# Fine-Grained Access Control for Querying Over Encrypted Document-Oriented Database

Maryam Almarwani(✉), Boris Konev, and Alexei Lisitsa

Department of Computer Science, University of Liverpool, Liverpool, UK
{M.almarwani,Konev,A.Lisitsa}@liverpool.ac.uk

**Abstract.** This paper presents two security models for document-based databases which fulfill three security requirements that are confidentiality, querying over encrypted data, and flexible access control. The first model which we refer to as dynamic is based on a combination of CryptDB [16] and PIRATTE [15] concepts. While CryptDB consists of a proxy between one user and a database for encrypting and decrypting data according to user queries, PIRATTE refers to a proxy wherein encrypted files are shared using a social network between the number of users and the data owner with the files being decrypted using the proxy key on the user side. The second model which we refer to as static is based on CryptDB concepts as well as CP-ABE [6]. CP-ABE is public key encryption which offers fine-grained access control regarding encrypted data and set of attributes that describe the user who is able to decrypt the data provided within the ciphertext. These two models enhance CryptDB security while also helping with data sharing with multi-users using CP-ABE or PIRATTE concept that helps in verifying authentication on the database or application level.

**Keywords:** Fine-grained access control · Querying over encrypted data · Document database · CryptDB · CP-ABE

## 1 Introduction

Databases tend to involve sensitive data including government and personal information. For such data, the security must be able to handle threats that are internal as well as external. Although external threats are reduced generally by applying access control, external as well as internal (such as by curious administrators) data leaks can be prevented by encryption. It should be noted, however, that using encryption has its own problems. For querying encrypted data, they either should be decrypted making data leaks possible or special techniques have to be implemented to query encrypted data which have limited querying power and efficiency. According to the authors of [9], there are three requirements for securing databases which are (i) confidentiality, (ii) enforcement access control, and (iii) querying over encrypted data. There have been numerous systems that have been developed which satisfy one or more of these requirements. Regarding relational databases, the confidentiality and querying over encrypted data requirements are addressed by the CryptDB system [16], whereas all three requirements are addressed by DBMask [18]. In the past few years, numerous NoSQL data models and DBMSs have become popular [1] because they can effectively store as well as process substantial amounts of unstructured and structured information. NoSQL databases can be classified into four types which are key-value store, document-based store, column-based store, and graph-based store. Concerning NoSQL databases security, the first and third requirements are dealt with for example, in [3] (Graph DB) and in [20] (document-based DB), whereas [4](document-based DB) addresses all three requirements and proposes SDDB design that adds more types of encryption methods to securely achieve more queries operations over encrypted data than in [20].

In the paper [4], the *dynamic SDDB* design, (called there just SDDB) utilizes the precise access control functionality provided by PIRATTE approach [15] with a particular benefit of preventing re-encryption data cost in the case of user revocation. This paper expands [4] as follows:

– We present the detailed description of the *dynamic model* from [4].
– We present novel *Static Secure DataBases*(S-SDDB) model. In short, in S-SDDB, PIRATTE is replaced by CP-ABE and in that case, the access control is achieved by using *Attribute-Based Encryption* in the layered encryption onions. This allows decreasing computations cost when applied to fixed data attributes which do not require users revocation feature.
– We compare dynamic and static SDDB models which provide flexible access control at *application* and *database* levels, respectively.
– We describe sets of MongoDB queries supported by two models.
– We report on experiments with an initial prototype of S-SDDB utilizing one layer of Attribute-Based Encryption.

The rest of the paper is organised as follows. Section 2 provides the background of CryptDB, PIRATTE, and CP-ABE systems. Sections 3 and 4 examine the Secure Document Database (SDDB) scheme and SDDB workflow, respectively. Sections 5 and 6 present comparisons between static and dynamic models, and security analysis, respectively. Section 7 defines a case study concerning SDDB scheme application. Section 8 discusses performance and security regarding the SDDB. Section 9 discusses a prototype implementation and Sect. 10 presents the related work. Section 11 provides the conclusion.

## 2 Background

This section provides the background regarding CryptDB, PIRATTE, and CP-ABE concepts. CryptDB [16] can be considered as the first practical database system supporting SQL queries over encrypted data. It is a proxy between the user and the database as it rewrites a query so that it can execute it over encrypted data on DBMS while not revealing plaintext and passing on the encrypted result it receives from DBMS to the user once decryption is done. Various types of encryption techniques are used based on the type of data and required operations. These include Deterministic (DET) and Random (RND) types of encryption. Onion layers are used to compose these techniques as shown in Fig. 1. CryptDB is unable to implement specific access control regarding columns and cells. It can only use a proxy-based reference monitor to implement row access because of encrypting every column data using one key. Moreover, CryptDB users are unable to share their data with user groups.

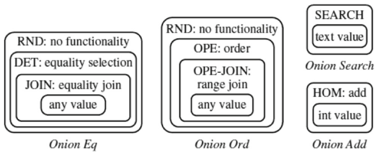

**Fig. 1.** Onion encryption layers in CryptDB [16].

For reducing CryptoDB limitations as well as adopting for the case of the document-based DB, implementing advanced cryptographic primitives including cipher-policy attributed based encryption (CP-ABE) [6] is that we propose. Attributes Based Encryption (ABE) is a public key cryptographic method in which encryption as well as decryption are based on the attributes. Restrictions on data access as per data owner imposed attributes are defined as access policy. ABE can be classified into Ciphertext-Policy AttributeBased Encryption (CP-ABE) and Key-Policy Attribute-Based Encryption (KP-ABE).

KP-ABE refers to the access policy which is labelled in the private key and the attributes are labelled in ciphertext whereas the reverse is true for CP-ABE. It provides a feature particularly for users who are able to decrypt the data specifically regarding the applications. Hence, it will be implemented in this paper to ensure specific access control for data. CP-ABE refers to public key encryption which offers fine-grained access control regarding encrypted data and can be considered to be similar to Role-Based Access control. Further, it identifies users who are authorized to decrypt data as per Access Policy including a set of attributes provided within the ciphertext. A set of attributes describes the user who is also issued a relevant private key. The user is able to decrypt the data only when private key attributes fulfil access policy regarding the ciphertext. Furthermore, there are four algorithms in CP-ABE as follows:

1. **Setup:** This algorithm includes only the implicit security parameter input while outputting the public parameters PK as well as a master key MK.
2. **Key Generation (MK, S):** This algorithm includes the master key MK as well as a set of attributes S describing the key as input while outputting a private key SK.

