Indonesian Journal of Electrical Engineering and Computer Science
Vol. 39, No. 2, August 2025, pp. 1299~1309
ISSN: 2502-4752, DOI: 10.11591/ijeecs.v39.i2.pp1299-1309

Processing queries on encrypted document-based database

Abdelilah Belhaj[1], Soumia Ziti[1], Karim Elbouchti[2], Noureddine Falih[3], Souad NajouaLagmiri[4]
[1]Intelligent Processing Systems and Security (IPPS) Team Faculty of Sciences, Mohammed V University in Rabat, Morocco
[2]Laboratoire Ingénierie des Systèmes Informatiques (LISI) Faculté des Sciences Semlalia, Université Cadi Ayyad Marrakech, Morocco
[3]LIMATI Laboratory, Computer Science Department, Polydisciplinary Faculty,
University of Sultan Moulay Slimane Beni Mella, Morocco
[4]IRSM, Institut Supérieur de Management d'Administration et de Génie Informatique Rabat, Morocco

Article Info

Article history:

Received Jul 27, 2024
Revised Mar 25, 2025
Accepted Jul 2, 2025

Keywords:

Encryption
MongoDB
NoSQL database
Oriented document
Secret sharing

ABSTRACT

Big data is a set of technologies and strategies for storing and analyzing large volumes of data in order to learn from it and make predictions. Since non-relational databases such as document-based have been applied in various contexts, the privacy protection must be taken into account by strengthening security to prevent the exposure of user data. In this paper, we focus mainly on secret sharing scheme that supports secure query with data interoperability to design a practical model for document-based databases, especially MongoDB. This approach, being based on secure query processing by defining elementary and suitable operators, allows us to perform operational computations and aggregations on encrypted data in the non-relational document database MongoDB. The obtained results, in the present work, could find places in various fields where data privacy and security are primordial such as healthcare, cloud computing, financial services, artificial intelligence and machine learning, in which user data remains secure and confidential during processing.

Corresponding Author:

Abdelilah Belhaj
Intelligent Processing Systems and Security (IPPS) Team Faculty of Sciences
Mohammed V University in Rabat
Morocco
Email: abdelilah_belhaj@um5.ac.ma

1. INTRODUCTION

Deploying a database in the cloud offers scalable, flexible, and cost-effective solutions for storing and processing data. It can reduce operational overhead and enable organizations to focus on their core applications. However, the outsourcing of private data storage and processing to third-party cloud providers introduces significant security and privacy risks. It exposes sensitive data to potential unauthorized access by external attackers or malicious insiders. Security and privacy in the cloud environment remain paramount for users who rely on these services for sensitive data management. Encrypting data before sending it to the cloud is a crucial measure to address the privacy and security challenges of cloud-based databases. These measures allow organizations to secure their data and protect it against both external threats and insider risks. However, querying encrypted data and performing secure computations directly on ciphertext present significant challenges and several schemes and techniques have been proposed for processing encrypted data in relational and NoSQL (Not Only SQL) databases [1], each with its strengths and limitations. These approaches allow for secure operations on encrypted data by performing aggregate queries over encrypted data. NoSQL is designed to handle large volumes of data and to store a wide variety of data types. Moreover, it provides significant advantages in terms of flexibility and scalability. However, it introduces unique security challenges that organizations must proactively address [2]-[4].

In relational database, several models have been proposed to ensure Transparent Data Encryption (TDE). These models offer solutions to provide security for data in-use by secure query processing over encrypted such as CryptDB [5]. CryptDB employs various encryption schemes depending on the operations needed on the data. Additive homomorphic encryption used in cryptDB enables limited arithmetic operations directly on encrypted data because multiplication and division are challenging to handle with partial homomorphic encryption. According to [6], MONOMI divides the execution of the query into two parts: one part consists of queries to be performed on outsourced encrypted data, and the other part involves executing queries on decrypted data on the user's side. El Bouchti *et al.* [7] propose a new implementation solution to protect encryption keys within DBMS by using a master key generated via an encryption on a table or a column. Omran [8], have proposed approaches based on dividing table attributes into multiple sub-columns according to the domain values of each attribute. It enables different queries using an order-preserving mapping fonction. A key limitation of such approaches is that they support only research on ciphertext.

Alternatively, Xu*et al.* [9] have adopted an additive homomorphic asymmetric cryptosystem to conceive an encrypted MongoDB called cryptMDB. It cannot support complex aggregations or operations on encrypted data due to the limitations of partial homomorphic encryption. Almarwani *et al.* [10] the autors propose a model known by a Secure Document Database (SDDB) for document-based databases which ensures an interrogation of encrypted data. Fully Homomorphic Encryption allows computation to be performed on encrypted data without putting it at risk. It has a potential to solve security issues and privacy-preserving data processing [11]-[16]. However, maintaining this property is challenging because fully homomorphic encryption also presents several challenges including performance overhead, complexity of implementation, several limitations in particular high latency and efficiency [17]-[20].

