Secure Database in Cloud Computing: CryptDB Revisited

Ziynet Nesibe Dayioglu[1,2], Mehmet Sabir Kiraz[1], Fatih Birinci[1], and Ihsan Haluk Akin[2]

***Abstract*—Databases are including valuable and critical data, therefore, they are the most desirable to the malicious adversaries. To preserve the security of database outsourcing scenarios, various techniques have been proposed so far which preserve a certain degree of confidentiality while still allowing to execute some SQL queries effeciently. CryptDB is an approach to outsourcing database while preserving confidentiality and performing a set of SQL queries in an effecient way.**

In this paper, we revisit CryptDB and describe it in more details for ease of understanding. We then highlight the drawbacks of CryptDB from security and efficiency points of view.

***Index Terms*—Secure Database, Encrypted Search, Cryptographic Protocol, Proxy Server**

I. Introduction

CLOUD computing's detailed definition exists at NIST's website [1]. In our work, we simply define cloud computing as the delivery of on-demand computing resources located on the remote servers. Through the evalution of tools and broadband connectivity, cloud computing is getting more feasible from a user's perspective [2], since it reduces cost and increases mobility [5].

With the increasing popularity of cloud computing the security issues also rise [3], [4]. Organizations considering migration to cloud based services must also consider and understand security, privacy, reliability, and regulatory issues. Important research agencies are aware of these risks and have produced reports [6], [7], [8], [9]. According to Gartner, cloud computing security risks can be summarized by seven categories: "Privileged User Access, Regulatory Compliance, Data Location, Data Segregation, Recovery, Investigative Support, Long-term Viability" [6]. Security issues are investigated more deeply under two main subjects namely, "loss of control over data" and "dependence on the Cloud Computing provider" [15].

Outsourcing databases into cloud can increase levels of availability, robustness, elasticity and efficiency as well as minimize administrative reasons. However the data in the cloud can be accessed by the cloud provider. So, the cloud provider, its employees or even subcontractors can deliberately or inadvertently access customers' data. In order to technically ensure data confidentiality, the data can be encrypted before being outsourced. However, executing queries as computationally perfect secrecy on an encrypted data cannot be generated efficiently. Performing non-trivial computations with the data on cloud such as searches, transformations, selections, and access control decisions is useful. Conventional encryption prevents processing this data on the cloud. Genrty solved a thirty year awaited problem in Cyrptography [14]. He propesed a novel encryption scheme which can allow processing on encrypted data. Nevertheless this encryption scheme is far away from being practical. DARPA separate $20 million to search for cryptography's this problem as a practical solution [16]. Popa and friends from MIT bring practical solution to processing encrypted database [10]. This paper focused on their work in more details and concentrated on their work's weeknesses.

(1)TUBITAK BILGEM UEKAE, Kocaeli, Turkey (2)Fatih University, Istanbul, Turkey

E-Mails:{ziynet.dayioglu, mehmet.kiraz, fatih.birinci}@tubitak.gov.tr, ihakin@fatih.edu.tr

A. Related Work

Researchers are trying to find solutions to keep data on the cloud confidential. Processing data securely on the cloud is complicated. In this section, we will give some well-known approaches for solving secure computing on the cloud.

Fully homomorphic encryption. As mentioned in the previous section, the data can be encrypted before uploading to the cloud. However, using conventional encryption schemes, the data cannot be processed on the cloud. In this case, the cloud can only be used for storage purposes. Homomorphic encryption may overcome this limitation upto a certain level, which allows logical operations on ciphertexts without decryption. Homomorphic encryptions like Paillier or ElGamal allow just one operation, namely either addition or multiplication [25], [26]. Fully homomorphic encryptions schemes allow addition and multiplication on ciphertexts, which allows an untrusted server to carry out arbitrary computation on encrypted data on behalf of a client without decryption. Namely, using fully homomorphic encryption schemes the cloud provider can run any program client wishes without obtaining any information about the plaintexts. Fully homomorphic encryption schemes are first invented by Gentry in 2009 [14]. Unfortunately, there is no practical scheme using fully homographic encryption today but a lot of work is being done in this field, some of them may be promising for cloud computing [20].

CryptDB: a weaker attacker model. CryptDB [10] is an implementation that allows query processing over encrypted databases. The database managed by the cloud provider, but database items are encrypted with keys that are only known by the data owner. SQL queries run over the encrypted database using a collection of operations such as equality checks and order comparisons. CryptDB uses encryption schemes that allow such comparisons to be made on ciphertexts. CryptDB represents a weak attacker model because it assumes the existence of a trusted cloud-based application server and proxy.

Nevertheless, CryptDB represents an interesting position on the trade-off between functionality and confidentiality from cloud providers. In this paper, we will go into details of CryptDB.

Private Information Retrieval. To enable search on remote database Secure Multi-party Computation can also be used [19], [21], [24]. These schemes are already used in practice and therefore, may be promising for cloud computing issues in the near future [22], [23]. Private Information Retrieval schemes solves the problem with absolute privacy (e.g., [17]). Curtmola et al. proposed two schemes against non-adaptive and adaptive adversaries with a good performance [18].

B. Contributions

The main contribution of this paper is to provide a detailed thread and performance analysis of CryptDB. For ease of understanding we first describe CryptDB with more detailed explanations. Next, we describe the server structure of CryptDB and we analyse the overhead of Proxy Server. Unfortunately, the original paper does not explain the search algorithm, therefore we have extended the search algorithm of the CryptDB. We show that the current search algorithm is not enough to meet all the search queries. We highlight the security and effeciency points of CryptDB. Finally, we present some remarks and open research areas for CryptDB.

II. CryptDB Architecture

A. Layered Encryption

The encryption of a data in the database is computed in a layered way. There are four different main goals to achieve, and for each goal there exists a different layered particle, which is called as onion [11]: EQ, ORD, SEARCH and ADD onion. EQ onion aims to adjust layers for equality queries, while ORD onion aims to adjust the order leakage for the queries including comparison. SEARCH onion is used to search a text in the database without leaking any information. This onion is not allowed to execute integer values. Finally, ADD onion aims to add encrypted values which only supports integer values. These onions have different layers each encrypted by using different algorithms. Furthermore, these algorithms have different security levels where the outer layer of an onion is more secure than the inner ones. Furthermore, a value has only one current layer in each onion. Once the database has been created, all the onions will be at the most secure layer.

