SSRG International Journal of Electronics and Communication Engineering
ISSN: 2348-8549/ https://doi.org/10.14445/23488549/IJECE-V10I5P102

Volume 10 Issue 5, 14-27, May 2023

Original Article

CryptNoSQL – A Methodology for Secure Querying and Processing of Encrypted NoSQL Data on the Cloud Environment

Sridhar Vemula[1], Ram Mohan Rao Kovvur[2], Dyna Marneni [3]

[1]*Osmania University, Hyderabad, India*
[1, 3]*CSED, Maturi Venkata Subba Rao (MVSR) Engineering College, Hyderabad, India.*
[2] *ITD, Vasavi College of Engineering, Hyderabad, India.*

[1]*Corresponding Author : sridhar_cse@mvsrec.edu.in*

Received: 02 March 2023 Revised: 08 April 2023 Accepted: 03 May 2023 Published: 31 May 2023

Abstract *- Businesses today use modern computing technologies such as Big Data and Machine Learning in their daily operations, which require effective management of large amounts of data. Relational data formats are no longer suitable for these applications, and NoSQL data formats are preferred. The use of cloud infrastructure offers advantages such as scalability, availability, and resource maintenance, but data security remains a challenge. Although cloud vendors provide encryption features, they may not be sufficient for sensitive data. To address this, some businesses use their encryption methods, but retrieving data from an encrypted form may not be possible. While specific encryption methods support the processing of encrypted data without decryption, there is no complete implementation of secure processing for NoSQL data from MongoDB or other databases. The proposed methodology, called CryptNoSQL, provides a secure way to query and process NoSQL data, including updates on encrypted data. We introduce a customized database design model that selects an appropriate encryption method during the insertion of a document based on the type of field and the operation it will be involved in. Our experimental results demonstrate that our approach is suitable for organizations with sensitive data hosted on the cloud and that require frequent query operations on this data.*

Keywords *- Secure data processing, Cloud security, Homomorphic, Order processing encryption, NoSQL.*

1. Introduction

As computing increases in various domains, a vast amount of data is generated. The domains are not specific, but it covers many areas like social networking, public relations, health care systems, banking systems, security management systems, government body management etc. These types of domains use various forms of data, including text data, numeric data, custom objects, date, email, custom objects, audio, video, animations etc. In some domains like banking and finance, transactions securing data while operating is of utmost importance. On the other side, as the data generated is enormous, organizations prefer to store and operate the data from Cloud. Many public cloud providers offer various data protection techniques to secure data stored in the cloud. However, the techniques offered by cloud providers to secure data are under the control of cloud administrators, and there are chances of data leakage by any curious cloud administrator. Because of this reason, security techniques provided by cloud services are not recommended. Hence it is required to encrypt the data using our encryption method before uploading it to the cloud. The application should support retrieving and querying this encrypted data without compromising security. Continuous research is ongoing on querying and retreating information from various forms of encrypted data without decryption. The various forms of data include plaintext, relational data(SQL), non-relational data(NoSQL), images, videos etc.

Data security is not new and was considered over a decade ago. Security Methods and level of access keep changing according to the type of applications used along with data, the format of data and its access methods. Suppose the data or information is in its format without storing it in a data store or database. In that case, security becomes easy as the same can be encrypted depending on its format. However, applications involving multiple data forms use data storage or database. In this scenario, careful observation is needed to identify which part of the data is to be secured. Initially, Relational Database is the popularly used data store depository best suited for various categories of applications. Some of them are Employee Management

Systems, Payroll Systems etc. The data stored in the database is considered secure if stored in a Private Secured Network of organizations. However, once the data is moved to the cloud environment, security is the primary concern as it is part of the Public Network [1-3]. CryptDB [4] provides one of the solutions for this problem. The author created a proxy where data is encrypted before uploading it to the database, and a given query can be operated directly on Encrypted content.

As computing technology improves, the type of data used in their applications are also changing. Nowadays, most applications are making use of data in a Non-relational format. NoSQL is a "Not Only SQL" format used in most applications. Data is stored in various types [5] in NoSQL, including Key-value pairs, Column-oriented, Document Store and Graph-Based.

Choosing an Appropriate Security Mechanism for data stored in these various models is very important [6]. In Secure NoSQL[7], the author proposed a proxy architecture which encrypts numeric data with Order Preserving Encryption (OPE) and Textual data with AES. The NoSQL query operates on Encrypted Data by encrypting the data values in the query and executing them on a NoSQL database. This work is implemented on Key-value pairs in JSON format.

The proposed system only compares sorting, groups and equality. Any modifications to data stored in databases, like additions and multiplications, are not supported in Secure NoSQL [7]. In [8], The author proposes a similar scheme as mentioned in Secure NoSQL [7] by removing separate security plans. In [9, 10], the author proposed Security-as-a-Service for NOSQL databases. This system does not support updating a numeric value over Encrypted data. In [11], The NoSQL database is decomposed into two parts - one with Numeric data, which contains indexed and numeric columns.

The remaining data does not contain fields to compare. The first part is encrypted with OPE to enable comparisons over Encrypted data [12]. The second content database part is encrypted with AES, which only retrieves documents by query results from the first document. As the data is decomposed, retrieving all data from both partitions takes extra processing time.

Moreover, this does not support updating a numeric value with addition or subtraction. All existing proposed models to secure NoSQL databases use OPE along with AES, which supports only Queries with comparisons, sorting, logical and grouping operations. They do not support updating queries by adding or subtracting a value.

This paper proposes a methodology that supports all query operations, including addition or subtraction in various queries. We use OPE, AES, along with Homomorphic Encryption to accomplish this. We also use a customized plan to increase throughput by using a customized plan to decide on which column to be used for encryption involving what type of operations.

The main contributions of this paper are:

- Provides customized NoSQL database design to choose the type of encryption used based on the type of operations
- Proposes a methodology which supports various types of querying on encrypted NoSQL database based on conditions
- Proposes a methodology to support updating encrypted NoSQL data, including adding or multiplying a value based on some conditions without data loss.

The rest of the paper is organized as follows. In section 2, we discuss standard encryption methods used in our work. Section 3 discusses designing a customized security plan according to application requirements, and the proposed methodology of CryptNoSQL is explained in Section 4.

Section 5 presents the system to implement the common NoSQL operations. In section 6, the Implementation of CryptNoSQL is discussed. Finally, in section 7, the results and performance of the work are presented.

2. Preliminaries

The backbone of any cryptosystem is its fundamental encryption methods used. This section discusses various public and private encryption methods proven to be strong enough for any secure processing models. As part of our proposed work, we use the following encryption systems.

2.1. AES-DET Encryption

Advanced Encryption Standard (AES) is a popular encryption method which provides strong security in the context of various attacks. The key size used in AES could be 128, 192 or 256 bits, and the block size is 128 bits. With Pseudo Random Initialization Vector (IV), AES generates different cypher text for the same plain text elements. Because of this feature, it is considered an RND encryption method. In the context of processing Encrypted data without decryption, AES-RND only helps retrieve a single document or complete database and does not help retrieve data with queries, including conditions.

In order to provide an equality comparison, DET encryption is used. DET encryption generates the same ciphertext values for the same plain text elements. Hence it provides support for queries involving equality comparison. AES with fixed IV can be implemented as DET encryption, wherein it provides Strong Encryption (AES) along with query execution (only equality) on Encrypted data.

