Cyberspace: Jurnal Pendidikan Teknologi Informasi
Volume 2, Nomor 1, Maret 2018, 69-83

ANALYZING CLIENT-SIDE ENCRYPTION IMPLEMENTED IN CRYPTDB

Mira Maisura
Information Technology Education, UIN Ar-Raniry
Banda Aceh, Indonesia
E-mail: mira.maisura@ar-raniry.ac.id

Abstract

Client side encryption has become one of the choice for data security in outsourced database as it is believed to provide more security than the server side encryption. It allows users to protect their data and prohibit access to that data from unauthorized user. Here, key use for encryption and decryption are all belong and known only to the user. On the other words, data stored in the database are all in encrypted form. CryptDB, a new cryptographic technique, where the system acts as a proxy to protect the communication between the application server and database server, implementing the idea. The aim of this study is to look into more detail about the encryption scheme implemented in CryptDB in 2 different case study, using SEARCH command with the condition given. The result of the study will present how the statement change with the act of proxy, and the encryption scheme implemented here.

Keywords: security, database, cryptography

1. Introduction

Data and information, users, and software developers are into three unitary interrelated with each other. Information, which is renewable every second, is expected to reach the hands of the user at any time, whenever and wherever they are. This fast - paced conditions trigger the software developers to create technologies that will satisfy the desires of the user.

Outsourced database system (ODB) is the result of technology advancement in internet and networking, as user demand for more reliable technology, which can ease their daily work. If a company uses this system, it will ease the workload within the company, in terms of materials, funds, or time. Thus, they can also increase their performance in company main task.

One should be a concern in using outsourced database is that their security. Data can be accessed easily, the possibility of being lost, damage, and change during the data transmission process cannot be avoided. This problem can occur not only because the action of outside attackers, but even more alarming from the insider, which can be either a party within the company, or data server administrator. On the other hand such problem can be solved using encryption system Encryption is one of development in which data is stored cannot be read directly without the help of additional technology. Using the

encryption methods as the base, scientists developed a more complete technology, where data security is assured, the speed of data access can be overcome.

There are two types of encryption, server side, which is widely used, and client side encryption. In server side encryption, provider will manage the user data as well as the encryption key. These approaches ease the work of user or data owner. But as the number of attack from insider getting higher, the trust for data administrator is decreasing. And scientists propose the use of client-side encryption. As its name, client side encryption allow user to manage their own key and encrypt their data before storing it in the database [1]. The security of data is maintain by specifying who has the key and has ability to access, and modify the data. Database administrator for example, has less possibility to see inside the data, they can only see small amount of information regarding the stored data.

The focus on this research is on the implementation of client side encryption, implemented on CryptDB. We would like to see in more detail how the change happen during storing and retrieving data in encrypted database, what role and ability user has, and the role of proxy as the main component in CryptDB. Triggered by the fact that this type of encryption system is a great approach for data security, there is a need for further research and implementation in real life work.

2. Literature Review

CryptDB

CryptDB solves the security system in DBMS by providing a system where user does not need to trust the server or its administrator. Its main purpose is to ensure the data secrecy from interfere who has access and control over the data. Basically, CryptDB will work by intercepting the queries sent from the application machine, and translate it into a new form in a way that it can be executed over encrypted data [2]. Using this approach, CryptDB tried to minimize the possibility that an adversary / untrusted party has ability to access the data, learnt, and misuses it, unless he owns the key. There is no possible way that unauthorized party will be able to access the data unless he owns the key or stole it from the user, who managed it.

Architecture of CryptDB consists of two main parts:

- Frontend

This part is trusted client side, which keep the track of key use for encryption and decryption, and database scheme which can be seen by the user.

- Server

The untrusted server is used to keep track of the encrypted scheme as well as the auxiliary table used by CryptDB. Server will keep the encrypted form of data, including its schema, onion encryption scheme used for computation, and the key.