3. **Encrypt (PK, A, M):** This algorithm has the public parameters PK, an access structure A, and a message M concerning the attributes as input. The algorithm encrypts M while developing a ciphertext CT because of which only a user who has a set of attributes fulfilling the access structure can decrypt the message. It should be assumed that A is implicitly included in the ciphertext.
4. **Decrypt (PK, CT, SK):** This algorithm includes the public parameters PK, a ciphertext CT containing an access policy A, as well as a private key SK that is a private key regarding a set S of attributes as input. In case of the set S of attributes fulfilling the access structure A, the algorithm is able to decrypt the ciphertext as well as return a message M.

Jahid and Borisov proposed the PIRATTE scheme [12] wherein CP-ABE and user revocation mechanism are integrated through a proxy which takes care of attributes as well as user revocation for enabling dynamic users. While a proxy key and secret keys are issued by the data owner in PIRATTE, only the proxy key regarding user revocation is updated.

## 3 SDDB Overview

This section provides SDDB's overview concerning both dynamic and static variants, including system requirements, threat model, Entities, and SDDB Models.

As illustrated in Fig. 2, SDDB allows Document DB to perform the queries over encrypted data regarding particular operations depending on access policy restrictions that the data owner imposes. Using user secret key, proxy can decrypt secret keys concerning encrypted data as well as determine encryption layers that can be adjusted.

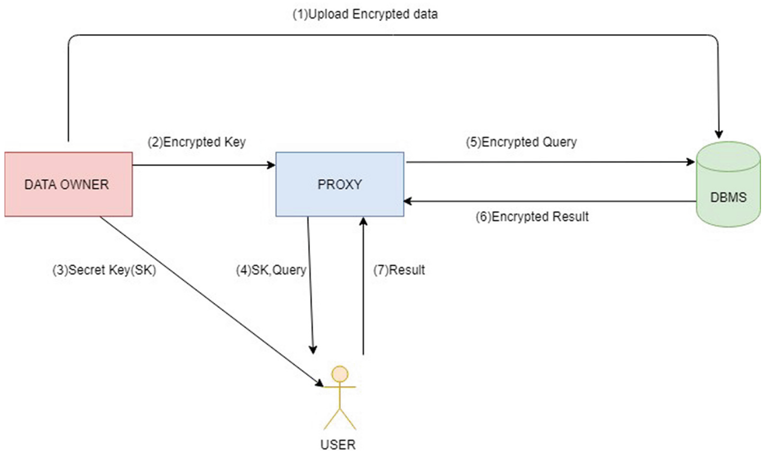

**Fig. 2.** SSDB design.

There are two models included in SDDB which are a static model a dynamic model [4]. Dynamic principle model is based on CryptDB concept including PIRATTE concept and fulfils fine-grained access control regarding application level that PIRATTE has on the proxy. On the other hand, static principle model is based on CryptDB concept including CP-ABE and fulfils fine-grained access control concerning database level as it encapsulates every onion into CryptDB by CP-ABE. Figure 2 presents four entries whose interactions in both models are as follows:

1. Data Owner (DO) refers to the authority that establishes access privileges concerning her/his data which this paper will call access policy (AP). DO, for example, can be a user sharing their data using applications. They may also use various keys for encrypting diverse aspects of the data based on the AP while uploading them to the DBMS server. This leads to the secret keys being created for every AP which are then distributed to the users. The authenticated user able to access the data is then verified using proxy in dynamic model or using DBMS in static model.
2. User Applications refer to users requesting data sharing regarding the DO. Here, the users' attributes are sent to the DO and their secret keys are received that helps in identifying themselves regarding the DBMS or proxy to gain access to parts of the data and to conduct queries.
3. Proxy refers to the intermediate server existing between DBMS, user application, and DO and helps verify the appropriate access regarding every user when dynamic model as well as rewriting queries should be executed concerning the encrypted data.
4. DBMS server concerns a server that offers database services including storage, retrieval, or verification of access control regarding static model. This also helps in storing encrypted data as well as conducting query while, if possible, not showcasing the plaintext data.

In both models of SDDB, processing query is conducting in six steps. The first step involves the user issuing the query as the data owner obtains the secret key. The second step involves the proxy rewriting the query of the user and replacing the anonymised fields, collection, and documents while encrypting the constants as per the necessary operation. In the third step, the proxy verifies the user query and determines whether it is able to execute regarding the current layer, failing which the proxy issues the update query for decrypting the current layer in case of the user being authorized for accessing such data. The fourth step involves the query being sent by the proxy to MongoDB which then sends the encrypted result to the proxy. In the fifth step, the results are decrypted by the proxy and sent to the user. In the sixth step, proxy re-encrypts the previous layer to ensure further security.

## 3.1 SDDB Requirements

SDDB design fulfils the requirements [4] given below such that both models fulfil c1–c4 while only the dynamic model fulfils c5–c7.

- **C1: Querying Over Encrypted Data:** This scheme can conduct operations as well as queries over encrypted data while not revealing the data.
- **C2: Flexible Access Control:** This grants users access to data parts as per data owner's policy which can be altered if required.
- **C3: Multi-user Sharing Support:** Depending on the policy of the data owner, numerous users can access data.
- **C4: Security and Performance Trade-off:** This scheme enables better security or performance to be developed.
- **C5: User Revocation Support:** This scheme can, as per requests by data owners, revoke the users and restrict them from accessing data.
- **C6: No Re-encryption Data:** There is no need for data re-encryption when user revocation has occured.
- **C7: No Re-distributed Key:** There is no need for keys to be re-distributed when user revocation takes place.

## 3.2 Threat Model

- **DO:** This is trusted and is offline following encryption and uploading of the encrypted data. It then distributes the keys to users except when a new user asks for permission or when the owner removes user access rights.
- **User Applications:** This is untrusted, and hence, the proxy has to verify them prior to allowing access and querying data. DO and Proxy and user applications do not share decryption keys.
- **Proxy:** This is semi-trusted and gains encrypted keys as well as is unable to decrypt data by itself.
- **DBMS Server:** This is semi-trusted and hence is unable to gain the keys for decrypting inner layer.

## 3.3 Document-Aware Encryption

This section examines encryption techniques that layers, security level, and onions use along with their work concept.

1. **Access Control (AC):** Here, encrypted value includes access policy which identifies which users have been authorized for accessing the data in which CP-ABE algorithm is implemented [6]. This does not provide any computation operations regarding the encrypted value. Further, as it does not reveal data information, it ensures optimum security as per Decision Bilinear Diffie-Hellam (DBDH).
2. **Random (RND):** Here, Blowfish-CBC or AES-CBC encrypt the same values regarding various values for numeric or string data, respectively, using a random initialization vector (IV). It appears to be similar to Access control layer which does not provide any computation operations or reveals any information. Further, it grants optimum security as per indistinguishability concerning adaptive chosen-plaintext (IND-CPA) [5].

