Journal of Computer Science

Original Research Paper

A New Database Encryption Model Based on Encryption Classes

Karim El Bouchti, Soumia Ziti, Fouzia Omary and Nassim Kharmoum

Department of Computer Science, Intelligent Processing Systems & Security Team, Faculty of Sciences, Mohammed V University in Rabat, Morocco

Article history
Received: 30-03-2019
Revised: 17-05-2019
Accepted: 25-06-2019

Karim El Bouchti
Department of Computer Science, Intelligent Processing Systems and Security Team, Faculty of Sciences, Mohammed V University in Rabat, Morocco

E-mail: elbouchtikarim@gmail.com

Abstract: Data encryption is one of the advanced measures used to consolidate data security inside Databases. It is an essential technique to protect data against theft, disclosure or modification from different typologies of attacks. In this work, we will propose a new database encryption model. It is based on a novel concept called "Encryption Classes". The proposed model is full compared to the existing models; it integrates many security mechanisms starting from the keys generation and their protection until the data encryption. Furthermore, our proposed model performed a new feature which is the Database structure encryption.

Keywords: Database Security, Database Security Model, Data Security, Computer Security, Database Encryption Model, Database Encryption

Introduction

Recently, the security of databases (DB) has been subject of multiple studies and researches conducted by the computer security worldwide community; it aims to protect sensitive data stored in centralized or distributed DB against attacks from malicious entities (Elbouchti *et al*., 2018). The conventional solutions and mechanisms to secure DB are based on the implementation of three security levels: 1/-Physical security level, 2/-Operating system security level, 3/- Database Management System (DBMS) security level (Elovici *et al*., 2018). The access control mechanism implemented inside the DBMS level is considered as a strong means that controls the access of subjects (users) to DB objects; it includes identification, authentification and auditing. Although this mechanism consolidates the DB security, it doesn't protect against administrator attacks. Furthermore, it is ineffective against DBMS bugs and physical access attacks as theft or destruction of data (Elbouchti *et al*., 2018; Jacob, 2012).

DB encryption is one of the necessary security measures to implement beside the access control mechanism. It can be implemented on several levels: application, DBMS and hard disk (Shmueli *et al*., 2010; Mattsson, 2005). For example, in a cloud, encryption is a crucial operation that ensures confidentiality and integrity of data exported to external service providers. Several architectures using encryption inside clouds have been developed (Ma *et al*., 2018; Chen *et al*., 2018)

The DBs encryption is based on developing efficient encryption models. A relevant model must meet criteria such as encryption granularity level, efficient encryption keys management and high performance (Shmueli *et al*., 2014; Elovici *et al*., 2018). In the literature, several DB encryption models have been proposed. The most relevant is the work developed by Elovici *et al*. (2018) and Shmueli *et al*. (2014). They suggested a model that protects data confidentiality and integrity using the DB cell coordinates. In order to improve the performance, they implemented this model in the DBMS just above the DB cache. Their approach uses a single encryption key with a single algorithm. Also, Sesay *et al*. (2005) presented another model that classifies data and users according to categories. The level of encryption and keys generation is determined according to these categories. This approach uses a single encryption algorithm and the encryption keys are generated from a master key which is stored in a tamper-proof controller. Another multi-level DB model with its prototype has been proposed by Sallam *et al*. (2012). It allows the insertion of an encryption system at the top of the multi-level DB. This solution will reduce the DB size and improve queries performance.

Any DB encryption model is limited by the policy of generation and protection of the encryption keys. Getting the encryption keys is the ultimate goal of any attacker. Besides the standard keys protection approaches presented in (Bouganim and Guo, 2009; Jacob, 2012), we have proposed in our previous published work, new models to protect encryption keys according to encryption granularity level (El Bouchti *et al.*, 2018). In another work, we have proposed another model to protect keys against internal attacks when the encryption is performed at the application level (El Bouchti *et al.*, 2019).