Queries are processed as columns in a database, therefore, CryptDB is also column-oriented. If one value needs to be decrypted to weaker layer, then the whole column is going to be decrypted. However, it causes more information leakage than requested. If we need a value to be the weakest layer, the whole column will be at this layer and leak the information. And also note that the inner layer is the weakest layer, but it also does not reveal any plaintext to DBMS Server. The inner layer of the onions which are Equi-JOIN, OPE-JOIN, SEARCH and HOM layers are never stripped off.

EQ onion.

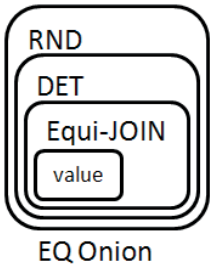

Fig. 1. EQ Onion

At the most secure layer of EQ onion is RND layer. This RND layer encrypts each value with AES (Rjindaels algorithm) in CBC mode. For integer, Blowfish in CBC mode is preferred because of its 64-bit block size instead of 128-bit to decrease the length of the ciphertexts. The initialization value for both encryption type is randomly chosen values. Each value goes to the different ciphertext with high probably even if the plaintexts are the same. For this reason, RND layer is indistinguishable under an adaptive chosen plaintext attack. However, this scheme does not provide an efficient functionality.

When the equality check of the values is needed, the RND layer should be stripped off. The next inner layer of EQ onion is DET layer. This layer is deterministic layer. After encryption is done, if two values are the same, then their corresponding ciphertexts will be the same.

For the values inside the same column, these two layers, RND and DET, are sufficient to process queries. However, some queries need to check either the equality of two different columns or two different columns of different tables. For these situations, the current layer should be updated Equi-JOIN layer. In this layer, JOIN-ADJ is concatenated to the encryption of DET layer.

$$\text{JOIN} = \text{JOIN-ADJ}||\text{DET}$$

JOIN-ADJ allows DBMS server to adjust the key of each column at run time. It is somehow a keyed hash with additional property that hashes can be adjusted to change their key without access to the plaintext. JOIN-ADJ is a deterministic function, also collision resistant (192-bit), non-invertible, and transitive. JOIN-ADJ function is defined below:

$$\text{JOIN-ADJ}_K(\text{value}) = \text{P}^{K.PRF_{k0}(value)}$$

Elliptic Curve Cryptography (ECC) is used as an algorithm. K is initial key for that table, column, onion, and layer. P is a point on an elliptic curve which is a public parameter. PRF is a pseudorandom function mapping values to a pseudorandom number, such as $\text{AES}_{K0}(\text{SHA}(\text{value}))$. K0 is the key which is the same for all columns and derived from Master Key.

The exponentiation is in fact repeated geometric addition of elliptic curve points, and it is faster than RSA exponentiation. Proxy computes $\Delta K = K/K'$ and sends it to DBMS Server. DBMS Server uses a UDF to make them share the same JOIN-ADJ by computing below with the assumptions that column c has K, and column c' has K' as keys at JOIN-ADJ

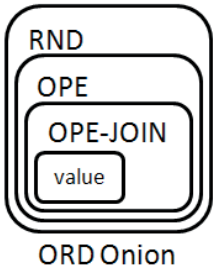

Fig. 2. ORD Onion

layer.

$$[\text{JOIN-ADJ}_{K'}(\text{value})]^{\Delta K} = [\text{P}^{K'.PRF_{k0}(value)}]^{K/K'} = \text{P}^{K.PRF_{k0}(value)} = \text{JOIN-ADJ}_{K}(\text{value})$$

By using the appropriate UDF, the JOIN-ADJ of two different columns are the same anymore. Therefore, it is possible to compare the equality of them by looking at the JOIN-ADJ. Because the DET layer uses different keys for each column for correlations between columns. Security of this scheme is based on the standard Elliptic-Curve Decisional Diffie Hellman hardness assumption.

ORD onion. The most secure layer of the ORD onion is exactly the same as the EQ onion, both is RND layer. The inner layer of the RND for ORD onion is OPE layer. Order preserving encryption algorithms have not enough security and efficiency until now. In the original CryptDB's paper [10], the authors do not consider the order-preserving encryption but they later pointed about this issue in their article "An Ideal-Security Protocol for Order-Preserving Encoding" in [11].

The limited information is given about this onion in CryptDB's article [10]. If the encryption of x is smaller than the encryption of y, then the value x is also smaller than y. If we find any encrypted value which is between these two encryptions, then the plaintext will also lie between x and y. Namely, this scheme leaks the order, therefore, it is a weaker scheme when compared to equality leakage.

For the values at the same column, this layer is enough to process the queries, but if two different columns are compared to check order, then we need to strip off the OPE layer, and reach the OPE-JOIN layer. This layer is lack of functionality than the EQUI-JOIN layer of the EQ onion. To adjust two columns are not possible because of the lack of the Order-Preserving algorithms' efficiency. There are two solutions. First one is, the application will declare the columns which can be joined, and while arranging the keys, the same key will be used for these columns. It is not logical in most situations to declare ahead of time. The second solution is the same key will be used for all columns at this layer. This solution is not a good solution also. Fortunately, range joins are not used too much.

SEARCH & ADD onions. SEARCH onion has just one layer, so there is no decryption process for this onion.

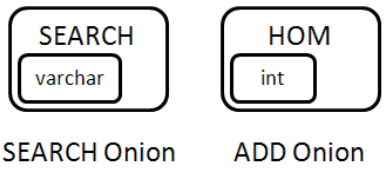

Fig. 3. SEARCH & ADD Onions

SEARCH onion has a value, and the SEARCH layer which covers this value. Search algorithm used in CryptDB is the Song's search algorithm which is described in Section III-D. The aim of the SEARCH onion is searching an encrypted value inside an encrypted table.

ADD onion also has the same situation. The HOM layer is the only one layer of ADD onion. This onion's aim is to provide some functionality with encrypted values without access to the plaintext. This can achieved with homomorphic encrytion. The algorithm used for Homomorphic encryption of CryptDB is the Paillier's algorithm.