3. **Deterministic (DET):** Here, AES-CBC and Blowfish-CMC [11] encrypt two equal values as the same value by concerning string and number data, respectively, with zero-IV. It results in leaking equivalent values, and hence, one is able to conduct equality operations on the encrypted data including equality predicate, Count, and Group. Moreover, it grants reduced security compared to AC while being effectively secure.
4. **Order-preserving Encryption (OPE):** Here, data encryption is done an OPE algorithm, such as Boldyreva' algorithm [7], and hence, it leaks data order as well as enables comparisons predicates including Order by, Min, and Max. It also offers less security compared to DET.
5. **Homomorphic Encryption (HOM):** Here, encryption of two same values is done to two different values and is able to conduct arithmetic operations concerning numeric data using Fully Homomorphic Encryption while being expensive and ineffective. On the other hand, partially Homomorphic encryption can support particular operations, such as summation or multiplication, that is less expensive and more effective. For instance, Pailier's algorithm [14] is implemented for supporting summation operations using Further, it provides similar security regarding AC layer.
6. **Word Search (WS):** Here, data encryption is done using Song's protocol [19] which supports LIKE Operator. Song's protocol is used to encrypt keyword in the user's query by proxy, while MongoDB searches aspects of encrypted data which includes the same encrypted keyword. Moreover, it does not leak information to a server, and hence ensures close security concerning AC.

### 3.4 Adjustable Query-Based Encryption

Every technique can support particular operations that include RND and AC regarding queries with no computations, OPE concerning comparison operations, DET concerning equality operations, WS concerning LIKE operator on string data, and HOM concerning summation (and other arithmetical operations) on numeric data. Every technique can offer different level of security. Hence, such techniques are arranged in the form of layers and onions as per types of data and operations. Further, the outmost layers provide optimum security while inner layers reduced security. For supporting data access restrictions as well as encrypting data using diverse keys using every technique as per who has access to data, AC technique is used to encapsulate the onions in the static model, as shown in Fig. 3(A), or to maintain CryptDB onions in the dynamic model, as shown in Fig. 3(B), while implementing PIRATTE on an application level.

Thus, when adjustable query-based encryption is used, the layers of encryption protecting the data has to be adjusted to allow to perform the required operations. For sharing data as well as restricting data access, the data owner provides a unique encryption key to every layer and onion regarding every access policy. The proxy verifies the access rights as well as clause within the query for identifying the suitable layer, onion, as well as field for encryption. Then, it issues the updated query which includes user-defined functions (UDFs) that should be

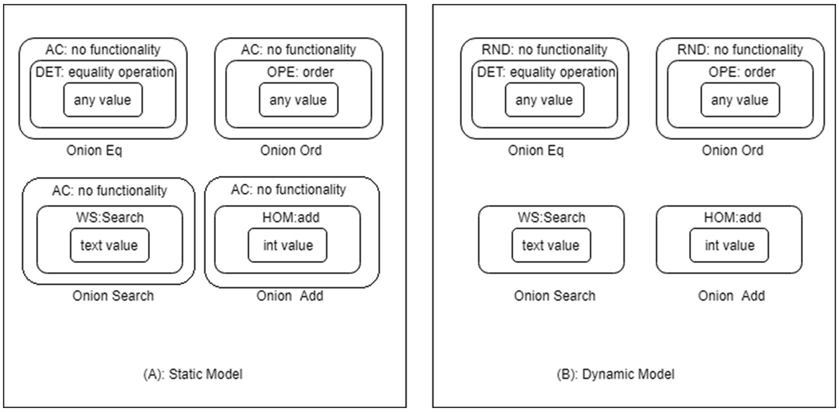

**Fig. 3.** Layers of onions in models.

executed on document DB. The user, for example, issues the query that includes equality operation, including checking ID = 23. The user also holds access rights with no regard to the query type. Regarding adjustable query-based encryption, the process in both models is as follows. Execution of this query is not possible if the outermost layer RND or AC. To decrypt the ID field's outermost layer concerning Onion Eq, the update query that must be on DET layer is issued. Proxy then rewrites user query which is executed on document DB. Proxy issues the update query to return to the outermost layer such as RND or AC to maintain further security.

### 3.5 Data Format and Query Language

It is possible to implement SDDB scheme for different document-based DBs, MongoDB environment is regarded as the major target. The data storage format [4] is defined along with the query language concerning MongoDB but we will define it here again with more details to understand the remainder of the paper. A binary coded format known as BSON is used for the representation of MongoDB data. BSON can be regarded as a form of JSON that has further data types, including *binary* and *date*, as well as embedding feature. It offers efficient encoding as well as decoding using various languages, further information for which can be found at http://bsonspec.org. Further, a particular data query language is not used by MongoDB although it adhered to simple query syntax which is appropriate for data representation by JSON which will be referred to as MongoDB Query. MongoDB Query syntax consists of calling database db followed by Collection-name and then operators such as find(), insert() and may be followed by any data condition in the case of request a representative in JSON and maybe also followed by functions such as sort() or count(). An example for this is given below:

```
db.collection–name.find('name':1)  .sort();
```

Table 1 presents MongoDB queries' common forms as per MongoDB documentation while ensuring the execution on this model's layers and the layers which are possible to adjust to ensure execution.

As shown in Table 1, there are three categories of MongoDB queries that possible in both models.

1. Executing query on the encrypted data while not including adjustable layers. Such as 1, 2 that be executed on the AC layer directly Following is how this group is executed:
   (a) The query is sent by the user to the proxy which is written on the MongoDB query.
   (b) The proxy encrypts the collection name because it is known in the uploading phase and then transfers it to Query router server.
   (c) If there is a find, the encrypted results are sent by the Query router server to the proxy, which then decrypts numerous times as per the number of onions or layers, such as first decrypting RND or AC layer and then the DET layer and providing the user with the result.

   **Insertion Example:**

   (a) The user issues I query

   ```
   db.mycol.insert({'title ':'MongoDB'})...(I)
   ```

   (b) The proxy encrypts 'MongoDB' on onion equal as well as onion order and sends II query to Query router server.

   ```
   db.mycol.insert({title-eq:'ghftygdyubnbc'},
         {title-ord:'tyuuiiowosak'},
         tilte-search:'xcbvbmnswe')...(II)
   ```

   (c) The query is executed by the query router by adding a new document which includes three fields as there is a string type title field.

