ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 6, June 2018

Homomorphic Encrypted MongoDB for Users Data Security

[1] Anil Kumar, [2] Harsha H L, [3] B. Swaroop Reddy, [4] K.Sunil Kumar Reddy, [5] Krishna N
[1] Asst.Professor, CSE, Vemana Institute of Technology, Bengaluru
[2 3 4 5] Student, CSE, Vemana Institute of Technology, Bengaluru

Abstract- **Database is used for storage of information in Software Applications. Normally, traditional RDBMS are used for storage purposes but with applications generating enormous amount of data, RDBMS is no longer efficient because RDBMS doesn't support quick data access and computations as it do not support processing of data in distributed manner. To overcome this problem, NoSQL based MongoDB is emerged which is document oriented database, it stores the data in the form of collections rather than tables therefore it supports quick data access and computations in distributed way and it provides flexibility by not enforcing the particular schema to be followed throughout. But very often MongoDB fails to provide security to the user data, which is very important these days. In this paper, security for users data is provided by using additive homomorphic asymmetric cryptosystem which encrypts the users data in MongoDB(CryptMDB) and achieve strong user's data privacy protection. This also supports the database operations over the encrypted data.**

Keywords— **Database, RDBMS, NoSQL, MongoDB, Homomorphic cryptosystem, CryptMDB.**

I. INTRODUCTION

Big data is one of the world's hottest vocabularies after the Internet of things and cloud computing. Big data has brought a great impact. In the Big data era, the applications based on it are generating huge amount of data and most of them works on real time basis so they require fast computation and huge storage capability [1],[2]. Generally, data should be stored in databases for easy access and utilization. Existing mainstream databases adopted by enterprises and individuals are relational databases, such as MySQL, Oracle, DB2, etc, in which data are stored as a item of tables and participated various Sql requests. But, RDBMS are not suitable for Bigdata storage purposes because of the following reasons:

First, the data size has increased tremendously to the range of petabytes—one petabyte = 1,024 terabytes. RDBMS finds it challenging to handle such huge data volumes.

Second, the majority of the data comes in a semi-structured or unstructured format from social media, audio, video, texts, and emails. However, the second problem related to unstructured data is outside the purview of RDBMS because relational databases just can't categorize unstructured data. They are designed and structured to accommodate structured data such as weblog sensor and financial data.

Third, data is generated at a very high velocity. RDBMS lacks in high velocity because it's designed for steady data retention rather than rapid growth. Even if RDBMS is used to handle and store big data, it will turn out to be very expensive. In order to overcome the disadvantages posed by the RDBMS platforms, we make use the NoSQL based databases like Cassandra, MongoDB for the storage of Bigdata. In this paper we considered MongoDB for the security analysis. There are several advantages of using NoSql over the RDBMS: 1.Schema Less: NoSQL databases being schema-less do not define any strict data structure. 2. Dynamic and Agile: NoSQL databases have good tendency to grow dynamically with changing requirements. It can handle structured, semi-structured and unstructured data. 3. Scales Horizontally: In contrast to SQL databases which scale vertically, NoSQL scales horizontally by adding more servers and using concepts of sharding and replication. This behaviour of NoSQL fits with the cloud computing services such as Amazon Web Services (AWS) which allows you to handle virtual servers which can be expanded horizontally on demand.4.Better Performance: All the NoSQL databases claim to deliver better and faster performance as compared to traditional RDBMS implementations.[3] Inspite of these advantages, existing MongoDB products fail to consider a crucial and practical issue in databases, i.e., privacy protection. It is well-known that data is stored which is deprived of any safety measures on commonly used databases, which is susceptible to attackers who are interested in users' sensitive information if adversaries can compromise databases to steal private data. Besides, MongoDB server is suspected as honest but curious, which may malicious peep data stored in databases due to it has the full access permission.