B. Adjustable Encryption

If we keep all the layers for each onion, it is not a good way. Because if the weakest layer is given, other layers are not necessary to create. However, if we use a layer for each onion, it is hard to know the most secure layer which meets our need ahead of time. For this reason, it is a requirement to adjust current layer dynamically without leaking the data. This problem is solved with adjustable functionality. There are different onions for different aims. All these onions are put into the table. According to our aim, we choose our onion, and use it as well. And each value is encrypted with all outer layers, beginning from the weakest to the most secure one. With this approach, if no query requests the weaker layers, then these layers are not available to use.

For each column in a table, the same key is used to encrypt the values inside, but for the different tables, columns, onions, and also layers, different keys are used. The key generation is done with a pseudorandom permutation like below:

$$\text{Key} = \text{PRP}_{MK}(\text{table t, column c, onion o, layer l})$$

In the beginning, all onions are the most secure layer which are RND, HOM, or SEARCH layers. If equality or order leakage is requested, the outer layer will be decrypted. This decryption does not give the plaintext because of the layered encryption. The remaining layers still cover our data to keep them secure. Furthermore, the weakest layers which are OPE-JOIN, Equi-JOIN, SEARCH, and ADD layer are never stripped off.

C. UDF: User-Defined Function

A function contains instructions in order to perform a specific task, and provides to repeat this task easily. User-defined function is also a function, and it is created by a user. This user needs to have permissions to perform the processes in the database, or database owner abbreviated as dbo can be

used. dbo is a user which can perform all activities in the database.

If we have a table called Square which has two columns as edge length and order number of the square, then we can create a UDF to calculate the area of the squares inside the database. The sample Square table can be in Table I.

TABLE I
THE SAMPLE SQUARE TABLE

Square	
Number	**EdgeLength**
1	10
2	5
3	7

There is no need to keep the area values causing extra overhead to the database. We can create a UDF, and call it whenever area is needed. Here is an example of creating a UDF.

```
CREATE FUNCTION dbo.area (edge FLOAT) RETURNS FLOAT
RETURN (edge * edge);
```

An example query to use this UDF can be like as follows:

```
SELECT Number, area (EdgeLength) AS Area FROM Square;
```

The returning result will be like Table II.

TABLE II
THE RETURNED RESULT OF THE SQL

Number	Area
1	100
2	25
3	49

Instead of keeping data for each information which can be required, it is better to create a UDF, and call it whenever this task is needed. In CryptDB, UDF is also a requirement. Decryption of the layered architecture to adjust the current layer is done by using user-defined functions. And also updating current state of the onion layers is done by using the UDFs. Changing all existing database mechanism is hard to achieve. Therefore, UDFs are very important to add functionality in the database systems.

D. Server Structure in CryptDB

In the database systems, users request some information or functionality from the applications. The SQL queries are used in order to meet the need of the users. The Application Servers send the queries to DBMS Server. DBMS stands for Database Management System. DBMS Server takes the SQL and responds the result set. This system is open to any eavesdropping and attacks of adversaries. Also DBMS Server may be curious about the queries and track the queries with their results. The other possibility is that the physical access to the disc can cause the data to be stolen. In CryptDB, the mechanism of the queries are the same, but server structure somehow changes, and an extra server called Proxy Server is added to the database system.

It is assumed that all the queries coming from the Application Server is taken by Proxy Server, and sent to DBMS Server after some modifications. The aim is to keep no meaningful information at DBMS Server's side in order to prevent the curious Database Management Administrator to see the contents of the tables in its database [27]. For this reason, we need to make the whole data meaningless.

The first thing is to create a table in order to keep some data inside DBMS Server's database. In CryptDB, the table of DBMS Server is totally different from the real one. Proxy Server changes the table's name. Then, according to type of the column (e.g. int, varchar), there exists some possible onions. Proxy Server keeps all possible onion's data in different columns separately in this table. Table III is a basic example.

New created table from Table III by Proxy Server is shown in Table IV.

The second thing is to add instances to the current tables. The names of the columns are changed, and each value inside the table are encrypted according to algorithm of its onion's layer. For instance, if the current layer of the EQ onion is RND layer, and query deals with the equality, then each value in the corresponding column will be encrypted with AES (Advanced Encryption Standard) in CBC (Cyclic Block Chaining) mode. Or the Paillier's algorithm will be used to encrypt the values which belongs to HOM layer of the ADD onion. All these encryption processes are done by Proxy Server.

When a query comes to Proxy Server, it changes the query. First of all, Proxy Server decides which onion is used for this query. Proxy Server has extra tables, one of them is for keeping the current state of each column's each onion. After deciding the onion of the query, Proxy Server looks at this table, and checks that the current state of the onion is at the needed layer for this query. If not, Proxy Server calls the UDF which is responsible for the stripping off the current layer to the needed layer. After this process, Proxy Server modifies the query, and sends it to DBMS Server. The whole data stored in DBMS Server's database is encrypted, and the queries are not meaningful for the Database Management Administrator.

DBMS Server takes the modified query, and returns an encrypted result to Proxy Server. Proxy Server takes the encrypted values, and decrypts them, and sends to the Application Server. The Application Server is not concerned about any encryption processes, it just sends its original plaintext, and take the results, and gives service to the clients with these results. The basic server structure of the CryptDB is described, and overhead which Proxy Server has to deal with is shown.

III. A BASIC EXAMPLE

In this section we describe an example to explain the basic structure of CryptDB. There is a single principal in our example, no modification or restrictions are done to the principle. Each table has only one Master Key abbreviated as MK throughout the paper. We have an example table called Students. This table has one Master Key which we abbreviate as MK. As mentioned in Section II-B, each table's each column's each onion's each layer has different keys from the following equation:

TABLE III
THE CREATED ORIGINAL TABLE AND ITS COLUMNS

Employees	
ID	**Name**

TABLE IV
THE MODIFIED TABLE FOR DBMS SERVER

Changed Table Name							
ID-IV	ID-EqOn	ID-OrdOn	ID-AddOn	Name-IV	Name-EqOn	Name-OrdOn	Name-SearchOn

$$\text{Key} = \text{PRP}_{MK}(\text{table t, column c, onion o, layer l})$$

A. Queries with security but without functionality

After reading the CryptDB's article [10], even if it easy to understand the structure, it is still hard to combine the CryptDB mechanism with SQL queries. For this reason, we illustrate a basic example in order to clarify the steps in more details.