   **Finding Example without Any Operations:**

   (a) The user issues I query

   ```
   db.mycol.find({})...(I)
   ```

   (b) The query is sent by proxy to Query router server.
   (c) Query router server sends fields-eq and Id such as id:7df78ad8902c and title-eq:'ghftygdyubnbc'.
   (d) Proxy decrypts the result by first peeling off RND or AC layer followed by DET layer for gaining 'MongoDB' value and forwarding it to user.

2. Query is executed over encrypted data using adjustable layers, including 3, 4, 7, 8, and 9 which are executed on inner layers. Following is how this group is executed:
   (a) The query is sent by the user to the proxy which is written on a MongoDB query.

(b) Proxy peels off RND or AC layer through issuing update query regarding operation field only and then sending to the Query router server for updating this field to make a comparison with the user's query value.
(c) The user query is then rewritten by the proxy and is sent to Query router server.
(d) Encrypted results are sent by the query router server to the proxy which then decrypts numerous times as per the number of onions or layers, and the results are sent to the user.

**Finding Example with Equal Operation:**

(a) The user issues I query

```
db.mycol.find({'tittle ':'MongoDB'})...(I)
```

(b) This query cannot be executed on the current layer (AC), and thus, proxy issues update query (II) for peeling off AC layer.

```
db.mycol.updateMany({},{\$set:{'tittle-eq'
:decrypt-AC(K, tittle-eq,Ksw))}} in static model...(II)
```

or

```
db.mycol.updateMany({},{\$set:{'tittle-eq'
:decrypt-RND(tittle-eq,IV))}} in dynamic model...(II)
```

This query includes the comparing field as well as key which was used on encrypting value for decrypting this field on DET layer.
(c) The proxy rewrites the existing user query for executing on encrypted data by encrypting ['tittle':'MongoDB'] as per the encrypted data on DET layer as III query.

```
db.mycol.find({'tittle-eq':'werwdgdjhhkjiu'})...(III)
```

(d) Query router server compare the value ('werwdgdjhhkjiu') on tilttle-eq fields and send all field on these documents contain this value to proxy.
(e) The result is decrypted by proxy by first peeling off the RND layer using field-IV or the AC layer using after which the DET layer is peeled off to gain plaintext value which is forwarded to user.

3. Querying is not supported over encrypted data. As existing encryption techniques are unable to support such queries, two solutions are recommended. Following is how this group is executed:
(a) Executing a part of the query that provides support to the existing encryption techniques and eliminates parts which cannot be executed, after which when the results are received by proxy and decrypted, user's queries are executed as per the result to gain the necessary results.

**Table 1.** Common MongoDB queries.

| No | Query type | Executing layer | Adjustable layer |
|---|---|---|---|
| 1 | - db.collection-name.insert(document) | AC or RND | NO |
| 2 | - db.collection-name.find()<br>- db.collection-name.find().pretty() | AC or RND | NO |
| 3 | - db.collection-name.find({"key":"value"}).pretty()<br>- db.collection-name.find({"key":{$ne:value}}).pretty() | DET | AC or RND |
| 4 | - db.collection-name.find({"key":{$lt:value}}).pretty()<br>- db.collection-name.find({"key":{$lte:value}}).pretty()<br>- db.collection-name.find({"key":{$gt:value}}).pretty()<br>- db.collection-name.find({"key":{$gte:value}}).pretty() | OPE | AC or RND |
| 5 | - db.collection-name.find({$and:[{key1:value1}, {key2:value2}]}).pretty() | Not support | - |
| 6 | - db.collection-name.find({$or:[{key1:value1}, {key2:value2}]}).pretty() | Not support | - |
| 7 | - db.collection-name.update(SELECTION-CRITERIA, UPDATED-DATA) | DET (equal) and OPE (other condition) | AC or RND |
| 8 | - db.collection-name.remove(DELLETION-CRITTERIA) | DET and WS | AC or RND |
| 9 | - db.collection-name.find().sort({KEY:1}) | OPE | AC or RND |
| 10 | - db.collection-name.ensureIndex({KEY:1}) | Not support | - |
| 11 | - db.collection-name.aggregate(AGGREGATE-OPERATION) | Not support | - |

## 4 SDDB Workflow

This section examined how access control can be verified and a query can be executed on SDDB regarding both models. PIRATTE concepts are used by the dynamic [4], model concerning Proxy and DO for verifying access control by allowing decrypting keys which are stored on the proxy. Hence, in case of the user's inability to obtain keys, user query is rejected. CP-ABE as encapsulation is used by the static model on every onion on DBMS. Thus, if the user does not hold right to access, the proxy rejects query. The two models are similar as given below apart from minor differences.

– **DO-DocumentDB Connection:**
  1. DO develops Policy Access (PA) including ((Doctor and L hospital) or Administer).
  2. DO conducts setup function for every encryption technique for gaining symmetric key concerning every policy. This key differs from other access policies.
  3. Regarding the Static Model, DO executes CP-ABE-Setup for gaining public key(pk) as well as master key(mk) for policy access. Regarding Dynamic Model, DO executes PIRATTE-Setup for gaining public key(pk) as well as master key(mk) along with Proxy Key-setup for developing proxy key to ensure policy access.
  4. DO conducts Enc-Layer for encrypting value for every document field through key from step 2.
  5. For Static Model, DO executes Enc-CP-ABE algorithms for encrypting value from step 3 through pk and PA.
  6. DO uploads encrypted documents in DBMS.
– **DO-Proxy Connection:**
  1. DO sends the keys(pk,mk, symmetric keys for encryption techniques, Proxy Key (Dynamic Model)) regarding every policy to proxy which encrypts them using Enc-CP-ABE (Static Model) or Enc-PIRATTE (Dynamic Model) as in case of a proxy attack, the adversary will fail to gain access to data on DBMs.
– **User-DO Connection:**
  1. User sends user attributes to DO.
  2. DO verifies which user attributes belongs to which policy access AP.
  3. DO executes CP-ABE-KeyGen (Static Model) or PIRATTE-KeyGen (Dynamic Model) for gaining secret key (skw) using master key.
  4. Do sends secret key to user.
– **User-Proxy Connection:**
  1. User sends attributes (skw) and Query (Q).
  2. Proxy verifies which attributes belong to which policy access and executes Dec-CP-ABE(Static Model) or Dec-PIRATTE(Dynamic Model) for decrypting keys through skw for gaining secret keys for policy access.
– **Proxy-DocumentDB Connection:**
  1. Proxy verifies whether operation on query is able to be executed on CP-ABE or RND layer. If it can, step 2 is done, and if it cannot, step 3.

