SECURE QUERY PROCESSING in CLOUD
NoSQL

Mohammad Ahmadian
Computer Science Department
University of Central Florida, Orlando, Florida 32816, USA
Email: ahmadian@cs.ucf.edu

Abstract—This paper proposes “SecureNoSQL,” a security scheme for secure querying over encrypted cloud NoSQL databases. The contributions of this paper are: (1) Introduces a novel descriptive language for defining security parameters; (2) Presents the design of a secure proxy enforcing a security plan on data and on query/response. A security plan defines the main attributes and the parameters of the security scheme.

Index Terms—Public cloud, Database as a Service, NoSQL, Document store, security, search, query, encrypted data.

I. INTRODUCTION AND MOTIVATION

In this paper we restrict our discussion to query processing particularly over encrypted NoSQL databases. We report first on a secure proxy called “SecureNoSQL.” The proxy enables access to a cloud server with state of the art efficient cryptographic primitives for query, response and data encryption/decryption. In addition, we introduce a descriptive language based on JSON notation, to generate a security plan. The security plan has four sections that introduces elaborately the data element, cryptographic modules and the mapping between them. In summary, the main contributions of this paper are:

• We introduce a descriptive language based on JSON notations to create a security plan that describes security parameters and maps crypto-modules to the data elements.
• A balanced system with a security level-proportional overhead. The overhead of the scheme is proportional to the desired level of security.
• SecureNoSQL is a secure proxy which transfers queries to run on the cloud server with respect to semantics of queries. In addition, with SecureNoSQL scheme, the server remained unmodified; therefore, it treats encrypted documents in the collection in a same way as plaintext database. With this feature, all properties of the distributed database system such as replication holds for encrypted data.

The rest of the paper is organized as follows: related work is presented in Section II. The organization of the system presented in Section III and the structure of the security plan and the notation of descriptive language for generation of the security plan is discussed in Section IV. Afterward, the mechanism of query processing is investigated in section V. Finally, in Section VI 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 length of 64, 128, 256, 512 and 1024 bit.

II. RELATED WORK

NoSQL databases lack a proper data protection mechanism. These databases have been designed to support high performance and scalability requirements. Privacy and security mechanisms are required to protect sensitive information of NoSQL databases residing on public clouds.

The integration of privacy aware access control features into existing big data are discussed in the [1, 2]. In [3, 4] the evolution of big data Systems from the perspective of an information security application is studied. A proxy is a very important element in the structure thus, special attention should be paid for its protection. The cloud based monitoring and threat detection system proposed by Chaw et al. in [5] is a critical component to for enforcing data security and privacy on the cloud computing infrastructure.

An investigation of Order Preserving Encryption (OPE) security against known plaintext attack with known N plaintexts is reported in [6, 7]. These papers concluded that the ideal OPE module accomplish one-awayness security.

The risk of disclosure caused by main memory attack is quantified by [8, 9]. An application of OPE in cloud environment is reported in [10] also application of classical cryptography on relational database system for embedded devices was studied in [11].

III. SYSTEM ORGANIZATION

The system we introduce in this paper is designed with several objectives in mind:

• Support multi-user access to an encrypted NoSQL database. Enforce confidentiality, privacy of transactions.
• Transparent from the end-user prospective and users do not engaged in the complexity of the security mechanisms.
• Avoid transmission of unencrypted data over public communication lines.
• Do not require any modification of cloud NoSQL server.
• Create an open-ended system; allow the inclusion of cryptographic modules best suited for an application.

These objectives led us to design SecureNoSQL which is secure proxy to leverage secure search on the cloud remote server with no modification and minimum required change on
the client side applications. The query processing in the system involves three phases:

1) Client-side applications issue queries in JSON format.
2) The query encryption and decryption done by designed proxy according to security plan.
3) Server-side query processing performed by an unmodified NoSQL database management system.

Two different system structural organizations for addressing these design objectives are possible. First, shown in Figure 1 is suitable when all database users belong to the same organization. Then the proxy runs on a trusted server behind a firewall, so the communication between clients and the proxy is secure. Second, when clients are unrelated to one another and access the system through public lines. In this case either each client software includes a copy of the proxy and only encrypted data is transmitted over public lines or using Secure Sockets Layer (SSL) protocol to establish secure connection to the proxy.

**IV. DESCRIPTIVE LANGUAGE FOR SECURITY PLAN**

For securing NoSQL databases, the first step is the development of a security plan that outlines and maps out the determined crypto-primitive with specific parameters to particular data element. Considering the fact that NoSQL databases are schema-less and each document can have different set of attributes (key-value pairs), makes it very hard to have security plan. For this problem, we designed a simple descriptive language to generate and read security plan automatically. For this purpose we used subset JSON notation which is readable by human and machine. We proposed to first find superset of all attributes then assign appropriate crypto-primitive to elements of superset. This technique helps us to have comprehensive security that considers all elements of all type of documents.

In fact the security plan identifies the mechanism that will be applied to maintain security of data elements in a database and how to interpreted queries that are issued from a specific user applications. The designed security plan details how those rules will be implemented. We organized the security plan in four sections which efficiently enable us to describe security rule not only for data elements but also for meta-data like field-name (Key), collection name. These sections are the building blocks of security plan, the structure of sections and are presented as follows:

1) **Collection**. Introduces the name of collection and assigns an encryption module for them since these names as meta-data are source of rich information for attackers.
2) **Cryptographic modules**. Presents the list of cryptographic modules that are available in the system.
3) **Data elements**. Defines the properties of each data field such as data-type, length and name.
4) **Mapping cryptographic modules to the fields**. Specifies the cryptographic modules for encryption of database fields.