Students table has two columns ID and Name. ID stands for identification number of the student and Name stands for student's name. ID column takes an integer value while Name column takes a string. Students table is created with the following statement:

```
CREATE TABLE Students (ID int, Name varchar(255));
```

After creating table, we are ready to insert some instances as follows.

```
INSERT INTO Students VALUES (1, "Alice");
INSERT INTO Students VALUES (2, "Bob");
INSERT INTO Students VALUES (3, "Eve");
```

When we run all these queries, the Application Server creates the following table V. This table has pure data, so it is not a good way to store this table inside the database of DBMS Server for the security reasons. Namely, Proxy Server creates a new encrypted table for DBMS Server. The created new encrypted table will be as in Table VI.

TABLE V
THE TABLE AFTER RUNNING THE ABOVE-MENTIONED SQL QUERIES

Students	
ID	**Name**
1	Alice
2	Bob
3	Eve

Since the type of ID column is an integer, SEARCH onion will not be available for this column. Furthermore, since the type of Name column is string, HOM onion will also not available for this column. As described earlier in Section II, all layers will be at the most secure layer at the beginning. There are seven layers. The most secure layers of the onions are the three layers which are RND, SEARCH and HOM. However, they also have less functionality. For example, RND layer does not leak any data, but it has no efficient functionality. If we come back to our example, by using the most secure layer, we can run the queries such as "SELECT * FROM Students;", "SELECT Name FROM Students;".

Note that Proxy must modify the queries in order to protect original table contents. If we continue with the SQL "SELECT Name FROM Students;", the modified SQL will be like below:

```
SELECT C2-IV, C2-Eq FROM DBMS_Table1;
```

DBMS Server will take this query, and return the result in Table VII.

TABLE VII
THE RETURNED RESULT FROM PROXY SERVER TO THE APPLICATION SERVER

C2-IV	C2-Eq
lpw9	dkfm
suc0	d82w
3mnu	snge

The corresponding table which Proxy Server decrypts and sends to the Application Server will be like in Table VIII.

TABLE VIII
THE RETURNED RESULT FROM PROXY SERVER'S SIDE

Name
Alice
Bob
Eve

This query returns the result which has no functionality. However, there is no change at the ciphertext inside DBMS Server's table. You can just read the returned data, but in general the queries which are coming from Application Server require some functionality such as updating current data, selecting and showing some specific rows of the table, calculating average of a column.

B. Queries with equality leakage

Note that equality in the databases is an important feature and is most frequently used one. For example, IDs of the students which got AA grade from a lecture at a university, the names of the clients which buy a specific product in a shopping mall and the phone numbers of the people who send cargo with wrong destination address can be solved by the equality feature in real life. If we come back to our example, we can create such SQL queries:

TABLE VI
THE ENCRYPTED TABLE WHICH IS CREATED ON PROXY SERVER

C1-IV	C1-Eq	C1-Ord	C1-Hom	C2-IV	C2-Eq	C2-Ord	C2-Search
q39f	ge88	hwip	dwna	lpw9	dkfm	fdkw	98wu
c8x3	8n4o	s09s	28bd	suc0	d82w	8mx0	x9ak
sk7x	x7wk	x6sh	s7w9	3mnu	snge	xuwn	u8sb

SELECT name FROM Students WHERE id = 1;

Proxy Server will take this query, and modify it according to desired functionality. Encryption layer of the computed column is checked whether the current layer is at a layer which can respond to the requested query. If not, Proxy Server sends UPDATE query. This UPDATE query provides to adjust the current layer of this column with given keys from Proxy Server. In our example, we need to use a UDF which strips off the RND layer to the DET layer. The reason why is, if we want to know the equality of the two data at the same column, we need to use a deterministic encryption. If we assume STRIP_RND is a UDF which takes the RND layer and decrypts it to the DET layer, then Proxy Server will send below UPDATE query at first. This query will send the decryption key to DBMS Server, and DBMS Server is able to strip off the RND layer.

UPDATE DBMS_Table1 SET C1-Eq = STRIP_RND (Key, C1-IV, C1-Eq);

Recall that the key is coming from the following equation:

$$\text{Key} = \text{PRP}_{MK}(\text{table t, column c, onion o, layer l})$$

After this UDF, the C1-Eq column is at the DET layer anymore. And Proxy Server will update this column's current onion state to the DET layer.

Decrypt_{Key}(C1-IV, C1-Eq) is the structure of the decryption function. Key is RND layer's decryption key for the first column in Students table. Assume that the decryption of this column is given as follows:

$$\text{Decrypt}_{Key}\ (\text{q39f, ge88}) = \text{djes}$$
$$\text{Decrypt}_{Key}\ (\text{c8x3, 8n4o}) = \text{ektd}$$
$$\text{Decrypt}_{Key}\ (\text{sk7x, x7wk}) = \text{3kw7}$$

After modifying DBMS Server's database, the Application Server will change the query for DBMS which will be as follows:

SELECT C1-IV, C1-Eq FROM DBMS_Table1 WHERE C1-Eq = djes;

Key1 is JOIN layer's encryption key for the C1 column of the DBMS_Table1. And also Key2 is DET layer's encryption key for the C1 column of the DBMS_Table1. Then encryption of a value will be generated by encrypting all layers until the current layer. For instance, encryption of the integer 1 will be generated as follows:

$$\text{Enc}_{Key2}\ (\text{Enc}_{Key1}\ (1)) = \text{djes}$$

In this part, we leak the information of the columns which are requested to check equality. If the data in the same column has the same value, the ciphertext of DBMS Server's database will be the same.

