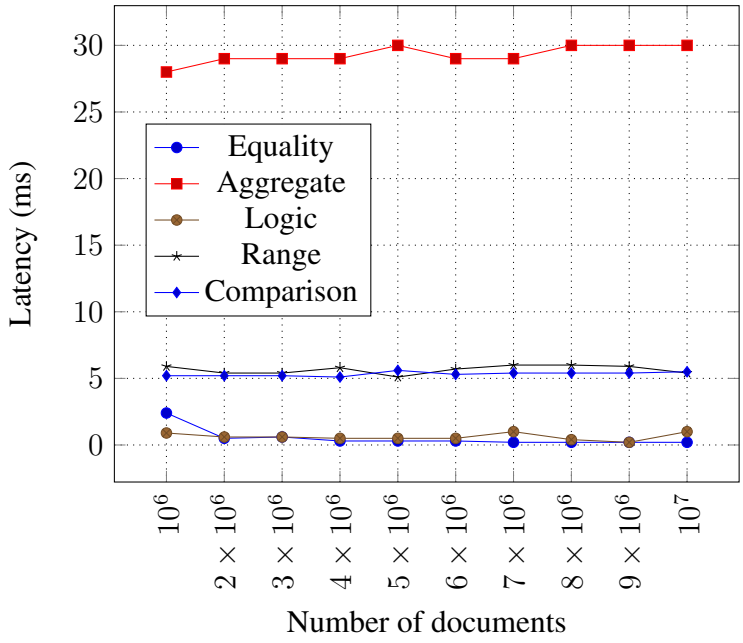
Performance cost and mitigations of disinformation. Insertion of disinformation documents increases the size of database and it can negatively affect the query execution time. To quantify the query latency in cloud DBaaS, an iterative method is employed to evaluate latency of several simple queries on the different databases that contain specific number of documents. In this way, we only focus on a single variable, which is the size of the database. The benchmark initially removes all documents from all databases and repopulates those with the required dataset size. Subsequently, two different tests are performed, with and without index. Five major query classes, including equality check, comparison, logic, range and aggregate are considered for query processing benchmark shown in Figure 5.4. To eliminate cache boost-up in the tests, the query caching is disabled. This process is repeated for all the specified database sizes and the measurement for the benchmark without using index is displayed in Figure 5.4a.

[1]If there are more buyers than sellers, it signifies a price increase; on the other hand, more sellers and high volume indicates a price drop.

The approach discussed in this work does not modify standard database servers thus, benefits from database technology features including indexing. Indexing allows to perform more sophisticated search on data, such as binary tree search, reducing the maximum search space drastically from $O(n)$ to $O(logn)$, a remarkable performance improvement. Figure 5.4b presents the improvement of the same queries execution time. The chart for a simple query on the non-indexed databases demonstrates that query latency steadily increases with rise of database size. However, the trend of query processing time remains steady, and shows no significant variations with increasing the size of indexed database. The indexed attributes guarantee an insignificant change in query processing time, especially for the encrypted databases which have the augmented size in comparison with the plaintext non-indexed database. This experiment is designed to examine indexing over ten ciphered and expanded databases with disinformation. Then, we measured query processing time with indexed encrypted database. The measurement process in both cases were automated and run under the control of the designed script which collected the processing time. The results are discussed in the next subsection.



(a)



(b)

Figure 5.4: The performance analysis of five different types of queries as a function of database size: (a) databases without indexes; (b) databases with indexes

Results and discussion. Typical cloud database services ensure advanced availability and scalability, but the data confidentiality and integrity are yet an area of interest to be explored further. The problem of secure processing of outsourced datasets with limited leakage is investigated in this study. The sensitive data in a single dataset can be protected with using crypto-systems. However, in the cloud DBaaS settings which is a pool of thousands of datasets, the aggregation of datasets introduces a new source of information leakage. A first challenge is to measure and limit the volume of information that an untrusted cloud database service can learn from the accumulation of data belonging to a group of users. The risks associated with an untrusted cloud DBaaS are investigated and a mitigation solution is proposed.

The second problem is that user applications expect to receive valid and accurate information in response of the issued queries, not the fake information. Most NoSQL databases have a different performance with processing the same query over different database size. Our experimental results show no significant variations in performance for a linear increase in database size, the performance penalty is negligible. This can be explained by the multilevel indexing which are utilized by NoSQL databases to provide a fast access time and short latency for query processing over larger databases. To overcome the second challenge, we propose and analyze an efficient algorithm based on the digital signature scheme to filter out the noisy documents.

5.4 Disinformation, Sensitivity Analysis, and Approximate Query Processing

Disinformation. A last resort method for information leakage prevention is disinformation, replication of collection documents with altered sensitive fields. The replication index is the cardinality of the set of documents created to hide the sensitive information in an original document. The larger the replication index, the more difficult it is for an attacker to identify the sensitive information, but the larger storage is required for the expanded collection.

The indiscriminate replication of all collection documents, not only increases the collection storage space dramatically, but also increases the response time for aggregate queries by a factor at least equal to the replication index. For example, a 100 TB collection becomes 1 PB collection when the replication index of every document in the collection is equal to ten; the query response time increases in average by an order of magnitude.

On-Line Analytical Processing (OLAP) applications extract information from massive datasets. The response time to a query posed to a very large collection can be prohibitive and limit the usefulness of data analytics. Many OLAP applications are latency sensitive and in some cases, e.g., in care of exploratory investigations, it is preferable to have an approximate answer to a query sooner than an accurate answer later. In such cases the Approximate Query Processing (AQP) solution proposed in [37] offers a tempting alternative and, as shown in this section, can also be useful for limiting information leakage.

AQP is based on a sampling technique for providing approximate responses to aggregated queries alongside an estimation of the implicit error produced by this method. An aggregate query is a query that calls aggregate functions to return a meaningful computed summary of specific attributes of a group of documents. Common aggregate functions are: *Average, Max, Min*, and *Count*. An AQP system supplies confidence intervals indicating the uncertainty of approximate answers. Indeed, an approximate answer without the specification of the errors involved would not be useful.

The documents in a collection have different levels of sensitivity, therefore an indiscriminate replication of all documents is not warranted. The solution proposed in this research requires a *sensitivity analysis* of the collection documents. Sensitivity analysis enables us to selectively apply disinformation to the collection documents. We choose AQP to carry out efficiently the collection sensitivity analysis.

Sensitivity analysis, samples, and errors. Sensitivity analysis has two stages: (i) establish sensitivity levels and (ii) determine the number of collection documents at each sensitivity level. The second stage of the sensitivity analysis requires an examination of all collection documents, a rather slow process. To facilitate fast sensitivity analysis, we shall use samples of the collection and report the estimation errors as required by AQP.