Wong *et al.* [21] have proposed SDB a secure query processing system that supports data interoperability. It divides data into sensitive and nonsensitive, with only sensitive data being encrypted allowing different operators of computation. The main idea of SDB is to split the sensitive data into two shares, one share is kept by the data owner (DO) and the second is kept by the service provider (SP). The encryption scheme used by SDB supports data interoperability by allowing queries on encrypted data without decrypting it. Similar studies can be found in [22]. It turns out that SDB provides data interoperability, making it possible to process complex queries by using an asymmetric secret-sharing scheme. It is a proxy between the user and the database. The SDB proxy is responsible for storing the column keys for sensitive data in its key store and rewriting SQL operators that involve sensitive columns to their corresponding user-defined functions (UDFs) [23], finally submitting the rewritten queries to the service provider (SP). The SDB system adopts an approach based on the secure multiparty computation (SMC) model and the secret sharing. Moreover, it prevents an attacker from recovering any sensitive data from their encrypted values. The cryptographic process of the secret sharing consists, on the one hand, of generating the item key encryption for each cell identified by the row number in each sensitive column on the other, of sharing the computation of the encrypted value from its plaintext one.

In the present investigation, we apply the cryptographic tools of SDB on a document-based database, referred to as secure query processing on document-based database MongoDB (SMDB). Employing SDB principles to MongoDB enhances data security and privacy, secure query processing in encrypted document-oriented databases. Our contribution is based on processing data using a secret sharing encryption scheme to perform operational computations and aggregations on encrypted data without revealing its content in non-relational document databases. In particular, the present investigation could solve a relevant problem in data security by preserving privacy. We expect that it could find a place in the bridging scenario of the cryptography and security in order to maintain data privacy such as financial and medical records [24]-[27].

2. METHOD

2.1. The proposal model

In this section, we present the architecture of SMDB being illustrated in Figure 1. In addition, we describe in details the encrypted tool which is the subject of the present investigation in this paper. In particular, the cryptographic tool will be studied.

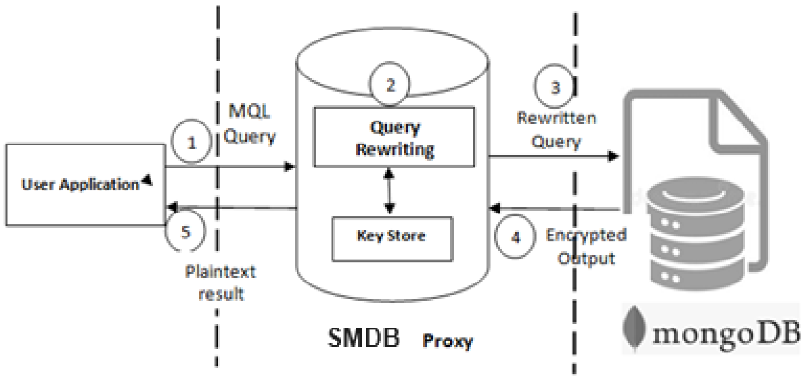


Figure 1. SMDB architecture

Here, the cryptosystem used in SDB is adopted. In SMDB, we use a common encrypted tool proposed in the reference [17]. This allows us to perform multiple operations on encrypted data. Before that, we consider the following materials. Specifically, g is a secret number and n is a public key being generated according to the RSA cryptosystem. n = pq is a large positive integer where p and q are prime numbers. In this way, g denotes a random number co-prime with n. To start, we take a sensitive field F of the collection COL in document-oriented database DB. In document-oriented database like MongoDB, each document includes a unique _id field. The latter is a simple identifier being generated automatically. It can be used to retrieve a document. Moreover, it can be considered as the first field of such a document which acts like a primary key. The document could involve several sensitive fields. In order to elaborate simplified analysis, we assume that the documents of collection myCOL have only one sensitive field F illustrated by:

{_ID: "value", "F": "value" // sensitive field, Other fields: 'data' }

We exploit two types of keys being field key and document encryption key. Indeed, the field key represents a pair of random number randomly generated by DO (Data Owner) for each sensitive field. As an example, we consider a sensitive field F. Indeed, we denote by $\mathrm{fk_F} = \langle \mathrm{m|x} \rangle$the field key of F where m and x are two random number such that $0 \le \mathrm{m, x} \le \mathrm{n}$.

Document Encryption Key denoted by $\mathrm{DK_F}$ is the item key to encrypt and decrypt the sensitive field F in each document. It can be generated from the field key $\mathrm{fk_F} < \mathrm{m, x} >$ and _id of document such that $0 \le \_\mathrm{id} \le \mathrm{n}$ according to the following relationship:

$$DK_F = key\_gen(id, fk_F(m,x)) = mg^{(id\ x\ mod\ \phi(n))} mod\ n. \tag{1}$$

It is denotedthat $\mathrm{DK_F}$ varies with the _id document. Certain notations will be used. V represents the plaintext of the sensitive filed which must be stored in an encrypted form. $\mathrm{V_e}$ denotes the encrypted value of V and $\mathrm{DK_F}^{-1}$ denotes the modular multiplicative inverse of $\mathrm{DK_F}$ such as:

$$DK_F . DK_F^{-1} = 1\ mod\ n. \tag{2}$$

The value of the sensitive field is encrypted by the public key FDK as:

$$V_e = E_{DK_F}(V) = V . DK_F^{-1}\ mod\ n \tag{3}$$

The decryption process is given as follows:

$$D_{DK_F}(V_e) = V . DK_F mod\ \mathrm{n} = V . DK_F . DK_F^{-1} mod\ n = V_s. \tag{4}$$

The_id of document is used to encrypt the value of the sensitive field F. Thus, the identifier _ID itself must be encrypted. It does not need to be encrypted in the same way as sensitive fields. To illsutre this scenrio, Figure 2 shows the process of encrypting the values of the field F in each document into encrypted values.