C. Queries with order leakage

Order preserving encryption is a difficult issue in private search mechanisms including CryptDB. In the original CryptDB's paper [10], the authors do not consider the order-preserving encryption in details, but later they highlight this issue in another article [11]. This part can be summarized as, if encrypted value of x is less than the encrypted value of y, then we know that the value x is less than y. If we find any encrypted value which is between these encrypted values, then the decrypted form is between x and y. This scheme leaks order, it is the weakest scheme. The queries "bigger than", "smaller than", ORDER BY, SORT, MAX, MIN can be performed with the encryption scheme. The implementation of the order-preserving encryption is not implemented until CryptDB, and also there is even no measure for the scheme's practicality. That is, CryptDB is the first implementation of order preserving encryption which uses Boldyreva's algorithm [12].

D. Queries with search functionality

Implementation of the Song's article is used for CryptDB. The queries with LIKE are done by using this scheme, but the search algorithm allows only to perform the full-word searches. In this section, we will explain the search algorithm of the Song [13]. Let's assume that we have lots of private documents and we have limited storage. Therefore, we can keep them on an untrusted server. This can also be applied to mail server as well, and we need to keep our personal e-mails there. For privacy and security reasons, the data stored on an untrusted party must be encrypted.

Without loss of generality, we will treat the words as N-bit strings. This is just an assumption that all words have the same length, smaller word are padded, and longer word can be divided into the N-bit blocks. While searching among the documents, we are interested in specific documents which contains some special words, a special keyword. If we have a low bandwidth we need to only download the requested files which contain our words instead of downloading all the files. Song's search algorithm is a good candidate for solving this issue.

Note that there two different search algorithms: index-based algorithm and sequential scan algorithm. Song's algorithm is based on the sequential scan algorithm. Using an index for searching big documents is generally faster than sequential scan. While reading data from a source, it is better to use index-based, but here storing and changing words is a hard

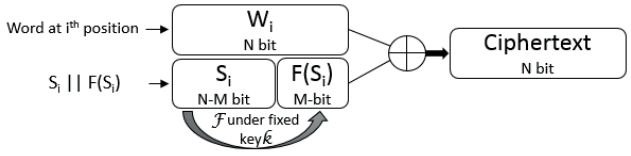

Fig. 4. Basic Scheme

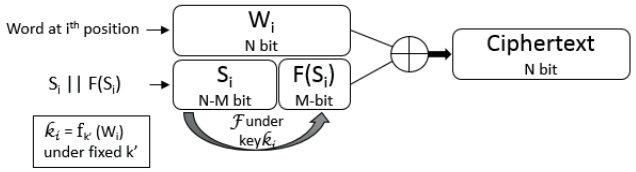

Fig. 5. Second Scheme

issue to consider. Index gives additional information, and this can lead to the statistical attacks. For these reasons, Song's algorithm prefers sequential scan algorithm.

1) Song's Search Algorithm on Encrypted Data: There exist four schemes in the Song's algorithm. First one is the basic scheme which is provably secure. However, there is no controlled searching and the searches are not hidden. To make the system more functional, Song presented three more schemes.

The basic scheme can be roughly described as follows: Let's assume that we search a word on encrypted documents on the untrusted party, but at the same time we do not want to reveal any information about the plaintext. The encryption mechanism of the basic scheme provides the provable secrecy. If you have a document which contains ℓ words of N-bits long, then you create a sequence of ℓ pseudorandom values of $N-M$ bits long. For encrypting an N-bit word at the i-th position, the corresponding sequence value S_i and $F(S_i)$ are concatenated where F is a secure pseudorandom function, i.e., $S_i \| F(S_i)$. This F function is secure pseudorandom function and it uses a key to encrypt. This key can be the same for all words inside the document, or it can be different for each location independently. This scheme is provable secure which means that the untrusted server cannot learn anything about the plaintext.

From the point of basic scheme, if we want to search a word, we give the word itself and keys of the locations which this word may appear. If we do not know anything about locations, we will give all keys. The untrusted party will take them, and XOR the ciphertext with the word. If the result has $S_i \| F(S_i)$ for some S, the matching occurs and the location of the word comes to us. If untrusted party does not know the key of a location, nothing will leak about that plaintext. However, knowing anything about locations causes all keys to be leaked, and this means all document is decrypted to the untrusted party.

Second scheme extends the basic scheme by supporting controlled searching. In this scheme, key generation is tied to another pseudorandom function G which takes the words under a random secret key k'. For each word, it creates the keys for the F function, shown as $k_i = f_{k'}(W_i))$ where W is a word inside the document. After encrypting the document, if we want to search a word, we will give the word and the key which is generated by using the word itself and our secret key k'. This scheme does not reveal any information about the locations which are not including our searching word. This provides controlled searching.

Until now, the word is given to the untrusted server as plaintext, in general this is not an accepted situation. Third scheme extends second scheme, and supports hidden searches. Before we search for a word, we encrypt each word with a deterministic algorithm like ECB (Electronic Code Book) mode encryption. The word we search is encrypted and also our key is generated by using the G function in the second scheme again. However, the key is dependent to the encrypted word, not to the original word anymore. The remaining mechanism is the same as second scheme.

The problem with the second and third scheme is that the decryption is not feasible. After trying concatenation of some S and the $F(S)$, and xoring them with a word, we encrypt the word. However, at the decryption part, to generate $F(S)$ from some S, we need to know the key of this specific word. And this key is dependent to the encrypted word. It is a contradiction that for decryption of an encrypted word requires the decrypted word itself. For this reason, some modification is done to the key generation function. Key is generated with $N-M$ bits of the encrypted word just as the length of S values. While decrypting the word, we use S and first $N-M$ bit of ciphertext to generate the first $N-M$ bit of the plaintext. With this information, we can generate the key, and using this key we find $F(S)$. And finally we find the last M bit of the plaintext by XORing the last M bit of the ciphertext and the $F(S)$ values. This final scheme is also provable secure just like the first scheme. Also we add query isolation feature, that is untrusted party just learns the search result, not additional information about the positions and plaintext forms of the words.