2.2. OPE

Order Preserving Encryption (OPE) was proposed [13]. As its name suggests, it preserves the order between data values even after encryption. Although it is not completely strong and leaks the order information, it is suited for requirements when data security and processing support on encrypted forms are required.

Let x and y be plain text elements and, after encryption with key K, generate OPE(x) and OPE(y).

As per the principle of OPE, it retains the order between the encrypted values. i.e

If x < Y, then OPE(x) < OPE(y) will be satisfied.

2.3. Homomorphic Encryption

Homomorphic Encryption allows computations to be performed on Encrypted Content without Decryption [14]. After much research, [15] provides the algorithm for Fully Homomorphic Encryption, which supports both Multiplication and Addition on encrypted data [16].

However, it is not easy to implement. Because of its difficulty, Fully Homomorphic Encryption can be categorised into two types [17]: Additive Homomorphic Encryption and Multiplicative Homomorphic Encryption. An additive homomorphic algorithm has the property as shown in Equation 1.

$$E(x) * E(y) = E(x + y) \quad (1)$$

As Equation 1 shows, the sum of two encrypted values can be found by multiplying two.

Whereas, Multiplicative homomorphic encryption has the following property.

$$E(x) * E(y) = E(x * y) \quad (2)$$

As mentioned in Equation 2, the multiplication of two values can be obtained by decrypting the multiplication of two encrypted values. Additive Homomorphic Encryption can be implemented by Paillier [18] Cryptosystem, and Multiplicative Encryption can be implemented by RSA[19] and ElGamal [20] schemes.

In our proposed work, Additive Homomorphic Feature supports Addition operations over Encrypted Content. Paillier Encryption method encrypts numeric values before storing them in a NoSQL database to support Addition operations on Encrypted Content. Similarly, Multiplicative Encryption supports Multiplications over Encrypted data. RSA Encryption is used to encrypt values before storing them in a database.

3. Customized Design Plan

The main contribution of this paper is to propose a methodology which makes querying and updating values possible even on encrypted data. We use various encryption algorithms to achieve this. Which security system to be used depends on the column type and required operations in which the column is going to involve. In general primary index key or field of any NoSQL collection is not involved with any comparison or update operations. As a result, the index fields are encrypted with only AES, which supports querying documents based on equality and provides high security. The properties of documents, which involves comparisons only, are encrypted with OPE. Another type of field involved in updations frequently is encrypted with Paillier Encryption and OPE encryption to support addition and multiplication, respectively.

The encryption can be customized with a customised database design before data uploading. The key points about customized database design are:

- For the more sensitive fields, they are to be encrypted with AES, which provides strong security. This field supports querying using only the Equality check.
- The fields that involve only comparisons can be encrypted with OPE so that this field supports querying based on equality and comparison.
- The fields that involve updations will be encrypted using Paillier and RSA to support additions and multiplications on encrypted data.
- Fields involving both Querying based on comparisons and updations will be encrypted with OPE, Paillier and RSA so that they supports Querying based on comparisons and updations.

This process of creating a customized security plan is shown in Figure 1 and described in Table 1. A sample employee JSON document representing it after storing it in the NoSQL database using a customized security plan is shown in Figure 2 for reference. While storing the documents, column and table names are also encrypted with AES. For understanding purposes, column names are shown with actual names in Figure 2.

4. Proposed Methodology

With the help of customized database design, the proposed system ensures to execute comparison and update queries on encrypted NoSQL data hosted on the cloud. To provide security to data at all levels of operations, a layer of functionality is required to encrypt new data to be uploaded or any involved in the query before sending it for execution. This functionality is implemented in the Intermediate layer or client system. At the server, another layer of functionality is required to implement homomorphic updations on encrypted data.