Therefore, it is crucial to propose a privacy-preserving approach which can ensure confidentiality of users' information on MongoDB [4]. In the last few years, several encryption schemes have been applied in relational databases. Raluca[5] et al. design a CryptDB system in MySQL, which uses an onion encryption structure to support 99.5% operations over encrypted data. Deshmukh[6] et al. propose a transparent data encryption scheme to provide high levels of security for columns, table and tablespace in Microsoft SQL Server 2008. Then Raluca[7] et al. present an ideal-security protocol for order-preserving encoding over relational databases, which not only can provide ideal security but also demonstrate the higher performance comparing with previous order-preserving approach. However, few specific encryption tools which have been applied in non-relational databases. In this paper, we propose a practical encrypted MongoDB, which can guarantee strong privacy protection and high performance in non-relational databases. In specific, the contributions of this paper can be summarized as follows:

1. We leverage an additive homomorphic asymmetric cryptosystem to design an encrypted MongoDB, which can achieve additive operations over encrypted data.
2. Security analysis shows that the proposed system can achieve strong privacy protection of users' information stored in databases. Besides, extensive experiments indicate that the proposed system is better than existing relational database (such as MySQL) in terms of data access and calculating.

The remainder of this paper is organized as follows. In Section II, we will describe the preliminaries. In Section III, we will propose a practical encrypted MongoDB and describe the details of our model. Then we carry out the security analysis and performance evaluation in Section IV and Section V, respectively. Finally, Section VII concludes the paper.

II. PRELIMINARIES

In this section, we will introduce the architecture of the proposed system and analyse threats of the system. Besides, encrypted tool also will be brought in this part, which will be served as the basic of our proposed scheme.

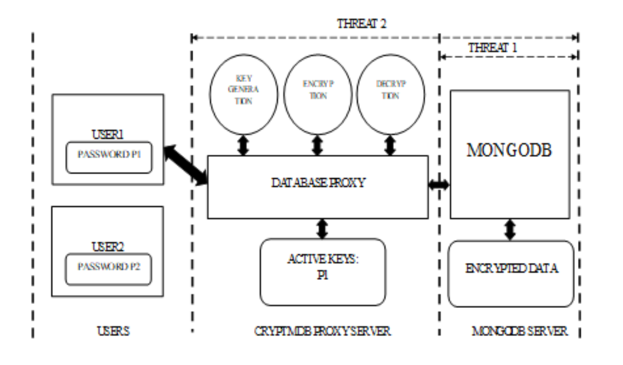


Fig1: CryptMDB Architecture

A. System Architecture:

As shown in Fig.1, CryptMDB mainly contains three parts: User's computers, CryptMDB proxy server and MongoDB server. Firstly, data provided by users will be encrypted by encryption tools and stored in MongoDB, when users want to query the contents of database, they should send some specific MongoDB query languages (Mql) to CryptMDB proxy server. Then these Mql queries will be rewritten by pre-set encrypted tools and sent to the MongoDB server. Next, the MongoDB server executes Mql to match corresponding ciphertexts which will be delivered to CryptMDB proxy server. Finally, the proxy server decrypts these ciphertexts and sends them to authorised users. We can see that in CryptMDB where MongoDB server executes Mql queries and return corresponding ciphertexts to users, it cannot gain access to the sensitive data of users, which ensures that user's private information cannot be leaked to any part in whole CryptMDB architecture, Besides, in CryptMDB, different users have their own disparate key to encrypt personal information. Therefore, even if the attackers full control the CryptMDB, they cannot get private data whose owner are not log in the CryptMDB systems. In this paper, although CryptMDB can protect the data confidentiality, it does not guarantee the data completeness, freshness, integrity and so on. Moreover, other attacks such as compromise user's computers, gain user's key, or a malicious DBA, are not the scope of our CryptMDB.

B. Threat 1: MongoDB Compromise:

Shown in Fig.1, in CryptMDB, we assume that the MongoDB server is honest but curious. On the one hand,it will strictly execute Mql queries provided by proxy server, on the other side, it may try to infer the contents of user's

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 6, June 2018

data and learn the relationship among user's information. Besides, proxy server is supposed to trustworthy in CryptMDB. Therefore, this threat mainly includes MongoDB software compromise, access to the databases, and access to the RAM of MongoDB machines. With the development of big data and lack private protection awareness in database areas, this threat turns more and more dangerous when tens of thousands data are stored in various databases. In this paper, we resist this threat by MongoDB server to execute Mql queries over encrypted data, in CryptMDB, all data will be encrypted by proxy server firstly, then these ciphertexts of each user will be stored in MongoDB. MongoDB server only to match corresponding ciphertexts when it receives the queried requests from proxy server. So it never gains access to plaintexts of data. Therefore, MongoDB server cannot get private data of users.