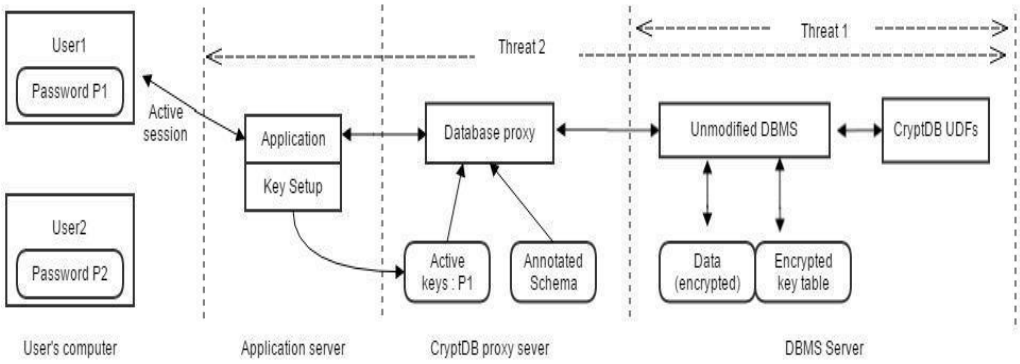

Figure 1 : CryptDB architecture [3]

Figure (2) below illustrate the working flow in CryptDB; (1) user will send a query, which will go through the query rewriter. Here, database name, tables and data inside it will be rewrite into encrypted form. (2) Onion Key Manager (OKM) will check and decide whether the data need an onion key for executing the query. Onion key will be provided if it is needed (3), then send it to the DBMS. Part (4) and (5) are part of retrieving the data from database and using the result decrypter to obtain the original result.

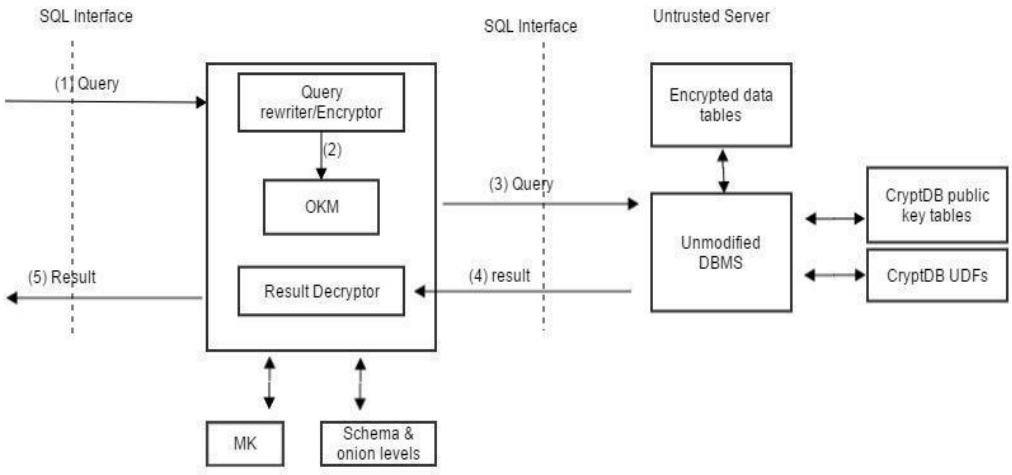

Figure 2 : CryptDB working flow

Security Overview

CryptDB provides several properties, such as:

1. If sensitive is in plaintext form, then it should not visible in the DBMS server.
2. CryptDB doesn't hide all information, it reveals several information to the server, depend on the application queries and its needed computation

3. Server cannot compute the (encrypted) result for the queries involve computation classes not requested by the application.

Popa et al., 2011 mentioned that there are 2 main safety issues CryptDB has found, namely as follow:

1. Database administrator (DBA)

DBA mentioned here refers to the curious-untrusted DBAs who has access to the system and might misuse the sensitive information stored in database, such as bank account, financial documents and many more. Both actions and the actor are harmful as it is violate the confidentiality of stored data. The curious DBA is in a type of passive attacker, who did nothing but learn about the data. He will not change the stored data, modifies query sent by the application, or change the retrieved information.

CryptDB provides at least two solutions for this problem. The first is by decreasing the level of sensitive information disclose to the server using encryption system. But then another problem come up from this system, it has either slow performance or unable to produce output with better confidentiality.