Collection samples consist of randomly selected documents from the original collection. Queries can be conducted in parallel on such samples. Given the set of documents in collection $\mathbb{C}$, and $\mathbb{S}$, the set of documents in a sample used by the AQP method, the *scaling factor*, σ, is defined as:

$$\sigma = \frac{\mid \mathbb{C} \mid}{\mid \hat{\mathbb{S}} \mid}. \tag{5.5}$$

The smaller the sample size, the larger is σ, and the shorter is the response time to a query posed to the sample, but also the larger are the estimation errors based on this sample.

Let S be a set of n sensitivity classes of documents in $\mathbb{C}$, $S = \{s_1, s_2, \ldots, s_n\}$. Call c_i the count of documents classified in sensitivity class s_i with $\mathbb{C} = \sum_{i=1}^{n} c_i$. Given the aggregate query θ, let $\hat{\theta}$ be the corresponding approximate query carried out using the documents in sample $\mathbb{S}$.

The response to the approximate query $\hat{\theta}$ may only include documents in $m \leq n$ sensitivity classes $\hat{s}_i$ of the set $\hat{S} = \{\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_n\}$. Call $\hat{c}_i \leq c_i$ the count of documents classified in sensitivity class $\hat{s}_i$. Then $\mid \hat{\mathbb{S}} \mid = \sum_{i=1}^{m} \hat{c}_i$. The sampling error for sensitivity class s_i is

$$\hat{e}_i = 100\frac{c_i - \hat{c}_i}{c_i}. \tag{5.6}$$

The error vector due to sampling is

$$\mathcal{E} = (\hat{e}_1, \hat{e}_2, \ldots, \hat{e}_n). \tag{5.7}$$

In a uniform random sampling $n - m$ classes may not appear in the response so components of the error vector for missing classes are 100%.

The Sampling-based Approximate Query Processing (S-AQP) with guaranteed accuracy provides bounds on the error caused by sampling [37]. A key element of any AQP system is to provide error bounds for the approximative results, allowing the user to decide whether the results are acceptable. Confidence Intervals (CI) represent the range of values centered at a known sample mean and used to calculate error bounds.

We use a close-form Central Limit Theorem(CLT) and Markov and Chebyshev inequalities to get the tightest bounds. As the number of elements in the sample n goes to infinity, the distribution converges into the standard normal random distribution $N(0, 1)$.

The tightness of the bounds resulted from the three aforementioned approaches are illustrated in Figure 5.5. Markov inequality provides larger deviation bounds than Chebyshev inequality. Close-form CLT provides the tightest bound among these three approaches [68].

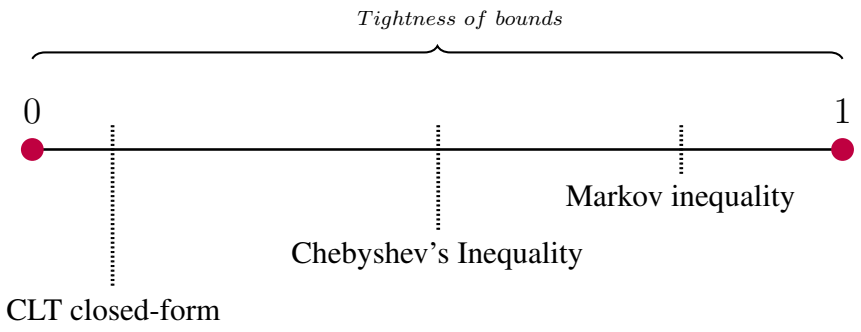


Figure 5.5: Bound tightness comparison obtained by using Markov, Chebyshev inequalities and close-form CLT.

If the estimation produced by AQP has the error percentage higher than the acceptable level sample size should be increased. This technique is known as bootstrapping. Sampling can be done with or without replacement. In sampling without replacement (disjoint samples), any two samples are independent whereas in sampling with replacement, sample values are dependent. Sampling without or with replacement is conducted during the *sampling phase*. The obtained results by resampling without replacement from the larger sample set are dependent on the original sample set, however it leads to a slightly more accurate estimation. The resampling process can be repeated to address user constraints related to acceptable approximation errors and response time latency [37, 40].

A bootstrap like resampling method is required to create a multi-layer sample set to meet different expectations. Assume θ is the query which is posed to process on the very large database D. With AQP the new query $\hat{\theta}$ will be composed to approximate the answer of θ using proper sample set. To rewrite an aggregate query θ to $\hat{\theta}$, one of the critical parameter is the scaling factor, denoted as $\delta = \frac{|D|}{|sample|}$ which is the ratio of the cardinality of the original database to cardinality of the sample. Figure 5.6 presents our resampling method.

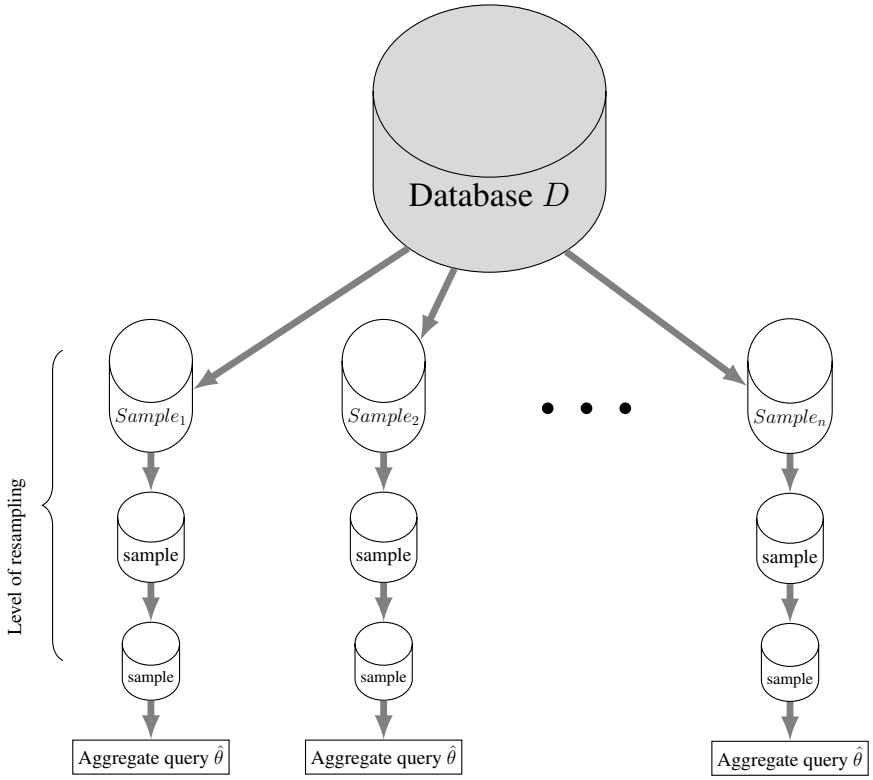


Figure 5.6: Execution of an aggregate query $\hat{\theta}$ on multiple random samples. The level of resampling is proportional to users' expectation in terms of error rate and latency.

