Flexible Access Control and Confidentiality over Encrypted Data for Document-based Database

Maryam Almarwani, Boris Konev and Alexei Lisitsa
Department of Computer Science, University of Liverpool, Liverpool, U.K.

Keywords: Document Database, Querying over Encrypted Data, Access Control, Confidentiality.

Abstract: In this paper, we present a SDDB scheme regarding document-based store that satisfies three security requirements: confidentiality, flexible access control, and querying over encrypted data. The scheme is inspired by PIRATTE and CryptDB concepts. PIRATTE is a proxy for sharing encrypted files through a social network between the data owner and the number of users and the files are decrypted on user side with the proxy key, whereas in CryptDB, it is proxy between a database and one user to encrypt or decrypt data based on user's queries. The scheme also improves CryptDB security and provides the possibility of sharing data with multi-users through PIRATTE concept which is used to verify authentication on the proxy side.

1 INTRODUCTION

Databases often contain sensitive data such as personal and governmental information. The security protection of such data has to address threats coming from both external and internal adversaries. While access control is commonly used to alleviate external threats, encryption is applied to prevent data leaks to both external and internal (e.g. curious administrators) attackers. Using encryption is not without issues. In order to query encrypted data, either they have to be decrypted, making possible data leaks, or special methods for querying encrypted data with limited querying power and efficiency have to be used.

The authors of (Ferretti et al., 2013) identified three requirements for secure databases: (i) Confidentiality, (ii) Enforcement access control, and (iii) Querying over encrypted data. Several systems satisfying some of these requirements have been proposed. In the context of relational databases CryptDB system (Popa et al., 2011) addresses requirements (i) and (iii), while DBMask (Sarfraz et al., 2015) deals with all (i)-(iii). For NoSQL databases the requirements (i) and (iii) are discussed, e.g. in (Xu et al., 2017) (document-based DB) and (Aburawi et al., 2018) (Graph DB).

Recent years saw rise in popularity (Tre,) of various NoSQL datamodels and DBMSs, as they able to store and process huge amounts of structured and unstructured information effectively. There are four basic types of NoSQL databases: (1).Key-value store, (2).Document-based store, (3).Column-based store, (4).Graph-based store. According to October 2018 ranking (ran,) document-based store (MongoDB) occupies the 1st rank of NoSQL database used. No comprehensive solutions for secure document-based stores addressing all requirements (i)-(iii) have been proposed so far to the best of our knowledge.

In this paper, we propose a particular scheme SDDB (Secure Document DataBase) for implementing secure NoSQL document-based databases. In this schema to satisfy (i)-(iii) we adopt CryptDB approach for querying encrypted data, originally proposed for relational DB and SQL (Popa et al., 2011), for the case of document-based databases. We further combine it with improved fine-grained access control originated in PIRATTE scheme (Jahid and Borisov, 2012).

The rest of the paper is organized as follows. Section 2 presents background of CryptDB and PIRATTE systems. Section 3 presents details of Secure Document Database (SDDB) scheme. Section 4 outlines a case study of an application of SDDB scheme. Performance and security of the SDDB are discussed in Section 5. Section 6 presents related work. Finally, Section 7 concludes the paper.

2 BACKGROUND

This section presents some background for CryptDB and PIRATTE concepts used in our proposal.

Almarwani, M., Konev, B. and Lisitsa, A.
Flexible Access Control and Confidentiality over Encrypted Data for Document-based Database.
DOI: 10.5220/0007582506060614
In *Proceedings of the 5th International Conference on Information Systems Security and Privacy (ICISSP 2019)*, pages 606-614
ISBN: 978-989-758-359-9

CryptDB (Popa et al., 2011) is the first practical database system which supports SQL queries over encrypted data. It functions as a proxy between the database and the user by rewriting a query to execute querying over encrypted data on DBMS without revealing plaintext and forwarding the encrypted result receiving from DBMS to the user side after decrypting. It uses different encrypted techniques depending on the data type and operation such as Random(RND) and Deterministic(DET). These types are implemented using onion layers as depicted below in Figure 1. CryptDB lacks the capacity to enforce fine-grained access control of cells and columns. CryptDB only enforces row access by using a proxy-based reference monitor due to encrypting each column data by the one key. Also, users on CryptDB cannot enable sharing their data with a group of users.