Despite the efforts made by researchers to improve DB encryption models, these latters still need to be developed to provide more security. Developing a model meeting the three criteria that were raised at the beginning of this introduction is not enough. It must be accompanied by other mechanisms and features which strengthens more data security, such as the use of multiple algorithms to encrypt data, create a method to encrypt data with multiple key values according to a level of encryption granularity (cell, column, ...) or according to data sensitivity level (public, secret, top secret, ...). In addition, it must include an approach of keys protection within DB server to avoid costs and risks associated with managing this protection outside the server (Itamar *et al.*, 2018; El Bouchti *et al.*, 2018). Unfortunately, the combination of all these elements in a single encryption model was not addressed inside works that we presented previously as well as in other works not mentioned in this article.

In this context, our work aims to propose a new DB encryption model called "Full Encryption Model". Our model offers a complete set of security mechanisms to data compared to models proposed in the literature. It is based on the use of a novel concept called "Encryption Classes". The "Full Encryption Model" is composed of four models: The data encryption model, the DB structure encryption model, the encryption keys generation model and the Master Key generation model. Our model encrypts data according to a classification of the sensitive columns. This latter allows the encryption/decryption only of these columns when queries execution. This concept will improve extremely the queries performance as well as offers perfect optimization of the DB size. Comparing to other existing models, the "Full Encryption Model" allows encrypting data with multiple algorithms and encryption keys according to data sensitivity. In addition, the protection of encryption keys is performed by their encryption using multiple other keys called "Master Keys".

Our work will be organized as follows: section 2 consists of giving a general overview of DB attacks models. We will also demonstrate the utility to encrypt DB structure. Section 3 describes the «Full Encryption Model» and explains the role of each model that composes it. In section 4, we present and discuss the results of tests made on all models described in previous sections.

Database Encryption Preliminaries

Database Models Attacks

We can classify attacks compromising a DB in three types (El Bouchti *et al.*, 2018; Shmueli *et al.*, 2010):

- **Internal attack:** This attack is made by a person who belongs to the group of trusted users of the DB. He may try to obtain information beyond his access rights (El Bouchti *et al.*, 2019)
- **External attack:** This attack is done by a person outside the DB's users group. In this case, this person has access to a computer system and attempt to extract sensitive information (Zabihimayvan and Doran, 2019)
- **Administrator attack:** This attack is performed by the DB administrator. He has specific rights and super privileges on it. This allows him to extract or alter sensitive and valuable information (Priebe *et al.*, 2018)

The strategies and means deployed by attackers are multiple because they can be located at any level of the three typologies level already discussed. In many cases, the attacker's strategies may exceed the forecasts of the DB security administrators. Thus, we can also distinguish between:

- **Direct Attacks:** If the attacker has physical access to the data, then the mechanisms of access controls are useless and he can attack stored data. The theft or removal of DB support is just an example of this attack type
- **Indirect attack:** The attacker can access to the DB data dictionary and retrieve some information relating to columns, tables and DB structure scheme, in order to deduct statistics to estimate data distributions (Liu and Gai, 2008)
- **Memory attack:** It is the most sophisticated DB attack. The attacker can directly access the DB server memory via advanced tools and steal some data or encryption keys

The Database Protection using Encryption

It is possible to implement four levels of security to ensure the DB security (El Bouchti *et al.*, 2018; Elovici *et al.*, 2019):

- Physical security level
- Operating system security level
- Database Management System security level
- Data encryption level

The interest of the first three security levels is evident. The description of their utilities is not the subject of this work. The implementation of the fourth level (data encryption) offered by the DBMS is essential, especially in the case of internal or administrator attacks. Currently, the commercial DBMSs offer efficient solutions for sensitive data encryption at rest, using symmetric encryption algorithms such as AES, DES, 3DES ... These algorithms differ in their complexity levels and the size of the encryption key they use (Dixit *et al.*, 2018). The DB encryption can be consolidated by the implementation of a new layer of security that can play an exciting role which is the protection of DB structure confidentiality.

Database Structure Protection

Protecting the DB structure (table and column names) using encryption is a way to secure the DB in addition to the data values stored inside columns. An attacker could use information from the DB structure to conduct attacks that aim to guess the data distributions.