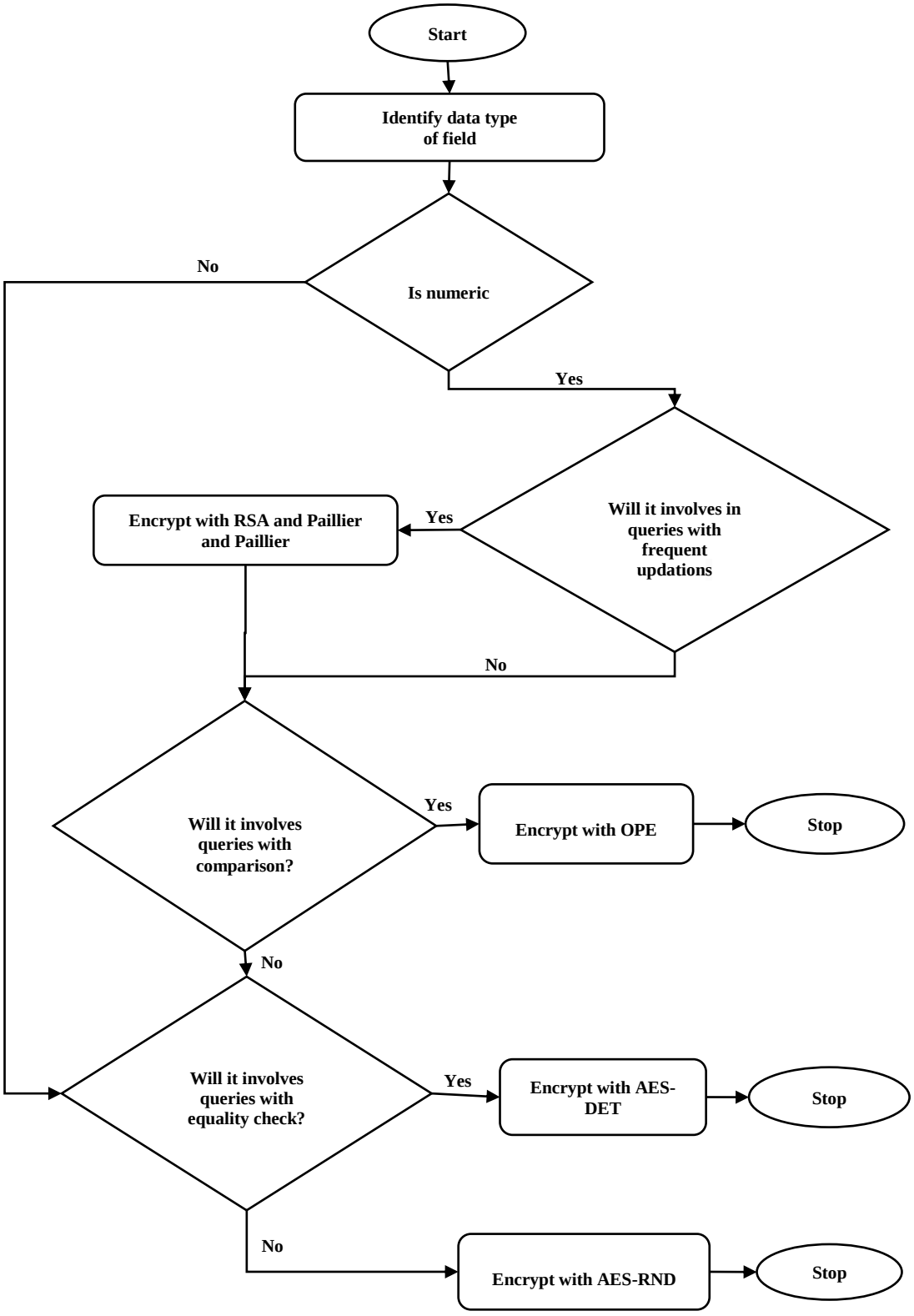

Fig. 1 Customized database design plan for NoSQL databases

```
_id: ObjectId('63cf8710eae2e24e5b2c52fa')
id: 1
name: "Mohammad, Ahmadian"
email: "turpis.non@id.ca"
Salary: 716046
ssn: 936136916
ccn: "4916-318-39-2134"
balance: 285462
```

```
_id: ObjectId('63e442ea1d59bd405d78330b')
Salary: 516046
Salary_ope: 131961126
Salary_rsa: 5186525461178877334
Salary_plr: 1952666222930786234
balance: 285462
balance_ope: 72983485
balance_rsa: 21253951774255162
balance_plr: 4800975883838568279
name: "JBV4aL2TTB25yqBlpt1eMn685sPtiQSigsZ2i+779WU="
id: 1
id_ope: 410
id_rsa: 1
id_plr: 4318305493261716527
email: "FoaTyT4QR5eKQ4pa3sCi9Cffefn7w6eRrICCLV1ZUzk="
ssn: 936136916
ssn_ope: 239645134391
ssn_rsa: 4051752315387351331
ssn_plr: 130732262728492435
ccn: "169CmBGMZ7sMYB2+zbOT0iffefn7w6eRrICCLV1ZUzk="
```


Fig. 2 Sample employee JSON representing encrypted document in database

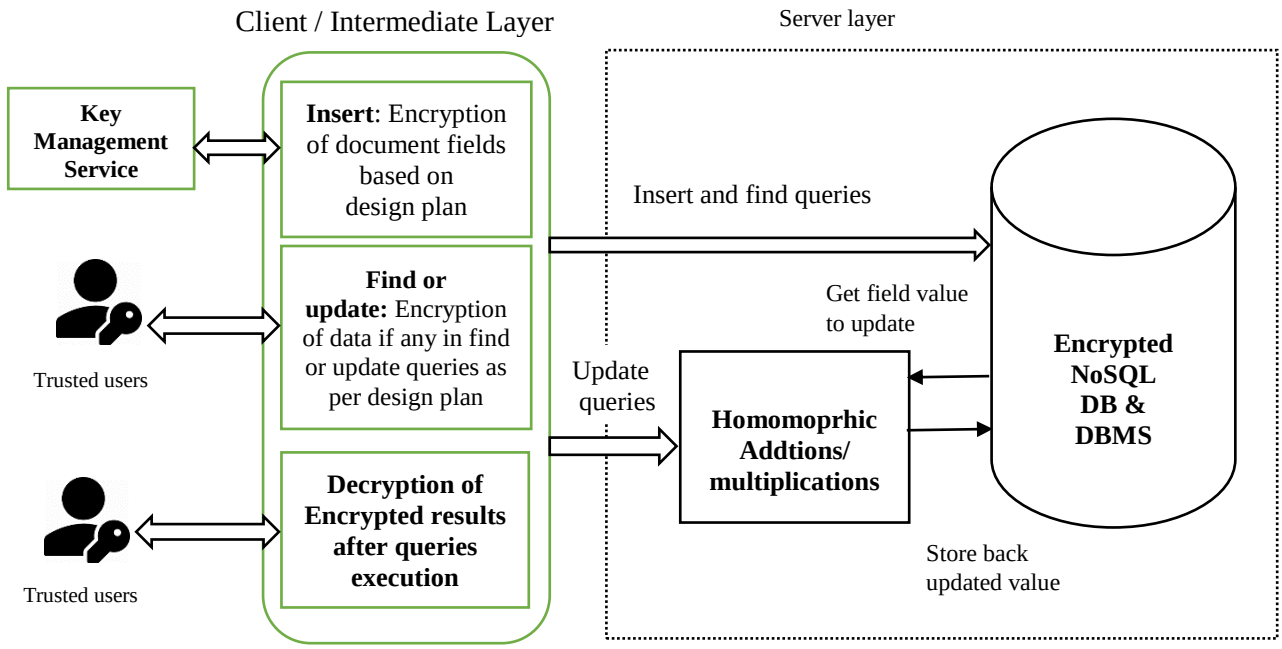

Fig. 3 System model

The system design of our proposed system is shown in Figure 3.

The proposed work is divided into three modules,

1. Intermediate Layer/ Client Layer
2. Server Layer
3. Key Management Module

4.1. Intermediate Layer / Client Layer

The functionality of this layer is to deal with various querying requirements of the client. One functionality is encrypting document keys and values according to customized database design and sending the encrypted document to the NoSQL database server running on the cloud. Another functionality encrypts any comparison values or operands in querying or updating queries. The administrator shares private keys with trusted clients based on requirements.

Table 1. Customized database design plan

Type	Encryptions used	Encryption Type
If the field involves only equality, check	AES	Private Key Encryption
If the field involves only comparison	OPE	Private Key Encryption
If the field involves only numeric updations	Paillier, RSA	Public Key Encryption
If fields involve both comparisons and updations	OPE, Paillier, RSA	Public Key Encryption

This key is used to decrypt the results of query processing. This layer can be included as a separate layer of functionality and run as a mediator between clients and server database systems. It is possible to include this functionality in the client system, provided the client should have the facilities to implement it.

4.2. Server Layer

This layer of functionality becomes active at the time of executing updations operations. As stated above, to support. In homomorphic encryption, the fields are encrypted using RSA and Paillier.

Adding or multiplying a value with any encrypted value stored in the database cannot be done as the size of the result will not fit into the database. In this case, the update operations are implemented as follows:

Step-1 : The server layer retrieves the value stored in that database that needs to be updated

Step-2 : It also receives encrypted operands from the client layer.

Step-3 : The server layer implements homomorphic addition/multiplication using modified algorithms based on the type of operation needed. The output of this step is adjusted so that it does increase in size.

Step-4 : The newly generated value is then stored back in the database.

All the above steps are carried out entirely on encrypted content. Hence complete security of the data is guaranteed despite the implementation of various operations.

The server Layer runs in the same enrolment where the database is hosted. The reason behind this is to execute update() queries faster. There is little delay in connecting and fetching the data from the NoSQL database as the server layer runs in the same environment where the database is hosted.

4.3. Key Management Service

Key Management Module responsible for generating vital keys for Encryption Schemes used in the system. The Encryption methods used in our system are

1. Advanced Encryption Standard (using fixed IV): This Encryption scheme encrypts text data and other values that are not involved as query parameters.
2. Order Preserving Encryption (OPE): This Encryption scheme supports queries with comparison and sort operations. This scheme maintains the order of plain text elements even after decryption; hence it helps in processing queries involving comparison and sort operations. We are using the Boldyreva [21] implementation of OPE.
3. Homomorphic Encryption: This encryption scheme allows computations on Encrypted content—the ordinary update operation on a field of NoSQL data stored in addition or multiplication. For additions, the Paillier Encryption method is called Additive Homomorphic Encryption. For multiplications, RSA encryption is termed Multiplicative Homomorphic Encryption.