Experimental results. Our experiments were conducted on a cluster of 100 AWS EC2 instances (t2.large) with two vCPU, 8 GB memory, and the Linux kernel version 4.4.0-59-generic. MongoBD version 3.2.7 was used as the NoSQL server. MongoDB supports variety of storage engines designed and optimized for various workloads. The storage engine is responsible for data storage both in memory and on disk; we chose *WiredTiger* storage engine. The OPE and AHOM cryptosystems are implemented locally and other crypto modules are imported from OpenSSL version 1.0.2g.

First, we measured the effect of the sample size on the estimation error. We created four sets of random samples from the original collection of 10^7 documents. Each set included 100 random

samples with 10^2, 10^3, 10^4, and 10^5 documents. The samples are selected with and without replacement. Figure 5.7 displays the error percentage for different sample size for the two different sampling modes. The measurement results show that samples without replacement exhibit slightly more accurate results than samples with replacement. For instance, the average error percentage is 0.22% for the largest sample of 10^5 documents, whereas the error is 5.08% for the smallest sample size of 100 documents. We concluded that a scaling factor of 1000 is perfectly suitable. The is likely to reduce the average response time to a query by two orders of magnitude.

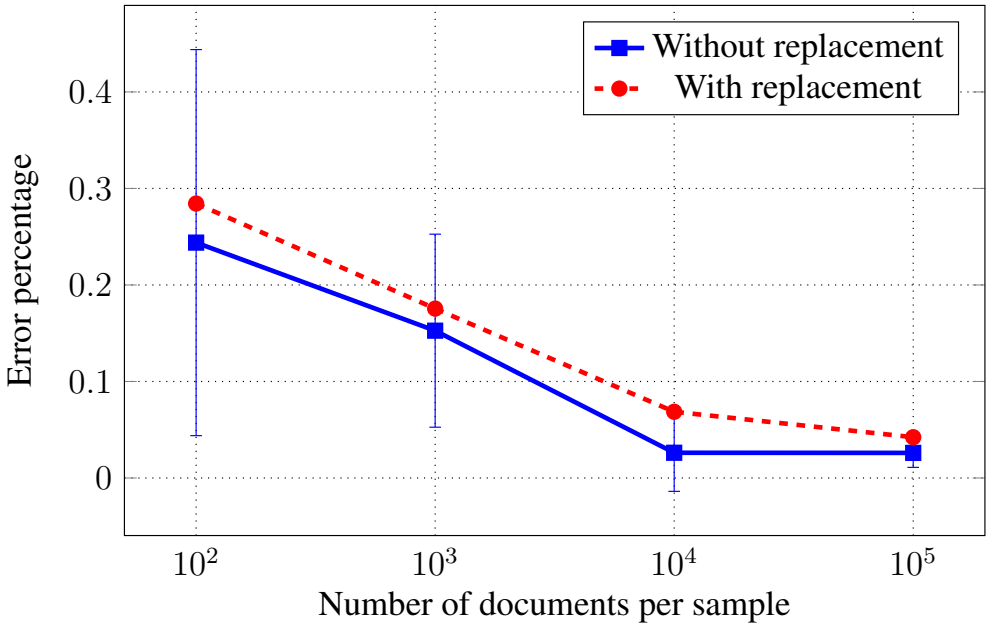


Figure 5.7: Estimation errors and confidence intervals for a collection with 10^7 documents. Results are shown for 10^2, 10^3, 10^4, and 10^5 documents per sample. Sampling without replacement consistently exhibit slightly more accurate results than sampling with replacement.

Next, we investigated sensitivity analysis. The first step of the sensitivity analysis is the determination the number of documents in each sensitivity class. The second step is the decision regarding the number of disinformation documents for each sensitivity class. In this experiment, we used a collection of ten million documents. Table 5.2 shows the eight sensitivity classes and the count and percentage of documents in each sensitivity class.

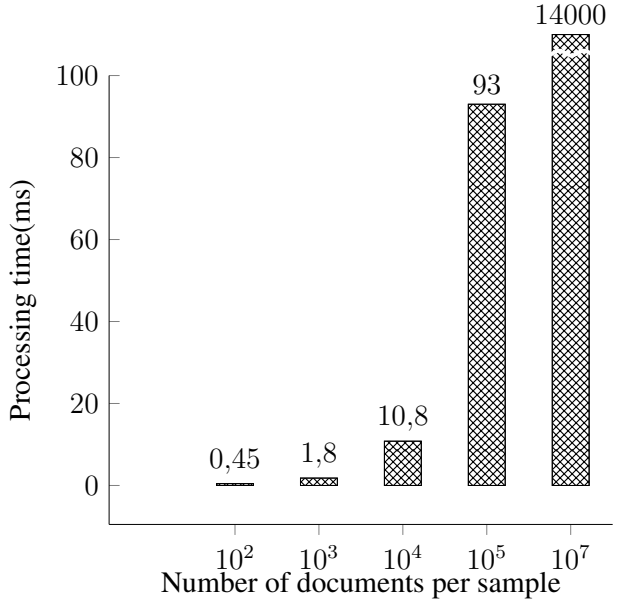
Table 5.2: Document counts in eight sensitivity classes for a collection with 10^7 documents.