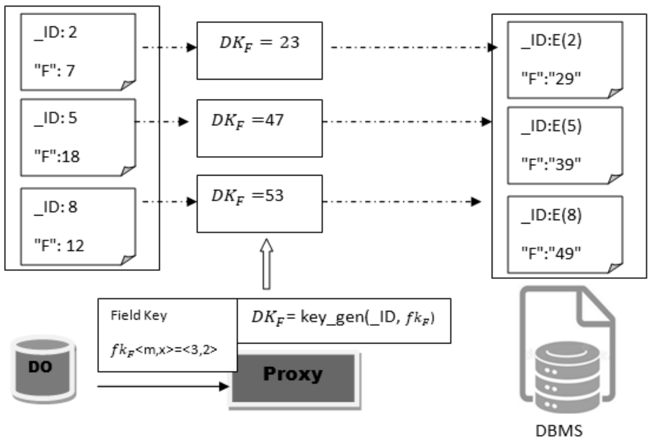


Figure 2. Encryption of field F in each document (n=55 and g= 3).

2.2. Secure operators in encrypted document-database

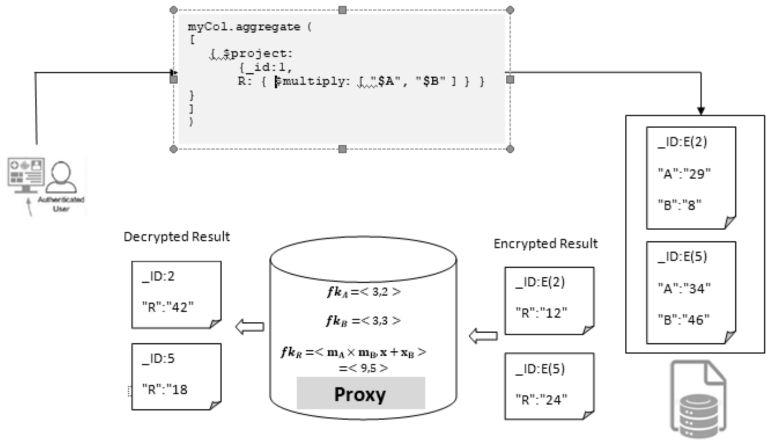
In this subsection, we provide relevant details of SMDB by giving comprehensive description and implementation of secure operators.

2.2.1. Multiplication operator (×)

First, we consider a collection called myCol in the document oriented database myDB containing two sensitive fields A and B as follow :{ _id:"", "A": "Encrypted value", "B": Encrypted value"}, $\mathrm{fk_A} = \langle \mathrm{m_A} | \mathrm{x_A} \rangle$ and $\mathrm{fk_B} = \langle \mathrm{m_B} | \mathrm{x_B} \rangle$are their key fields respectively.
Suppose a user send the following MQL (MongoDB Query Language) query to calculate the multiplication of the field A and the field B over encrypted values:

```
myCol.aggregate( [ { $project:      {_id:1, R: { $multiply: [ "$A", "$B" ] } } } ] )
```

First, the proxy rewrites and sends a rewritten query to MongoDB. Now, we show how this scenario works. Indeed, we consider R as the output field containing the result of the multiplication of the fields A and B in the encrypted form $R{=}A{\times}B$. For a document identified by id, a and b are the plaintext values of fields A and B, respectively. However, $\mathrm{a_e}$ and $\mathrm{b_e}$ denote their encrypted values stored in the database. Take r being the plaintext value of the output field Rsuch that r = a × b. Let $\mathrm{r_e}$ be the encrypted value of nR given by $r_e = a_e \times b_{\ e}$.
MongoDB executes the MQL query over encrypted data and returns results to the proxy server. For each document, the query computes the product of the encrypted values $\mathrm{a_e}$ and $\mathrm{b_e}$ and returns the result $\mathrm{r_e}$ which should be stocked in the output field R. This scenario is illustrated in Figure 3.



```
myCol.aggregate (
[
   { $project:
       {_id:1,
       R: { $multiply: [ "$A", "$B" ] } }
}
]
)
```

Figure 3. The multiplication of two sensitive fields. (n=55 and g= 3)

The proxy server constructs the key field fk_R of R according to the following relationship:

$$fk_R = \langle m_A \times m_B | x_A + x_B \rangle. \quad (5)$$

It determines the document key DK_R. The later is the key used to decrypt the field R in each document identified by _id as follows: $DK_R = m_A \times m_B g^{id(x_A+x_b)}(mod\ n)$ which is equivalent to: $DK_R = DK_A.DK_B(mod\ n)$. Indeed, we have :

$$r_e.DK_R \ = a_e.b_e.DK_R = a.DK_A^{-1}.b.DK_B^{-1}.DK_A.DK_B(mod\ n) = a.b(mod\ n)$$
$$r_e.DK_R = r. \quad (6)$$

Finally, the proxy server decrypts the cipher text of R which is the result of the multiplication by using DK_R and returns the plaintexts to the authorized user. In what follows, we give the assioated Algorithm 1:

```
Algorithm 1. EE multiplication
    Input : Field A, B with field key $\langle m_A|x_A\rangle$ and $\langle m_B|x_B\rangle$
    Output: R= A.B with R's field key $\langle m_R|x_R\rangle$
    Client-protocol:
    $m_R = m_A \times m_B$ mod $\phi(n)$
    $x_R = x_A + x_B$mod n.
    Server-protocol:
    for each document id do:
        Let $a_e$, $b_e$ be the encrypted value of A and B. Set $r_e = a_e \times b_e$mod n.
    End.
```