Second solution is by increasing the ability for query execution more efficiently Using Advance Encryption Standard (AES). Despite its strongest, AES restrains the server for executing many queries, which is contradictory with the approach solution. One logical solution would be to let the server to have access to the decryption key. But this solution is a window for second threat to enter the system.

2. Adversary

Adversary gains complete access and control of the applications as well as the server. The threat can attack the server or the proxy, which might be jeopardized. Compare to the first threat, threat from adversary is more harmful, as he has complete access and control to the application and server. Here we found the proof that CryptDB cannot ensure the integrity or completeness of data stored in the database. Adversary, which has access to the key used for encryption scheme in the database, can modify, or even worse, delete the data stored on it.

CryptDB offers several possible solution. To increase the data security, we need to decrease the amount of data revealed when they are endanger both the application and the server. However, in order to implement this solution, we need to find a way to ensure that the jeopardized application can gain only certain amount of decrypted data which can be access by the application itself. Another possible solution is simply by providing each user with unique decryption key. This approach will be difficult to implement in some platform. For example in an application which has common page such as bulletin board. To overcome this matter, CryptDB come up with idea of using different keys to perform encryption on different data items. By doing so, each data item stored in database will have its own unique key.

With the information mentioned on the threats above, CryptDB provide three main key ideas, that is:

1. SQL-aware encryption.

Using SQL-aware encryption strategy to perform an execution on the encrypted data. Such strategy define that all SQL queries are formed by primitive operators, such as equality check and joins. By knowing this CryptDB then will encrypt each data item before storing it. Detail information about this approach will be explained in chapter 3.

2. Adjustable Query-based Encryption

In designing CryptDB, choosing the right of encryption scheme is very important. It should be the most secure scheme which enables the query to be processed. To avoid data being leak, CryptDB choose carefully the SQL encryption scheme for any data end to the server. Using onion encryptions, in which each data will be encrypted accordingly, depend on its type. Adjustable query based decrypt the database column to the least secure onion layer. It takes short time and fast to perform such computation, as it takes only once to visit a column and decrypt it.

3. Key chain

Key chain means creating a connection link between the encryption key and the user password. In this manner, when user logged in to the application, he can decrypt the data item only through a chain of key connected to the password user poses to access their data. Otherwise, if the user is not log in, and the adversary has no information regarding the password, then there is no possibility that he can decrypt any data, even though the system is already jeopardized.

Client-Side Encryption in CryptDB

In CryptDB [7], there are several type of encryption is used to implement computation over sql query, called SQL-aware encryption. Each of them has different security level and needs in CryptDB. The use of it is also depend on the data type in database.

1. Random (RND)

RND has the highest level of security, such as IND-CCA2, where it only used when the user need to retrieve data from database without additional operation required on the server side. Using generated initial vector, RND produces a cipher text from a column name. Even though there are advantage of using RND such as it provides powerful encryption and has ability to handle sensitive data, RND has limit some SQL computation such as ORDER By and SUM.

2. DET

Deterministic (DET) exposes some information about the data, which make this scheme slightly weaker than the previous one. The server can perform the computation such as equality filter, joins, group by in this encryption level. Because of that, there are some information reveals to the user.

3. OPE

Order Preserving Encryption (OPE)[6] is an encryption scheme which allow computation over encrypted data in database or/and other application. It let the server to perform order comparison which is used for data sorting, range checking, and etc, on the cipher text.

OPE conserves the order of cipher text and order of the plaintext to be remained the same. For example, if, for any key, a < b , then OPE(a) < OPE(b) . Because it reveals data order, this method of encryption is weaker in compare to DET. This computation can be seen in case study 1.

Boldyreva et al., 2009 declared that there is a small chance to define the ideal goal of OPE, which is the limit the information leakage of the plain text beside what is ordered.

4. HOM

Homomorphic encryption (HOM) is an encryption scheme which is IND-CCA secure, where the unauthorized party who are curious with user data lost the ability to distinguish between encryptions of different messages, even when allowed to make encryptions and decryptions of its choice.