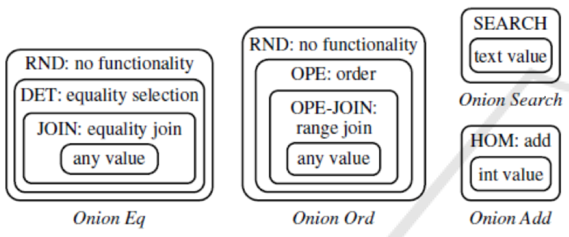


Figure 1: Onion encrypted layer in CryptDB(Popa et al., 2011).

To mitigate CryptoDB limitations and adopt it for document based DB, we suggest using advanced cryptographic primitives such as cipher-policy attributed based encryption(CP-ABE) (Bethencourt et al., 2007). Jahid and Borisov proposed the PIRATTE scheme (Jahid and Borisov, 2012) in which CP-ABE is integrated with user revocation mechanism using a proxy that handles attributes and user revocation to allow dynamic users. The data owner in PIRATTE issues user secret keys and a proxy key once, but for user revocation, only the proxy key is updated.

3 SDDB SCHEME

This section presents an overview of SDDB scheme, i.e., system requirements, architecture, threat model, SDDB workflow and its algorithms.

3.1 SDDB Requirements

Our proposed design satisfies the following requirements :

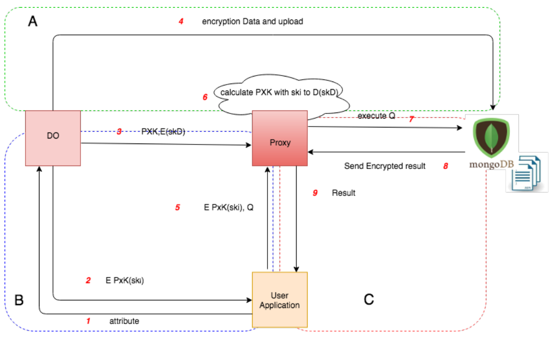


Figure 2: SDDB Scheme Architecture.

- **C1: Querying over Encrypted Data.** The scheme is able to perform operations and queries over encrypted data without revealing the data.
- **C2: Flexible Access Control.** Users have access to parts of the data based on data owner's policy and the policy can be changed if it is necessary.
- **C3: User Revocation Support.** The scheme can revoke the users based on data owners requests, so they cannot access the data after that.
- **C4: No Re-encryption Data.** Data does not need to be re-encrypted in the case of an user revocation.
- **C5: No Re-distributed Key.** Keys don't need to be re-distributed in the case of an user revocation.
- **C6: Multi-user Sharing Support.** Based on data owner's policy (parts of) data can be accessed by multiple users.
- **C7: Security and Performance Trade-off.** The scheme allows configuring for better security or performance.

3.2 SDDB Architecture

Figure 2 illustrates the proposed that includes four entries: Data Owner (DO), User Application, Proxy, and DBMS Server. Their interaction can be described as follows:

1. DO is the authority responsible for establishing access privileges for her/his data which will be called access policy(AP) in the rest of the paper. For example, DO may be a user who shares his or her data through applications. This entity uses a different keys to encrypt different parts of the data depending on the AP and uploads them to the DBMS server. It creates the secret keys for each access policy (AP) and distributes the secret keys to the users. Then, the proxy verifies authenticated user who can access the data.

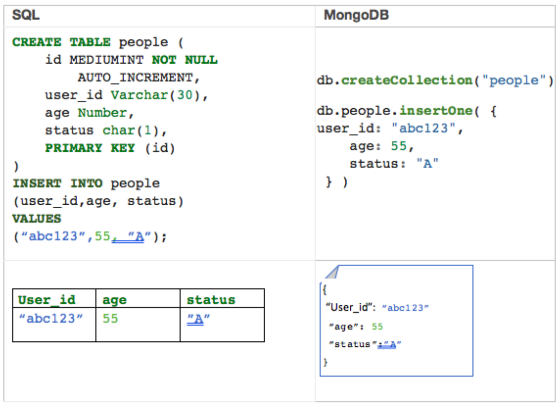
2. User Applications are users who request data sharing for the DO. The users send their attributes to the DO and receive their secret keys using which they identify themselves to the proxy for accessing parts of the data and executing their queries.
3. Proxy is the intermediate server between DO, user application, and DBMS. It is responsible for verifying the right access for each user and rewriting queries to be executed on the encrypted data.
4. DBMS server is a server that provides database services such as storage and retrieval. It is responsible for storing encrypted data and executing the query without revealing the plaintext data (if possible).