2.2.2. Key field update $k$ey_update

In order to update the key field $fk_A = \langle m_A|x_A\rangle$ of the field A to $\langle m_C|x_C\rangle$ without affecting stored data, one needs an operator denoted by key_update(A, $\langle m_C|x_C\rangle$). Key update is a helper operator that takes as input a field A and a target field key $\langle m_C|x_C\rangle$. It gives as output a field C with $fk_C = \langle m_C|x_C\rangle$ is its key field. C shares some plaintext of the field A without revealing sensitive value. First of all, we add a new auxiliary field S with the field key $\langle m_s|x_s\rangle$ in the collection with the plaintext value given by $S = 1\ mod\ n$. According to the encryption protocol mentioned above, we have: $s_e = DK_S^{-1}(mod\ n)$ where s_e is the encrypted value of S. The proxy computes two numbers p and q such that:

$$p = x_S^{-1}(x_C - x_A)mod\ \phi(n) \quad (7)$$

$$q = m_A.m_s^p.m_C^{-1}.mod\ n. \quad (8)$$

For each document, MongoDb computes the value of the output field C via $C_e = q.a_e.s_e^p$ following to the Algorithm 2:

```
Algorithm 2: _update(A, $\langle m_C|x_C\rangle$)
    Input: Field A with field key $\langle m_A|x_A\rangle$
    Output: C= A with C's field key $\langle m_C|x_C\rangle$
    Client-protocol :
    Let the field S with the field key $\langle m_s|x_s\rangle$
            $p = x_S^{-1}(x_C - x_A)$mod $\phi(n)$, $q = m_A.m_S^p.m_C^{-1}$.mod n.
    Server-protocol: Obtain p,q
    for each document id do:
        Let $a_e$, $s_e$be the encrypted value of A and S. Set $c_e = q.a_e.s_e^p$mod n .
    end.
```

The rewriting MQL query is as follow:

```
Xs, ms=generate_key_field(), Add_auxilary_field(S)
P, q=compute_pq(ma,xa,mn,xn,ms,xs)
myCol.aggregate([{$project {_id:1, N: {$multiply:[ p,"A","$pow":{"S",q}]}},}
```

The proxy decrypts the result stored in the field N using the key update $\mathrm{fk_N} = \langle \mathrm{m_N} | \mathrm{x_N} \rangle$. The $\mathrm{key\_update}(\mathrm{A}, \langle \mathrm{m_N} | \mathrm{x_N} \rangle)$ operator will help us to define other secure operators, serving as a relevant tool.

2.2.3. Addition/Subtraction (+,-)

Given a request from the user to calculate the sum of two sensitive fields A and B stored in encrypted form in document-oriented database MongoDB:

```
myCol.aggregate( [ { $project:      {_id:1,   C: { $sum: [ "$A", "$B" ] } } } ]
```

The query performs calculations on encrypted data. Our objective is to calculate the sum without revealing the plaintext value of the sensitive fields. The result of the sum is stored in the output field C returned by query such that C=A+B. For each document, MongoDB computes $\mathrm{C_e}$ which denotes the sum of the encrypted values $\mathrm{a_e}$and $\mathrm{b_e}$ given by $C_e = a_e + b_e$. In order to make our protocol behaves like an isomorphic additive function, we should perform the same field key for A, B and C such that: $fk_A = fk_B = fk_C = k$. We can use the key update operator to assign them the same field key without modifying the contents of fields. The proxy rewrites the query by first applying the key update operator to the fields A and B. To do this, the server proxy generates a field key $\langle \mathrm{m_C} | \mathrm{x_C} \rangle$ and computes the output fields A1 and B1 by applying the key update protocol to fields A and B and using the same key field$\langle \mathrm{m_C} | \mathrm{x_C} \rangle$. This leads to A1 = $\mathrm{key\_update}(\mathrm{A}, \langle \mathrm{m_C} | \mathrm{x_C} \rangle)$ and B1 = $\mathrm{key\_update}(\mathrm{B}, \langle \mathrm{m_C} | \mathrm{x_C} \rangle)$. The field key of A1and B1 and C are all $\langle \mathrm{m_C} | \mathrm{x_C} \rangle$, as shown in Algorithm 3.

```
Algorithm 3. EE addition
    Input: Field A, B with field key $\langle m_A | x_A \rangle$ and $\langle m_B | x_B \rangle$
    Output: C= A+B with R's field key $\langle m_C | x_C \rangle$
    Client-protocol: Generate random $\langle m_C | x_C \rangle$
    A1 =key_update(A, $\langle m_C | x_C \rangle$), B1=key_update (B, $\langle m_C | x_C \rangle$)
    Server-protocol:
    for each document id do:
        Let $a_e, b_e$ be the encrypted value of A1 and B1, Set $c_e = a_e \times b_e \bmod n$.
    end.
```

The SMDB proxy server rewrites the query as follows:
myCol.aggregate([{ $project { id:1, A1 : {$multiply:[pa,"A","$pow":{"SA",qa}]},
B1: {$multiply:[pb,"B","$pow":{"SB",qb}]}, C :{ $sum: ["A1", "B1"] }}])
The proxy decrypts the result of the sum stored in the field C by using$\mathrm{DK_C} = (\mathrm{id}, \langle \mathrm{m_C} | \mathrm{x_C} \rangle)$ and returns the plaintext value to the user.