CryptDB assembles several of this techniques so that many sql operations can be computed over the encrypted data. Basically this encryption system sustains the relationship between two different plain text. By using certain computation, HOM, which is used to compute and analyze mathematical operations, generates an encrypted data, which when it is decrypted, it will return the same result as the operation done in plaintext.

Scheme	Operation	Example
RND	None	AES in UFE
HOME	+, *	Pailier
DET	Equality	AES in CTR
JOIN	Join , OPE-JOIN	**new**
SEARCH	ILIKE	**Song algorithm**
OPE	Order	**Boldyreva**

Table 1: Security Level of Encryption Scheme

5. Join (JOIN and OPE-JOIN)

There are 2 join operations supported in CryptDB. First is for finding data using equality, and the other for checking data order.

For finding equality from two different column, we need to use different key for DET, so that we can avoid the possibility of cross-column correlation to happen. DBMS

also should not have ability to connect the columns beside what user asked for. Therefore, only connected columns are encrypted with the same key, while the others are not.

While all operation conducted in DET is supported by JOIN, as well as finding the equality of different columns, OPE-JOIN use order relation to enable all computation using JOIN. OPE scheme has a lack in structure, because of that it is not possible to compute the order check in the similar procedure as in equality check.

Here, the 2 columns which is in pair will need to be disclosed earlier by the application, or else in all layer of OPE, the same key will be used.

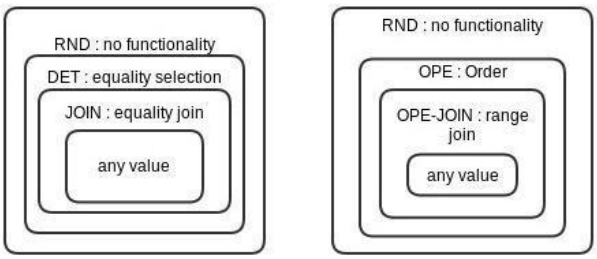

Figure 3: Eq and Ord onion

6. Word Search (SEARCH)

As its name, this method is used for searching on encrypted data, by implementing the Song's Algorithm. In security level, it is almost as secure as RND, because there is no data leakage after the computation. The only drawback of this method is that it reveals the frequency of word repetition, if there exist such, which ease the adversaries to calculate the possibility of repeated words.

3. Result

During the 2 case studies, we will use the original SQL statement as comparison to check how the data transform into the new form. It includes the changes of SQL statement and the result displayed. We would like to also see what if, for some reason, some adversaries has a knowledge about the table name in encrypted form, whether he will be able to perform any operation or not.

We created simple database called *uni_bonn* as shown in table (1). Database *uni_bonn* consists of original data we manually inserted. For security reason, storing original data into the database is not recommended. Here, proxy will act as a data translator, and transform all information displayed above in encrypted form

id	username	department	grade	sid
1	Alice	biology	3	40
2	Bob	math	2	50
3	Eve	medizine	5	80
4	Mary	arts	1	110
5	Claire	social science	2	220

Table 3: Database created for case studies

If the table is already encrypted, as shown in table (4), then, we can be sure that our data is saved. We see that the table name already changed from username to table *PEPKJXJWFR*.

tables_in_uni_bonn
table_PEPKJXJWFR

Table 4 : encrypted table

Case Study 1: Search with condition

In case study (1) , we will perform search query over encrypted data, with the given condition.

1. SELECT * from username where id >3

For such query, CryptDB will return all information which satisfy the condition. In the proxy side, the result will printed out in encrypted form as shown in Table 4, which is clearly part of the completed table which printed out in the case study 0.