C. Threat 2: Arbitrary Threats:

In this section, we describe the arbitrary threats, which means that CryptMDB proxy server and MongoDB server have been compromised by attackers. In this case, attackers can get access to the databases and utilize the keys to encrypt or decrypt user's data arbitrarily. Compared with the threat 1, the hazard of threat 2 is more deadly and dangerous. To prevent user's data from being leaked to malicious attackers, we adopt different keys to encrypt data for each user. Besides, developers pre-annotate the database schemas to determine which key will be used for each data set, and these keys only be activated by corresponding users who are logging in the MongoDB. Thus, even attackers entire control the proxy and MongoDB server, they just decrypts the data of current users (who are logging in MongoDB), other user's private data do not reveal to attackers.

D. Cryptographic Tool:

In this paper, an additive homomorphic asymmetric cryptosystem is adopted. In the CryptMDB, we utilize a common encrypted tool proposed by Paillier et al. [8] which can achieve additive operations over encrypted data.

Pailier Algorithm:

Key generation:

1. Choose two large primes *p* and *q* randomly and independently of each other such that ***gcd (pq,(p-1)(q-1))=1.*** This property is assured if both primes are of equal length.
2. Compute ***n=pq*** and ***λ=lcm(p-1,q-1).***
3. Select random variable ***g*** such that $g \epsilon Z_n^{2*}$
4. Ensure ***n*** divides the order of ***g*** checking the existence following modular multiplicative inverse: $\mu=(L(g^{\lambda} mod\ n^2)^{-1} mod\ n$, where L function is defined as: $L(x)=\frac{x-1}{n}$
5. The public (encryption) key is ***(n,g)***
6. The private (decryption) key is ***(λ,μ)***

Encryption:

1. Let ***m*** be the message to be encrypted $0 \leq m \leq n$
2. Select random number ***r***, $0 \leq r \leq n$
3. Compute ciphertext as: $c=g^m.r^n\ mod\ n^2$

Decryption:

1. Let be the ciphertext to decrypt, where: $c\ \epsilon\ Z_n^{2*}$
2. Compute the plaintext message as: $m=L(c^{\lambda}\ mod\ n^2).\mu\ mod\ n$

Homomorphic properties:

A notable feature of the Paillier cryptosystem is its homomorphic properties along with its non deterministic encryption. As the encryption function is additively homomorphic, the following identities can be described:

Homomorphic addition of plaintexts: The product of two ciphertexts will decrypt to the sum of their corresponding plaintexts $D(E(m_1,r_1)\ .\ E(m_2,r_2)\ mod\ n^2)= m_1 + m_2\ mod\ n$

The product of a ciphertext with a plaintext raising ***g*** will decrypt to the sum of the corresponding plaintexts,

$D(E(m_1,r_1)\ .\ g^{m2}\ mod\ n^2\ )= m_1 + m_2\ mod\ n$

Homomorphic multiplication of plaintexts: An encrypted plaintext raised to the power of another plaintext will decrypt to the product of the two plaintexts,

$D(E(m_1,r_1)^{m2}\ \ mod\ n^2\ )= m_1\, m_2\ mod\ n$

$D(E(m_2,r_2)^{m1}\ \ mod\ n^2\ )= m_1\, m_2\ mod\ n$

III. DESIGN OF CRYPTMDB

In this section, we will introduce the details of CryptMDB. Before executing Mql queries , we create an encrypted user's document in MongoDB as follows:

```
> post = {"Epk(name)" : "Epk(Harsha)",
..."Epk(age)" : Epk(23),
..."Epk(sex)" : "Epk(male)",
..."Epk(Location)" : "Epk(Bengaluru)", }
> db.users.insert(post)
```

Since, pailier Algorithm works on numbers, we convert each key and value into number format. After pailier algorithm is applied on those numbers.