2.2.4. Comparison (= / >)

Given two sensitive fields A and B stored in encryption form in document-based database. W consider two comparison operators including equality (=) and ordering (>) that compare values and return true or false. The proxy first calculates $\mathrm{C} = \mathrm{F} \times (\mathrm{A} - \mathrm{B})$ where F is a random field in a collection and C is the output field. Secondly, the proxy applies key update operator to the output field C. Let Z be the output field of the key update operator such that Z= $key(C, \langle 1 | 0 \rangle)$. Finally, the proxy can determine the comparison's outcome based on the value z of Z. If $z > 0$, then one has $a > b$ and if $z = 0$ then one has $a = b$. The comparison is often used in selection queries. Suppose a user sends a Mql query to proxy server as follows:
db.users.find({ ”A” : "B"})
This Mql query will be rewritten by the proxy. Firstly, the Proxy computes two numbers pz and qz according to the previous relationships, and adds a new auxiliary field Z, as shown in Algorithm 4

```
Algorithm 4. Comparison
Input: Field A, B with field key $\langle m_A | x_A \rangle$ and $\langle m_B | x_B \rangle$
Output: field of comparison results C= 1 if A>B; C=0 if A=B; C=-1 if A<B
Client-protocol:
$R = F \times (A - B)$ , Z= key_update(R, $\langle 1 | 0 \rangle$).
Server-protocol:
```

```
for each document id do:
    Let $z_e$ be the values on Z. if $z_e > 0$: $c_e = 1$ elif $z_e = 0$: $c_e = 0$ Else $c_e = -1$
end.
```

Then, the proxy server rewrites the MQL query as follows:
add_auxillary_field("SZ",ms,xs)
p,q=calcul_pq(mc,xc,1,0,ms,xs)
myCol.aggregate([{ $match:{"$$Z" :0}
$project {_id:1,C: {$multiply:["F","$subtract":{"$A", "$B"}]},
Z: {$multiply:[pz,"C","$pow":{"SZ",qz}]},}])

2.2.5. Aggregation operations, sum, count, average and group

Several documents are processed via aggregation operation, which then produce results. Consider the collection **myCol**, which has n documents and an encryptedvalue in the sensitive Field A. Our objective is to calculate the sum of the values in the field A from encrypted data: $S_n = \sum_{i=1}^{n} a_i$ where a_i are the plaintext values of the field A in a document To achieve this, the proxy first generates a random number m_C and applies the key update protocol to the field A. Take C the output field such that $C = \text{key\_update}(A, \langle m_C | 0 \rangle)$ and C_n is the sum of $(c_i)_e$ which denotes encrypted value of the field C in the document

$$C_n = \sum_{i=1}^{n} (c_i)_e = m_C^{-1}.S_n. \tag{9}$$

Indeed, we have:

$$S_n = m_C\,.\,C_n. \tag{10}$$

A user can send the following methods to calculate the sum of values in a field A in MongoDB:
db.myCol.aggregate([{$group: { _id:null, sum_val:{$sum:"$A"}}}]).
The proxy rewrites the query by executing key updates protocol and stores the result in the output field C as follows:

```
Xs,ms=Generate_key_field() ,add_auxillary_field("S")
mc=generate_number()
p,q=calcul_pq(ma,xa,mc,0,ms,xs)
db.myCol.aggregate([{$group: { _id:null, C: {$multiply:[ p,"A","$pow":{"S",q}]},
sum_val:{$sum:"$$C"}}}]).
```

The proxy decrypts the result by using the following relationship:

$$plaintext_{sum_{val}} = sum_{val} \times mc \tag{11}$$

In the same way, we can define the other aggregation operators, including:$count, $group, $avg.

2.3. Implementation

The implementation of this system has three basic entities: client, proxy SMDB representing the main logic of this system, and a server provider. The latter performs a number of functions through UDFs which are the user defined JavaScript functions stored on Google Cloud storage buckets. The proxy uses secret sharing encryptions to encrypt sensitive filed. The Id field is encrypted using the AES-CMC deterministic encryption algorithm which allows the server to find the document. SMDB proxy is composed of four main componentswhich are Query Parser, Query analyser, Query Rewriter and Query Executer.

We must emphasize that choosing the appropriate length of key for SMDB cryptosystem is essential to achieve a balance between security and performance. We notice that the length of ciphertexts is quite long since we use 1024-bit key length. To ulistrate, we provide certain example:For key size=1024 bit, To generate the field key of the sensitive field A $\langle m_A | x_A \rangle$, we give the following scheme:

xa=random.randint(1,n)
ma=random.randint(1,n)

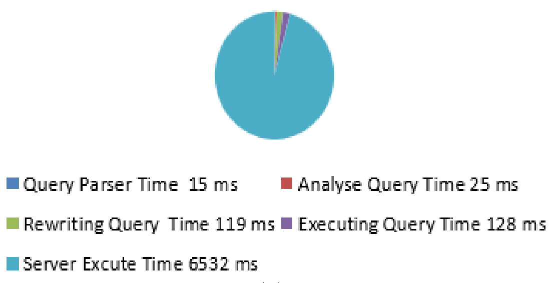
Consider the following document containing two sensitive fields A and B. {'A': 1235 , 'B': 524}
The _id is a identifier of each document arbitrarily generated as follow *:id= random.randint(1,n).*
The id is added in encrypted form using AES [28] encryption scheme in particular AES-CBS [29]. According to cryptosystem of secret sharing SMDB, a cipheretext document is created and sent to MongoDB server by proxy server.