The Key management module generates and manages the keys between users and data administrators. The Encryption algorithms used in our works are AES-DET, OPE, Paillier and RSA. Table 2 shows the Encryption algorithm name and size of the key and cipher text.

The 128-bit key size is the most commonly used key size for AES, and it provides a very high level of security. The 192-bit and 256-bit key sizes provide even more robust security but may be slower and more resource-intensive. The same thing applies to OPE also. Hence we are using AES and OPE with key sizes of 128 bits.

RSA and Paillier, being a public key encryption system with a 2048-bit key, provides a high level of security and is generally considered secure for most purposes. However, as computing power increases, larger key sizes may be necessary to maintain the same level of security. As RSA and Paillier are involved in computations, we tested performance and the operations with key sizes of 128 bit, 256 bit, 1024-bit, and 2048-bit. The key management module's role is to share the Keys generated in the initial process with respective users in a secure channel. All the generated keys are stored as Map, which is also encrypted separately. The

Encryption keys are shared with authorized clients so that retrieved data can be decrypted on the client side after completion of querying.

5. NoSQL Operations

This section covers the implementation process for commonly used NoSQL operations. These operations are essential functions for applications that use NoSQL data to meet business requirements. In MongoDB, these operations are called functions, not commands.

5.1. Storing Documents – db. collection.insert()

The storage module is part of the Client / Intermediate layer responsible for encrypting the data to be uploaded into a NoSQL database. The new data to be uploaded can be a single document or multiple documents in the form of a JSON file. If it is multiple documents, each document will be read from a JSON file and encrypted before storing it in the NoSQL data store.

The functionality of the storage module is depicted in Figure 4. After reading the data from input documents, the element type is retrieved. The encryption algorithm will depend on the Customized design plan discussed in the previous section. The encryption method is applied to field values in such a way that it supports respective query operations. This is summarized in Table 3.

5.2. Reading the Documents Based on Search Criteria – db. collection.find()

Retrieving documents from any NoSQL database can be done in various ways. Retrieve one particular record based on an equality condition on the index field, retrieve some documents based on a condition or retrieve all documents. The query contains input values in the first two cases to check for equality and comparisons. Hence the input values are to be encrypted using OPE before executing the query.

In the third case, because there are no input values, the query with only encrypted column names can be executed directly. In the first two cases, the type of encryption is chosen based on an input value. If the input value is numeric and the search criteria contain comparison, then the given input value is encrypted with OPE. The table and column names are also with AES for all queries before sending them to the MongoDB server. If the input value is not numeric and the search criteria are equal, it is encrypted with AES to compare for equality. This complete process is shown in Figure 5.

5.3. Updating One or More Documents - db.collection. update() with $inc and $mul

There are some situations where values in some documents need to be updated. An example salary of an employee needs to be incremented in the payroll system, or the account balance needs to be added to the deposited amount in the banking system etc. Using the Homomorphic encryption property, updating the value of numeric fields can also be done on encrypted data without decryption.

According to the homomorphic property, to perform addition or multiplication on encrypted data, the ciphertexts need to be operated upon, which yields the same result as the actual addition or multiplication of the plaintexts. However, the multiplication of two ciphertext numbers results in large numbers that may not fit the database filed size. With the algorithms proposed in our previous contribution [22], it is possible to perform unlimited additions or multiplications on ciphertexts without increasing the size of the result, making it suitable for storage in a database at any time. Update queries generally contain an operand to be added or multiplied with document property value.

In this case, the type of encryption used to code a given operand depends upon the type of updation. Suppose the query is trying to add an operand value to the database. In that case, it is encrypted with Paillier encryption, or if the query is trying to multiply some value with the database, then the RSA cryptosystem will be used. Because of their Homomorphic property, we can update the Database content with additions or multiplications without even decrypting the destination value. This process is shown in Figure 6.

6. Implementation

The proposed work CryptNoSQL applies to document-based NoSQL databases. This work can be extended to other models of NoSQL databases with slight modifications in the module. MongoDB is one of the most popular document-based NoSQL databases, so the proposed work is implemented on this.

The same can be applied to other document-based NoSQL databases also. We have installed MongoDB 5.0.9 Enterprise version on Windows 10 operating system running on Intel Core-i5 8th generation platform with 8GB RAM. The proposed CryptNoSQL is tested on a sample json dataset downloaded from Github generated by MdAhemdian [23]. We have used one lac document of plaintext data from this dataset. We have created four NoSQL databases with 256-bit, 512-bit, 1024-bit and 2048-bit key sizes concerning RSA and Paillier encryption systems.

The processing time of inserting, querying, and updating data are compared. All four databases are uploaded in MongoDB, and performance is monitored by executing various queries on all three databases. The work CryptNoSQL is implemented in JAVA, and it is running under Java VM version 8. The Java components are connected to the MongoDB database using the API provided by MongoDB [24, 25].