2) CryptDB's Encrypted Search: CryptDB uses the implementation of the Song's search algorithm. This algorithm performs full-word searches with LIKE queries. In CryptDB, repetition of the words are removed, and the words are permuted to provide more security in searching. However by comparing the number of RND ciphertext with SEARCH ciphertext, it is possible to know the number of the duplicated words. After that, the words are padded to the fixed N-bit length to encrypt according to Song's algorithm. The queries can be run as follows:

```
SELECT * FROM Students WHERE Name LIKE "% Alice %";
```

After this query come to Proxy Server, it will modify the query with the encryption of the string "Alice". The encryption is assumed as 98wu in the example.

```
SELECT * FROM DBMS_Table1 WHERE C2-Search;
        LIKE "% 98wu %";
```

One of the UDFs makes DBMS Server to check the matching of the ciphertext in the SEARCH column with the encryption of the string "Alice". The only leakage to DBMS

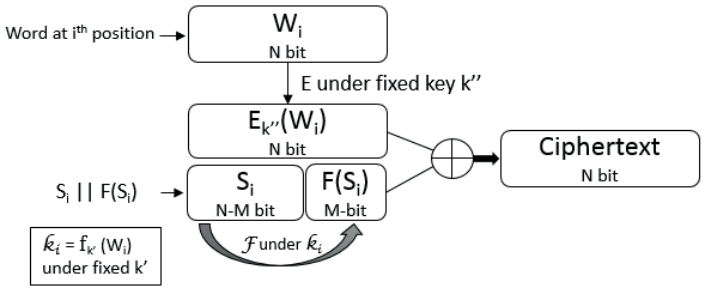

Fig. 6. Third Scheme

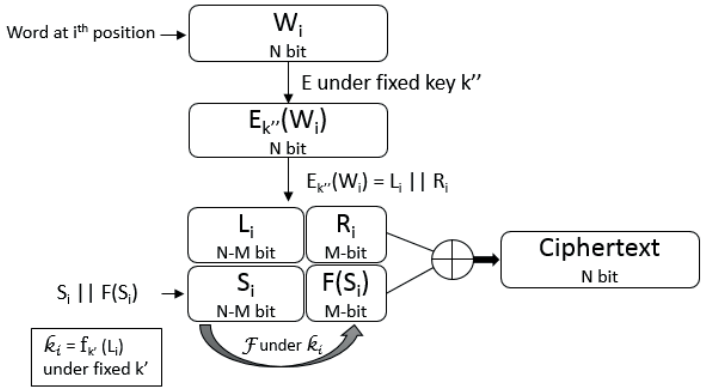

Fig. 7. Final Scheme

Server is that there is any matching with this full-word or not. Regular expressions and multiple words are not supported, just the full-word searches can be done with this algorithm, so this search part of CryptDB is not enough to meet our need for search queries.

TABLE IX
THE UPDATED TABLE WITH GRADE VALUES ADDED

StudentsWithGrades		
ID	**Name**	**Grade**
1	Alice	86
2	Bob	77
3	Eve	95

E. Queries with homomorphic addition functionality

From the point of the SQL queries, summation of the integer is used, not just for SUM queries, but also it is a part of counting average. The way of adding extra functionality to ciphertexts can be achieved by using homomorphic property. If one function is homomorphic, then multiplication of two ciphertexts is the encryption of the multiplication or addition of plaintexts of these two ciphertexts. For CryptDB, multiplication of the plaintext is not used from the point of the SQL queries, so additively homomorphic algorithm is necessary for this structure. Additively homomorphism can be demonstrated as follows:

m_1 and m_2 are the plaintext of the two values.

c_1 is the encryption of the m_1.

c_2 is the encryption of the m_2.

$c_1 \times c_2$ = Enc $(m_1 \times m_2)$ where Enc is an encryption function which has additively homomorphic property.

The homomorphic algorithm used in CryptDB is Paillier's algorithm. There is also Ge & Zdonik's algorithm. This algorithm, in general, aims to prevent the compromise of the database by processing queries on the ciphertext without decrypting them. It combines all the values of a row into one HOM ciphertext for each row. However it requires two times space overhead. For this reason, in CryptDB this algorithm is not implemented.

1) CryptDB's Homomorphic Property: In CryptDB, we keep the values as encrypted in the database. For this reason, homomorphic property is a solution for CryptDB. Queries with SUM and AVG are included in this part. We can create a query according to our example. We will update our table to make a meaningful query, and calculate the summation of the grades of the students. Our updated table is illustrated in Table IX.

Table IX is not kept inside the database as it is. Proxy Server encrypts the values for creating another table for DBMS Server. The created database for DBMS Server will be like in Table X.

Our example query can be like as follows:

```
SELECT SUM (grade) AS average FROM Students;
```

Proxy Server will modify the query, and send to DBMS Server. According to DBMS's Server's table, the modified

TABLE X
THE UPDATED TABLE WHICH IS CREATED ON PROXY SERVER'S SIDE

DBMS_Table2											
C1-IV	**C1-Eq**	**C1-Ord**	**C1-Hom**	**C2-IV**	**C2-Eq**	**C2-Ord**	**C2-Search**	**C3-IV**	**C3-Eq**	**C3-Ord**	**C3-Hom**
smsk	skwl	9dsi	32k0	cos8	qsaw	udb7	6sq8	wj9s	wus8	21do	qmdl
aksi	w8f9	v7sj	3ld0	sk0w	kd8s	lvmh	s99s	kcsk	dfsi	xnsk	ahc0
wipd	msuw	9s9k	389r	kduc	8sjf	k9sk	cnbs	bosm	xl9s	jwmd	h9dj

TABLE XI
AN EXAMPLE TABLE AND ITS ENCRYPTED FORM IN DET LAYER OF EQ ONION

Original_Table				**Encrypted_Table**			
ID	**Track-ID**	**Driver-Name**	**Location**	**col-1**	**col-2**	**col-3**	**col-4**
1005	1	Bob Marley	f.point	29ab74	b3ef12	ed34ef	12ada2
1006	1	Alice	g.point	6d5e5a	b3ef12	132ad3	27aa3b
1007	3	McDonald	g.point	b5aeb3	a214ba	3781e2	27aa3b
1008	3	Bob Marley	g.point	ae22ac	a214ba	ed34ef	27aa3b
1009	3	Bob Marley	a.point	8ebaba	a214ba	ed34ef	34ba9a

query will be as follows:

SELECT SUM (c3-Hom) AS average FROM DBMS_Table2;

After taking this SQL, DBMS Server will call the UDF which is responsible to calculate the summation of the values. This summation will be done by using homomorphic property. That is, the encrypted values of the C3-Hom column will be multiplied, and the result will be the encrypted format of the plaintexts.