3.3 Threat Model

- DO: It is trusted and remains offline after encrypting and uploading the encrypted data and distributes the keys to the users unless a new user requests permission or until a user access rights is removed by the owner.
- User Applications: It is untrusted. Therefore, the proxy must verify them before allowing them to access and query the data. Proxy and DO do not share decryption keys with them.
- Proxy: It is semi-trusted. It obtains encrypted keys and cannot decrypt data alone.
- DBMS server: It is semi-trusted and thus cannot obtain the keys to decrypt inner layer.

3.4 Data Format and Query Language

While SDDB scheme can be implemented for various document-based DBs, we consider MongoDB environment as a primary target. We briefly outline here the data storage format and the query language for MongoDB that will be needed for understanding of the rest of the paper. Firstly, MongDB represents its data in binary-encoded format called BSON. BSON is an extension for JSON with additional data types, such as *date* and *binary*, and embedding feature. It provides high-efficiency of encoding and decoding with different languages and can be found a more detailed information at http://bsonspec.org/. Secondly, MongoDB query language has its own syntax compatible with JSON structure and is nearly as powerful as SQL. For example, a database contains a collection (table) called "people" that consists of three fields (columns) represented by SQL and MongoDB query as shown in Figure 3. Some examples of MongoDB querying below on the previous example



SQL	MongoDB
`CREATE TABLE people ( id MEDIUMINT NOT NULL AUTO_INCREMENT, user_id Varchar(30), age Number, status char(1), PRIMARY KEY (id) ) INSERT INTO people (user_id,age, status) VALUES (“abc123”,55, ”A”);`	`db.createCollection("people") db.people.insertOne( { user_id: "abc123", age: 55, status: "A" } )`
User_id / age / status: “abc123” / 55 / ”A”	{ “User_id”: “abc123” “age”: 55 “status”:”A” }

```
CREATE TABLE people (
    id MEDIUMINT NOT NULL
        AUTO_INCREMENT,
    user_id Varchar(30),
    age Number,
    status char(1),
    PRIMARY KEY (id)
)
INSERT INTO people
(user_id,age, status)
VALUES
(“abc123”,55, ”A”);
```

```
db.createCollection("people")

db.people.insertOne( {
user_id: "abc123",
    age: 55,
    status: "A"
 } )
```

User_id	age	status
“abc123”	55	”A”

```
{
 “User_id”: “abc123”
  ”age”: 55
  ”status”:”A”
}
```

Figure 3: SQL vs MongoDB Structure.

and for more types of queries in details see at https://www.tutorialspoint.com/mongodb/index.htm./.

1. Find all Documents in the collection:

   ```
   db.people.find({})
   ```

2. Find all Documents on the collection containing user-id="abc123":

   ```
   db.people.find
   ({"user_id":"abc123"})
   ```

3. Update age on all Documents in the collection containing user-id="abc123" to 56:

   ```
   db.people.UpdateMany({"user_id":
   "abc123"},{\$set:{"age":56}})
   ```

3.5 SDDB Workflow Overview

The DO has documents that are written in BSON format, and the data sharing between the Data owner and the users is achieved in four stages.

1. **Data Encryption (Stage A).** This stage includes system initialization and encrypting data and uploading it. The DO issues access policy (AP) ,that holds attributes to determine who users can access data, for each group of documents. It then issues secret keys such as public key and proxy key based on PIRATTE SETUP algorithm and onion key for each AP based on algorithms used in such as AES-CBC and Homomorphic Encryption based on CryptDB SETUP algorithm. Then, the DO uses the secret keys, i.e. master key, collection key, document key, onion key and layer key, for encrypting data for several onions and layers, as shown in figure 3(A) similar to SQL-aware Encryption on CryptDB. To optimize performance, the DO does not have to encrypt all the documents in all onions and layers and instead

chooses the appropriate onions and layers as per the level of sensitivity of DO's data and desired queries. The DO then encrypts the secret keys using the public key and sends them to proxy with the proxy key.