**Collection.** A collection is defined as a group of NoSQL documents which is an equivalent for table in relational database. A collection has some properties like name which need to be protected by encryption.

**Cryptographic modules.** Several crypto-systems with different strengths and weaknesses exist. The choice of a particular crypto-system depends on the security policy of applications. Several criteria impact the choice of an encryption algorithm including: the desired level of security, the efficiency of encryption and decryption, whether the encryption and decryption can be parallelized in cpu pool like cloud computing, the memory requirements, known weaknesses of the algorithms, and the integration in the overall system design. According to the proposed format, the Cryptographic modules introduces all encryption modules and their parameters such as key, key-size, initialization vector and output-size.

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 crypto-systems 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[12]. 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. To ensure the desired level of security the security plan should provides 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 specifies all cryptographic modules for all sensitive data fields.
V. PROCESSING QUERIES ON ENCRYPTED DATA

According the proposed scheme, in order to process queries over encrypted data the queries should be transferred to the encrypted version with respect to security plan, this task is designed to be conducted by our secure proxy. The security plan is discussed in Section IV, supplies the identity of the cryptographic modules to be applied to the different fields of the query. Figure 2 displays the processing and rewriting of a sample query.

![Diagram](image)

Fig. 2: The query `db.customers.find({salary: {$gt:5000}, balance: {$lt:2000}})` received from an application. (a) The parsing tree of the query (b) The cryptographic modules applied to the data elements according to the security plan

VI. RESULTS AND DISCUSSION

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

1) Encrypt/decrypt time of data and query;
2) Communication time between the server and clients;
3) Response time of server.

For our experiments we first created a sample database with million(s) documents and then determined the overhead of searching an encrypted database. To do so, we measured the database response time for queries when the documents were decrypted and when they were encrypted. Then, we measured the encryption and decryption time for different sizes of the ciphertext. We wanted to avoid reporting the response time, as the communication time introduces extraneous factors, irrelevant to the performance of the system we discuss.

The test environment setup was based on Linux operating system and we chose MongoDB 3.0.2, classified as a NoSQL document store database. The random data generator in JS, PHP, and MySQL format was generated by using a ready databases to produce million(s) plaintext documents in the data set. Each document has seven different attributes including name, email, salary. Afterward, we applied to the numeric data type, OPE 64, 128, 256 and 512 bit and the AES-DET 128 bit for the string data type of plaintext data set and generated four encrypted data sets of millions documents each. Finally, we uploaded all five datasets on the EC2 instance of AWS one the cloud. Once the MongoDB databases were created we run several types of queries including equality, greater than, less than, and equal to, less or equal to, and OR logical operations. For full description of the experiments refer to our technical report [13].

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. This explains why for three of the five types of queries, Q2, Q3, and Q4, database response time is slightly shorter the plain one when the keys are 32 bit integers. The results reported in Table I and in Figure 3 show the database response time for the five MongoDB experiments. Each query performed 100 times with disabled query cache and the average time calculated in milliseconds.

<table>
<thead>
<tr>
<th>Query type</th>
<th>Record count</th>
<th>32-bit plaintext</th>
<th>64-bit ciphertext</th>
<th>128-bit ciphertext</th>
<th>256-bit ciphertext</th>
<th>512-bit ciphertext</th>
</tr>
</thead>
<tbody>
<tr>
<td>Q1: Comparison</td>
<td>361,620</td>
<td>240</td>
<td>280</td>
<td>305</td>
<td>335</td>
<td>390</td>
</tr>
<tr>
<td>Q2: Equality</td>
<td>991,225</td>
<td>990</td>
<td>990</td>
<td>990</td>
<td>990</td>
<td>990</td>
</tr>
<tr>
<td>Q3: Range</td>
<td>991,225</td>
<td>370</td>
<td>350</td>
<td>360</td>
<td>380</td>
<td>400</td>
</tr>
<tr>
<td>Q4: Logical</td>
<td>1,031,820</td>
<td>660</td>
<td>660</td>
<td>660</td>
<td>660</td>
<td>660</td>
</tr>
<tr>
<td>Q5: Aggregation</td>
<td>1</td>
<td>660</td>
<td>660</td>
<td>660</td>
<td>660</td>
<td>660</td>
</tr>
</tbody>
</table>

TABLE I: 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.

Our measurements show that the response time of the NoSQL database management system to encrypted data depends on the type of the query. The shortest and longest database response time occur for Equality and Aggregated queries, respectively; for these two extremes the time for the unencrypted database almost doubles, 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. As expected, the OPE encryption time increases significantly with the size of the encryption space; it increases almost ten fold 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 to 1024 bit.

Proxy server is an important element for the proposed architecture; therefore, the potential attacks that could affect the proxy, also should be taken to considerations. 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 to the different targets. 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 best practical solutions.

VII. CONCLUSIONS AND FUTURE WORK

While 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 II. 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 that increasing the size of the co-domain of the OPE mapping function from $2^{64}$ to $2^{128}$, $2^{256}$, to $2^{512}$ results in an increase of database response time up to 5%, except for the case Q3-type queries in which the increase is significant. The penalty for using encrypted, rather than unencrypted NoSQL databases such as MongoDB is less than 5% for Q2, Q4, and Q5 which is considered to be relatively small. Moreover, the overall query response time is found to be dominated by the communication time depending on the connection bandwidth. The secure proxy is a critical component of the system, it is multi-threaded and the 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. Experimental results for multiple large datasets with millions documents show that SecureNoSQL is rather efficient.

REFERENCES