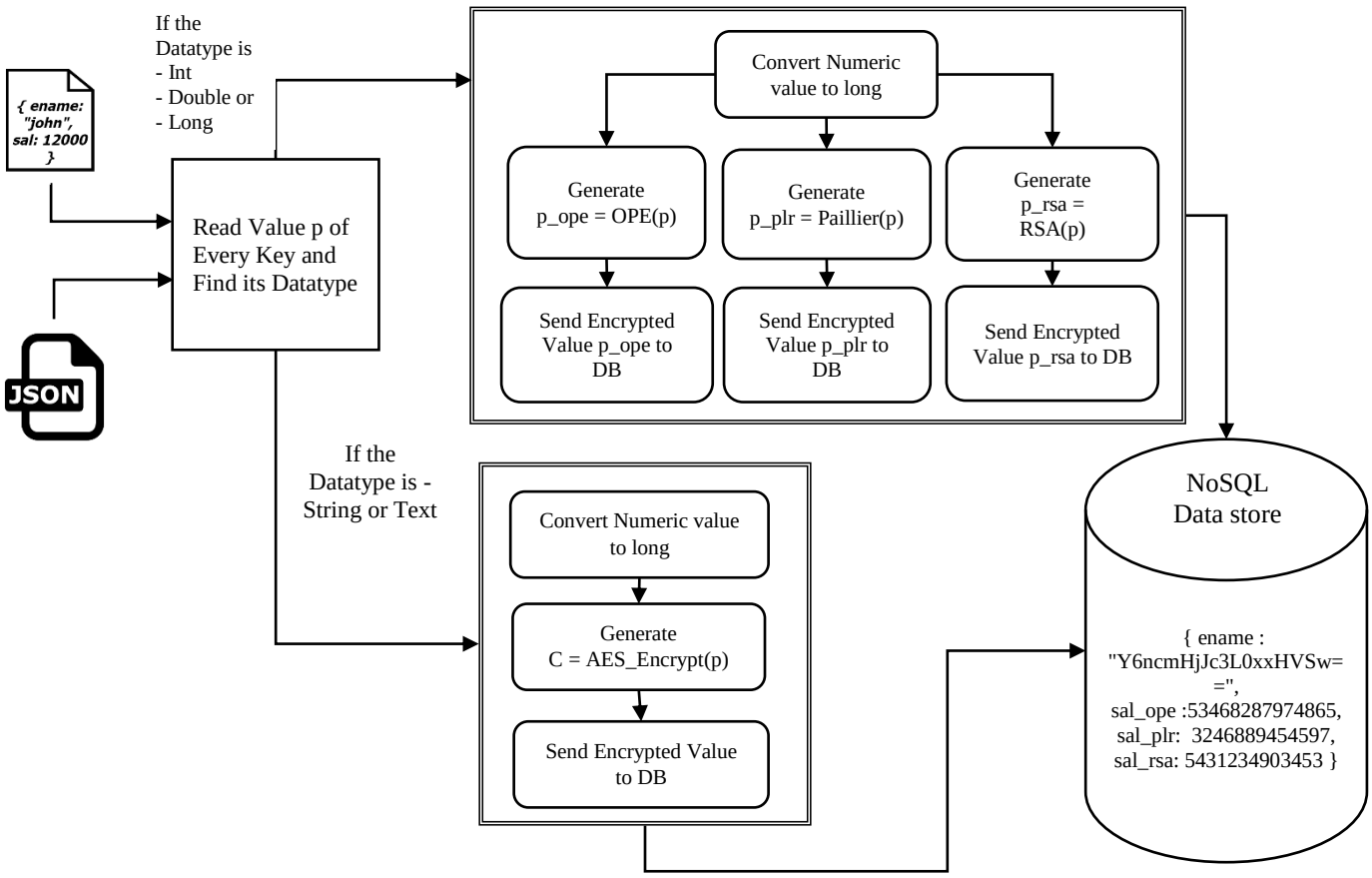

Fig. 4 Storage model

Table 2. Key sizes of various encryption methods used in our work

Name of the algorithm	AES DET	OPE	Paillier	RSA
Key Size	128 bits	128 bits	512 – 2048 bits	512 – 2048 bits

Table 3. Summary of encryption methods and their supported NoSQL operations

S.No	Encryption methods	Supported NoSQL operations	Example
1	AES	find() with an equality condition	db.emp.find() db.emp.find({ name:"John"} }
2	OPE	find() with comparisons and equality MIN() and MAX() aggregations	db.emp.find({$gt: {sal : 500000} } $min and $max operators
3	RSA	updateOne() or updateAll() with $mul operator	db.emp.updateOne({ name:"John"} , { $mul : {sal: 2} })
4	Paillier	updateOne() or updateAll() with $inc operator supports SUM(), AVG()	db.emp.updateOne({ name:"John"} , { $inc : {sal: 100000} }) $sum , $avg

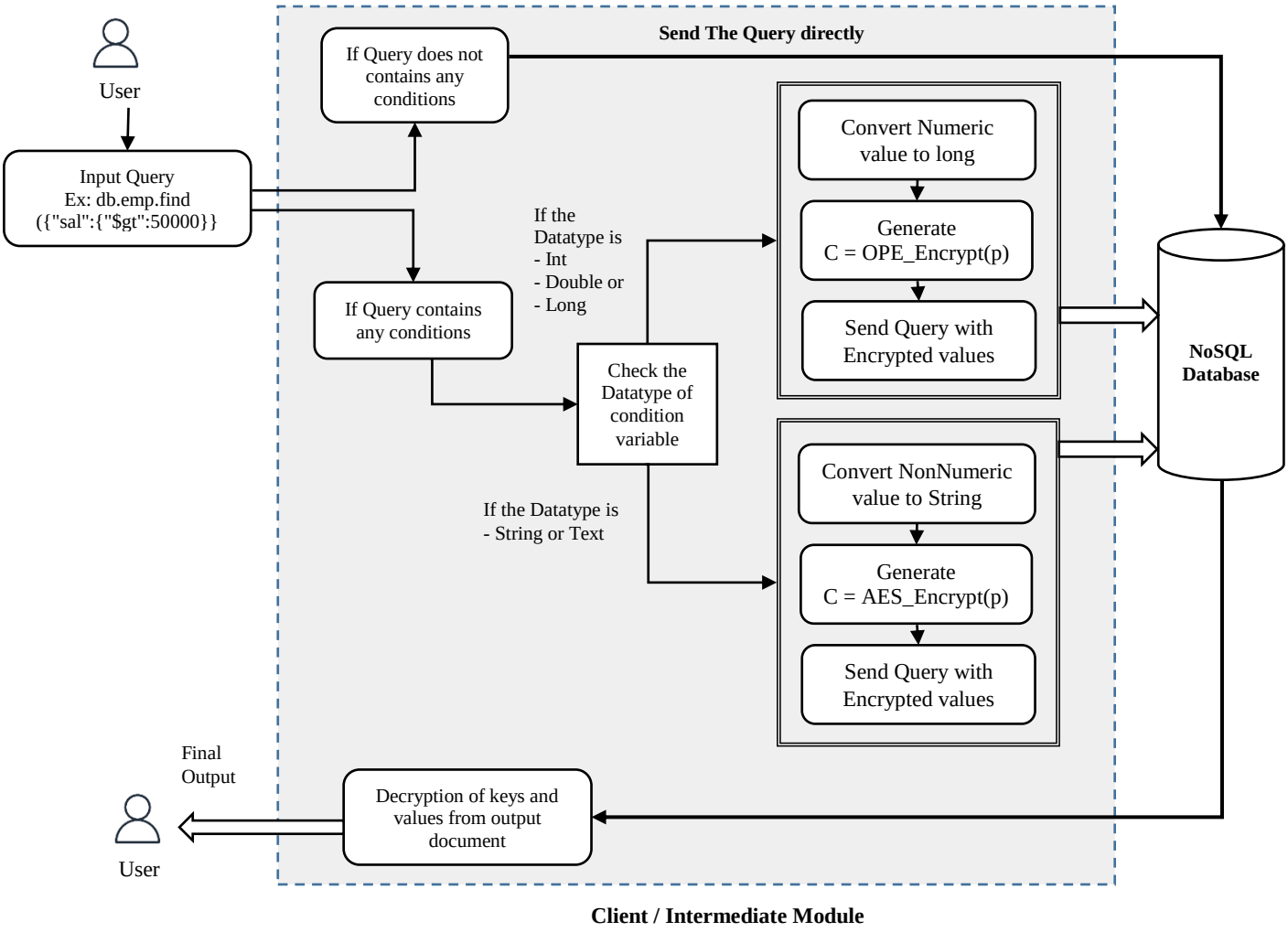

Fig. 5 Processing of queries with conditional operations

7. Results and Discussion

Various queries are executed on four databases prepared with data encrypted with different key sizes to measure the performance. The module developed in JAVA works as a proxy and takes care of the following activities:

Upload process: Encrypt the numeric with OPE, RSA, PLR encryption and non-numeric data with AES while uploading any dataset into the NoSQL database.

Query process: Encrypt condition variables, retrieve the documents based on condition and decrypt the results at the client side.

Update process: Encrypt the operand variables, perform modifications on the encrypted document(s) and decrypt the results if any.

The performance of this system can be measured by executing the various queries over encrypted and unencrypted data. The processing time of an operation contains the following components:

1. Time required to encrypt table name, column name and any values used in the query
2. The communication delay between the client and the server
3. The response time of database systems to execute queries on the NoSQL database.
4. If any, time is required to decrypt the results at the client side.

