I. INTRODUCTION

With the rapid development of network and information technology, data are woven into every corner of our lives, such as websites, social networks, public platforms, etc. Along with the convenience and immediacy of data acquisition, big data technologies, and many other new data analysis tools [1], [2] have prevailed and applied in various fields of people's lives, which have generated enormous social impact and economic benefits [3], [4].

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. However, with the data explosion, especially in big data era (data are growing at an alarming rate every day), traditional relational databases cannot address users’ demands for quick data access and calculating, since they cannot process data in a distributed way. To tackle this problem, MongoDB, a new non-relational database, which has been favored in more and more enterprises and individuals due to its strong ability in distributed data processing. It is generally known that MongoDB is one kind of non-relational databases based on documents. Different from other relational databases, arbitrary type data can be stored in a document in MongoDB. Therefore, compared with relational databases, data can be more easily stored on other servers. As a result, data can be processed in distributed servers for large-scale access and calculation tasks.

However, existing MongoDB products nearly fail to consider an urgent and practical issue in databases, i.e., privacy protection. It is widespread that data is stored without any safety measures on commonly used databases, which is vulnerable 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 maliciously peep data stored in databases due to it has the full access permission.

Therefore, it is urgent to propose a privacy-preserving approach which can ensure confidentiality of users’ information on MongoDB. In the last few years, several encryption schemes have been applied in relational databases. Raluca et al. design a CryptDB system in MySQL, which uses an onion encryption structure to support 99.5% operations over encrypted data. Deshmukh 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 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, i.e., CryptMDB, 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:

• We leverage an additive homomorphic asymmetric cryptosystem to design an encrypted MongoDB (i.e., Crypt-MDB),
which can achieve additive operations over encrypted data.

- Security analysis shows that the CryptMDB can achieve strong privacy protection of users' information stored in databases. Besides, extensive experiments indicate that the CryptMDB 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. METHODOLOGY

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

### A. CryptMDB 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 authorized 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

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.

#### D. Cryptographic Tool

In this paper, a symmetric cryptosystem is adopted. In the CryptMDB, we use AES-256-ECB encryption algorithm to encrypt the users data

i. Class Hierarchy several classes are defined in order to implement the security enforcement and their relationships are shown in Table I.

ii. Class TobeEncrypted We define a new class TobeEncrypted to store to-bestored data that need encryption. Inside this class is an Object instance and basic getter and setter methods.

iii. Class EncryptedBasicDBObject This class is extended from BasicDBObject. It mainly overrides BasicDBObject class's put and get methods. Put method asks for two parameters: String key and Object obj. By judging the class of the second parameter, we can decide whether it need encryption or not. We serialize the tobeencrypted object in order to convert it into byte form and then encode it. The result of encryption is of class byte[]. In case that some clear message is also in the format of byte[], we attach a flag information before the user's input to indicate that this is something that need encryption. When retrieving the object from an EncryptedBasicDBObject instance, we call override function get. Before returning the value, we check if it need decryption. If it began with the encrypted flag information, we decrypt it first and then return the result. Fig. 1 and Fig. 2 are the flow charts which show the process of override put and get method. Besides, we override method toString. This function will decrypt the coded key-value pair in a EncryptedDBObject object and then return a string in JSON format. We also define two
more converting methods that can facilitate other Encrypted classes. Method convert(EncryptedBasicDBObject encobj) is designed to convert EncryptedBasicDBObject instance into BasicDBObject instance by iteratively getting values from encobj, decrypting them and then put into a BasicDBObject instance. Method convertJe(DBObject obj) just do the opposite work. It gets values from obj and put them into an EncryptedBasicDBObject instance without doing anything to the values.

iv. Class EncryptedDBCollection This class is of vital importance because it offers most of the method to operate on the database. The processing, however, is comparatively easy. It only calls methods convert and convert_re, if necessary, before throwing those 908 parameters got from users to its father methods defined in DBCollection.

v. Other Classes as for EncryptedMongo, EncryptedDB, EncryptedDBCollection and EncryptedDBCursor, we simply extend them from proto-classes and set a private member of the prototype instance. All these newly defined classes is more like an encapsulation because most operation are all done by calling original methods. The major difference between new classes and proto-classes are the different types of the parameters.