Here,
Name(key) is converted to 1851878757.
Harsha(value) is converted to 79583369193569.
Similarly, all other key and value pair converted to number format as shown above.
For the convenience of description, there we use Epk(mi) to denote the ciphertext of mi. According to the encrypted tool mentioned above, a cipheretext document is created and sent to MongoDB server by proxy server as follows:

```
{ "_id" : ObjectId("5ace6cd1b1ea970474a64577"),
"17880361333142338130021354392805348986976264811
80960625955438344610479590279581409183159248228 8
63000924563559039490133429076036096345031916588 4
76717868024290198118515810144917625760851040225 2
71736029530513511139863073061533426396386010318 7
75326031347901490315612167573547795285727044903 0
40762025402823114800 8" :
"22381110553932608687818338184962220349675534160
22516487969456671056306809966768568588826834791 3
74432729632001075343845366909818512508683928199 2
42332378502331020332498783296964729510954427781 2
27719868780666937750897919605007367489241446056 5
83612148657626372870581592615544325843742692163 5
87238880813480139296 6",

"16170122257890772960153143187894865712000403780
85341972657175011830565124897384583512737477952 7
75378072118194326733677174540621360425626331808 7
88423529630516268714917612359451209062992363439 4
36205271164303831024920281593813721389677164207 9
15175620633918988018534678734865242367185507667 6
46848976530632990330 5" :
"16397646801038108864807494747918362839201960234
19664659294971673452241123837184680858256873614 8
80946018845061660328932424251479988380509001185 4
30799329293323439339577075367473592523208104167 6
78894511370030609219820122166364133847982675558 5
05009032658921699445519423140700230581603809439 9
14669602009725889616 9",

"79000193997077946529721920215651428129302888590
79398014083111405806375288265084523148655044454 3
29418437212324001735813301616397258098577077846 4
03252152410797899622175119908832758331951228722 1
92515600578587880653241795914091550528412546685 9
19490125246342878917485017711372066922414331284 4
96668630024254585236" :
"26054346867392485882595058538145247010327670004
93260008122259725977578543153740576436021951315 1
45124241468711613007764181220148677774558706421 5
57210138778706906896229011024191069112307078317 5
56485920855990299294397666091728080764340218046 9
62825087928292882001327986709500304881425334696 1
80568957046427303149 8",

"36825062527421293448663208928987121808804033369
95958911035655825259727220888907275306235057655 4
13680815353212728720336448128923716183580532235 2
83445203180960597425832194911700694780524106574 0
05794043823633160623625663380487234491278706679 9
17425121293590147899965040493578517910236553857 1
51527275666049119270" :
"25660265960004634490946841429023377320199927686
28247757761110298423465802927676691045622926736 4
99852807003074034930583256431071523235444199155 0
24600554088849337680702175943776477414739408740 3
70784224209358853812240688697477379811814676581 3
00039597935765267186248371085900118964609812325 5
342098508909066703283"}
```

where the id is a identifier of each document assigned by MongoDB automatically. Because the key lengths we adopt are 512 bits, so the lengths of ciphertexts are fairly long..

A. Insert Document :

As mentioned above, user's data will be encrypted by proxy server before executing Mql queries. If we need to add some new information in MongoDB, such as insert a ”favorite book” to user's document, user's computers firstly send a insert Mql request to proxy server as follows:

```
>db.users.update({”id”:ObjectId(”5ace6cd1b1ea970474a64577”)},
...{”$set” : { ”favorite book” : ”Wings of Fire”}})
```

Then this Mql request will be rewritten by proxy server as follows:

```
>db.users.update({”id”:ObjectId(”5ace6cd1b1ea970474a64577”)},
...{”$set” : {Epk(”favorite book”) : Epk(”Wings of Fire”)}
```

Next, proxy server delivers the rewritten Mql request to MongoDB server. Finally, MongoDB server queries over encrypted data and insert Epk (f avorite book) information to corresponding user's document.

B. Query Document:

Similarly, in CryptMDB, if users want to query some document information encrypted in MongoDB, users should send a Mql query to MongoDB proxy server firstly, then this Mql query will be rewritten and sent to MongoDB. For example, f avorite book information have been inserted in user's document above, we can use a Mql query to check whether this information has been stored in MongoDB. Firstly, user sends a Mql query to proxy server as follows:

```
> db.users.find({”name”:”Harsha”, ”age” : 23})
```

Then this Mql query will be rewritten as follows:

```
 >db.users.find({”178803”:”163976”,”161701”:163976})
```

We can see that all the Mql queries of plaintexts are encrypted by proxy server, then these Mql requests are sent to MongoDB server, which only to execute the Mql query of ciphertexts over encrypted data, and return corresponding results to proxy server as follows:

```
{” id”:ObjectId(”5ace6cd1b1ea970474a64577”),
”178803”:”223811”,
”161701”:163976 ,
”790001”:”260543”,
”368250”:”256602”,
”254881”:”209542”,}
```

Here ” 254881”: ” 209542” denotes the ciphertexts of ”favorite book”:” Wings of Fire”. Finally, the proxy server decrypts the ciphertexts and returns the plaintexts to authorised users.