DB structure encryption might add further security to a DB and increase the complexity of an attacker to decrypt its data. In some attacks like SQL injection (El Bouchti *et al.*, 2018; Lee *et al.*, 2012), an attacker may be interested in attacking only specific data of some tables or columns such as customers table or column of the payment cards numbers, for example. Adopting a DB structure in clear and whatever its size, small or large in number of tables or columns and even though the encryption concerns only the data, an attacker can easily focus on one or more columns to decrypt them. On the other hand, if the structure is encrypted too (case of a DB containing 1000 tables for example), the complexity of the attack is multiplied by 1000. In a DB with encrypted structure, as the number of tables and columns increases, the longer the complexity to attack it, becomes enormously high.

In addition, companies that produce business software face significant challenges to establish and implement specific DB designs adapted for complex business areas. After a major modeling and design efforts, the resulted DB structure is considered an original work and a company secret which must be protected and not disclosed, hence the possibility to protect it can be made in two ways: 1/ - legal protection by copyright (DECJ, 2013), 2/ - structure encryption and that before it goes reproduced or hacked by competitors.

Feature of an Effective Database Encryption Model

The big challenge of a DB encryption model is its ability to take into consideration the elements mentioned below:

a. **Encryption granularity:** The granularity level of encryption is considered a fundamental problem in DB encryption (Shmueli *et al.*, 2010). The most relevant levels where encryption must be implemented are cell, record, table and page. A relevant encryption granularity level should bring to the DB security the following advantages:

   - It is necessary to encrypt only the sensitive data and leave the insensitive data in clear. This implies that the DBMS encrypts and decrypts only sensitive data when executing a user's query
   - An encryption granularity must ensure extreme security of data; the breaking of the encryption has to be impossible
   - Encrypting sensitive data with a single key doesn't provide a high security level; it can be catastrophic, even if it is associated with an access control mechanism. If an attacker gets encryption keys, he can decrypt everything without leaving any traces. A relevant encryption level must provide the ability to encrypt data inside DB with multiple key values
   - If the granularity level is not properly implemented, it may cause serious DB security problems as data leakage or unauthorized modification
   - An encryption granularity must ensure independence between DB record decryption and other records

b. **Encryption keys management**: The management of encryption keys is a fundamental point in any DB encryption model. It defines the method whose keys are generated, stored and protected during their generations until destructions (Galushka *et al.*, 2018; Bouganim and Guo, 2009; Mattsson, 2005). An effective encryption keys management in a DB encryption model must take into consideration the following elements:

   - It must define a secure method of encryption keys generation
   - It must give the possibility to use several values of keys to encrypt data according to their encryption granularity level (cell, column, line ...), or according to the data sensitivity level (secret data, top-secret data, confidential data....)
   - It must define a method to protect DB encryption keys against attackers

c. **Performance:** The deployment of encryption inside DB generates a calculation overhead that influences the DBMS performance and impacts the user's queries automatically. The first step that we should consider when designing an encryption model is to adopt a selective encryption strategy, i.e., encrypt sensitive data and leave insensitive data in clear. Another important factor is to minimize the time of encryption/decryption by using just a single operation when encryption/decryption.

d. **Database Size:** The encrypted DB shouldn't be too large compared to the original DB.

e. **Influence in the DBMS architecture:** The model wouldn't generate any changes in the internal architecture or DBMS functionalities. A new implementation must keep the internal DBMS functionalities intact (example: Index, primary and foreign key…).

Description of our "Full Encryption Model"

The "Full Encryption Model" is a new model that we propose to encrypt DBs; it is composed of the following 4 models:

1. The Data Encryption Model
2. The DB Structure Encryption Model
3. The Encryption Keys Generation Model
4. The Master Key Generation Model

In this part, we will begin by defining the implementation of our "Full Encryption Model". Afterwards, we will introduce the notion of "encryption classes" on which our model is based and finally, we will describe in details the four models of the "Full Encryption Model".

The "Full Encryption Model" Implementation

We propose to implement the four models of the "Full Encryption Model" at the DBMS level, precisely inside the Database Management System Layer as shown in Fig. 1. All operations performed via these models are performed in the following three blocks of Fig. 1.