3. RESULTS AND DISCUSSION

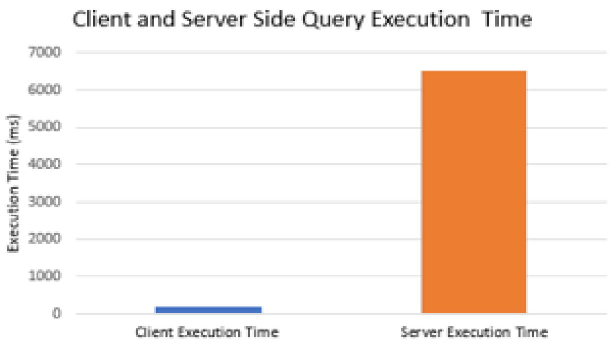
Given the following MQL query which computes the product of the sensitive field A and B and only returns documents where the value of output field C is greater than 1255.

```
db. mycol.aggregate([ { $ project: { id: 1, C: { $multiply: ["$A", "$B"] } } },
{ $match: { C: { $gt: 1255 } } } ])
```

All the experimental procedures are performed on an Intel(R) Core (TM) i5-1035G1 CPU @ 1.00 GHz, 1190 MHz, 4 cores, 8 processors. Overview of query performance: stage breakdown and client vs. server execution time as shown in Figure 4. Figure 4(a) shows the performance of the query and the amount of time taken by each component of the proxy.



(a)



(b)

Figure 4. Performance of the query (a) query performance time and (b) client-side and server-side query performance

The cost of client-side and server-side query execution is illustrated in Figure 4(b). To evaluate the execution time of the proxy server, we divide it into several components. The total proxy execution Time is measured from the moment the query enters the proxy until the final result is returned to the client. server must handle all processing, which can be costly. It depends on several factors including encryption and decryption overhead, query complexity and performance. The proposal model has been tested by considering numbers of documents ranging from 500 to 2500. The time taken by a query in a proxy system SMDB plus response time from the server. Figure 5 illustrates a comparison of the time required for aggregate operations between an unencrypted database and a database encrypted using our SMDB Model

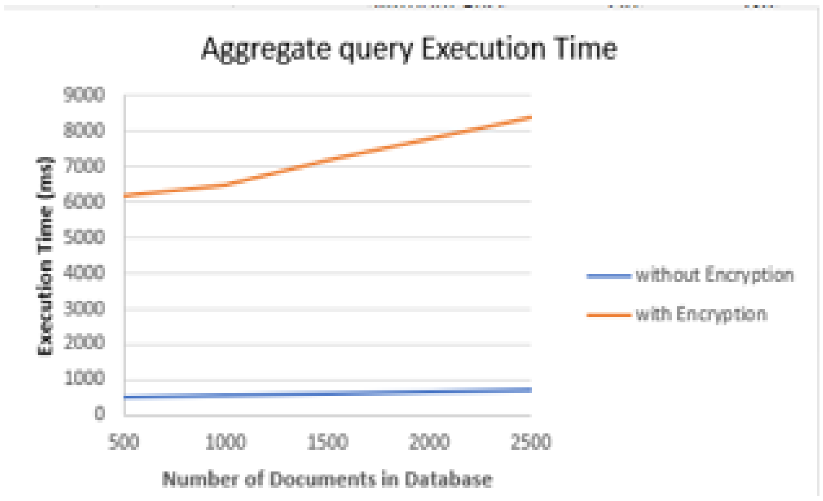


Figure 5. Aggregate query execution time using SDMB

Applying SDB approach based on secure query processing with data interoperability in a cloud database to a document-based database like MongoDB involves several key considerations. Although SDB provides robust data protection and privacy for securely processing queries in encrypted databases, it also encounters certain limitations and challenges including high computational costs, storage overhead due to encrypted data size, finally key management and Security Risks. As an expected behavior, it has been remarqued that the execution time increase rapidly in terms of number of documents. In the end of this work, we would like to discuss the security behavior. Multiple systems have indeed been implemented to detect and prevent attacks on SQL and NoSQL database**s** [30]-[32]. First, we provide the treats of our model illustrated in Figure 6. There are two main types of threats. For treat 1, the SMDB proxy server can be compromised by attackers. In this case, the attackers can use the keys to arbitrarily encrypt or decrypt user data.

For the threat 2, an attacker can see the requests sent to the SP and all intermediate encrypted results from any operator involved in the request. An attacker can also intercept messages exchanged over the communication channel between the client and the server. The attacker cannot deduce the plaintext of the sensitive field or the encrypted result.

To summarize, we have approached a secret sharing scheme promote secure query with data interoperability to design a practical model for document-based databases, especially MongoDB. Since the application of fully homomorphic encryption is not easy task, this study can be exploited to perform operational computations and aggregations on encrypted data in the non-relational document database MongoDB. The present work could find places in various fields where data privacy and security are primordial such as healthcare, Cloud Computing, financial Services, artificial intelligence and Machine Learning, in which user data remains secure and confidential during processing. However, the empirical discussions need more sophisticated hardware. We hope address such questions in futures researches.