The communication delay depends on various factors like server locations, network speed and other characteristics. The communication delay is unrelated to our work and is not considered. Decrypting the results is also not considered because it does not apply to all queries. For example, this part is not considered for queries related to updating the contents of documents. In summary, the time required for encrypting the table name, column name, operand values and time required for response time by the database management

system is considered for performance evaluation. The queries to be executed on CryptNoSQL can be categorized into three groups.

1. Find() queries to Read the documents based on certain conditions
2. Update queries which include addition or substitution operations
3. Update queries which include multiplication or division operations

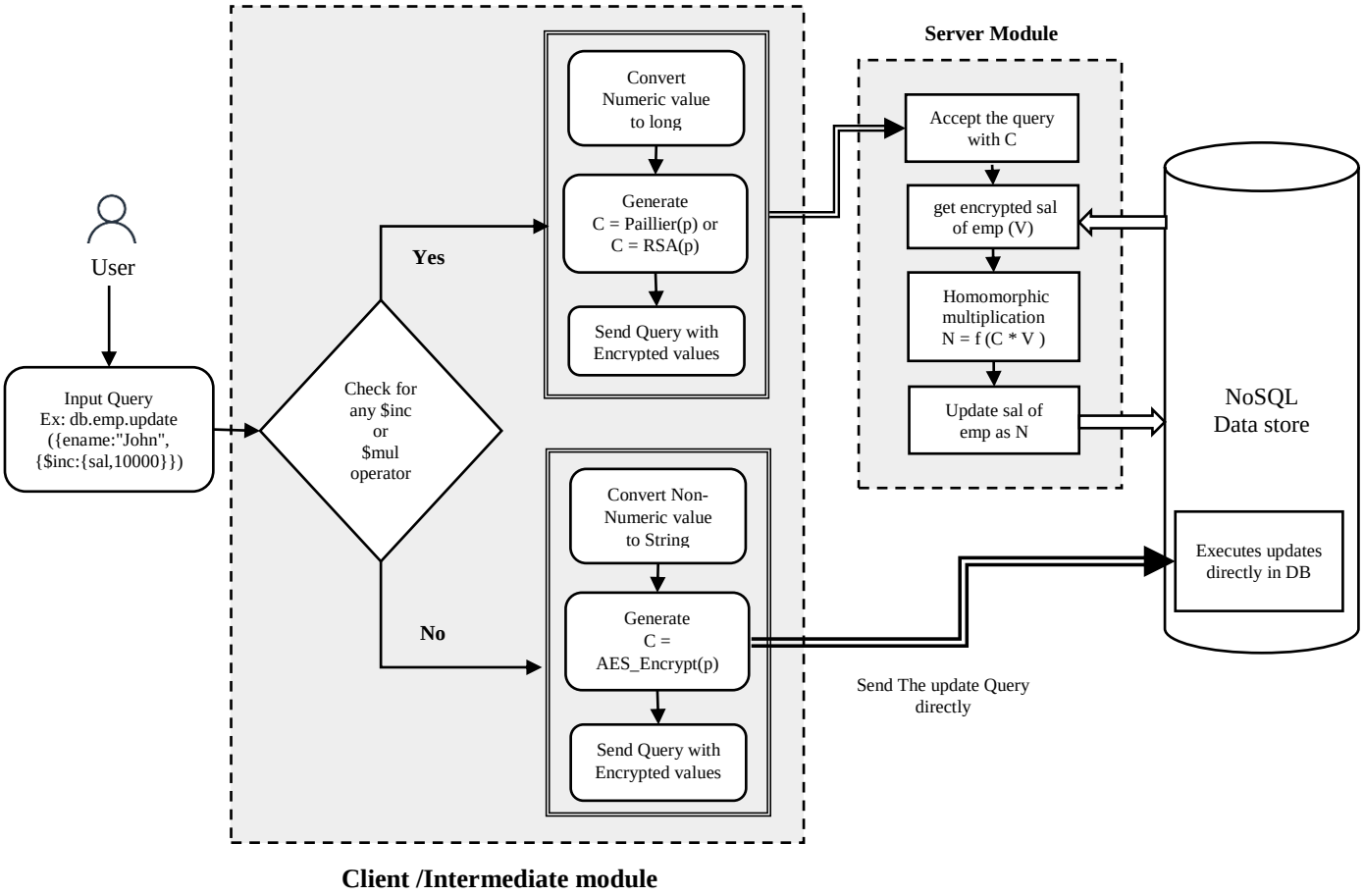

Fig. 6 Processing of queries with update operation

In NoSQL database applications, data is queried more often than inserted. Data insertion can occur through initial bulk uploads or dynamically inserting individual documents as needed. As a result, the performance of insertion operations is not typically evaluated because they occur infrequently and have minimal impact on the overall system performance. In the following subsections, we will discuss the query evaluation for find() and update() operations.

7.1. Reading Documents with Comparison Operators

In the first category, the find() method is used to select or retrieve all documents or some documents based on specific conditions on document variables. For example, a query can retrieve documents of all employees where the employee salary is greater than some threshold value. The corresponding query is given below:

```
db.data.find({sal:{$gt:500000}})
```

This query retrieves all the documents of employees with CTC greater than five lacs. In this proposed system, all numeric values are encrypted with Order Preserving Encryption, Paillier Cryptosystem and RSA Encryption to Support comparisons and perform addition and multiplication operations on documents.

In comparison, the operand value used in the query is encrypted with OPE and table and column names are encrypted with AES. The Processing time of the above query includes the time required for encrypting the table name, column name using AES and comparator value using OPE. The table name and field name encryption is constant for all types of queries and hence is not included in performance analysis.

We have generated three MongoDB databases containing 100,000, 500,000, and 1,000,000 records, respectively. We evaluated the processing time required to execute a sample query on all three databases, and the results are presented in Table 4. The corresponding graph is shown in Figure 7.

Figure 7 illustrates that the response time does not increase proportionally with the database size. Consequently, there is a slight decrease in response time as the number of documents in the database increases.

7.2. Updating Documents

Some update operations include replacing a property's existing value with a new value or performing an operation with existing value. In the first case, it is possible to replace the other existing value with a new encrypted value using the equality condition in the query.