C. Update Document:

In CryptMDB, all Mql requests are encrypted by the proxy server, to the users, they execute Mql queries over CryptMDB without difference compared with unmodified MongoDB. So, as a user, if he want to modify some information which have been stored in CryptMDB(such as the ages of Harsha), a normal Mql request will be sent to the proxy server:

```
> db.users.update({”name”:”Harsha”}
...{”$set” : {”age” : 25}})
```

Then this request is encrypted by the proxy server as follows:

```
> db.users.update({”178803”:”223811”}
 ...{”$set” : {”161701” : 209795}})
```

where 161701denotes the ciphertexts of ages. Next, the proxy server sends the Mql request to MongoDB server which executes the ciphertext orders. Finally, the ages of Harsha will be changed.

D. Remove Document:

Data stored in CryptMDB may be outdated or inaccurate sometimes, in this case, we can utilize the Mql queries to delete these data. Similarly, as mentioned above, if Harsha want to delete his age information, he should send a Mql request to proxy server as follows:

```
> db.users.remove({”age”:23})
```

Then the proxy server rewrites the order as follows:

```
> db.users.remove({”161701”:163976 })
```

E. Aggregation Operation:

In this paper, we adopt an additive homomorphic asymmetric cryptosystem which can achieve additive operations over encrypted data, such summation, average, count, etc. For example, we create several user’s documents in CryptMDB, utilizing aggregate orders provided by MongoDB to add the ages of all users. Similarly, users send Mql requests to the proxy server as follows:

```
        > db.users.aggregate([{$group : { id : ”sex”, num_total : {$sum : $age}}}])
```

Then this Mql request will be rewritten by the proxy server as follows:

```
> db.users.aggregate([{$group : { id : ”790001”, num_total: {$sum : $161701}}}])
```

This is going to return the appropriate results and the reason of additive encrypted tool we adopt, we can utilize the aggregate order installed in MongoDB to execute average, count, and many other aggregate operations.

IV. SECURITY ANALYSIS

A. Confidentiality of Users’ Data :

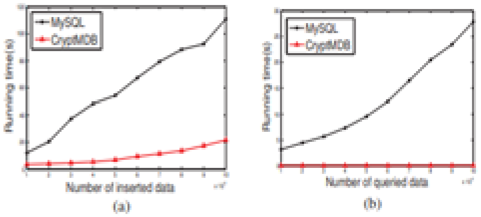
As mentioned before, there has two security threats in CryptMDB. For the threat 1, the MongoDB server is supposed to honest but curious, which can utilize the computing power to infer the user’s information when it executes the Mql queries. But in CryptMDB, all data are encrypted by the proxy server before storing in MongoDB, besides, user’s Mql requests also are encrypted before sending to MongoDB server. Therefore, the tasks of MongoDB server are to execute the encrypted Mql queries over encrypted data, and returns the matching ciphertexts to corresponding users, which cannot deduce any information of plaintexts. Thus, the confidentiality of user’s data can be protected well in CryptMDB.

B. Resist Arbitrary Threat

For the threat 2, when the MongoDB server and proxy server are compromised by attackers, they can use the proxy server to encrypt ciphertexts returned by MongoDB server and get the plaintexts. For this case, we adopt different keys to encrypt each user's information in CryptMDB. In this way, user's keys only be activated by user logged in MongoDB at that time. Thus, although the proxy server and MongoDB server have been compromised by attackers, they only can steal the information from current users, and other user's data (which are not logging in MongoDB) do not reveal to any attacker.

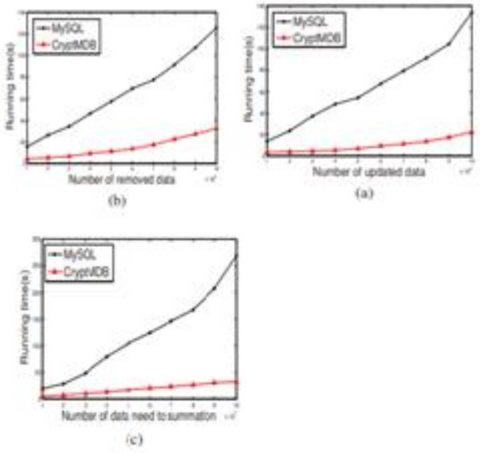
V. PERFORMANCE EVALUATION

In this section, we will evaluate the performance of CryptMDB by comparing with MySQL in terms of insert, query, update, remove, and aggregation operations. For the authority of experiment, the same encryption tool is used in MySQL. Besides, all the experimental procedures are performed on an Intel Core i5 3.2GHZ system.