The "Data Encryption Module" Block: This block implements the model (1). It performs the data encryption/decryption. It communicates directly with the "Keys Generation" block which provides the encryption keys necessary during each encryption/decryption operation.

The "Data Base Structure Encryption" Block: This block implements the model (2). It performs the DB structure encryption. It communicates with the "Keys Generation" block which provides the keys needed to encrypt the DB structure.

The "Keys Generation" Block: This block implements the model (3) and (4). It is in charge of generating the keys to encrypt data and DB structure.

As illustrated in Fig. 1, the Discretionary Access Control (DAC) is the mechanism that allows the creator of DB objects, to define the access policy on these objects. The creators may delegate some permission on these objects to other users.

The "Encryption Classes"

The Encryption Class represents a combination of parameterable arguments which are used to encrypt (the data or column name) of a column or several columns in a DB. It has the form of the following vector:

$$Class(i)\begin{pmatrix} Id\_Class(i),\ Key\_Class(i), \\ Algorithm(i),\ Sensitivity(i) \end{pmatrix}$$

Where:

Id_class(*i*)	= Represents the identification of the *Class*(*i*); it is coded on 4 digits
Key_class(*i*)	= The key used to generate the data encryption key or for column names
Algorithm(*i*)	= Algorithm used for encryption
Sensitivity(*i*)	= The sensitivity degree of the data or for column names

Example:

Class(1)('0001','Mycolor012345678has@ml@p', AES192, 'Confidential').
Class(2)('0002','@mysonmyson@12345678@mydaugther@', 'AES256', 'Secret')

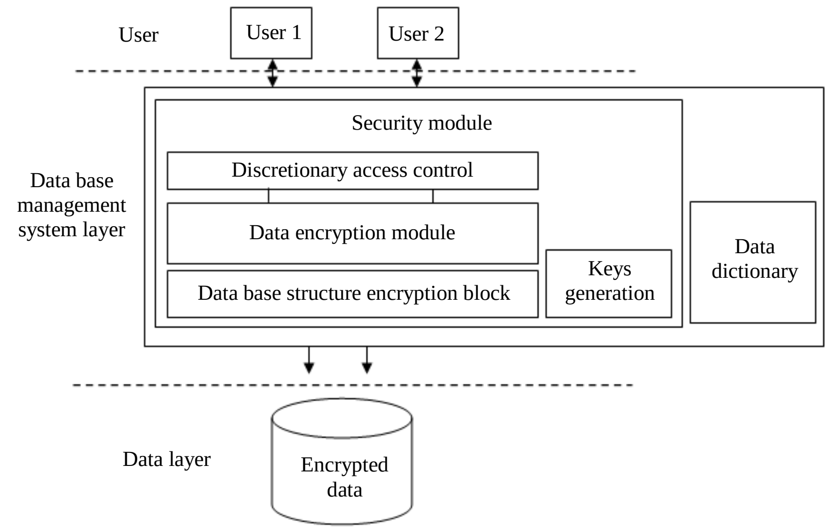

Fig. 1: Implementation of the "Full Encryption Model" inside DBMS

Model for Defining an Encryption via a Class

The definition of an encryption on a table in our model is done on two elements:

- The sensitive columns data
- The sensitive columns names

Assuming a table created in a DB defined as: R1 (COL(i), COL(i+1), COL (i+2),…………,COL(n)) and let's take the 2 encryption classes: Class(x) and Class(y) assuming COL(i) as a sensitive column:

a. Model for defining encryption on column data
The definition of an encryption on the column data COL(i) follows the model:

$$COL(i)\,Type\ encrypt\ with\ Class(x) \qquad \text{(A)}$$

b. Model for defining encryption on column name
The definition of an encryption on the column name COL(i) follows the model:

$$Struct(R1.COL(i))\,encrypt\ with\ Class(y) \qquad \text{(B)}$$

Example:

Assuming a table defined in a DB as R1 (COL1, COL2, COL3).

We suppose that COL(2) and COL(3) as a sensitive columns.