2. Proxy alters the outermost layer through issuing update query by skw concerning the intended Onion.
3. Proxy rewrites Q and then replaces the constant with encryption using existing layer for sending query to DocumentDB.
4. DocumentDB forwards encrypted result to proxy.
5. Proxy decrypts the result by conducting Enc-ourmost-layer and then decrypts it again through Enc-inner-layer and forwards the result to user.

## 5 Static Model VS Dynamic Model

This section focuses on the crucial differences between dynamic and static models. Concerning the static model, every onion's top layer includes Access control technique using CP-ABE, as illustrated in Fig. 3A. For allowing data sharing, diverse keys are provided to inner layers as per the top layer, and only the user who holds the access right can decrypt the data. Hence, if the user does not have access, the user query is rejected. Thus, this model is unable to allow user revocation, in which case re-encryption of data or keys distribution is needed. Regarding the Dynamic model [4], layers and onions regarding CryptDB, as shown in Fig. 3B, are used, and PIRATTE is used for allowing data sharing as well as access enforcement for enabling data encryption using various keys. Hence, data can only be accessed by the user who is from that group. Access control fulfils the application level between the user and proxy with no need for Database sharing. In case the user is not given access, the query is refused by proxy. This model allows user revocation with no need for re-encryption data or keys distribution. The two models are compared in Table 2.

**Table 2.** Static model VS dynamic model.

| | Dynamic model | Static model |
|---|---|---|
| AC technique | PIRATTE | CP-ABE |
| AC level | Application | Database |
| Onion and layers | As CryptDB | AC technique as outermost layer for onions |
| Unauthorized user | Rejecting query | Rejecting query |
| User revocation | YES | NO |

## 6 Security Analysis

Regarding the proposal design, as shown in Fig. 2, as well as the encryption techniques types, the design offers protection against two types of threats:

1. **DBMS Threats:** DBMS threats are intended here ADB curious and external threats against full access and data leaks:

(a) Full access is hindered as encryption keys are not shared with ADB.
(b) On the encrypted data, query is executed for offering confidentiality so that data leaking can be prevented, whereas the leakage level is based on the encryption techniques that are used, as depicted in Table 3. OPE, for example, is the weakest security as it leaks order as well as duplicates and can be part of the plaintext along with DET providing more security which only leaks duplicates. Then are AC, RND, HOM, and WS which have no leakage.

**Table 3.** Leakage information level for encryption techniques.

| Encryption technique | Leakage |
|---|---|
| AC or RND | None |
| DET | Duplicates |
| OPE | Order, duplicates + partial plain-text |
| HOM | None |
| WS | None |

2. **Arbitrary Threats:** Arbitrary Threats are intended here any attacks on users, proxy and DBMS. If there is an attack on the proxy, the keys will be accessible to the attacker for detecting the stored data while also gaining full access to database. This can be prevented using CP-ABE so that full access to data can be prevented because of various keys being used for data encryption as well as keys encrypted on proxy as per access policy. This, however, does not prevent data leakage of a group which is part of the logged-in user if there is an attack. It also helps in preventing collision resistance threat if users or entities collide.

## 7 Case Study

This section assumed that the DO includes a data set that has a collection which includes two documents, both of which have two fields, ID and Name, which are integer and string data type, respectively. The execution of Q1 is done using MongoDB querying language to execute query through SDDB model and not using SDDB [4].

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

### 7.1 Without SDDB

This scenario is implemented when the system fails to offer data encryption or access verification. The data is uploaded by DO in DBMS as Plaintext, while it is assumed that access privilege is provided to users by providing them with password (PW). Hence, the Q1 and PW is used by the user through user application which sends it to the DBMS server for execution, the results of which are sent to the user and Name = Alice is matched to ID = 23 [4].

## 7.2 With SDDB

This scheme can be classified into cases as per computation classes needed by the application's queries. In this case study, for example, because the application needs queries (insert, delete, update, select) that have equality operation, the encryption of data is done using onion equality (DET), as depicted in Fig. 4, in the dynamic model or between DET and AC, as depicted in Fig. 5, in the static model.

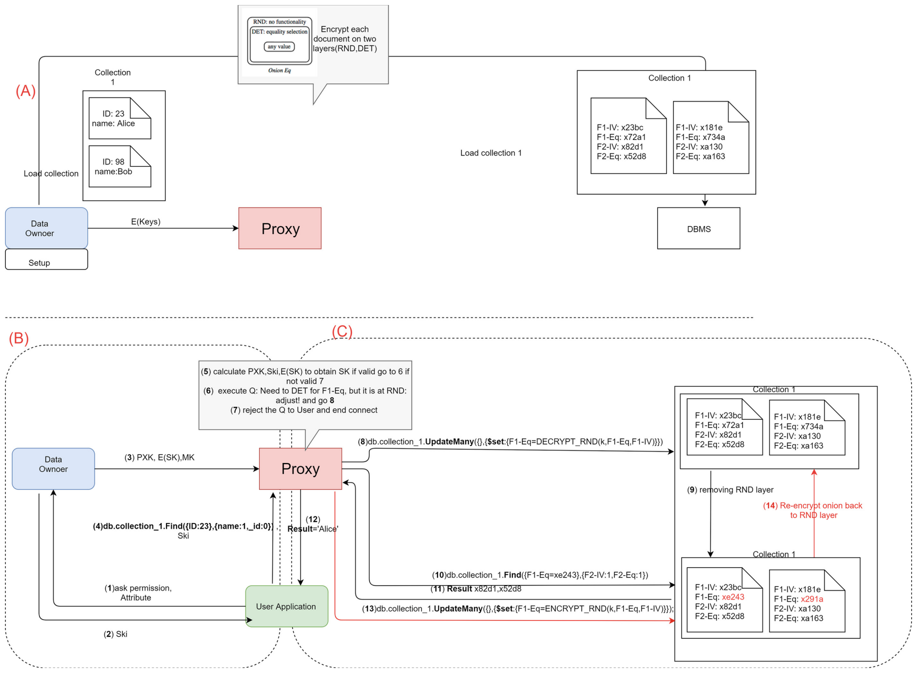
RND: no functionality
DET: equality selection
any value
Onion Eq
Encrypt each document on two layers(RND,DET)
(A)
Collection 1
ID: 23 name: Alice
ID: 98 name:Bob
Load collection
Load collection 1
Collection 1
F1-IV: x23bc F1-Eq: x72a1 F2-IV: x82d1 F2-Eq: x52d8
F1-IV: x181e F1-Eq: x734a F2-IV: xa130 F2-Eq: xa163
Data Ownoer
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 Ownoer
(3) PXK, E(SK),MK
Proxy
(8)db.collection_1.UpdateMany({},{$set:{F1-Eq=DECRYPT_RND(k,F1-Eq,F1-IV)}})
(4)db.collection_1.Find({ID:23},{name:1,_id:0}) Ski
(12) Result='Alice'
(1)ask permission, Attribute
(2) Ski
User Application
(9) removing RND layer
(14) Re-encrypt onion back to RND layer
(10)db.collection_1.Find({F1-Eq=xe243},{F2-IV:1,F2-Eq:1})
(11) Result x82d1,x52d8
(13)db.collection_1.UpdateMany({},{$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