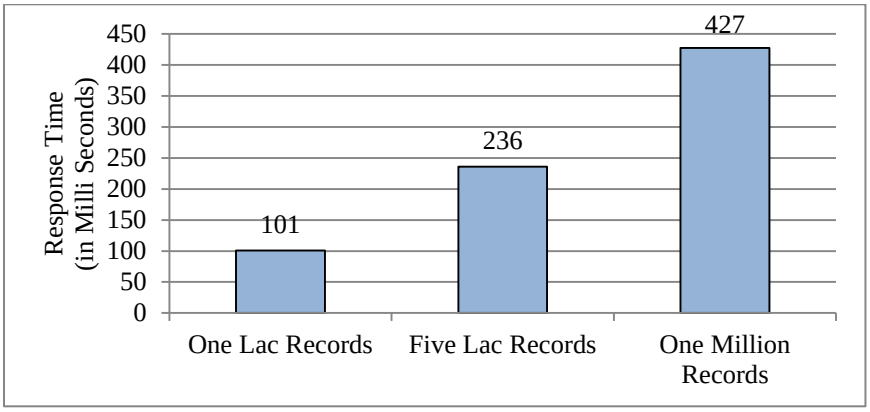

Fig. 7 Response time comparison for query operation with OPE

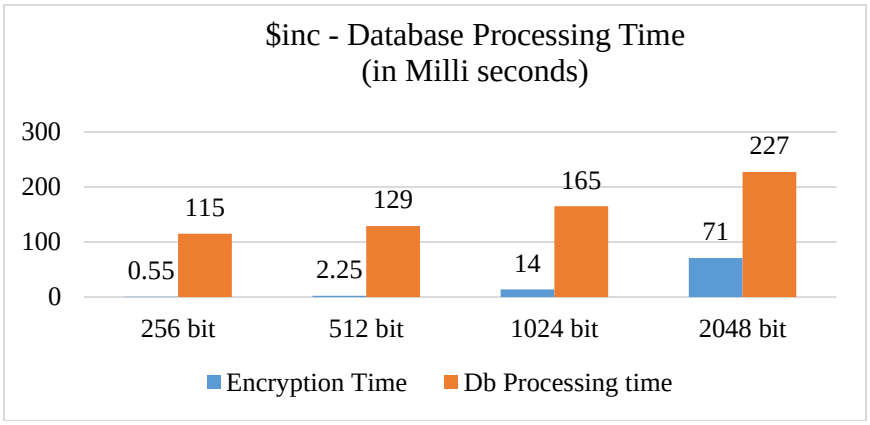

Fig. 8 Encryption and processing time of updating a document with $inc operator

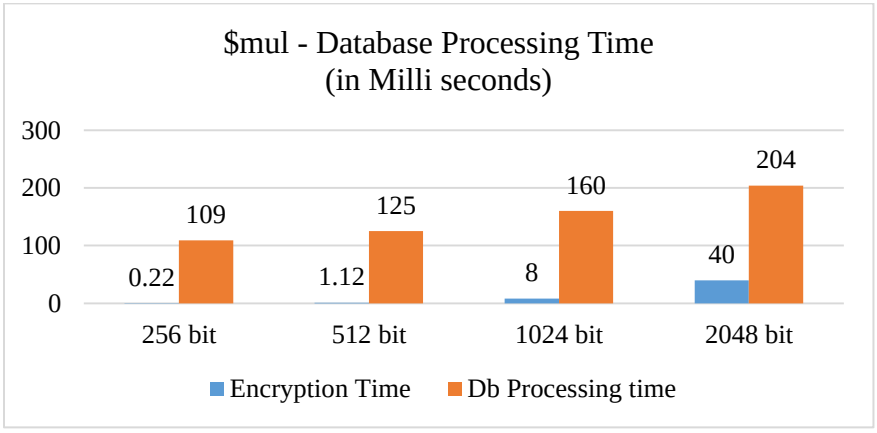

Fig. 9 Encryption and processing time of updating a document with $mul operator

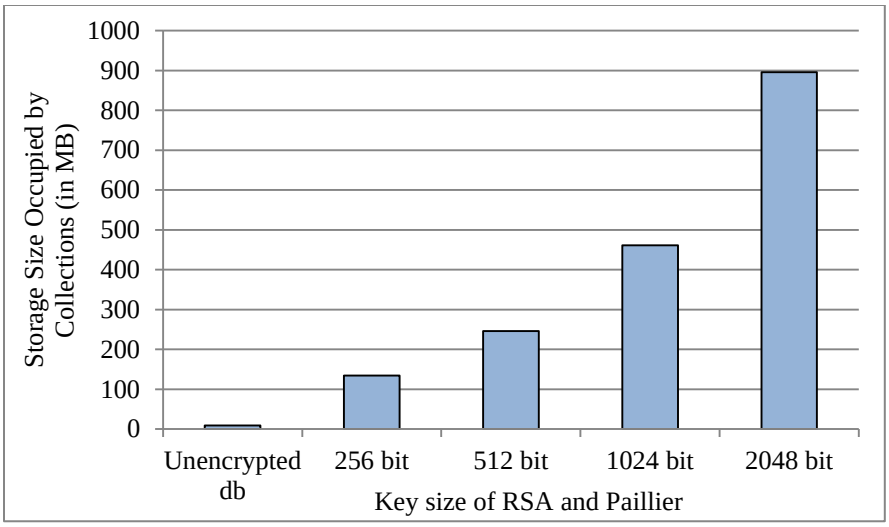

Fig. 10 Storage size of NoSQL databases with different key sizes

Table 4. Processing time of executing find() with comparison in various database sizes

	Processing time in milliseconds		
	Database with one lac documents	**Database with five lac documents**	**Database with ten lac documents**
Number of records returned	55606	277846	556112
Average encryption time of OPE (to encrypt comparator value)	6	6	6
Database processing time (time taken for query execution by MongoDB)	95	230	421
Total time taken for find() query with comparison	101	236	427

Table 5. Encryption and processing time of updating a document with Paillier encrypted field

	Paillier encryption Key-size			
	256 Bit	**512 bit**	**1024 bit**	**2048 bit**
Encryption Time (in Milliseconds)	0.55	2.25	14	71
Db Processing time (in Milliseconds)	115	129	165	227

Table 6. Encryption and processing time of updating a document with $inc with RSA encrypted field

	RSA encryption Key-size			
	256 bit	**512 bit**	**1024 bit**	**2048 bit**
Encryption Time (in Milliseconds)	0.22	1.12	8	40
Db Processing time (in Milliseconds)	109	125	160	204

Table 7. Storage overhead

	Key size of RSA and Paillier				
	Unencrypted db	**256 bit**	**512 bit**	**1024 bit**	**2048 bit**
Storage size in (MB)	9.27	134	246	461	896

In the latter case, existing value will generate new value. Some of the operations to generate new value include adding, subtraction or multiplication with new value. Paillier or RSA cryptosystems with Additive or Multiplicative homomorphic properties support these operations.

The performance of update operations is evaluated against four NoSQL databases of One lac records encrypted with 256-bit, 512-bit, 1024-bit and 2048-bit key encryption with RSA and Paillier encryption systems. The document already has different encrypted values using Paillier and RSA to support additions and multiplications.

An example update query is implemented with two cases.

Case-1 : To increase an employee's salary by Rs.1,00,000-00, the operand value is first encrypted using the Paillier encryption scheme. The encrypted value is then transmitted to the server module, which retrieves the corresponding encrypted salary of the employee and performs a homomorphic multiplication between them. The resulting value is capped to the actual size of the salary field and stored back into the database. Figure 8 and Table 5 provide information on the processing times required for adding Rs.1,00,000 to an employee's salary.

Case-2 : To triple an employee's salary, the operand value of 3 is first encrypted using the RSA encryption scheme. The encrypted value is then transmitted to the server module, which retrieves the corresponding encrypted salary of the employee and performs a homomorphic multiplication between them. The resulting value is capped to the actual size of the salary field and stored back into the database. Figure 9 and Table 6 provide information on the processing times required for multiplying an employee's salary by a value of 3.

The Figure 8 and Figure 9 shows that, the processing times of the NoSQL database are not increasing with encryption key sizes of RSA and Paillier encryption. It performs well with the increased key size and provides better security.

7.3. Storage Overhead

Encryption algorithms generate cypher text of more length for a given plain text to ensure maximum security. In CryptNoSQL, all the fields are encrypted to provide secure processing of NoSQL data. This results in an increase in storage size. Table 7 and Figure 10 show the storage size occupied by four NoSQL databases, each with one lac records encrypted with 256-bit, 512-bit, 1024-bit and 2048-bit encryption.