Let consider also four encryption classes: Class(1), Class(2), Class(3), Class(4) defined as:

Class(1)('0001','Bigstorm1234567812345678',AES192, 'Confidential').
Class(2)('0002','Myuniquekey@12345678@mysecretkey ', 'AES256', 'Secret').
Class(3)('0003','ilove@DBsecurity','AES128','Confidential').
Class(4)('0004','&@I@encrypt@my@sensitive@data1 @&', 'AES256', 'Secret')

We define the encryption (A), (B), (C) and (D) on the table R1 as below:

A. COL2 varchar2(100) encrypt with Class(1)
B. Struct (R1.COL2) encrypt with Class(3)
C. COL3 varchar2(100) encrypt with Class(2)
D. Struct (R1.COL3) encrypt with Class(4)

In (A), an encryption has been defined on the sensitive data of the column COL2 using Class (1). That's mean that these data have a confidential nature. Thus, they will be encrypted with the algorithm AES192 with an encryption key that will be generated basing on the key k = 'Bigstorm1234567812345678'. Similarly, in (C) an encryption has been defined on the data of column COL3 which have a secret nature. The data will be encrypted with the AES256 algorithm with a key that will be generated on the base of the key k = 'Myuniquekey @ 12345678 @'.

In (B), we have defined an encryption on the column name of COL2 using the Class(3). It means that this column name is classified in the "Confidential" sensitivity degree. It will be encrypted with the algorithm AES128 with a key that will be generated basing on k = 'ilove @ DBsecurity' '. Likewise, in (D) we have defined an encryption on the column name of COL3 using Class(4). This column name is classified in the "Secret" sensitivity degree". It will be encrypted with the algorithm AES256 with a key that will be generated using k = '& @ I @ encrypt @ my @ sensitive @ data1 @ & '. The result of these operations is shown in Tables 1 and 2.

The Class management is an operation controlled by the DB administrator; it is subject to the following rules:

- Each Class(i) has a unique Id_class(i) and a unique Key_class(i)
- Each Class(i) can be used to encrypt data in one or more columns. Identically, each Class(i) can be used to encrypt a column name or multiple columns names
- Defining an encryption on the data of column COL(i) using Class(i) can't work if we didn't define an encryption on the column name of this latter using either Class(i) or another Class(j) and vice versa
- The DB administrator manages the creation, updating, attribution or the revocation of the classes to columns according to the security requirements. This management is controlled inside the DBMS by the Discretionary Control Mechanism (DAC) as shown in the above Fig. 1

Table 1: Table (R) before encryption

COL1	COL2	COL3
1000	Dupont	PW@124
2482	James	PW@884

Table 2: Table (R) after encryption

COL1	C7E220367559EE77B5221D27B92AE495	3F99A273812785D025100177BBDFA307
1000	490A39A1BBA9888DCB1DD4158D5975F9	4DDF495C731928AFA79EC598CDB31E2
2482	2091BF0941B816B149654605D33A4685	9A2B6B09120DCB98B56FFF0233C27270

Encryption/Decryption in the Data Encryption Model

Inside the DB, our Data Encryption Model encrypts/decrypts data at the table cell level. In other words, we adopt an encryption granularity at the cells level. Each cell value in a column is encrypted differently from the others.

When inserting in a sensitive column, plaintext values are encrypted according to the model defined below:

$$E\left(Data1\right) = E\,(K_{COL(i),}Data1 \parallel P\,((rownum+1) + Column\_id + Table\_id + Id\_class(i)) \quad (3.1)$$

Let consider X a ciphertext value as defined below:

$$E\left(Data1\right) = X$$

When consulting sensitive columns, ciphertext values are decrypted according to the model defined below:

$$D(X) = Rep\left(D\left(K_{COL(i)}, X\right), P\begin{pmatrix}(rownum) + Column\_id + \\ Table\_id + Id\_class(i)\end{pmatrix}\right) \quad (3.2)$$

In the model (3.1) and (3.2), we have:

- E: A symmetric encryption algorithm which encrypts a clear value
- D: A symmetric decryption algorithm which decrypts a ciphertext value
- Data: A plaintext value to be encrypted. It is located in a sensitive column COL(i)
- X: A ciphertext value to be decrypted. It is located in a sensitive column COL(i)
- $K_{COL(i)}$: The encryption key of column COL(i)
- Rownum: It's an integer that represents the last record in a table
- Column_id: It's a unique integer that represents the identifier of the column COL(i)
- Table_id: It's a unique integer that represents the identifier of the table where COL(i) is located
- Id_class(i)): It's the integer that represents the identifier of the class (Class(i)) assigned to encrypt the data of the column COL(i)
- Rep(str1, str2): function that cancels the (str2) string which exists in the string (str1).