Class (s_i)	**Cardinality**(c_i)	**Percentage**
Top Secret	782 471	7.823
Secret	1 475 118	14.751
Information	3 134 844	31.348
Official	1 475 603	14.756
Unclassified	783 443	7.834
Clearance	783 024	7.830
Confidential	782 698	7.826
Restricted	782 799	7.828
Total	**10 000 000**	**100.00 %**

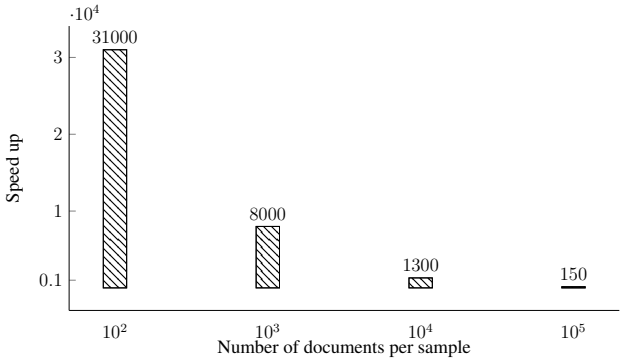
The ultimate objective of the sensitivity analysis is to reduce the query response time, defined as the interval between the time when the sever receives a query and the time it starts forwarding the query result. Most database servers cache the most recently used data to reduce the response time. In our experiments, we disabled query caching and prefetching in order to force the query optimizer to serve the next matching queries directly from the database not cache memory. The aggregate query displayed in Figure 5.8 is used for computation of cardinality and percentage of each sensitivity class.

```
 db[collection].aggregate([
{"$group":{"_id":{"clearance":"$clearance"}, "count":{"$sum":1}}},
{"$project":  { "count":  1, "percentage":{
"$concat":[{"$substr":[{"$multiply":[{"$divide":["$count",
{"$literal":Sample size }]},100]}, 0,6]},"", "%"]} } } ]);
```

Figure 5.8: Aggregate query for sensitivity analysis of collection. This query will be executed on the original database and 100 sample databases.



(a)



(b)

Figure 5.9: The performance of AQP based classification over 100 samples set with $10^5, 10^4, 10^3$ and 10^2 documents: (a) the processing time of aggregate query; (b) speedup obtained of AQP based classification.

The results show that the average speedup due to AQP is better than linear and the estimation

errors are quite low. Both metrics are plotted for the four sample size in Figure 5.9. For the largest sample of 10^5 documents, the average processing time is 93 ms, whereas for the original collection (including 10^7 documents) the processing time of the same query is $14\,000$ ms, a speedup of 150.

Table 5.3, displays the estimation for cardinality of each class , when the sample size is 10^4 and scaling factor of $\sigma = 10^3$. In this case the speedup is $1\,300$ and the error is about 1%. It is worth comparing the approximate result with the exact values in Table 5.2.

Table 5.3: The effect of sampling on the number of documents in each sensitivity class when the sample size 10^4. The column labeled Deviation shows the cardinality differences between the sample and the original collection.

Class(s_i')	**Cardinality(c_i')**	**Percentage**	**Deviation**
Top Secret	785 600	7.856	-3 129
Secret	1 462 000	14.620	13 118
Information	3 152 200	31.522	-17 356
Official	1 463 700	14.637	11 903
Unclassified	787 200	7.872	-3 757
Clearance	784 800	7.848	-1 776
Confidential	783 900	7.839	-1 202
Restricted	780 300	7.803	-2 499
Total	**9 945 260**	**99.4526**	**54 740**

After running the approximate sensitivity classification, the disinformation replication factor $\mathcal{V}$ can be assigned for each class according to their sensitivity class. For example, knowing the approximated cardinality value c_i', replication factors of 100, 25, 0, 5, 10, 0, 50 and 15 are considered for "Top Secret", "Secret", "Information", "Official", "Unclassified", "Clearance", "Confidential", and "Restricted" classes, respectively, from Table 5.3. The expansion factor E is defined as the ratio of the cardinality of the collection with disinformation to the original collection. In this example, the expansion factor using SDDP method is $E = 0.18$ while indiscriminate disinformation insertion leads to an expansion factor $E = 100$. The overhead is drastically reduced while

providing a similar leakage prevention level.

We concluded that AQP is a powerful method to substantially reduce query latency with bounded and small estimation errors. AQP with uniform random sampling provides sensible results for classification aggregate query workload, with a sensible compromise between sample size and query latency. However, for queries with different workloads such as aggregate functions that involve multiple correlated collections, the uniform sampling cannot provide accurate responses and we designed a new technique for biased sampling solution for this problem. In the next section, we highlight approximated answers for correlated collections.

5.5 Warehouse Information Leakage

Can the attribute correlation method discussed in Section 5.3 be extended to a cloud data warehouse hosting a large number of databases? Leakage prevention in case of a data warehouse requires an exhaustive cross-correlations analysis among all cloud datasets. For a warehouse hosting n databases, each one with m collections, each with q documents will require $N = (m \times n \times q)^2$ operations. For example, when $m = 10^3$, $n = 10^6$, and $q = 10^9$ then $N = 10^{36}$. The solution we propose is also based on AQP.

Cross-correlation size estimation. The approximation method has two phases: first, identifying the correlated keys; second, devising an optimum sampling method appropriate for cross-correlation cardinality approximation. We elaborate each phase as follows.

Phase 1: Correlated keys identification. The underlying idea is to create a graph of collections which has common attribute keys using the selected samples. In the resulting graph, the vertices i and j are connected through an edge if they both have the same identifier attribute. The identifier

attribute could be postal address, phone number, social security number, patient ID, and so on for any collection related to healthcare, utilities, financial records, etc.

Exploring connected components in an undirected graph can be done with graph search algorithms such as Depth First Search (DFS) or Breadth First Search (BFS) starting from every unvisited vertex. As a result, the nodes with higher degree indicate attributes that cause more cross-correlations. The sorted list of vertices based on their degree exposes the attributes that cause more cross-correlation in a pool of cloud databases.

To facilitate the discussion, we first consider a 2-way cross-correlation which is an unintentional join relation between two collections $\mathbb{C}_1$ and $\mathbb{C}_2$. This relation is a result of a common identifier attribute, namely *linkage attribute*. Intuitively, a k-way ($k > 2$) cross-correlation relation is a k-way join relation between a sequence of k correlated collections $\mathbb{C}_1, \mathbb{C}_2, \ldots, \mathbb{C}_k$ which is equivalent to a combination of several 2-way cross-correlations [39, 69].

Approximation of cross-correlation cardinality based on the uniform random samples suffers from a large error, which is addressed in our new sampling method. Along with accuracy, quality assessment of the approximated answer is another essential constituent which is a crucial property for any approximation method. In this section, we focus on the new sampling method that produces confidence intervals to capture the exact cardinality of cross-correlation with high probability. In particular, the confidence interval obtained from the sample sets were drawn by the sampling method define a range for estimated cardinality values ($\tilde{\mathbb{C}} \pm \alpha$) such that the exact value of cross-correlation cardinality $\mathbb{C}$ lies within the range with probability of at least $1 - \epsilon$, where $0 < \epsilon \ll 1$.