IUOHJZMRYXoEq	SKIASJJMCUoEq	SRNLGSVJBSoEq	cdb_saltZXWVQPYFWY	WDRMQXELIoEq	cdb_saltMAOIYEZWLY	FNEZKFVSFZoEq	cdb_saltYDARRSQVGY
7702453780396703159	??5?????AK????#I<?+6??=????7??u]	B???{?xij5j????9?C?xV???????]??????!??/???????	2596416236337042212	9774416075842806816	1390011771679222052 0	5818831939587111618	1543596622587943732 3
3029401382327740934	z???H??W???o!?zx??Y??????B?&?????	P??$1?6?I?|8??????lmz???F??????5??0j??????d8???`	1239549034 6234820682	1402400026192350158 4	4618352711516160 18	1384806205033714754 3	1090233362003779493 9

Table 5: return result for id>3

Case Study 2: Like command

1. SELECT * from username LIKE %c%;

```
mysql> select * from username where username LIKE %c% ;
ERROR 1105 (07000): Error: Bad Query: [select * from username where username LIKE %c%]
Error Data: parse_sql
FILE: main/rewrite_main.cc
LINE: 1380
```

Figure 4: LIKE command-1

2. SELECT * from username LIKE “%Alice” ;

```
mysql> select * from username where username=Alice ;
ERROR 1105 (07000): Error: Bad Query: [select * from username where username=Alice]
Error Data: JOIN::prepare
FILE: main/rewrite_main.cc
LINE: 1380
```

Figure 5: LIKE command-2

In this case study, we tried several combinations of using LIKE command, to see if there is any small chance that it will return some value. But at the end, all of them always return false. At this point, we can assume that LIKE command is one of the restriction in CryptDB.

4. Analysis

In CryptDB, before storing the data in database, it will translated into an encrypted form by proxy. We will divide the process, from encrypting the data, store and search execution perform in CryptDB, by using the case studies.

Queries over encrypted Data

While doing the experiment in case studies before, we see that the sql statement changed significantly from the original, with the help of proxy. We will use the statement in case study 1;

```
SELECT *
FROM username
WHERE id > 3
```

```
select
`uni_bonn`.`table_PEPKJXJWFR`.`IUOHJZMRYXoEq`,`uni_bonn`.`table_PEPKJXJWFR
`.`SKIASJJMCUoEq`,`uni_bonn`.`table_PEPKJXJWFR`.`SRNLGSVJBSoEq`,`uni_bonn`.
`table_PEPKJXJWFR`.`cdb_saltZXWVQPYFWY`,`uni_bonn`.`table_PEPKJXJWFR`.`W
DRMQXELIIoEq`,`uni_bonn`.`table_PEPKJXJWFR`.`cdb_saltMAOIYEZWLY`,`uni_bon
n`.`table_PEPKJXJWFR`.`FNEZKFVSFZoEq`,`uni_bonn`.`table_PEPKJXJWFR`.`cdb_sal
tYDARRSQVGY`

from `uni_bonn`.`table_PEPKJXJWFR`

where (`uni_bonn`.`table_PEPKJXJWFR`.`PHGIZGUTJYoOrder` > 15734563768)
```

In proxy, the statement change into gibberish message as shown above. Application ask for every information in database, by calling all columns name in encrypted form. From query translated into `uni_bonn`.`table_*PEPKJXJWFR*`, which is in the form of : username ENCRYPTED , and the condition given is with id > 3, translated with the use of OPE. In general, the statement above can be written as follows:

```
SELECT
        id_OPE , username_DET , department_DET , grade_OPE , id_OPE
FROM
    username_ENCRYPTED
WHERE
    id_OPE > ENCRYPT_OPE(3)
```

In case study 1, proxy directly transform the sql statement in the format above. We would like to see, what will it do if we use the sign of != , as follows :

```
SELECT
FROM username
WHERE department != ‘math’
```

While computing this query, in the proxy side, it show how the data is being translated and encrypted step by step, it printed out as:

```
QUERY: select * from username where department != 'math'
Adjusting onion!
onion: oEq
ADJUST:
 UPDATE `uni_bonn`.table_PEPKJXJWFR
SET SRNLGSVJBSoEq = cryptdb_decrypt_text_sem (`uni_bonn`.`table_PEPKJXJWFR`.`SRNLGSVJBSoEq` AS
`SRNLGSVJBSoEq`,'??????o???????d?',`uni_bonn`.`table_PEPKJXJWFR`.`cdb_saltZXWVQPYFWY` AS
`cdb_saltZXWVQPYFWY`);
```

From this query, we see how CryptDB actually read the query execution over cipher text. After receiving query from application, CryptDB will do the following:

a. Adjust which onion should be used. In Table 3, it shows that each table is written in onion encryption
b. Take the encrypted value of department, in this query is correspondent to SRNLGSVJBSoEq and use in UPDATE and SET statement.
c. Take result from (b) and applied in the original query but in encrypted form. In general we can write it as:

SELECT id_OPE, column_ENCRYPT, FROM table_ENCRYPT, WHERE (UPDATE, SET value)

In all case studies, server return the expected value to the application without decrypting it. Question raise from here, how can the system read the data without decrypting it first? What type of algorithm is used to sort the requested value?