2. **Flexible Access Control (Stage B).** This stage is from enforcing access control step for PIRRATTE but it executes on the proxy side. It is responsible for verifying a user. Users send their attributes to the DO to obtain users' secret keys. Users then send their secret key, attributes, and query to proxy. Proxy, in turn, combines the proxy key, the encrypted secret data's secret keys, and the users' secret keys to obtain data secret keys,as shown in figure 3(B). If the proxy obtains correct secret keys, it will move to the next stage(C) otherwise the users' query will be rejected.
3. **Data Query and Retrieval (Stage C).** This stage is from querying and retrieving data steps for CryptDB system. At this stage, a query is executed and high security is provided, as shown in figure 3(C). Proxy rewrites the query according to the layer that can be executed on it, for which some querying may require adjustment layer before the user query is executed. Back to the previous example on 2.3 section, if the collection encrypted on two layers (RND, DET see section 5 for details of these algorithms), therefore, query 1 will be executed immediately on RND layer but query 2 will be executed after peeling off RND layer. Some queries cannot be executed on encrypted data and this case will not be discussed in this paper left for future work. Then, the proxy sends user query to DBMS which executes it and sends the encrypted result. Proxy decrypts the encrypted result and sends it back to the user. For higher security, we suggest that the proxy re-encrypts the peeling layers after each query or user session ends.
4. **Revoked Users (Optional).** This stage is exactly from revocation step for PIRATTE. This stage is executed if the DO wants to revoke a user. The DO will update only the proxy key without affecting other users in the system.

3.6 SDDB Algorithms

An SDDB scheme can be outlined by nine algorithms, which is taken from PIRATTE and CryptDB algorithms. These algorithms are classified according to which stage they are used in, as follows:

- **Stage A Algorithms.** The algorithms of this category are taken as follows:
  (i)AttrGen(),KeySetup(),ProxyKeySetup() from PIRATTE scheme.
  (ii)LayerKeySetup() and EncKey() from CryptDB.
  - **AttrGen ().** Data Owner defines sets of attributes and generates access policy (AP) for each set of attributes.
  - **KeySetup ().** Data Owner generates Public Key (PK) and Master Key (MK) for each access policy.
  - **LayerKeySetup ().** Data Owner runs setup function for each encryption algorithm, such as AES, using it to encrypt data for each access policy in order to obtain secret layers keys (SLK).
  - **ProxyKeySetup (PK, MK, AP).** Data Owner generates Proxy Key (PXK) by PK and MK for each AP to decrypt (SKL).
  - **EncKey (PK, K).** Data Owner encrypts SKL and MK by PK.
- **Stage B Algorithms.** The algorithms of this category are taken from PIRATTE.
  - **KeyGen (MK, AP).** DO generates a set of secret keys for 'i' users (SKi) from MK corresponding to AP.
  - **DecKEY (E, PYX, SKi).** Proxy decryptes secret keys.
- **Stage A and B Algorithm.** The algorithm of this category is taken from CryptDB.
  - **LayerEnc (SLK, D).** Data Owner or Proxy runs Encryption function for each encryption algorithm by using data (D) and SLK to obtain decryption data(C).
- **Stage C Algorithm.** The algorithm of this category is taken from CryptDB.
  - **LayerDec (SLK, C).** Proxy runs Decryption function for each of the encryption algorithms by using C and SLK to obtain data (D).

4 CASE STUDY

In this section, it is assumed that the DO has a set of data consisting of one collection that comprises two documents, each with two fields. These two fields, ID and Name, are integer and string data type respectively and Q1 will be executed through MongoDB querying language for both without-SDDB and with-SDDB.

```
Q1:(db.collection-1.find({ID:23},
{name:1}))
```