P: polynomial function which is defined as:

$$P : N \rightarrow N$$

$$P(n) = a_0n^0 + a_1n^1 + a_2n^2 + a_3n^3 + \ldots\ldots\ldots\ldots\; a_nn^n \quad (4)$$

The coefficients of the polynomial (a0, a1, a2,......, an) are integers. They are fixed in the encryption/decryption general algorithm which executes the models (3.1) and (3.2). Indeed, data encryption/decryption in the Data Encryption Model can be modified by changing these coefficients. In the model (3.1), the value delivered by P ((rownum+)+Column_id+Table_id+Id_class(i)) is unique as (rownum) changes from a cell to another. Therefore, all equal cells values are encrypted differently from each other. This technique increases extremely the encryption security and avoids the probability of frequency analysis attacks (Elovici *et al*., 2018).

The Database Structure Encryption Model

The DB structure encryption is a way to protect its confidentiality by making incomprehensible its real structure. Our "Full Encryption Model" adopts this concept in order to exploit it to take benefit from two major advantages:

- Encrypt the DB structure to take benefit from all the advantages mentioned above
- Beside data encryption, implement another level of security or control that must be crossed before accessing the data (during encryption or decryption). The following rule is adopted by our "Full Encryption Model": A user who owns a class that encrypts/decrypts data in a column can't access to data without having the class that encrypts/decrypts the name of that column. The execution of (A) and (B) is mandatory to define encryption on a sensitive column

```
/* Definition of an encryption on the data of COL(i)
COL(i) Type encrypt with Class(x);     (A)
```

```
/* Definition of an encryption on the name of COL(i)*/
Struct (R1.COL(i)) encrypt with Class(y);    (B)
```

Assuming a table in a DB that has the following structure:

R(i) (COL(i), COL(i+1), COL(i+2), COL(i+3))

Let consider the two encryption classes, Class(i) and Class(j):

Class(i)(id_class(i), key_class(i), Algorithm(i), Sensitivity(i))
Class(j)(id_class(j), key_class(j), Algorithm(j), Sensitivity(j))

Assuming both encryption definitions on the names of the two sensitive columns COL(i+1) et COL(i+3):

Struct R(i).COL(i+1) encrypt with Class(i);
Struct R(i).COL(i+3) encrypt with Class(j);

The encrypted structure of R(i) follows the model:

$$R(i)\begin{pmatrix} COL(i),\ H\left(E\left(COL(i+1)\right)\right), \\ COL(i+2),\ H\left(E\left(COL(i+3)\right)\right) \end{pmatrix} \quad (5)$$

Where:

H = Hash function.
E = A symmetric encryption algorithm

Encryption Keys Generation Model

The "Keys Generation" block generates two key types. The keys for encrypting the columns data and the ones for encrypting the columns names. Both of these key types are generated following the models defined below:

$$K_{COL(i)} = H\ \left(Table\_id \,\|\, Column\_Id \,\|\, key\_class(i)\right)$$

$$K_{Struct\ (COL(i))} = H\ \left(Table\_id \,\|\, Column\_Id \,\|\, key\_class(i)\right)$$

Where:

- $K_{COL(i)}$: The encryption key for data of the column COL(i)
- $K_{Struct(COL(i))}$: The encryption key for name of the column *COL*(*i*)
- H: A hash function
- Table_Id: A unique integer that represents the identifier of the table where *COL*(*i*) is located
- *Column_id*: A unique integer that represents the dentifier of the column *COL*(*i*)
- *key_class*(*i*)): The key defines in the class that is assigned to encrypt the (name or data) of the column *COL*(*i*). This key is unique for each class