As all data are being encrypted before storing it in database, it is common known that in order to process some information requested by user, the overall data must decrypted first. But in this way, there will be data leakage, much more than what user asked for. For doing so, user also need to trust the untrusted server the perform of the operation and it can become a security thread.

So far, from both case studies, we can actually see that CryptDB assign each value with different type of encryption. There are many approaches for computation over encrypted data without decrypting any database such using fully homomorphic encryption which is prohibited in practice.

Song Algorithms

In order to make the data more secure, CryptDB rearrange the words and remove if there is word repetition in database. Song et al., 2000 mentioned that there is a cryptographic scheme which provide a security proof in encryption system, by segregate the search ability if the user is not authorized. It also has feature for searching hidden queries, where user can ask the server to search specific word without revealing the secret word itself to the server. In this way, the adversaries or untrusted party will have no ability to learn about the plaintext given the only cipher text if such scheme is applied.

Figure 6: Gobberish Message Gibberish image for restricted access

The flow of the use of Word search method is the following:

a. The key word will be split into several parts for every column where SEARCH is needed. It can be done by using keyword extraction for example
b. Remove all the words which has duplication or repetition.
c. Rearrange the positions of the word.
d. Encrypt the word by using the Song's algorithm
e. Group together if there are words with the same size.

In the basic scheme of Song's Algorithm [3], the scheme work as follows; First, user will generate a sequence of pseudorandom value using a generator G with *n-m* bits long each. To encrypt that word, user will take the value of and set of and pseudorandom function with the specific key. This computation will give an output in cipher text form. Set of and can only be generated by authorized user and no one else can decrypt it.

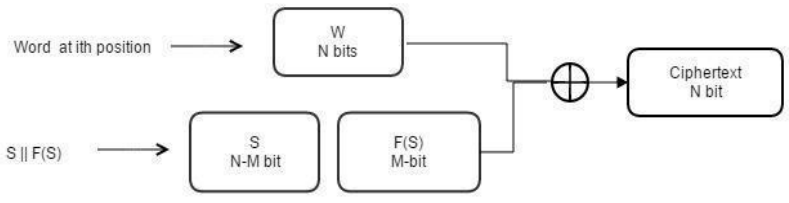

Figure 7: Basic song's scheme

In case study 2, we tried to get the data about specific user by sending the query with LIKE command;

```
SELECT * FROM username WHERE username LIKE %c% ;
```
And
```
SELECT * FROM username WHERE username LIKE "%alice%".
```

Based on R.A Popa et al., 2011, this operation can be done in CryptDB using Word Search. As far as the word sent is in a full word, then it can be completed. Application will send such as query and proxy will take its encrypted form and send it to the DBMS, which will search for the match word using UDFs function.

In our experiment, such query will return nothing but invalid statement. This is a contradiction with the previous work. It is possible the word search can only return some result if the query sent by the application is providing the exact word, such as:

```
SELECT id FROM username WHERE username = 'Alice'
```

which works in the similar way with the case study 1. This query will take the encrypted value of Alice, find the matching word in the database and return the exact id to the application.

5. Conclusion

Server side encryption ask user to put their trust on the untrusted server completely. User let the provider to manage their data together with the key. It might be efficient enough and less work for user, but it become a security thread. On the other hand, Client-side encryption provides the opposite service of server-side encryption. It allows the user to manage their own data. Starting with encrypting their data in the local machine and keep the encryption key for themselves. Unless the user loss the key or let other user to learn their key, it is less possible that the data can be stolen easily.

From this point of view, we can assume that Client side encryption provide better security comparing to server side encryption. But even so, there is still a security thread need to be considered happen in this type of encryption.