Phase 2: Heterogeneous biased sampling. The heterogeneous biased sampling is inspired by a recently improved sampling technique which takes the frequency of values into consideration [70]. Biased sampling creates sample sets with respect to the repetition frequency of each value of the

intended attribute. The higher frequency value, the higher selectivity probability which means it is more likely to be included in the sample set. Furthermore, infrequent values from both collections have small contribution in the sample set, however, if there are large number of infrequent values their impact adds up.

We customized the biased sampling algorithm consistent with cross-correlation analysis queries which are join-driven to probe and extract the leaked attributes from the given collections. For any 2-way join query there are two collections: the left and the right collection. The cross-correlation analytical query returns new set of attributes from the right collection based upon the evaluation result of the join-predicate.

To balance between sample size and accuracy, a tunable threshold T_i for each collection $\mathbb{C}_i$ is defined, so that the values with frequency of $f_v > T_i$ are definitely added to the sample, otherwise they will be included with probability of $p_v = \frac{f_v}{T_i}$. Higher values of T_i result in smaller sample set. The cross-correlation approximation, using biased sampling for an attribute value v between two collections $\mathbb{C}_L$ and $\mathbb{C}_R$ with the corresponding threshold parameters T_L and T_R, respectively, is demonstrated in Equation 5.8.

$$c_v : \begin{cases} f_L(v).f_R(v) & if\ f_L(v) \geq T_L\ and\ f_R(v) \geq T_R \\ T_L.f_R(v) & if\ f_L(v) < T_L\ and\ f_R(v) \geq T_R \\ f_L(v).T_R & if\ f_L(v) \geq T_L\ and\ f_R(v) < T_R \\ f_L(v).f_R(v).\max(\frac{T_L}{f_L(v)}, \frac{T_R}{f_R(v)}) & if\ f_L(v) < T_L\ and\ f_R(v) < T_R \end{cases} \tag{5.8}$$

We modify the threshold parameters of biased sampling algorithm to generate a larger sample set taken from the right collection than that of the left collection. We manage the threshold value

of right collection T_R to be significantly smaller than T_L to increase selectivity possibility which results in a larger sample size. By this adjustment, two samples are heterogeneous in terms of their size. Next, the cross-correlation analysis query is performed over the samples. An instance of cross-correlation extractor query which is implementation of Equation 5.1 is displayed in Figure 5.10.

```
db[Left].aggregate([
{$lookup :{from :Right,
localField:value,foreignField:value,
as :"correlation" }},
{$match :{"correlation":{$ne :[]}}},
{ $out :  saveToCollection }]);
```

Figure 5.10: The aggregation join query for discovery of attributes from the Right collection based on evaluation of equality check on the value of the common attribute.

After samples are created, the 2-way cross-correlation analysis query is processed over the samples instead of the original data. The sampling probability for collection $\mathbb{C}_i$ is $p_i = \frac{|\mathbb{S}_i|}{|\mathbb{C}_i|}$, where $\mathbb{S}_L$ and $\mathbb{S}_R$ are sample set are taken from $\mathbb{C}_L$ and $\mathbb{C}_R$ respectively. The exact cross-correlation between $\mathcal{C}_L$, $\mathcal{C}_R$ is denoted by $\mathbb{C}_{LR}$ while $\mathbb{S}_{LR}$ is the approximated value computed for $\mathbb{S}_L$ and $\mathbb{S}_R$. Utilizing the biased sampling, the cross-correlation size approximation is computed by $\tilde{\mathbb{C}} = \sum_v c_v$. The scaling factor $\frac{1}{\min(p_L, p_R)}$ is used to scale up the result from the size of samples to the size of original dataset.

Experimental results. We use four cross-correlated databases from different areas consisting of social media profiles, phone directory, medical and financial records. Each collection includes 10^7 documents. The pairwise exact cross-correlation cardinalities are known in advance, as displayed in Figure 5.11.

The proposed estimation method based on heterogeneous biased sampling is evaluated on the different datasets and compared with the random sample selection method. The optimized biased sampling method can provide more accurate response in the speed of interactive time. Although, the proposed method requires more precomputation regarding the frequency of values, it can be done offline. In contrast to the other independent sampling methods, approximation using the heterogeneous biased sampling provides more accurate cross-correlation size estimation with the same sample size within interactive time budget. The comparison between random sampling and the proposed method is illustrated in Figure 5.12.

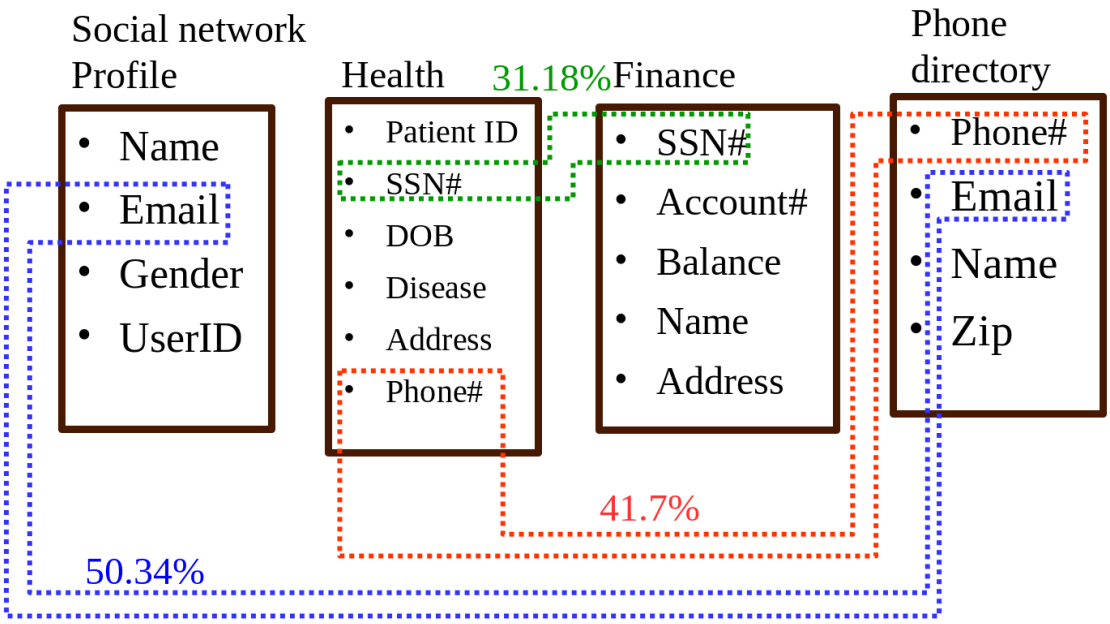


Figure 5.11: Four datasets containing 10^7 documents with a pre-defined level of cross-correlations are used by the estimation algorithm.