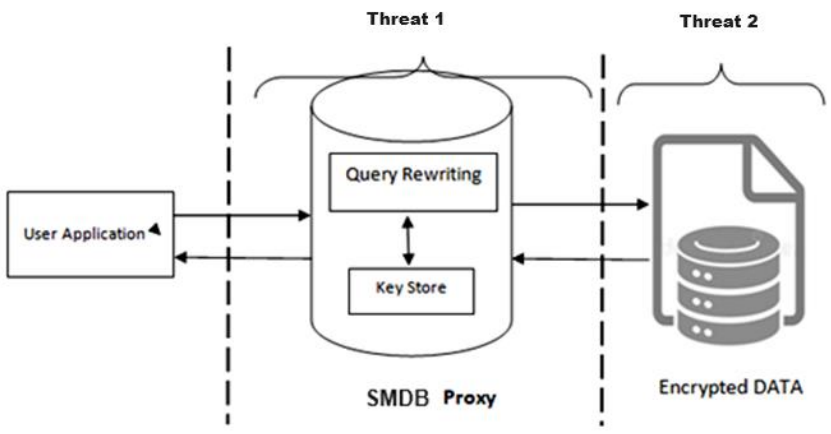


Figure 6. Threats of scheme

4. CONCLUSION

In this paper, we have presented a practical model for Secure MongoDB document-oriented database SMDB based on secret sharing. We have provided an exhaustive explanation of SMDB's query rewriting and analysis of performance. It has remarked that this query processing system based on a asymmetric secret-sharing is practically efficient that supports multiple secure operators while maintaining data interoperability. This is crucial in sensitive sectors like healthcare, financial services, banking and government. Moreover, it has been believed that the obtained results could find places in various applications and uses in connection with AI, in which the neural networks could be applied to encrypted data. However, our model provides robust data protection and privacy, its application in NoSQL databases introduces certain challenges related to performance, scalability, and complexity. This approach could be adopted for appropriates applications where the importance of this study appears more relevant for extended hardware. This may need more thinking and further investigations.

REFERENCES

[1] J. Xia, Q. Huang, Z. Gui, and W. Tu, "NoSQL databases," in *Open GIS*, Cham: Springer International Publishing, 2024, pp. 143–171.

[2] M. Kantarcioglu, "Securing big data: new access control challenges and approaches," in *Proceedings of ACM Symposium on Access Control Models and Technologies, SACMAT*, May 2019, pp. 1–2, doi: 10.1145/3322431.3326330.

[3] S. Sicari, A. Rizzardi, and A. Coen-Porisini, "Security&privacy issues and challenges in NoSQL databases," *Computer Networks*, vol. 206, p. 108828, Apr. 2022, doi: 10.1016/j.comnet.2022.108828.

[4] S. B, "'Using MongoDB to understand the underlying methods techniques encryption in NoSQL database,'" *International Journal of Research Publication and Reviews*, pp. 862–866, Sep. 2022, doi: 10.55248/gengpi.2022.3.9.23.

[5] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: protecting confidentiality with encrypted query processing," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, Oct. 2011, pp. 85–100, doi: 10.1145/2043556.2043566.

[6] 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, Mar. 2013, doi: 10.14778/2535573.2488336.

[7] K. El bouchti, S. Ziti, F. Omary, and N. Kharmoum, "New solution implementation to protect encryption keys inside the database management system," *Advances in Science, Technology and Engineering Systems Journal*, vol. 5, no. 2, pp. 87–94, 2020, doi: 10.25046/aj050211.

[8] O. M. Omran, "Data partitioning methods to process queries on encrypted databases on the cloud." 2016.

[9] G. Xu, Y. Ren, H. Li, D. Liu, Y. Dai, and K. Yang, "CryptMDB: a practical encrypted MongoDB over big data," in *IEEE International Conference on Communications*, May 2017, pp. 1–6, doi: 10.1109/ICC.2017.7997105.

[10] M. Almarwani, B. Konev, and A. Lisitsa, "Fine-grained access control for querying over encrypted document-oriented database," *Communications in Computer and Information Science*, vol. 1221 CCIS, pp. 403–425, 2020, doi: 10.1007/978-3-030-49443-8_19.

[11] R. Hamza *et al.*, "Towards secure big data analysis via fully homomorphic encryption algorithms," *Entropy*, vol. 24, no. 4, p. 519, Apr. 2022, doi: 10.3390/e24040519.

[12] N. R, V. K, M. A, I. P, and A. P, "A hybrid improved zhou and wornell's inspired fully homomorphic encryption scheme for securing big data computation in cloud environment." Apr. 16, 2021, doi: 10.21203/rs.3.rs-356001/v1.

[13] A. Alabdulatif, I. Khalil, and X. Yi, "Towards secure big data analytic for cloud-enabled applications with fully homomorphic encryption," *Journal of Parallel and Distributed Computing*, vol. 137, pp. 192–204, Mar. 2020, doi: 10.1016/j.jpdc.2019.10.008.

[14] A. Jadama, A. Mohammed, and F. Rashid, "Fully homomorphic encryption," 2024.

[15] Z. Zheng, K. Tian, and F. Liu, *Fully Homomorphic Encryption*. 2023.

[16] T. Z. Erlanovna, T. Sakhybay, A. Z. Muratovna, and T. Gulzat, "Development Paillier's library of fully homomorphic encryption," *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 34, no. 3, pp. 1989–1998, Jun. 2024, doi: 10.11591/ijeecs.v34.i3.pp1989-1998.

[17] G. Dimitoglou and C. Jim, "Performance evaluation of partially homomorphic encryption algorithms," in *Proceedings - 2022 International Conference on Computational Science and Computational Intelligence, CSCI 2022*, Dec. 2022, pp. 910–915, doi: 10.1109/CSCI58124.2022.00163.

[18] N. Jain and A. K. Cherukuri, "Revisiting fully homomorphic encryption schemes," *arXiv*, 2023, doi: 10.48550/arXiv.2305.05904.

[19] A. Maqousi, M. Alauthman, and A. Almomani, "Homomorphic encryption enabling computation on encrypted data for secure cloud computing," in *Innovations in Modern Cryptography*, 2024, pp. 215–240.