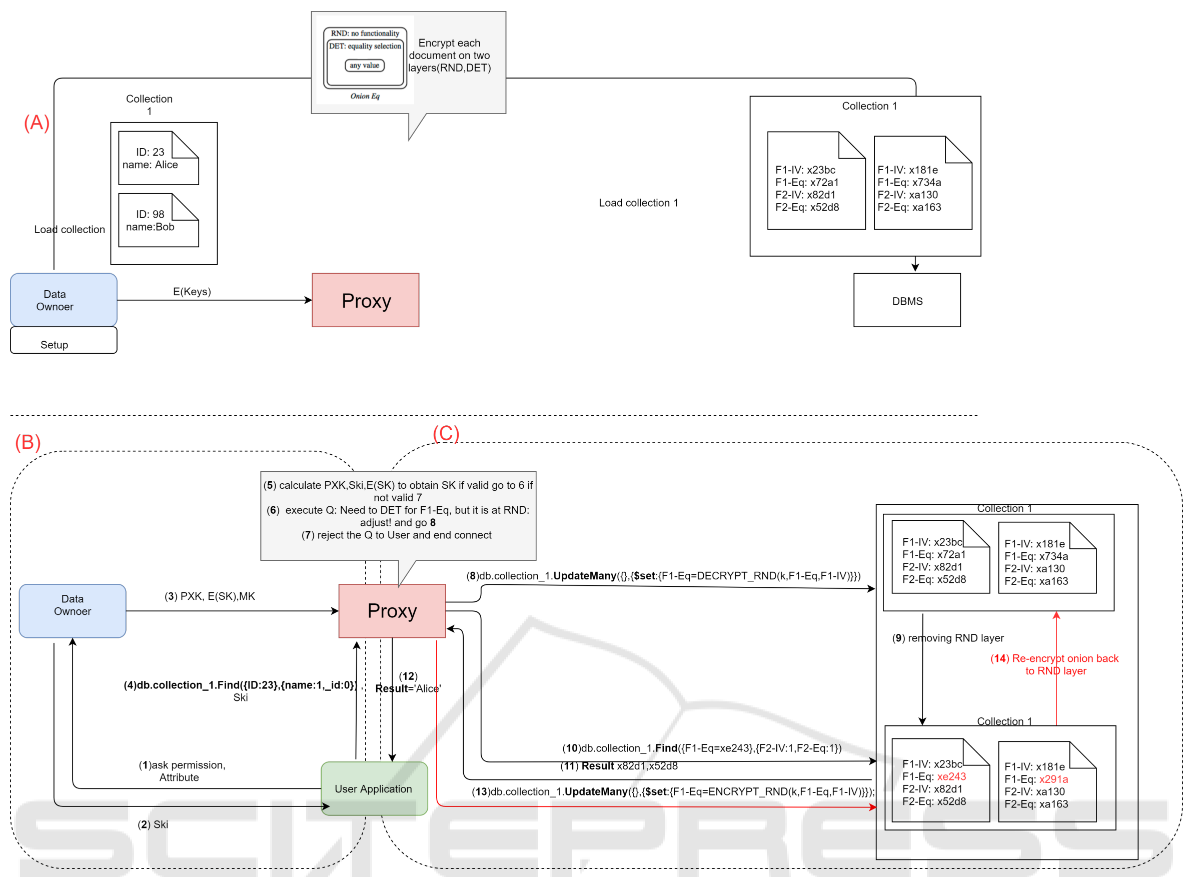

(A)
RND: no functionality
DET: equality selection
any value
Onion Eq
Encrypt each document on two layers(RND,DET)
Collection 1
ID: 23 name: Alice
ID: 98 name:Bob
Load collection
Load collection 1
Collection 1
F1-IV: x23bc F1-Eq: x72a1 F2-IV: x82d1 F2-Eq: x52d8
F1-IV: x181e F1-Eq: x734a F2-IV: xa130 F2-Eq: xa163
Data Owner
Setup
E(Keys)
Proxy
DBMS
(B)
(C)
(5) calculate PXK,Ski,E(SK) to obtain SK if valid go to 6 if not valid 7
(6) execute Q: Need to DET for F1-Eq, but it is at RND: adjust! and go 8
(7) reject the Q to User and end connect
Data Owner
(3) PXK, E(SK),MK
Proxy
(8)db.collection_1.UpdateMany({},{$set:{F1-Eq=DECRYPT_RND(k,F1-Eq,F1-IV)}})
(4)db.collection_1.Find({ID:23},{name:1,_id:0}) , Ski
(12) Result='Alice'
(1)ask permission, Attribute
User Application
(2) Ski
(9) removing RND layer
(14) Re-encrypt onion back to RND layer
(10)db.collection_1.Find({F1-Eq=xe243},{F2-IV:1,F2-Eq:1})
(11) Result x82d1,x52d8
(13)db.collection_1.UpdateMany({},{$set:{F1-Eq=ENCRYPT_RND(k,F1-Eq,F1-IV)}});
Collection 1
F1-IV: x23bc F1-Eq: x72a1 F2-IV: x82d1 F2-Eq: x52d8
F1-IV: x181e F1-Eq: x734a F2-IV: xa130 F2-Eq: xa163
Collection 1
F1-IV: x23bc F1-Eq: xe243 F2-IV: x82d1 F2-Eq: x52d8
F1-IV: x181e F1-Eq: x291a F2-IV: xa130 F2-Eq: xa163

Figure 4: Case Study.

4.1 Without-SDDB

This scenario is used in case the system does not provide encryption of data or verified access. DO uploads the data in DBMS as Plaintext and assumes that access privileges are granted to users by sending its password (PW) to them. Therefore, the user uses PW with Q1 through user application that, in turn, sends it to the DBMS server to execute and then returns the result to the user, where Name=Alice is matched to ID=23.

4.2 With-SDDB

This scheme can be divided into cases based on classes of computation required by the application's queries. For example, in this case study, as the application requires queries(insert,delete,update,select) with equality operation, the data is encrypted considering onion equality between two layers, RND and DET, as shown in figure 4.