Here, the values are $qmdl$, $ahc0$, and $h9dj$. Let's assume that the result of the multiplication is $a83g$.

$$(qmdl \times ahc0 \times h9dj) = a83g$$

Note that DBMS Server will not be able to know the values and the result because of encryption. The ciphertexts are multiplied which will result in a new fresh ciphertext. Proxy Server will take this encrypted result and will decrypt it. This decrypted value will be the the summation of the values inside the Grade column due to the underlying additively homomorphic property.

$$a83g = \mathsf{Enc}\ (86 + 77 + 95)$$

If Dec is the decryption function of the Enc which has additively homomorphic property, then the following equality will also be valid.

$$\mathsf{Dec}\ (a83g) = 86 + 77 + 95$$

Finally, the summation is done accurately on DBMS Server's side without revealing any information on the data itself. Namely, DBMS Server will not be able see the plaintexts, but it can still do the summation by only computing the multiplication over the ciphertexts.

IV. SECURITY & COMPLEXITY OF CRYPTDB AND FURTHER REMARKS

In this section, we are going to analyse the security and complexity of CryptDB and finally conclude our paper by giving further remarks.

CryptDB is open to frequency attack where the adversary knows the frequency of the plaintext. If the RND layer is decrypted to the DET layer in EQ onion, then the frequency attack is possible to apply because deterministic encryption is used in the DET layer. In this attack, the adversary that observes the queries can determine the ciphertext simply by looking at the results' row count. This attack can only be fixed by the RND layer, which has no usable functionality in practice.

For example, assume that we have left part of Table XI, and its encrypted form is the right part in Table XI. By using the knowledge of the frequency, one can learn the corresponding plaintexts from the right encrypted part in Table XI. This issue can be solved easily by using random IV based symmetric encryption, however this will prevent executing queries. The queries are always open in CryptDB which may leak information, too. By using the queries and their results, one can start to group the ciphertexts which is then possible to apply the frequency attack. We note that Private Information Retrieval (PIR) is required to eliminate this threat.

CryptDB is designed to move the database to cloud securely. The cloud is considered cheap for maintenance and administrative reasons. To evaluate the performance of CryptDB, a machine with two 2.4 GHz Intel Xeon E5620 4-core processors and 12 GB of RAM to run the MySQL 5.1.54 server, and a machine with eight 2.4 GHz AMD Opteron 8431 6-core processors and 64 GB of RAM to run the CryptDB proxy and the clients are used. [10] We believe that CryptDB may increase the cost significantly, i.e, one need an additional six core Proxy Server. On the other hand, the security of Proxy Servers must be ensured. This rises another administrative and security cost to the system.

To evaluate the performance, the assumptions are that CryptDB is trained on query set, i.e., onion adjustment is not happened during the test. The TPC-C Throughput is shown in following figure. The hardest hit is at the SUM and incremented UPDATE, the reason is they are using HOM onion. The overall throughput reduction is 26%. As average time, 0.60 ms is added to a query by Proxy Server. [10]

Deployment is another problem in CryptDB. Namely, if

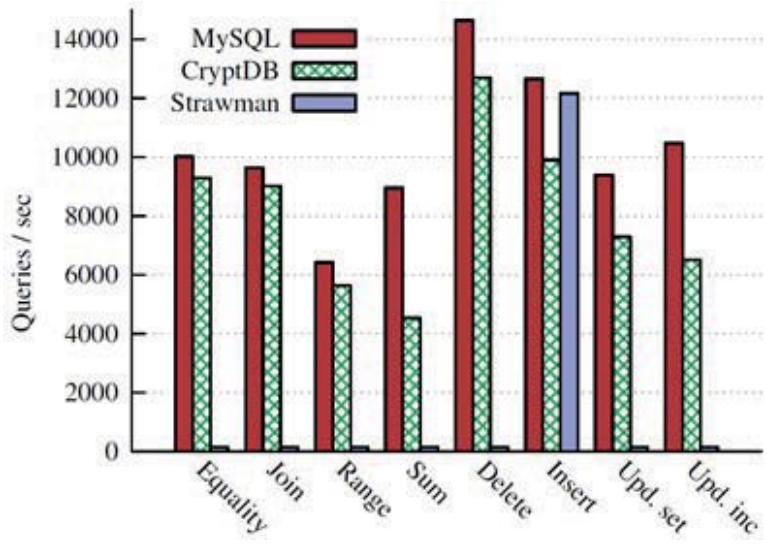

Fig. 8. TPC-C Throughput when CryptDB is trained on query set [10]

there is a bug at the production stage, debugging will be hard because of the encrypted values in the database. Therefore, in the real scenario, the companies will not be interested in using CryptDB.

Reducing the cost of Proxy Server is important in order to use CryptDB. Otherwise, CryptDB will only be useful for government agencies. Eliminating the frequency attack and hiding the queries must be taken into consideration in order to make the CryptDB more secure.

V. Conclusion

In this paper, we first explained CryptDB in a detailed way. CryptDB is the first practical Database Management System for running most standard queries on encrypted data. It does not make any changes to the DBMS. We revisited the server structure of CryptDB and pointed out the large overhead of the Proxy Server. We next explained the search algorithm of CryptDB which is based on Song's algorithm with the inabilities. We showed that the current search algorithm is not enough to meet all the search queries. We give a detailed analysis about the efficiency and security aspects. With some modification to the current form, the system can be more secure and can provide all necessary functionalities in order to execute all possible queries.

References

[1] NIST SP 800-145. The NIST Definition of Cloud Computing, National Institute of Standards and Technology. Retrieved 24 July 2011. URL: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf .

[2] G. Rosen. Amazon usage estimates and updates, October 2009, URL: http://www.jackofallclouds.com/.

[3] Yinqian Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In Proceedings of the 2012 ACM conference on Computer and communications security (CCS '12). ACM, New York, NY, USA, 305-316.