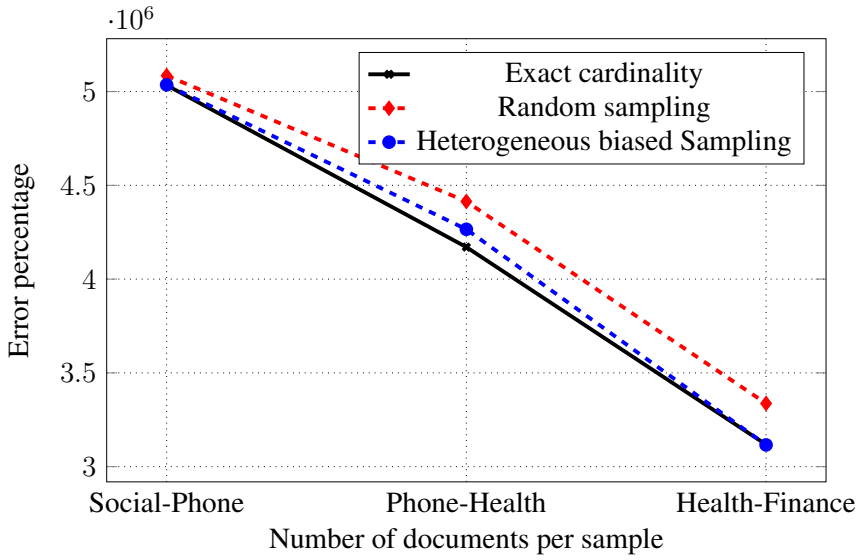


Figure 5.12: Approximation of cross-correlation cardinality using two sampling methods.

The calculation time of the exact cross-correlation cardinality between social media and phone directory collections requires almost 12 000 seconds of processing time. However, with the proposed approximation method, it is done in 1.5 seconds in trade-off with 1% error. This method significantly improves time complexity of leakage analysis in a cloud DBaaS warehouse.

5.6 Conclusion

Information leakage is a new type of threat to public clouds where data warehouses maintain numerous databases from many organizations. The sensitivity analysis based on approximate query processing introduced in this work identifies the most valuable information to be protected. Insertion of disinformation documents can reduce the information leakage resulting from attribute cross-correlations among a group of datasets. Yet, indiscriminate insertion of disinformation increases dramatically the size of the datasets and the query response time. Thus, the need for

sensitivity analysis is a fundamentally important requirement.

Attribute cross-correlation between hosted databases in a cloud warehouse, involves accessing enormous amounts of data, information leakage analysis is prohibitively time consuming. To mitigate this process, an approximation method is presented to estimate the size of cross-correlation using heterogeneous biased sampling with reasonable level of inaccuracy. The optimum sample size results in substantial speed up in cross-correlation cardinality analysis with closer answer to the exact value.

The solution discussed in Section 5.5 suggest introduction of leakage detection cloud services that can offer organizations guidance on how to better protect their data and minimize the risks of information leakage. Such recommendations include the use of data encryption for particular fields, or other nonlinear transformations diminishing cross-correlation leakage.

Future work. Sensitivity and cross-correlation analysis at cloud warehouse level can only be conducted by a CSP with access to all datasets. We suggest individual Service Level Agreements that include a clause related to information leakage protection, allowing the CSPs to periodically compute Cross-Correlation Indexes (CCIs).

CHAPTER 6: CONCLUSION

Fine-grained encryption schemes add another layer of protection for data stored in the third-party storage; however, most applications in the cloud platform desire not only secure storage, but also processing functionality on the encrypted data sets. The primary hurdle that restricts utilization of schemes that only protect data at-rest is that encrypted data can not be queried which is very crucial for most applications. Intuitively, for processing encrypted data in a cloud service, the decryption key must be compromised, which is a big security violation. This is considered a big conflict of cloud computing, and using traditional cryptosystems. Motivated by these limitations a new set of cryptosystems is needed to exercise in the cloud settings to maintain data security meanwhile taking advantages of cloud platform. This issue is addressed in our study by investigating new cryptosystems that support computation on the ciphertext.

Theoretically, FHE is considered the safest cryptosystem that allows all type of general computation to be performed on the encrypted data. FHE also has great semantic security and it leaks almost nothing about the data. However, it is prohibitively inefficient even after lots of improvement on practicality of FHE still it is nine orders of magnitude slower than plaintext computation. Beside, the huge cryptographic overhead of FHE, another main reason for impracticality of FHE is the query should be expressed as circuit over the entire database. This means, for every single query the entire database must be processed while in plain database by using indexes only small fraction of data need to be scanned. We tried to come up with intermediate point ideally, almost as fast as plaintext while at the same time having high degree of security.

One main contribution of our solution is its modest overhead which is roughly below 20% throughput (number of query per second that server can execute) loss in comparison with plaintext database. Interestingly, SecureNoSQL makes changes neither in the existing database service nor in the ap-

plications. Thus, the sever use special data structure such as indexes to run queries on encrypted data in a short amount of time. Also, the application can run on top of SecureNoSQL without getting involved in the complexity of encryption/decryption operations. The insight of SecureNoSQL is that in fact most queries posed by clients are using limited set of query operators, so we tried to support those operations on the encrypted data.

Impracticality of FHE motivated us to use specific kind of homomorphism that supports some operations such as summation and multiplication. First, DET encryption was used to support equality check and full-word search operations; therefore, a group of operations such as *Count ($count)*, *Group by ($group)*, *Equality match ($eq)*, *Not equal ($ne)*, and *Selects if value specified is in the array ($in)*. We used OPE to support large number of operations that are dealing with the order of data such as *Greater than ($ge)*, *Less than ($lt)*, *$orderby*, *$sort*, *$max*, and *$min* as well as range queries on encrypted data. OPE encryption is suitable for data fields with high entropy where the order of values does not leak much, so OPE is still provides an acceptable level of protection.

For protection of data in transit we rely on Secure Socket Layer (SSL) and Transport Layer Security (TLS) protocols, especially data already have been encrypted with variety of the schemes.

Aggregation of large number of databases in a cloud data warehouse poses a new type of security risk called information leakage threatens sensitive information even in a encrypted database. The sensitivity analysis of a very large scale NoSQL databases based on approximated query processing is presented in which provides classification information in interactive speed. The result provided by this method usually comes with negligible deviation from the exact result. The output of fast classification method can be used to focus on the security of documents which are classified in the top sensitivity class.