III. DESIGN OF CRYPTMDB

In this section, we will introduce the details of Middleware. Before executing Mql queries , we create an encrypting tool which sends data to MongoDB to store in encrypted form and only authorized user is having a key to access the same data.

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 We provide an “Add Book” functionality to insert books in BookData Store in MongoDB.

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, 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({"title" : "Jaguar", "Pages" : 30})
```

Then this Mql query will be rewritten as follows:

```
> db.users.find("0x22aa48", "0x7c5e95", "0x6e60a31")
```

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("4b25b067525f35f94b60a31"),
  "0x22aa48" : "0x7c5e95",
  "0x6e60a31" : "0x3f64a4",
  "0x7c5e95" : "0x6e60a31",
  "0x7c5e95" : "0x6e60a31"
}
```

Here "0x22aa48" : "0x7c5e95" denotes the ciphertexts of "book" : “ Do and Die”. Finally, the proxy server decrypts the ciphertexts and returns the plaintexts to authorised users.

C. 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 Alice want to delete him location information, he should send a Mql request to proxy server as follows:

```latex
> db.users.remove("title": "Jaguar")
```

Then the proxy server rewrites the order as follows:

```latex
> db.users.remove("0x1135b5": "0x29e092")
```

Next, the MongoDB server queries over encrypted data and deletes the title information. Moreover, we can use the query operations to check whether this information has been deleted:

```latex
> db.users.find("0x3f ec40": "0x37f9f6", "0x6ed803": "0x2f83ff")
```

The results provided by MongoDB server as follows:

```latex
{
"_id": ObjectId("4b253 b067525f135f94b60a31"),
"0x3f ec40": "0x37f9f6",
"0x6ed803": "0x2f83ff",
"0x7ad4b": "0x193ad3",
"0x22aa 48": "0x7c5e95",
}
```

From the results above, the title information has been deleted.

### IV. SECURITY ANALYSIS

In this section, we analyze the privacy protection of Crypt-MDB in querying over encrypted data. In this paper we mainly focus the security threat in II, other security features are not the focus of our concern.

#### A. Confidentiality of Users’ Data

As mentioned before, there are 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 Crypt-MDB 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 Inter Core i5 3.2GHZ system.

#### 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 21.518s to complete inserted operations while MySQL needs 121.032 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 inserted data reach 100000, the CryptMDB only takes 21.518s to complete inserted operations while MySQL needs 121.032 to finish the same operations.
Fig. 2: Running time. (c) For the different number of removed data.

C. Remove Operations

Similarly, we also analyze the removed performance of CryptMDB by comparing with MySQL in same experimental environment. As shown in Fig. 2(c), 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.

VI. RELATED WORK

With information explosion and development of network and information technology, encrypted database, which is designed to achieve user’s data privacy protection has drawn more and more attention [11]–[14], and has been proposed in various application scenarios. For example, Raluca et al. [7] design a CryptDB system based on MySQL, which uses an onion encryption structure to support 99.5% operations over encrypted data. Deshmukh et al. [8] propose a transparent data encryption scheme to provide high levels of security for columns, tables and tablespace in Microsoft SQL Server 2008. Then Raluca et al. [9] 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.

We know that user’s privacy have been proposed in many fields [15]–[18], also including relational database. However, few specific encrypted tools which have been applied in non-relational database. Recently, Tian et al. [19] proposes a transparent middleware implementation in MongoDB, which can efficiently encrypt sensitive data specified by users on a dataset level, however, it cannot achieve large scale aggregation operation. With the data explosion, especially in big data era (data is growing at an alarming rate every day), due to users’ demands for speed of data access and calculating [20] – [22]. Therefore, it is urgent to propose a privacy-preserving approach which can ensure confidentiality of users’ information over non-relational database. In this paper, we propose a practical encrypted MongoDB, i.e., CryptMDB, which can guarantee strong privacy protection and high performance in non-relational database. Furthermore, extensive experiments demonstrate that the CryptMDB is better than existing relational database in terms of data access and calculating.

VIII. ACKNOWLEDGMENT

I would like to express my deep gratitude to my guide Prof. Mona Mulchandani for their valuable guidance and encouragement.

I would also like to sincerely thank all the faculty members of CSE Department and my colleagues for helping me throughout this project.

REFERENCES