**Fig. 4.** Dynamic model [4].

There are two major processing stages:

1. Encrypt and upload data
   (a) DO chooses AP for these two documents: AP: (Doctor AND Surgery Department).
   (b) DO runs Setup functions to obtain PK, MK, LK-RND (dynamic model), LK-AC (static model), LK-DET, PXYK (dynamic model).

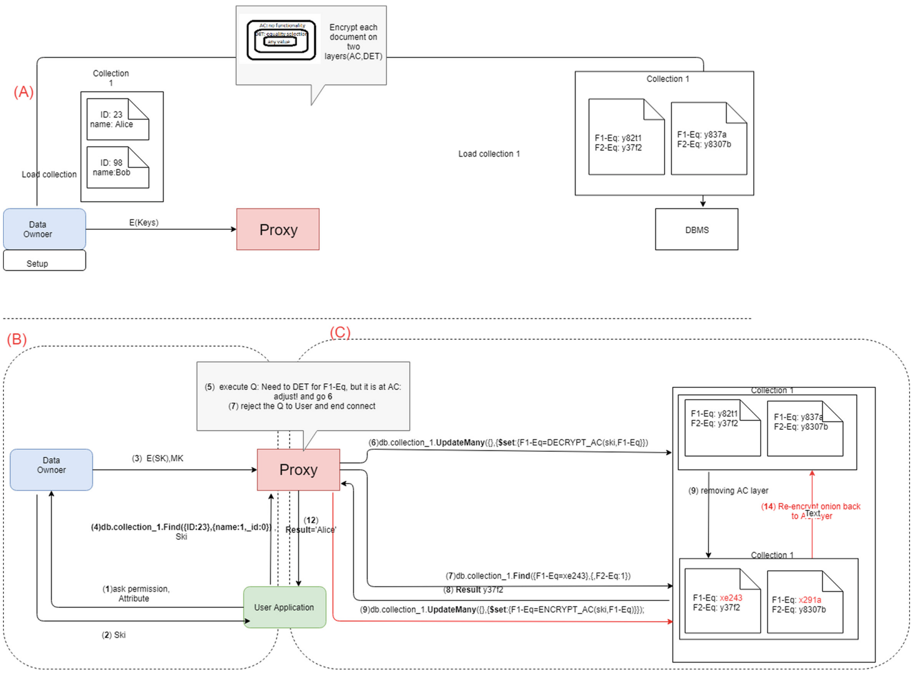
(A)
Collection 1
ID: 23
name: Alice
ID: 98
name:Bob
Load collection
Data Ownoer
Setup
E(Keys)
Proxy
AC: no functionality
DET: equality selection
any value
Encrypt each document on two layers(AC,DET)
Load collection 1
Collection 1
F1-Eq: y82t1
F2-Eq: y37f2
F1-Eq: y837a
F2-Eq: y8307b
DBMS
(B)
(C)
(5) execute Q: Need to DET for F1-Eq, but it is at AC: adjust! and go 6
(7) reject the Q to User and end connect
Data Ownoer
(3) E(SK),MK
Proxy
(6)db.collection_1.UpdateMany({},{$set:{F1-Eq=DECRYPT_AC(ski,F1-Eq}})
(4)db.collection_1.Find({ID:23},{name:1,_id:0}) , Ski
(12) Result='Alice'
(1)ask permission, Attribute
User Application
(2) Ski
(7)db.collection_1.Find({F1-Eq=xe243},{,F2-Eq:1})
(8) Result y37f2
(9)db.collection_1.UpdateMany({},{$set:{F1-Eq=ENCRYPT_AC(ski,F1-Eq)}});
Collection 1
F1-Eq: y82t1
F2-Eq: y37f2
F1-Eq: y837a
F2-Eq: y8307b
(9) removing AC layer
(14) Re-encrypt onion back to AC layer
Text
Collection 1
F1-Eq: xe243
F2-Eq: y37f2
F1-Eq: x291a
F2-Eq: y8307b

**Fig. 5.** Static model.

   (c) In dynamic, DO encrypts data by LK-RND and LK-DET. While, in static, DO encrypts data by PK, LK-DET.
   (d) DO uploads data on DBMS.
   (e) DO sends encrypted keys by PK to proxy.
2. Verify access control and execute the query
   (a) User sends attributes(Doctor and surgery department) to DO.
   (b) Do sends secret key(ski)
   (c) User sends ski and attributes and query Q1 to proxy.
   (d) in case of dynamic model, Proxy check attributes and decrypt keys if ski is correct, go to next steps, if it is incorrect the query rejects. In case of static model, proxy does to next setps.
   (e) Proxy check operation of query in this example is equality operation therefore, outermost layer adjust by LK-RND for dynamic or ski for static model by update query Q2.

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

```
 Q2−static:(db.collection −1.updateMany(
${},{$set=F1−Eq =DECRYPT_AC(
ski ,F1−Eq}))
```

(f) Proxy rewrites query Q1 to Q3 and sends to DBMS:

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

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

(g) DBMS sends result to proxy which is in dynamic (F2-IV=x82d1, F2,x52d8) or in static (y37f2).
(h) proxy decrypts the result twice in RND and DET (dynamic) or in AC and DET (static) and sends to user.
(i) Proxy come back to outermost layer to more security by issue opposite query in step (e).

```
 Q4−dyanimc:(db.collection −1.updateMany(
${},{$set=F1−Eq =Encrypt_RND(
k,F1−Eq,F1−IV}))

 Q4−static:(db.collection −1.updateMany(
${},{$set=F1−Eq =Encrypt_AC(
ski ,F1−Eq}))
```

## 8 Discussion

This section will examine the security as well as performance of the two models in this case study along with the studies that it inspired [4]. This scheme will also be compared with existing works.