The concept of information leakage resulted from attribute cross-correlation in the pool of cloud-

hosted databases investigated in this work. For leakage management, insertion of disinformation documents studied to limit leakage among a group of datasets. Meanwhile, indiscriminate insertion of disinformation introduces two disadvantages to the systems. First, the growth in data overhead causes system performance degradation. Second, there will be a substantial increase in communication cost between the DBaaS sever and the clients' applications.

To overcome these challenges, two mitigation solutions were presented. First, the leakage quantitative methods to detect the documents that have more leaked information in datasets. A fast sensitivity analysis method based on approximate query processing for any dataset with multi-level sensitivity was presented. To enhance the performance and speed up of the AQP based classification, we used multi-layer sampling technique to provide different sizes to deliver query response with variety of acceptable error rates. Using AQP, elevates the scalability of sensitivity analysis to any database size in the interactive speed. With the proposed method, the large latency of analytical aggregate queries which severely limits the feasibility of many analysis applications will be resolved.

Second, we have extend application of AQP to design a method to approximate cross-correlation cardinality in the cloud DBaaS warehouse level with enormous number of BP level databases. With the presented method, the attribute cross-correlation can be discovered in almost interactive speed. The proposed technique to estimate the size of cross-correlation using biased sampling showed both theoretically and empirically effectiveness of our method. They main reason for taking this approach is that in this way the optimum sample size results in substantial speed up in cross-correlation analysis with closer answer to the real value.

LIST OF REFERENCES

[1] M. Stonebraker, "SQL databases v. NoSQL databases," *Commun. ACM*, vol. 53, 4, pp. 10–11, April 2010.

[2] D. C. Marinescu, *Cloud computing: theory and practice,* Morgan Kaufmann, May 2013.

[3] M. Tremante, "Amazon web services growth unrelenting," `https://news.netcraft.com/`, [Online; accessed 27 October 2017].

[4] H. Liu, "Amazon data center size," `https://huanliu.wordpress.com`, [Online; accessed 27 October 2017].

[5] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," *ACM Transactions on Computer Systems (TOCS)*, vol. 26, no. 2, p. 4, 2008.

[6] J. Baker, C. Bond, J. C. Corbett, J. Furman, A. Khorlin, J. Larson, J.-M. Leon, Y. Li, A. Lloyd, and V. Yushprakh, "Megastore: Providing scalable, highly available storage for interactive services," *Proceedings of the Conference on Innovative Data system Research (CIDR)*, pp. 223–234, 2011.

[7] S. Sivasubramanian, "Amazon dynamodb: A seamlessly scalable non-relational database service," *Proceedings of ACM SIGMOD Int. Conference on Management of Data*, pp. 729–730, 2012.

[8] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85–100, 2011.

[9] D. Cash, J. Jaeger, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Rosu, and M. Steiner, "Dynamic searchable encryption in very-large databases: Data structures and implementation," *Network and Distributed System Security Symposium (NDSS14)*, 2014.

[10] D. Cash, S. Jarecki, C. Jutla, H. Krawczyk, M.-C. Roşu, and M. Steiner,"Highly-scalable searchable symmetric encryption with support for boolean queries," *Advances in Cryptology–CRYPTO 2013*, pp. 353–373, 2013.

[11] D. X. Song, D. Wagner, and A. Perrig, "Practical techniques for searches on encrypted data," *Proceedings IEEE Symposium on Security and Privacy*, pp. 44–55, 2000.

[12] J. H. Cheon, M. Kim, and M. Kim, "Optimized search-and-compute circuits and their application to query evaluation on encrypted data," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 1, pp. 188–199, 2016.

[13] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich, "Processing analytical queries over encrypted data," *Proceedings of the VLDB Endowment*, vol. 6, no. 5, pp. 289–300, 2013.

[14] R. Cattell, "Scalable sql and nosql data stores," *ACM SIGMOD Record*, vol. 39, no. 4, pp. 12–27, 2011.

[15] A. Boldyreva, N. Chenette, Y. Lee, and A. Oneill, "Order-preserving symmetric encryption," *Advances in Cryptology-EUROCRYPT 2009*, pp. 224–241, Springer, 2009.

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

[17] L. Xiao, I. Yen, *et al.*, "Security analysis for order preserving encryption schemes," *46th Annual Conference on Information Sciences and Systems (CISS)*, pp. 1–6, 2012.

[18] F. Kerschbaum, "Frequency-hiding order-preserving encryption," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 656–667, 2015.

[19] M. Canim, M. Kantarcioglu, B. Hore, and S. Mehrotra, "Building disclosure risk aware query optimizers for relational databases," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 13–24, 2010.

[20] S. Bajaj and R. Sion, "Trusteddb: A trusted hardware-based database with privacy and data confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 3, pp. 752–765, 2014.

[21] M. Ahmadian, A. Paya, and D. Marinescu, "Security of applications involving multiple organizations and order preserving encryption in hybrid cloud environments," *IEEE International Conference on Parallel Distributed Processing Symposium Workshops (IPDPSW)*, pp. 894–903, May 2014.

[22] M. Ahmadian, "SECURE QUERY PROCESSING in CLOUD NoSQL," *2017 IEEE International Conference on Consumer Electronics (ICCE)*, pp. 90–93, Jan. 2017.

[23] M. Ahmadian, J. Khodabandehloo, and D. Marinescu, "A security scheme for geographic information databases in location based systems," *IEEE SoutheastCon*, pp. 1–7, April 2015.

[24] P. Colombo and E. Ferrari, "Privacy aware access control for big data: a research roadmap," *Big Data Research*, vol. 2, no. 4, pp. 145–154, 2015.

[25] M. Islam and M. Islam, "An approach to provide security to unstructured big data," *8th International Conference on Software, Knowledge, Information Management and Applications (SKIMA)*, pp. 1–5, Dec 2014.

[26] J. Gantz and D. Reinsel, "The digital universe in 2020: Big data, bigger digital shadows, and biggest growth in the far east," *IDC iView: IDC Analyze the Future*, vol. 2007, pp. 1–16, 2012.

[27] C. Tankard, "Big data security," *Journal of Network Security*, vol. 2012, no. 7, pp. 5–8, 2012.

[28] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, and J. Molina, "Controlling data in the cloud: outsourcing computation without outsourcing control," *Proceedings of the ACM workshop on Cloud computing security*, pp. 85–90, 2009.

[29] R. A. Popa, C. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85–100, ACM, 2011.

[30] M. Ahmadian, F. Plochan, Z. Roessler, and D. C. Marinescu, "SecureNoSQL: An approach for secure search of encrypted nosql databases in the public cloud," *International Journal of Information Management*, vol. 37, no. 2, pp. 63 – 74, 2017.