The table 1 shows the notations that will be used in this case study. If the DO wishes to share the data with the Ui and only allow for equality computation, the system will have to be followed in four stages.

Table 1: Scheme's Notations.

Notation	*Description*
DO	Data Owner
Ui	Users or user applications
KUi	Key for user i
DET	Deterministic Algorithm
RND	Random Algorithm
AP	Access policy
PK	Public key
MK	Master key
CK	Collection key
DK	Document Key
OK	Onion Key
LK_{RND}	$RNDLayerKey$
LK_{DET}	$DETLayerKEy$
PXK	Proxy key
Qi	User's or Proxy's Query

1. **Data Encryption (Stage A)**

   1.1 **System Initialization.** For this substage, Do runs AttrGen, KeySetup,LayerKeySetup,ProxyKEySetup,EncrKey once for documents.The DO creates AP, for example (Doctor AND Surgery Department), and establishes the keys for PK, MK, CK, DK, OK, LK_{RND} , LK_{DET}, and PXK according to each AP and encrypts all previous keys, except PK and PXK by PK (i.e. E_{PK}(MK), E_{PK}(CK),E_{PK}(DK), E_{PK}(OK), E_{PK}(LK_{RND}),E_{PK}(LK_{DET})), and sends them with PXK and AP to the proxy and store in-memory, as shown in table 2.

Table 2: Scheme Keys.

AP	(Doctor AND Surgery Department)
MK	EPK(MK)
CK	EPK(CK)
DK	EPK(DK)
OK	EPK(OK)
LKRND	EPK(LKRND)
LKDET	EPK(LKDET)
PX	PXK

1.2 **Data Encryption and Upload.** DO runs LayerEnc and uses MK, CK, DK, OK, and LKs to encrypt the data in an equality onion and upload it to BDMS.

2. **Flexible Access Control (Stage B)**

2.1 **User Registration.** Ui sends their attributes to the DO to obtain KUi. For example, assume U1 and U2 are Doctor and Surgery Department, and U3 is a Nurse and Surgery Department. The DO runs KeyGen to create KU1 and KU2 and sends it to U1 and U2, respectively, and rejects the U3 request that does not satisfy AP.

2.2 **Verifying User.** For example, U1 sends KU1 with his/her attributes and Q1 to the proxy which runs DecKey. Before executing Q1, the proxy must verify the user's authentication to obtain the secret keys. The proxy calculates PXK and KU1 with each encrypted key to obtain MK, CK, DK, OK, and LKs.

3. **Data Query and Retrieval (Stage C)**
The proxy follows the following steps:

3.1 Examines the current layer of ID field and its suitability to Q1 if Q1 requires DET layer and the current is RND.

3.2 Decrypts the field (i.e. F1-Eq) corresponding to ID by issuing a query Q2 and sends it to DBMS by LayerDec.

```
Q2:(db.collection-1.updateMany(
${},{$set=F1-Eq =DECRYPT_RND(
k,F1-Eq,F1-IV}))
```

3.3 Rewrites Q1 query by replacing the constants with the corresponding value on DET and swaps the fields corresponding to the DB fields as Q3 for executing it on the DBMS server, after which the result F2-Eq, F2-IV is sent to the proxy.

```
Q3:db.collection-1.find({F1-Eq=
xe243},{F2-IV:1,F2-Eq:1})
```

3.4 Decrypts F2-Eq twice in RND and DET on the proxy side by LayerDec, as the F2-Eq field is still in RND, to obtain the result (name="Alice") and send it to U1.

3.5 Returns F1-Eq to RND to obtain higher security by issuing an opposite query Q4 to Q2 by LayerEnc.

```
Q4:db.collection-1.updateMany({},
{\$set=F1-Eq=ENCRYPT_RND(
k, F1-Eq,F1-IV)})
```

4. **Revoked Users (Optional)**
This is an optional step that is executed if the DO wants to revoke the user. For this, the DO only updates PXK and resends it. For example, to revoke U1, PXK is updated without having to data re-encryption or key distribution and without affecting the rest of the users. After that, U1 sends a query to a proxy after revocation that its requests cannot be executed because he/she has lost authentication and rejected.

5 DISCUSSION

In this section, there will be a discussion regarding the security and performance of the two previous models in the case study, as well as studies inspired by it. Finally, we briefly compare our scheme with some existing works. In term of security, in the without-SDDB model, if the DBMS server is exposed to compromise or curiosity, it will reveal plaintext data because it is not encrypted. In addition, it is likely that the adversary will be able to obtain the password and impersonate the data owner to manipulate the data. Therefore, this requires a secure channel to exchange it.