Security Issue

Comparing the security between the basic outsourced database, CryptDB, and Monomi, is quite obvious. CryptDB work better by applying the client-side encryption, which reduce the possibility for adversary or unauthorized user to sneak into the data. Even though the data is stored after being encrypted by user, other user can still learn about it. S. Tu et al., 2013 summarized it as the following:

Encryption Scheme	SQL operations	Leakage
Randomized AES + CBC	None	None
Deterministic AES + CMC or FFX	a = const, IN, GROUP BY, equi-join	Duplicates
OPE	a> const. MAX, ORDER BY	Order+partial plaintext
Paillier	a + b , SUM(a)	None
SEARCH	a LIKE pattern	None

Table 6: Information reveals in encrypted database [8]

Some encryption scheme reveals some information regarding the data, either it is more what the user request, or information leakage which is need to process the query.

Storage Limitation

There are several reasons why one should have a big space for data storing before implementing CryptDB, such as:

1. For each data store in database, it will be encrypted using onion encryption, so that there will be multiple column for the same field exists in database.
2. Depend on the scheme used, the cipher text size can change to smaller, or even the same size. So then, we will have double data size from the original
3. CryptDB will encrypt all data, it does not distinguish between sensitive or not.

Contradictory fact

In our experiments, we are focusing on two main case studies, one with the ">" sign is corresponded to OPE, and LIKE command to SEARCH approaches. We found that the computation using OPE in case study 1 will reveals some information to the application, such as data order of plaintext, as mentioned in the above table.

There is one point from the claim about data reveals that contradictory with the query in case study 2. We computed a LIKE pattern in CryptDB. Based on R.A Popa et al., 2011 and S. Tu et al., 2013, CryptDB will reveal in which rows the match word that we are searching is located. In our case study, there is no data reveals. After sending such query, CryptDB return it as invalid. We have tried by using different combination for search with LIKE, but it still return the same. Instead, SEARCH can be computed if the query use an equal sign with the exact word in it.

If the result we have got so far is true, then we might assume that CryptDB is one level more secure than what it claims in the previous work.

References

[1] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB : Protecting confidentiality with encrypted query processing. In Proc. of the 3rd SOSP, pages 85-100, Cascais, Portugal, Oct 2011

[2] M. G. Solomon, V. Sunderam, L. Xiong , Towards Secure Cloud Database with fine-grained access control Book of Data and Application Security and Privacy, pages 324-338 , Springer berlin Heidelberg, 2014

[3] R. A. Popa, C. M. S. Redfield, N. Zeldovich, and H. Balakrishnan. CryptDB : A practical encrypted relational DBMS. In Proc. of 23rd ACM Symposium on Operating Systems Principles, pages 85-100, USA, 2011

[4] S. Tu , M.F. Kaashoek, S. Madden, N. Zeldovich. Processing analytical queries over encrypted data. In Proc. of the VLDB Endowment Volume 6, pages 289-300, March 2013

[6] R. A. Popa, F. H. li, N. Zeldovich . An ideal-security protocol for order-preserving encoding. In Proc Sp'13 of the 2013 IEEE Symposium on Security and Privacy, pages 463-477, Washington, DC, USA, 2013

[7] CryptDB https://css.csail.mit.edu/cryptdb/

[8] S. Tu , M.F. Kaashoek, S. Madden, N. Zeldovich. Processing analytical queries over encrypted data. . In Proc. of the VLDB Endowment Volume 6, pages 289-300, March 2013

[9] A. Boldyreva, N. Chenette, Y.Lee, A. O'Neil. Order-Preserving Symmetric Encryption. In Proc of the 28th Annual International Conference on Advances in Cryptology : The Theory and Applications of Cryptographic Techniques, pages 224-241, Berlin, Heidelberg, 2009

[10] CryptDB source Code, available at https://github.com/CryptDB/cryptdb

[11]D. X. Song, D. Wagner, A. Perrig. Practical Techniques for Search on Encrypted Data. In Proc of the 2000 IEEE Symposium on Security and Privacy, page 44, USA, 2000