[31] M. Naveed, S. Kamara, and C. V. Wright, "Inference attacks on property-preserving encrypted databases," *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 644–655, ACM, 2015.

[32] C. Liu, L. Zhu, M. Wang, and Y.-a. Tan, "Search pattern leakage in searchable encryption: Attacks and new construction," *Information Sciences*, vol. 265, pp. 176–188, 2014.

[33] R. Ostrovsky, "Efficient computation on oblivious rams," *Proceedings of the twenty-second annual ACM symposium on Theory of computing*, pp. 514–523, ACM, 1990.

[34] O. Goldreich and R. Ostrovsky, "Software protection and simulation on oblivious rams," *Journal of the ACM (JACM)*, vol. 43, no. 3, pp. 431–473, 1996.

[35] N. Li, T. Li, and S. Venkatasubramanian, "t-closeness: Privacy beyond k-anonymity and l-diversity," *2007 IEEE 23rd International Conference on Data Engineering*, pp. 106–115, April 2007.

[36] S. E. Whang and H. Garcia-Molina, "Managing information leakage,", *Stanford InfoLab*, 2010.

[37] S. Agarwal, H. Milner, A. Kleiner, A. Talwalkar, M. Jordan, S. Madden, B. Mozafari, and I. Stoica, "Knowing when you're wrong: Building fast and reliable approximate query pro-

cessing systems," *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, SIGMOD '14, (New York, NY, USA), pp. 481–492, ACM, 2014.

[38] K. Zeng, S. Gao, B. Mozafari, and C. Zaniolo, "The analytical bootstrap: a new method for fast error estimation in approximate query processing," *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pp. 277–288, ACM, 2014.

[39] S. Acharya, P. B. Gibbons, V. Poosala, and S. Ramaswamy, "Join synopses for approximate query answering," *ACM SIGMOD Record*, vol. 28, pp. 275–286, ACM, 1999.

[40] B. Babcock, S. Chaudhuri, and G. Das, "Dynamic sample selection for approximate query processing," *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pp. 539–550, ACM, 2003.

[41] C. Gentry, "Computing arbitrary functions of encrypted data," *Communications of the ACM*, vol. 53, no. 3, pp. 97–105, 2010.

[42] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," *Advances in cryptologyEUROCRYPT99*, pp. 223–238, Springer, 1999.

[43] L. Xu, X. Zhang, X. Wu, and W. Shi, "Abss: An attribute-based sanitizable signature for integrity of outsourced database with public cloud," *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*, pp. 167–169, 2015.

[44] L. Xu, C. Jiang, J. Wang, J. Yuan, and Y. Ren, "Information security in big data: Privacy and data mining," *Access, IEEE*, vol. 2, pp. 1149–1176, 2014.

[45] X. Yu and Q. Wen, "A view about cloud data security from data life cycle," *International Conference on Computational Intelligence and Software Engineering (CiSE)*, pp. 1–4, Dec 2010.

[46] N. E. Weiss and R. S. Miller, "The target and other financial data breaches: Frequently asked questions," *Congressional Research Service*, 2015.

[47] J. Silver-Greenberg, M. Goldstein, and N. Perlroth, "Jpmorgan chase hack affects 76 million households," *New York Times*, vol. 2, 2014.

[48] C.-H. Lin, S.-H. Tsai, and Y.-P. Lin, "Secure transmission using MIMO precoding," *IEEE Transactions on Information Forensics and Security*, vol. 9, pp. 801–813, May 2014.

[49] S. Halevi and V. Shoup, "Algorithms in HElib," *International Cryptology Conference*, pp. 554–571, Springer, 2014.

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

[51] L. Ducas and D. Micciancio, "Fhew: Bootstrapping homomorphic encryption in less than a second," *Advances in Cryptology–EUROCRYPT 2015*, pp. 617–640, 2015.

[52] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," *Advances in Cryptology–CRYPTO*, pp. 505–524, 2011.

[53] D. Micciancio and O. Regev, "Lattice-based cryptography," *Post-quantum cryptography*, pp. 147–191, Springer, 2009.

[54] D. Cash, D. Hofheinz, E. Kiltz, and C. Peikert, "Bonsai trees, or how to delegate a lattice basis," *Journal of cryptology*, vol. 25, no. 4, pp. 601–639, 2012.

[55] S. Gorbunov, V. Vaikuntanathan, and H. Wee, "Attribute-based encryption for circuits," *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, pp. 545–554, 2013.

[56] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 563–574, ACM, 2004.

[57] D. of Energy, "Grid Modernization and the Smart Grid." `http://energy.gov/oe/services/technology-development/smart-grid`, [Online; accessed 27 October 2017].

[58] E. Union, "Smart grids and meters." `http://ec.europa.eu/energy/en/topics/markets-and-consumers/smart-grids-and-meters`, [Online; accessed 27 October 2017].

[59] R. E. Brown, "Impact of smart grid on distribution system design," *Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, 2008 IEEE*, pp. 1–4, IEEE, 2008.

[60] R. DeBlasio and C. Tom, "Standards for the smart grid," *2008 IEEE Energy 2030 Conference*, 2008.

[61] Amazon, "Amazon Web Services." `http://www.aws.amazon.com/`, 2016. [Online; accessed 27 October 2017].

[62] K. Xie, Y.-q. LIU, Z.-z. ZHU, and E.-k. YU, "The vision of future smart grid," *Electric Power*, vol. 41, no. 6, pp. 19–22, 2008.

[63] F. Galiegue and K. Zyp, "Json schema: core definitions and terminology," *Internet Engineering Task Force (IETF)*, 2013.

[64] E. Dede, M. Govindaraju, D. Gunter, R. S. Canon, and L. Ramakrishnan,"Performance evaluation of a mongodb and hadoop platform for scientific data analysis," *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pp. 13–20, 2013.

[65] K. Ben, "Generate sample data."https://www.generatedata.com/, [Online; accessed 27 October 2017].

[66] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 51–58, December 2001.

[67] K. Chatzikokolakis, T. Chothia, and A. Guha, "Calculating probabilistic anonymity from sampled data," *Manuscript*, vol. 200, no. 9, 2009.

[68] P. J. Huber, "The behavior of maximum likelihood estimates under nonstandard conditions," *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, vol. 1, pp. 221–233, 1967.

[69] D. Vengerov, A. C. Menck, M. Zait, and S. P. Chakkappen, "Join size estimation subject to filter conditions," *Proceedings of the VLDB Endowment*, vol. 8, no. 12, pp. 1530–1541, 2015.

[70] C. Estan and J. F. Naughton, "End-biased samples for join cardinality estimation," *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pp. 20–20, IEEE, 2006.