8. Conclusion and Future Work

This work proposes a secure implementation of various NoSQL operations, including updations. The work is implemented as a proxy, which operates on the top of the NoSQL database and takes care of the encryption and decryption process while uploading documents into the NoSQL database and while querying. The proxy is also responsible for updating documents based on given criteria without decryption.

This work provides a secure way of executing most of the common operations on NoSQL databases without decrypting the data on the server side. Hence it ensures that no data leakages the while retrieving and querying also.

This work is suitable for NoSQL data models of Key-value pairs and Documents based, and we will extend the same work to other NoSQL database models, graph-based and object-based models.

References

[1] N. Chandrakala, and B. Thirumala Rao, "Migration of Virtual Machine to Improve the Security in Cloud Computing," *International Journal of Electrical and Computer Engineering*, vol. 8, no. 1, pp. 210-219, 2018. [CrossRef] [Google scholar] [Publisher Link]

[2] Suraj Krishna Patil, and Suhas B. Bhagate, "Protecting Data in Relational Database Management System using Purpose and Role Based Access Control," *International Journal of Computer Engineering in Research Trends*, vol. 4, no. 8, pp. 336-340, 2017. [Publisher Link]

[3] S. Ravichandran, and R. Rajkumar, "Design and Development of Communication Salvage upon Encrypted Information in Cloud Computing," *International Journal of Recent Engineering Science*, vol. 6, no. 6, pp. 17-22, 2019. [CrossRef] [Google Scholar] [Publisher Link]

[4] Raluca Ada Popa et al., "CryptDB: Protecting Confidentiality with Encrypted Query Processing," *In Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pp. 85-100, 2011. [CrossRef] [Google Scholar] [Publisher link]

[5] Ameya Nayak, Anil Poriya, and Dikshay Poojary, "Type of NoSQL Databases and Its Comparison with Relational Databases," *International Journal of Applied Information Systems*, vol. 5, no. 4, pp. 16-19, 2013. [Google Scholar] [Publisher link]

[6] D. Shravani, "Review of Literature on Web Services Security Architecture extended to Cloud, Big Data and IOT," *International Journal of P2P Network Trends and Technology*, vol. 6, no. 4, pp. 7-12, 2016. [Publisher Link]

[7] Mohammad Ahmadian et al., "Secure NoSQL: 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. [CrossRef] [Google Scholar] [Publisher link]

[8] Mamdouh Alenezi et al., "An Efficient, Secure, and Queryable Encryption for NoSQL-Based Databases Hosted on Untrusted Cloud Environments," *International Journal of Information Security and Privacy*, vol. 13, no. 2, pp. 14-31, 2019. [CrossRef] [Google Scholar] [Publisher link]

[9] G. Dumindu Samaraweera, and J. Morris Chang, "SEC-NoSQL: Towards Implementing High Performance Security-as-a-Service for NoSQL Databases," *arXiv e-prints (2021): arXiv-2107*, 2021. [CrossRef] [Google Scholar] [Publisher link]

[10] Mahesh M. Baradkar, and Bandu B. Meshram, "A Survey on Cloud Security: Infrastructure as a Service," *SSRG International Journal of Computer Science and Engineering*, vol. 6, no. 6, pp. 17-21, 2019. [CrossRef] [Publisher Link]

[11] Muhammad Ali Raza et al., "Secure NoSQL Over Cloud using Data Decomposition and Queryable Encryption," *Intelligent Technologies and Applications: Second International Conference*, vol. 1198, pp. 409-421, 2020. [CrossRef] [Google Scholar] [Publisher link]

[12] Richa Kunal Sharma, and Nalini Kant Joshi, "Security and Privacy Problems in Cloud Computing," *International Journal of Computer and Organization Trends*, vol. 9, no. 4, pp. 30-39, 2019. [CrossRef] [Publisher Link]

[13] Rakesh Agrawal et al., "Order Preserving Encryption for Numeric Data," *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, pp. 563-574, 2004. [CrossRef] [Google Scholar] [Publisher link]

[14] Vemula Sridhar, and K. Ram-Mohan Rao, "Multi Keyword Search on Encrypted Text without Decryption," *Second International Conference on Computer Networks and Communication Technologies*: *ICCNCT 2019*, vol. 44, pp. 256-263, 2020. [Google Scholar] [Publisher Link]

[15] N. Swetha, and S. Ramachandram, "A Survey: Query Processing Techniques for Secure Cloud Databases," *International Journal of Computer Engineering in Research Trends*, vol. 2, no. 12, pp. 1257-1262, 2015. [Publisher link]

[16] K. Karuppasamy, F. Margret Sharmila, and Tharani, T., "Survey on Cloud Security and Algorithms," *SSRG International Journal of Computer Science and Engineering*, vol. 6, no. 11, pp. 40-42, 2019. [CrossRef] [Publisher Link]

[17] Payal V. Parmar et al., "Survey of Various Homomorphic Encryption Algorithms and Schemes," *International Journal of Computer Applications*, vol. 91, no. 8, 2014. [CrossRef] [Google Scholar] [Publisher link]

[18] Pascal Paillier, "Public-Key Cryptosystems based on Composite Degree Residuosity Classes," *International Conference on the Theory and Applications of Cryptographic Techniques*, vol. 1592, pp. 223-238, 1999. [Google Scholar] [Publisher link]

[19] Xin Zhou, and Xiaofei Tang, "Research and Implementation of RSA Algorithm for Encryption and Decryption," *Proceedings of 2011 6th International Forum on Strategic Technology*, vol. 2, 2011. [CrossRef] [Google Scholar] [Publisher link]

[20] T. ElGamal, "A Public Key Cryptosystem and A Signature Scheme Based on Discrete Logarithms," *IEEE Transactions on Information Theory*, vol. 31, no. 4, pp. 469-472, 1985. [CrossRef] [Google Scholar] [Publisher link]

[21] Alexandra Boldyreva et al., "Order-Preserving Symmetric Encryption," *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Technique, Proceedings 28*, pp. 224-241, 2009. [Google Scholar] [Publisher link]

[22] Sridhar Vemula, Ram Mohan Rao Kovvur, and Dyna Marneni, "Algorithms for Implementing Repeated Homomorphic Operations on Restricted Data Type Ranges," *2023 Somaiya International Conference on Technology and Information Management (SICTIM)*, pp. 106-111, 2023. [CrossRef] [Google Scholar] [Publisher Link]

[23] [Online]. Available: https://github.com/MoAhmadian/SecureNoSQL

[24] M. Purna Chary, Srinivasa S. P. Kumar, B., and T. RamDas Naik, "A Survey on Implementation of Column-Oriented NoSQL Data Stores (Bigtable and Cassandra), " *International Journal of Computer Engineering in Research Trends*, vol. 2, no. 8, pp. 463-469, 2015. [Google scholar] [Publisher Link]

[25] Sridhar Vemula, Ram Mohan Rao Kovvur, and Dyna Marneni, "Secure E-Voting System Implementation using CryptDB," *SN Computer Science*, vol. 2, no. 217, 2021. [CrossRef] [Google Scholar] [Publisher Link]