In the with-SDDB example, the DBMS server does not reveal the information except where it is detected based on the encryption algorithm in the current layer used, such as equal in DET. Therefore, the decryption layer will be returned to high-security layers other than CryptDB. In this model, there is also no need to share encryption keys because different keys are used for both user authentication and

Table 3: Comparison of our scheme with some existing work.

	(Popa et al., 2011)	**(Xu et al., 2017)**	**(Aburawi et al., 2018)**	**(Sarfraz et al., 2015)**	**(Ferretti et al., 2013)**	**(Pirretti et al., 2006)**	**SDDB**
C1	✓	✓	✓	✓	✓	X	✓
C2	☐	☐	X	✓	✓	✓	✓
C3	☐	☐	☐	✓	☐	✓	✓
C4	☐	☐	☐	Only user's group belong	☐	✓	✓
C5	☐	☐	☐	Only user's group belong	☐	✓	✓
C6	X	✓	✓	✓	✓	✓	✓
C7	✓	X	✓	X	X	☐	✓
DB	Relational DB	Document DB	Graph DB	Relational DB	Relational DB	☐	Document DB
PS	✓	☐	✓	✓	X	☐	In the future

Notes.{✓} :satisfies. {X} : does not satisfy. {☐} : Out of scope the paper.

data encryption, according to access privileges other than CryptDB. PIRATTE runs decryption data on the user's side and is not trustworthy and therefore, this permission is given to the proxy in our scheme. In the case that the adversary is able to obtain the keys, they cannot impersonate the data owner by associating these keys with the user's attributes and the Proxy Key. Furthermore, in the case that the proxy is exposed or compromised, there will not be a leakage of data because it cannot decrypt keys alone.

In terms of performance, the without-SDDB model can execute any type of query or computations at high speed. Meanwhile, the with-SDDB model provides only desired queries based on the type of algorithm used, as a result of reducing the number of layers in CryptDB to meet the level of sensitivity of the data owner and the application requirements. In addition, the size of the encrypted data is constant, as opposed to PIRATTE that increases reliance on AP. The PIRATTE concept in this scheme is thus used in the verification of access and not for the data encryption. Therefore, SDDB provides a trade-off between security and performance. Finally, table 3 shows the properties of our scheme in comparison with some existing works, that is further explained in detail in Section 6. The reported properties include C1-C7 the database type (DB) and practical status (PS) which means scheme has been implemented and evaluated.

6 RELATED WORK

CryptDB (Popa et al., 2011) as explained on background section is secure system worked on relational database for SQL queries over encrypted data. In term of its security, in the case of a proxy attack, only data for users who are logged in at that time will be vulnerable to leakage.

With regards to NoSql, CryptMDB(Xu et al., 2017)is a practical encryption system on MongoDB that employs an additive homomorphic asymmetric cryptosystem to encrypt data. It applies the proxy concept at the top of MongoDB, setting it to only perform an additive operation over encrypted data.

CryptGraphDB (Aburawi et al., 2018) is a system that executes queries to encrypted data stored in graph store (i.e. Neo4j database) through the transfer CryptDB concept. It provides traversal-aware encryption adjustment (using dynamically adjusting encryption layers) that is synchronised with the query execution - unlike static adjustment which precedes execution - thereby offering increased security(Aburawi et al.,).

The previous studies focuses on confidentiality and querying over encrypted data while (Ferretti et al., 2013) and (Sarfraz et al., 2015) also focuses on access control for relational database. In (Ferretti et al., 2013) the data owners encrypt data by SQL-aware encryption on CryptDB without the need to a proxy and store it as relational database over a cloud. Then they pass the encryption keys to Database administrator who has the authority to access all the data over the cloud and in turn responsible for distributing the keys to users based on legitimate access.
DBmask(Sarfraz et al., 2015) is the system that offers fine-grained access control built on attribute-based group key management(AB-GKM) scheme, in which users have attributes that should satisfy data policies to grant access and execute SQL queries over encrypted data based on user permissions. The system's architecture is inspired by CryptDB and uses the proxy that filters clause queries that do not execute over encrypted data, instead execute on in-memory from the proxy side. The DBmask system has been implemented through two schemas: (1) DBmask-SEC offers maximum security, (2) DBmask-PER offers the best performance. The limitation is that access policy groups belongs to each cell are revealed to DBMS.