(a) (b)

Fig. 2: Total running times. (a) For the different number of inserted data. (b) For the different number of queried data.



(b) (a) (c)

Fig. 3: Running time. (a) For the different number of updated data. (b) For the different number of removed data. (c) For the different number of data need to summation.

A. Insert Operations

As shown in Fig. 2.(a), we can see that the MySQL and CryptMDB are inserted users' records from 10000 to 100000 respectively. The results shows that CryptMDB has higher insertion speed compared with MySQL. For example, when the number of inserted data reach 100000, the CryptMDB only takes 21.513s to complete inserted operations while MySQL needs 111.025 to finish the same operations.

B. Query Operations

Similarly, Fig. 2.(b) shows that the running times with different number of queried data, from the picture it is obvious that the CryptMDB has stronger queried ability compared with MySQL. For example, when the number of queried data reach 100000, the total running times of MySQL rapidly up to 27.884s but CryptMDB only takes 0.064s.

C. Update Operations

As shown in Fig. 3.(a), with the increase of user's data, it is easy to find that the running time of two databases both are increased quickly. But compared with MySQL, we can see that the updated performance of CryptMDB is better than MySQL, one of the major reasons is that the strong ability of CryptMDB to achieve large scale data access and calculating. Thus, it can conduct updated operations by combing other servers to improve the execution speed. For example, when the number of updated data reach 100000, MySQL need 133.821s to execute all the sql requests but CryptMDB only takes 22.513s to achieve the same tasks.

D. Remove Operations

Similarly, we also analyze the removed performance of CryptMDB by comparing with MySQL in same experimental environment. As shown in Fig. 3.(b), we remove the users' data from 10000 to 100000 orderly, it is not difficult to find that CryptMDB has higher performance to remove user's information, especially when users' data are huge.

E. Aggregation Operations

Because an additive homomorphic asymmetric cryptosystem is adopted in this paper, we evaluate the aggregation ability of two databases. As mentioned before, although all the user's data are stored in database in the form of ciphertexts, we also can execute some aggregation operations such as summation, average, count, etc. In this section, we take the summation as a example. Fig. 3.(c) shows that the running times with different number of data need to summation, because the strong ability of distributed data processing, it is undoubted that the CryptMDB has lower running time to achieve same operations compared with MySQL.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a practical encrypted MongoDB (i.e., CryptMDB) to achieve the privacy protection of user's data stored in database. The key idea of the CryptMDB is utilizing an additive homomorphic asymmetric cryptosystem to encrypt user's data. Security analysis demonstrates that the cryptMDB can achieve strong privacy protection for user's data and prevent adversaries from illegally gaining access to the database. Future works include: 1. Incorporate all the dynamic operations supported by mongo DB into our portal. 2. Integrate our portal with cloud service providers. 3. restoration techniques to get back the encrypted data in CryptMDB deleted by unauthorized person.

REFERENCES

[1] Z. Zhang, K. Barbary, F. A. Nothaft, E. R. Sparks, O. Zahn, M. J. Franklin, D. A. Patterson, and S. Perlmutter, "Kira: Processing astronomy imagery using big data technology," IEEE Transactions on Big Data, 2016.

[2] J. Chen, Q. Jiang, Y. Wang, and J. Tang, "Study of data analysis model based on big data technology," in IEEE International Conference on Big Data Analysis (ICBDA), March 2016, pp. 1–6.

[3] https://dzone.com/articles/when-use-mongodb-rather-mysql

[4] http://people.csail.mit.edu/nickolai/papers/raluca-cryptdb.pdf

[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. ACM, 2011, pp. 85–100.

[6] Deshmukh, A. Pasha, and D. Qureshi, "Transparent data encryption solution for security of database contents," International Journal of Advanced Computer Science and Applications, vol. 2, no. 3, pp. 25– 28, March 2011.

[7] R. A. Popa, N. Zeldovich, and F. H. Li, "An ideal-security protocol for order-preserving encoding," in IEEE Symposium on Security and Privacy. IEEE, 2013, pp. 463–447.

[8] https://en.wikipedia.org/wiki/Paillier_cryptosystem