[4] Cloud Security Alliance URL: https://cloudsecurityalliance.org .

[5] Marcos Dias de Assuncao, Alexandre di Costanzo, and Rajkumar Buyya. 2009. Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. In Proceedings of the 18th ACM international symposium on High performance distributed computing (HPDC '09).

[6] J. Brodkin. Gartner: Seven cloud-computing security risks. Infoworld, 2008.

[7] Cloud Computing Security Considerations, A Microsoft Perspective, Microsoft Whitepaper, 2010, http://www.microsoft.com/malaysia/ea/whitepapers.aspx.

[8] Cloud Computing: Benefits, Risks and Recommendations for Information Security, ENISA Report, 2009, http://www.enisa.europa.eu/act/rm/files/deliverables/cloud-computing-riskassessment.

[9] Security Guidance for Critical Areas of Focus in Cloud Computing V2.1, Cloud Security Alliance (CSA) Report, 2009, http://www.cloudsecurityalliance.org/csaguide.pdf. P. Mell and T. Grance. The NIST definition of cloud computing. National Institute of Standards and Technology, 53(6), 2009.

[10] CryptDB: Protecting Confidentiality with Encrypted Query Processing. In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011), Cascais, Portugal, October 2011.

[11] Raluca Ada Popa, Frank H Li, Nickolai Zeldovich. An Ideal-Security Protocol for Order-Preserving Encoding. Symp on Security and Privacy, 2013.

[12] Order-preserving symmetric encryption. A. Boldyreva, N. Chenette, Y. Lee and A. O'Neill. In Eurocrypt '09, pp. 224-241. Springer, 2009.

[13] Practical techniques for searches on encrypted data. D. X. Song, D. Wagner, and A. Perrig. In Proceedings of the 21st IEEE Symposium on Security and Privacy, Oakland, CA, May 2000.

[14] C. Gentry. Fully homomorphic encryption using ideal lattices. In Proceedings of the 41st ACM Symposium on Theory of Computing STOC 2009, pages 16978. ACM, 2009.

[15] Marko Hölbl, Cloud Computing Security and Privacy Issues, The Council of European Professional Informatics Societies (CEPIS), 15.03.2011. URL: http://www.cepis.org/media/CEPIS_Cloud_Computing_Security_v17.11.pdf .

[16] A.Greenberg. DARPA will spend 20 million to search for crypto's Holy Grail. Forbes, April 2011.

[17] Dan Boneh, Eyal Kushilevitz, Rafail Ostrovsky, William Skeith, Public Key Encryption That Allows PIR Queries. Preliminary version appeared in CRYPTO 2007: 50-67.

[18] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable Symmetric Encryption: Improved Definitions and Efficient Constructions, Journal of Computer Security, 2011.

[19] Andrew C. Yao, Protocols for Secure Computations, IEEE, 1982.

[20] López-Alt, Adriana and Tromer, Eran and Vaikuntanathan, Vinod, On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption, Proceedings of the 44th symposium on Theory of Computing, 2012.

[21] Damgård, Ivan and Zakarias, Sarah, Constant-Overhead Secure Compu-

tation of Boolean Circuits using Preprocessing, Theory of Cryptography, 2013.
[22] Damgård, Ivan and Pastro, Valerio and Smart, Nigel and Zakarias, Sarah, Multiparty Computation from Somewhat Homomorphic Encryption, Advances in Cryptology CRYPTO 2012.
[23] Ivan Damgård, Sebastian Faust, Carmit Hazay, Secure Two-Party Computation with Low Communication, TCC, 2012.
[24] Chen, Hao and Cramer, Ronald, Algebraic Geometric Secret Sharing Schemes and Secure Multi-Party Computations over Small Fields, Advances in Cryptology - CRYPTO 2006.
[25] P. Paillier. Public key cryptosystems based on composite degree residuosity classes. In Proc. of Eurocrypt 99, volume 1592 of LNCS, pages 223238. IACR, Springer-Verlag, 1999.
[26] T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. IEEE Transactions on Information Theory, IT-31(4):469472, 1985..
[27] J. A. Halderman, S. D. Schoen, N. Heninger, W. Clarkson, W. Paul, J. A. Calandrino, A. J. Feldman, J. Appelbaum, and E. W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Proceedings of the 17th Usenix Security Symposium, San Jose, CA, JulyAugust 2008.

Ziynet Nesibe Dayioglu completed her bachelor studies at the department of computer engineering, Fatih University in İstabul, Turkey in 2012. She also completed her double major at the department of mathematics in Fatih University. Following the graduation, she started her masters studies at the department of computer engineering, Fatih University. Since February 2013, she has been working at the cryptology department of TUBITAK BILGEM as a researcher.

Mehmet Sabir Kiraz finished his bachelor studies at the department of mathematics, Middle East Technical University (METU) in Ankara, Turkey in 2000. Following the graduation, he worked at Pamukbank and Yapi Kredi Bank as software analyst and software programmer respectively. In October 2002, he started his masters studies at the International Max-Planck Institute for Computer Science in Saarbrücken, Germany and finished at the end of 2003 with a specialization on security protocols. From March 2004 to March 2008, he was a Ph.D. student at the Coding & Crypto group of the Technische Universiteit Eindhoven, under the supervision of prof. Henk van Tilborg and prof. Berry Schoenmakers. After his Ph.D. studies he worked at Phili, Eindhoven and TomTom, Amsterdam. Since 2010, he has been working as a researcher at the cryptology department of TUBITAK BILGEM.

Fatih Birinci received his Bachelor's degree and Master's degrees from the department of Mathematics, METU in 1995 and 1998 respectively. He also received Master's degree from department of Computer Engineering, Gebze Institute of Technology, in 2002. He has been working in TUBITAK BILGEM since 1997.

Ihsan Haluk Akin finished his bachelor studies at the department of mathematics, METU in Ankara, Turkey in 1999. He finished Master's degree at Sabanci University, Turkey in 2002. He finished his Ph.D. at METU in 2009. He worked at several places including TUBITAK BILGEM between 2000 and 2010. He worked at Fatih University between 2010 and 2013. Currently he is working as postdoc at Worcester Polytechnic Institute.