The models (6) and (7) dedicated to generate the keys are identical. Inside a DB table, if *COL*(*i*) and *Struct*(*COL*(*i*)) are encrypted using distinct Classes, we will have different values of $K_{COL(i)}$ and $K_{Struct\ (COL(i))}$. On the other hand, if *COL*(*i*) and *Struct*(*COL*(*i*)) are encrypted using the same Class, we will have $K_{COL(i)}= K\ Struct_{(COL(i))}$. Two sensitive columns *COL*(*i*) and *COL*(*ii*) that belong to two different tables will never be encrypted by the same keys even if they use the same class. It is the same case if they belong to the same table.

The « Master Key » Generation Model

The security of sensitive data encrypted in a DB depends on the protection of the encryption keys. It is a fundamental process for the global DBMS security (Itamar *et al*., 2018; El bouchti *et al*., 2018). We propose to protect the encryption keys of our "Full Encryption Model" via their encryptions with a Master Key (Km). The generation of the Master Key follows the model:

$$Km_{(COL(i))} = H\ \left(Table\_name\ \|\ COL(i)\right) \quad (8)$$

Where:

H	=	Hash function
COL(*i*)	=	The name of the sensitive column
Table _name	=	The name of table which contains the column *COL*(*i*)

The Km value is unique for each sensitive column *COL*(*i*). Its generation is dedicated to protect only the encryption keys for data in our "Full Encryption Model".

Results and Discussion

In this part, we will present an analytical comparison between our "Full Encryption Model" and two other models. The first one is the model described by Sesay *et al*. (2005), whereas the second model has been proposed by Shmueli *et al*. (2014). They seem to be the most relevant models in the literature in terms of providing maximum security.

The comparative Tables 3 and 4 show the result of the comparison.

In order to concretize the functioning of the "Full Encryption Model" and to conduct an objective discussion of the comparison result analysis, an implementation of a real case will be jointly as presented below.

Case Study

Let consider the table "employee" in a DB1 which has the following sensitive columns: First_name, Last_name, Salary (Table 5). Assuming that all these columns own the same data sensitivity level and also the same sensitivity level of the columns names. They will be all encrypted using Class (1) which is defined as:

Class (1)('0001','ilove@DBsecurity','AES256', 'Confidential')

After defining the encryption on the sensitive columns using the models (A) and (B), the table "employee" will own a new structure after encryption using the models (3.1) and (5) as illustrated by Table 6.

The data encryption keys and their protection keys are generated by the (6) and (8) models as shown in table 7, whereas the next table 8 shows the encryption keys of the columns names. These keys have been generated by the (7) model.

Table 3: Comparison (1)

	Algorithms uses	Keys (Uses/Generation)	Data Sensitivity Level
Full Encryption Model	• The model uses several Algorithms to encrypt data • The model uses several Algorithms to encrypt the DB structure • The model allows the possibility to implement other encryption algorithms besides the usual encryption algorithms such as (DES, AES,…..)	• The model generates and uses multiple encryption keys values : the keys for encrypting the columns data and the keys for encrypting the columns names	• The model defines multiple levels of data sensitivity • The model allows creating other levels of data sensitivity • The model allows defining finer levels from a single level of data sensitivity
(Sesay *et al*., 2005)	• The model uses a single algorithm to encrypt data	• The model uses a single key to encrypt "Classified" sensitivity data and multiple keys to encrypt "Private" sensitivity data	• The model defines three level of data sensitivity (Unclassified, Classified, and Private) • The model doesn't allow creating other level of data sensitivity
(Shmueli *et al*., 2014)	• The model uses a single algorithm to encrypt data	• The data encryption uses a single key value.	• The model defines one level of data sensitivity. • The model doesn't allow creating other level of data sensitivity

Table 4: Comparison (2)