AB-GKM adds an extra column corresponding for each column in a table to determine the group belongs to a cell and this requires fixed data structure format as relational database as well as in case of database attack, an adversary will reveal the cells belonging to the same group. Consequently, it is important to find an adequate mechanism for non-relational database and provides property to hide access control from the database side. However, in 2005, Sahai and Waters proposed attribute-based encryption (ABE)(Sahai and Waters, 2005), which is a type of public key encryption, that uses user identity for encrypting and decrypting data to access control of document data. ABE can further be classified into two types: key-policy-ABE(KP-ABE) and ciphertext-policy-ABE(CP-ABE). In 2006, Goyal developed KP-ABE (Goyal et al., 2006) and stated that the ciphertext is associated with a set of attributes with the secret key associated with access policy (AP). A user can decrypt data only if the corresponding attributes of ciphertext satisfies the AP of a user's key. The disadvantage of this type of ABE is that the Data Owner cannot determine which users can decrypt the data. Therefore, KP-ABE is not suitable for applications which share data. However, in 2007, Bethencourt developed CP-ABE (Bethencourt et al., 2007) and stated that the cipher-text is associated with access policy (AP) with the secret key associated with a set of attributes to overcome the disadvantage of KP-ABE and more suitable for applications. Both KP-ABE and CP-ABE lack user revocation mechanism. Though previous studies such as in (Pirretti et al., 2006; Boldyreva et al., 2008; Liang et al., 2013) have noted that revocation mechanism has been added to CP-ABE, it requires either key re-distribution or data re-encryption. In 2012, Jahid and Borisov proposed the PIRATTE scheme (Jahid and Borisov, 2012) to address these limitations as explained on background section.

7 CONCLUSIONS

This paper presents the main idea of Secure Document Database (SDDB) scheme satisfying three main security database requirements, which are confidentiality, flexible access control and querying over encrypted data for a document-based store. Future work will concentrate on the choice of encryption primitives appropriate to construct onions. Then, SDDB will be implemented on the MongoDB and trade-off between security and performance will be evaluated.

ACKNOWLEDGMENTS

Maryam Almarwani was supported by Royal Embassy of Saudi Arabia. Alexei Lisitsa was partially supported by EPSRC funded RAI Hub FAIR-SPACE (EP/R026092/1).

REFERENCES

Db-engines ranking. https://db-engines.com/en/ranking. Accessed: 2018-11-14.

Nosql, rdbms - explore - google trends. https://trends.google.com/trends/explore?date=all &q=NoSQL,RDBMS. Accessed: 2018-07-14.

Aburawi, N., Coenen, F., and Lisitsa, A. Traversal-aware encryption adjustment for graph databases.

Aburawi, N., Lisitsa, A., and Coenen, F. (2018). Querying encrypted graph databases. In *Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, January 22-24, 2018.*, pages 447–451.

Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *Security and Privacy, 2007. SP'07. IEEE Symposium on*, pages 321–334. IEEE.

Boldyreva, A., Goyal, V., and Kumar, V. (2008). Identity-based encryption with efficient revocation. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 417–426. ACM.

Ferretti, L., Colajanni, M., and Marchetti, M. (2013). Access control enforcement on query-aware encrypted cloud databases. In *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 219–219. IEEE.

Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based encryption for fine-grained access control of encrypted data. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 89–98. Acm.

Jahid, S. and Borisov, N. (2012). Piratte: Proxy-based immediate revocation of attribute-based encryption. *arXiv preprint arXiv:1208.4877*.

Liang, K., Fang, L., Susilo, W., and Wong, D. S. (2013). A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security. In *Intelligent Networking and Collaborative Systems (INCoS), 2013 5th International Conference on*, pages 552–559. IEEE.

Pirretti, M., Traynor, P., McDaniel, P., and Waters, B. (2006). Secure attribute-based systems.

Popa, R. A., Redfield, C., Zeldovich, N., and Balakrishnan, H. (2011). Cryptdb: protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, pages 85–100. ACM.

Sahai, A. and Waters, B. (2005). Fuzzy identity-based encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 457–473. Springer.

Sarfraz, M. I., Nabeel, M., Cao, J., and Bertino, E. (2015). Dbmask: fine-grained access control on encrypted relational databases. In *Proceedings of the 5th ACM*

Conference on Data and Application Security and Privacy, pages 1–11. ACM.

Xu, G., Ren, Y., Li, H., Liu, D., Dai, Y., and Yang, K. (2017). Cryptmdb: A practical encrypted mongodb over big data. In *Communications (ICC), 2017 IEEE International Conference on*, pages 1–6. IEEE.