Regarding security, in case of the without-SDDB model, in case of the DBMS server's exposure to curiosity or compromise, it will show plaintext data as it is not encrypted. Further, the adversary may also gain the password as well as impersonate the data owner so that they can manipulate the data. Hence, a secure channel is necessary for exchanging it [4].

Concerning the with-SDDB, the information is revealed by the DBMS server only if it is identified as per the encryption algorithm in the existing layer used, including equal in DET. Thus, the decryption layer is sent back to high-security layers except CryptDB. Concerning SDDB, encryption keys also do not need to be shared as various keys are used for user authentication as well as data encryption, as per access privileges except CryptDB. For the user, PIRATTE executes data decryption and cannot be trusted. Hence, this permission is provided to the proxy in the dynamic model, while for the proxy or DBMS, CP-ABE executes data decryption in the static model. If the adversary can gain the keys, they will be unable to impersonate the data owner by connecting the keys with the

**Table 4.** Comparison of our scheme with some existing work [4].

| | [16] | [20] | [3] | [18] | [9] | [15] | Dynamic model [4] | Static model |
|---|---|---|---|---|---|---|---|---|
| **C1** | ✓ | ✓ | ✓ | ✓ | ✓ | X | ✓ | ✓ |
| **C2** | ☐ | ☐ | X | ✓ | ✓ | ✓ | ✓ | ✓ |
| **C3** | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **C4** | ✓ | X | ✓ | X | X | ☐ | ✓ | ✓ |
| **C5** | ☐ | ☐ | ☐ | ✓ | ☐ | ✓ | ✓ | X |
| **C6** | ☐ | ☐ | ☐ | Only users group belong | ☐ | ✓ | ✓ | X |
| **C7** | ☐ | ☐ | ☐ | Only users group belong | ☐ | ✓ | ✓ | X |
| **DB** | Relational DB | Document DB | Graph DB | Relational DB | Relational DB | ☐ | Document DB | Document DB |
| **PS** | ✓ | ☐ | ✓ | ✓ | X | ☐ | In the future | In the future |

Notes: Satisfies (✓), Does not satisfy (X), Out of scope (☐).

user attributes or with the Proxy Key. In addition, if the proxy is compromised or is exposed, data leakage will not occur as it is unable to decrypt keys in the dynamic model or gain accurate results in the static model.

Regarding performance, the without-SDDB model is able to execute any query type or computations at a high speed. The with-SDDB model, on the other hand, offers necessary queries as per the algorithm type because of decreased number of layers in CryptDB for attaining the sensitivity level of the data owner as well as the application requirements. Moreover, concerning the dynamic model, the encrypted data size remains constant compared to PIRATTE which increases dependence on AP. In this scheme, the PIRATTE concept is used to verify access and not for data encryption. In the static model, there will be a decrease in the communication cost between the DO and proxy as it will not need a connection for updating the keys which occurs in the dynamic model. Thus, SDDB grants a trade-off between performance and security.

Table 4 [4] depicts the scheme's properties compared to existing works which Section examines in detail. In the reported properties, C1-C7 the database type (DB) as well as practical status (PS) are included which depicts scheme being implemented and evaluated.

## 9 Implementation

The prototype of the simplified static variant of SDDB has been implemented. It includes a simplified model for data encryption using one layer (CP-ABE) so that flexible access control, querying over encrypted data as well as confidentiality could be ensured and tested in the Document Database. Further, this implementation is written in Java as a monolithic code (no separation between user and data owner components, as yet) and executes on Windows 10. It allows experimenting with the proposed design and assumed workflows. In particular a scenario is implemented in which document fields(name, salary and Credit Card number) and the user name as well as password are first checked, following which user attributes are verified so that they satisfy policy access. Once access is granted, the following list of queries is executed:

- Q1- Insert 100 documents (No computation)
- Q2- Find 100 documents (No computation)
- Q3- Delete 100 documents (No computation)
- Q4- Enter one document (Equality computation)
- Q5-Insert one document (Equality computation)
- Q6- Delete one document (Equality computation)
- Q7- Update one document (Equality computation)

In the query Q4–Q7, the user is asked to enter the value for a particular field to compare encrypted value for it by CP-ABE by data stored then find or delete or update documents in the case of equality. The exeperiments were conducted using Local MongoDB server and a client, implementing user and data owner functionality. Each case was executed 15 times. Then, average execution time is taken as shown in Table 4. As shown in Fig. 6, a significant time increase is observed in Q1, Q2, and Q5, with minimal increase in the remainder of the queries. This performance sacrifice, however, provides enhanced security which will be explored and further verified in the future work (Table 5).

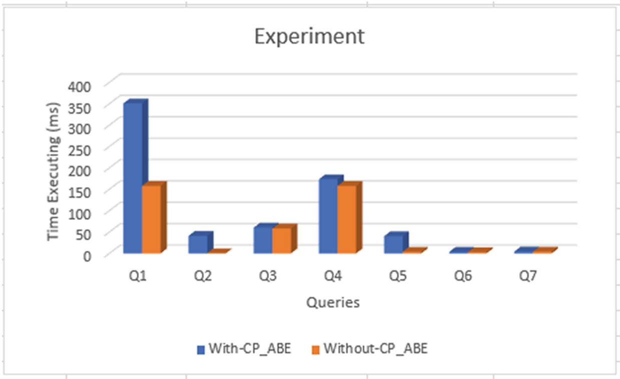

**Fig. 6.** Result analysis.

**Table 5.** average execution time of seven queries.

| | Q1 | Q2 | Q3 | Q4 | Q5 | Q6 | Q7 |
|---|---|---|---|---|---|---|---|
| With-CP-ABE | 350.8 | 40.9333 | 60.2667 | 173.4 | 40.1333 | 3.6667 | 4.5333 |
| Without-CP-ABE | 157.8667 | 0.4667 | 58.4 | 157.8667 | 3.9333 | 3.1333 | 4.3333 |

## 10 Related Work

CryptDB [16] refers to a secure system that is executed as per relational database concerning SQL queries over encrypted data. Regarding its security, for a proxy attack, only that data is at risk of being leaked which belongs to users logged in then.

Concerning NoSql, CryptMDB [20] refers to a practical encryption system regarding MongoDB using an additive homomorphic asymmetric cryptosystem for data encryption. It implements the proxy concept at MongoDB's top so that it only perform an additive operation concerning encrypted data. Crypt-GraphDB [3] refers to a system which conducts queries over encrypted data that are stored within a graph store, such as Neo4j database, using CryptDB-like technique. Further, it uses dynamically adjusting encryption layers to offer traversal-aware encryption adjustment which is coordinated with the query execution, thus providing enhanced security [2].