[20] W. Dai and B. Sunar, "cuHE: a homomorphic encryption accelerator library," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9540, 2016, pp. 169–186.

[21] W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, and S. M. Yiu, "Secure query processing with data interoperability in a cloud database environment," in *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, Jun. 2014, pp. 1395–1406, doi: 10.1145/2588555.2588572.

[22] M. R. Asghar, G. Russello, B. Crispo, and M. Ion, "Supporting complex queries and access policies for multi-user encrypted databases," in *Proceedings of the 2013 ACM workshop on Cloud computing security workshop*, Nov. 2013, pp. 77–88, doi: 10.1145/2517488.2517492.

[23] J. Luo, L. Zhang, and X. Li, "A model-driven parallel processing system for IoT data based on user-defined functions," in *2020 IEEE 5th International Conference on Cloud Computing and Big Data Analytics, ICCCBDA 2020*, Apr. 2020, pp. 463–470, doi: 10.1109/ICCCBDA49378.2020.9095646.

[24] D. Beaver, "Efficient multiparty protocols using circuit randomization," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 576 LNCS, Berlin, Heidelberg: Springer Berlin Heidelberg, 1992, pp. 420–432.

[25] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via MiniONN transformations," in *Proceedings of the ACM Conference on Computer and Communications Security*, Oct. 2017, pp. 619–631, doi: 10.1145/3133956.3134056.

[26] T. Sander, A. Young, and Moti Yung, "Non-interactive cryptocomputing for NC/sup 1/," in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, 2003, pp. 554–566, doi: 10.1109/sffcs.1999.814630.

[27] B. Sarkar, A. Saha, D. Dutta, G. De Sarkar, and K. Karmakar, "A survey on the advanced encryption standard (AES): a pillar of modern cryptography," *International Journal of Computer Science and Mobile Computing*, vol. 13, no. 4, pp. 68–87, Apr. 2024, doi: 10.47760/ijcsmc.2024.v13i04.008.

[28] M. Vaidehi and B. J. Rabi, "Design and analysis of AES-CBC mode for high security applications," in *Second International Conference on Current Trends In Engineering and Technology - ICCTET 2014*, Jul. 2014, pp. 499–502, doi: 10.1109/ICCTET.2014.6966347.

[29] R. Macedo *et al.*, "A practical framework for privacy-preserving NoSQL databases," in *2017 IEEE 36th Symposium on Reliable Distributed Systems (SRDS)*, Sep. 2017, vol. 2017-Septe, pp. 11–20, doi: 10.1109/SRDS.2017.10.

[30] M. Shachi, N. S. Shourav, A. S. S. Ahmed, A. A. Brishty, and N. Sakib, "A survey on detection and prevention of SQL and NoSQL injection attack on server-side applications," *International Journal of Computer Applications*, vol. 183, no. 10, pp. 1–7, Jun. 2021, doi: 10.5120/ijca2021921396.

[31] C. Blanco *et al.*, "Security policies by design in NoSQL document databases," *Journal of Information Security and Applications*, vol. 65, p. 103120, Mar. 2022, doi: 10.1016/j.jisa.2022.103120.

[32] A. M. Eassa, "NoSQL security in web application," in *The Role of Cybersecurity in the Industry 5.0 Era*, IntechOpen, 2025.

BIOGRAPHIES OF AUTHORS

Abdelilah Belhaj in 2013, he earned his Master's degree in applied informatics and offshoring from Mohammed V University in Rabat, Morocco. Currently, he is pursuing his Ph.D. His research in in cryptography, big data and deep learning. He can be contacted at email: abdelilah_belhaj@um5.ac.ma.

Soumia Ziti is a full professor and researcher at Mohammed V University in Rabat since 2007. She obtained her PhD in computer science specializing in graph theory from the University of Orleans in France, along with a diploma in advanced studies in fundamental computer science. Furthermore, she earned a master's degree in science and technology in computer science from the same institution. Her research interests encompass a wide range of topics including graph theory, information systems, artificial intelligence, data science, software development, database modeling, big data, cryptography, and numerical methods and simulations. She can be contacted at email: s.ziti@um5r.ac.ma.

Karim El Bouchti in 2020, he obtained a Doctor of Computer Science degree from the Faculty of Sciences of Mohammed V University in Rabat Morocco. Since 2023, he has been working as an associate professor at the Faculty of Sciences Semlalia, University Cadi Ayyad Marrakech, Morocco. With 10 years of professional experience in CNESTEN the National Center for Energy Sciences and Nuclear Technology, his field of research focuses on cyber security, big data analytics. He can be contacted at email: k.elbouchti@uca.ac.m.

Noureddine Falih in 2013, he obtained a Doctor of Computer Science degree from the Faculty of Sciences and Technologies of Mohammedia, Morocco. Since 2014, he has been working as an associate professor at the Polydisciplinary Faculty of Sultan Moulay Slimane University in Beni Mellal, Morocco. With 18 years of professional experience in several renowned companies, his research interests revolve around information system governance, business intelligence, big data analytics, and digital agriculture. He can be contacted at email: nourfald@yahoo.fr.

Souad Najoua Lagmiri she received her diploma for doctor degree from the Mohammadia School of Engineers Rabat. Currently Director of the Higher Institute of Management, Administration and Computer Engineering Rabat Morocco. She has published several scientific articles in international journals and conferences. His field of research focuses on encryption algorithms, cryptography, and information security based on complex mathematical equations. She can be contacted at email: snajoua.lagmiri@gmail.com.