Previous studies have examined confidentiality as well as querying concerning encrypted data whereas [9] and [18] emphasise access control regarding relational database. The study by [9] shows that the data owners use SQL-aware encryption regarding CryptDB for data encryption while not requiring a proxy which is then stored as relational database over a cloud. The encryption keys are provided to Database administrator authorised for accessing all the data over the cloud while also passing the keys to users as per legitimate access.

Further, DBmask [18] refers to a system which provides fine-grained access control based on Attribute-Based Group Key Management (AB-GKM) scheme wherein users' attributes fulfil data policies for providing access as well as executing SQL queries over encrypted data according to user permissions. The architecture of the system is based on CryptDB while implementing the proxy which refines clause queries which are unable to execute over encrypted data rather than executing on in-memory concerning the proxy. Two schemas have been used for conducting the DBmask system which are DBmask-SEC providing maximum security and DBmask-PER providing optimum performance. Here, the restriction concerns the access policy groups belonging to every cell being revealed to DBMS. An additional column is added by AB-GKM corresponding to every column in the table for identifying the group which belongs to a cell. For this, a fixed data structure format is needed because relational database and an adversary for database attack helps in determining the cells that are part of the same group. Hence, determining a suitable mechanism is important regarding non-relational database as it offers property for not revealing access control concerning the database. In 2005, however, an attribute-based encryption (ABE) [17] was suggested by Sahai and Waters referring to a form of public key encryption in which user identity is used for data encryption and decryption concerning access control of document data. Moreover, ABE can be classified into ciphertext-policy-ABE (CP-ABE) and key-policy-ABE (KP-ABE). Goyal created KP-ABE [10] in 2006 and noted that there is an association between the ciphertext and a set of attributes that have secret key related to AP. It is possible for a user to decrypt data in case of the ciphertext's corresponding attributes fulfilling the user key AP. Here, the limitation concerning such type of ABE concerns the Data Owner being unable to identify which users are able to decrypt the data. Hence, KP-ABE is inapplicable regarding applications in which data is shared. In 2007, however, Bethencourt created CP-ABE [6] and noted that there is a relationship between cipher-text and AP and the secret key concerns a set of attributes for surpassing the limitations of KP-ABE and further suitable for applications. KP-ABE as well as CP-ABE do not include user revocation mechanism. Although existing studies including [8,13,15] note that revocation mechanism is added to CP-ABE, data re-encryption or key re-distribution is necessary. Moreover, in 2012, the PIRATTE scheme [12] was suggested by Jahid and Borisov concerning the limitations previously stated in the background section.

## 11 Conclusion

This paper examines the primary idea regarding the Secure Document Database (SDDB) scheme which fulfils three major security database requirements that are flexible access control, confidentiality, and querying over encrypted data regarding a document-based store. The dynamic and static models are used for presenting SDDB. The dynamic model fits dynamic applications which can need changing users' attributes who are able to access data while not re-encrypting data as well as distributing keys. On the other hand, the static model is suitable in static applications which do not alter users' attributes necessary for accessing data. We reported also on the experiments with the simplified prototype implementation of the static model. In future work, we are going to implement both models using MongoDB as a document store, and evaluate trade-offs between performance and security.

## References

1. No SQL, RDBMS - explore - Google trends. https://trends.google.com/trends/explore?date=all&q=NoSQL,RDBMS. Accessed 22 June 2019
2. Aburawi, N., Coenen, F., Lisitsa, A.: Traversal-aware encryption adjustment for graph databases (2018)
3. Aburawi, N., Lisitsa, A., Coenen, F.: Querying encrypted graph databases. In: Proceedings of the 4th International Conference on Information Systems Security and Privacy, ICISSP 2018, Funchal, Madeira - Portugal, 22–24 January 2018, pp. 447–451 (2018). https://doi.org/10.5220/0006660004470451

4. Almarwani., M., Konev., B., Lisitsa., A.: Flexible access control and confidentiality over encrypted data for document-based database. In: Proceedings of the 5th International Conference on Information Systems Security and Privacy, vol. 1: ICISSP 2019, pp. 606–614. INSTICC, SciTePress (2019). https://doi.org/10.5220/0007582506060614
5. Bellare, M., Rogaway, P.: Symmetric encryption. In: Introduction to Modern Cryptography (2004)
6. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: IEEE Symposium on Security and Privacy, SP 2007, pp. 321–334. IEEE (2007)
7. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33
8. Boldyreva, A., Goyal, V., Kumar, V.: Identity-based encryption with efficient revocation. In: Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 417–426. ACM (2008)
9. Ferretti, L., Colajanni, M., Marchetti, M.: Access control enforcement on query-aware encrypted cloud databases. In: 2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom), pp. 219–219. IEEE (2013)
10. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
11. Halevi, S., Rogaway, P.: A tweakable enciphering mode. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 482–499. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-45146-4_28
12. Jahid, S., Borisov, N.: Piratte: proxy-based immediate revocation of attribute-based encryption. arXiv preprint arXiv:1208.4877 (2012)
13. Liang, K., Fang, L., Susilo, W., Wong, D.S.: A ciphertext-policy attribute-based proxy re-encryption with chosen-ciphertext security. In: 2013 5th International Conference on Intelligent Networking and Collaborative Systems (INCoS), pp. 552–559. IEEE (2013)
14. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
15. Pirretti, M., Traynor, P., McDaniel, P., Waters, B.: Secure attribute-based systems. J. Comput. Secur. **18**, 799–837 (2006)
16. Popa, R.A., Redfield, C., Zeldovich, N., Balakrishnan, H.: Cryptdb: protecting confidentiality with encrypted query processing. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles, pp. 85–100. ACM (2011)
17. Sahai, A., Waters, B.: Fuzzy identity-based encryption. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 457–473. Springer, Heidelberg (2005). https://doi.org/10.1007/11426639_27
18. Sarfraz, M.I., Nabeel, M., Cao, J., Bertino, E.: DBMask: fine-grained access control on encrypted relational databases. In: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, pp. 1–11. ACM (2015)

19. Song, D.X., Wagner, D., Perrig, A.: Practical techniques for searches on encrypted data. In: Proceedings of the 2000 IEEE Symposium on Security and Privacy S&P 2000, pp. 44–55. IEEE (2000)
20. Xu, G., Ren, Y., Li, H., Liu, D., Dai, Y., Yang, K.: CryptMDB: a practical encrypted mongoDB over big data. In: 2017 IEEE International Conference on Communications (ICC), pp. 1–6. IEEE (2017)