	Protection of encryption keys	Encryption granularity level	Protection of Database structure
Full Encryption Model	• The model protects encryption keys according to "The Master Key generation model"	• The model encrypts data at cell level	• The model encrypts DB structure according "The Database structure encryption model"
(Sesay *et al*., 2005)	• No model defined to protect encryption keys.	• The model encrypts private data at cell level and classified data at column level	• No model defined to protect DB structure
(Shmueli *et al*., 2014)	• No model defined to protect encryption keys. • Proposal of classical approaches to protect keys such as: Wallet, HSM.	• The model encrypts data at cell level	• No model defined to protect DB structure

Table 5: Table "employee" before encryption

Code	First_name	Last_name	Salary
0001	Paul	Williams	2000
0002	Paul	Watson	4000
0003	Paul	Stevens	3000
0004	Paul	Diaz	6000

Table 6: Table "employee" after encryption

Code	E7762D87BE5F945200E1D6D6FB4BAE13	6CD6345907C75C6024A9F30EF511412	12D000F12B13967460FD1BDDFDC76A39
0001	9A92AED8F7556155627AB4A3C5F04E23	47CF2D656BF168CA58757A88A78E5AFE	7C83A5BF4DBB903FD707D581E05CA78B
0002	0A9DDD8178B598858BF601AA80285671	B29D5A3EBC24E9C8E60C40927FCE3539	CD1D6512C2AEFB7D48CF343BECE9E3DC
0003	62DE828D74C326B872897703CD29FCAF	F7DAF7E6EC8EFB3993ACEAC1617F55AD	4D5E1EF3C302F3E190EC4A93F6C94DAA
0004	5B30556038840D1CE6D4030616B77E0A	BE8BBEB4BAFF6C63297619F9B1293B48	36BD51A6E388781DFAF33B7647F66642

Table 7: Encryption keys generated for data and the Master keys

Column name	keys generated for encrypting the columns data	Master key
First_name	9E9A833CD2243929E932A212610AE8C4	3A58508BE387C9142ED397047EEC5BE1
Last_name	280633F706590011FE8DE6B8C2807B04	51995FCA5D1A5F628BE8029B95E414B5
Salary	C3D81B8FEE00F1880BD57EC2031521ED	C052C4B7867D8497D1D3F9D4BC366325

Table 8: Encryption keys generated for columns names

Column name	keys generated for encrypting the columns names
First_name	9E9A833CD2243929E932A212610AE8C4
Last_name	280633F706590011FE8DE6B8C2807B04
Salary	C3D81B8FEE00F1880BD57EC2031521ED

Discussions

According to the comparison represented in Table 3 and Table 4 and taking in consideration the implementation performed in the case study, we have chosen to represent our analysis of results based on 6 criteria as shown.

1st Criterion: *Algorithms use*

The notion of the class with arguments especially the Algorithm(i) argument provides the uses of several algorithms to encrypt data in the columns and also the DB structure. That implies strong security for data in DB. This feature is unique to our "Full Encryption Model" compared to other models in Table 3. Another advantage provided using the Algorithm(i) argument in the class is the possibility to create and implement new specific classes which support other encryption algorithms other than the usual encryption algorithms (DES, AES, 3DES, ...etc). For example, implementing the algorithm proposed by Sekhar and Sivaranjani (2018).

2nd *Criterion*: *Keys (Uses/Generation)*

The notion of the class with arguments provides to our "Full Encryption Model" the specificity, compared to other models in Table 3, to use multiple keys for encrypting data in columns and also the DB structure. Therefore, it avoids naive data encryption and respects the requirements of an efficient DB encryption model discussed in the DB encryption preliminaries part. As we can see in the implementation results illustrated in Table 7, each sensitive column (First_name, Last_name, Salary) of "employee" table possesses its encryption keys and it will be encrypted using these keys. The number of generated keys for encrypting the columns data equals to the number of sensitive columns. The values of the Key_class(i) arguments participate in the generation of the real encryption keys (models (6) and (7)) with a random way and an unknown value for the administrator. Modifying this argument changes the real encryption key and thus increases data security.

Another feature of the "Full Encryption Model" provided using classes, is the creation of derived classes from one class. For example, we can use several classes to encrypt DB which have the same encryption algorithm; i.e., the same values of the arguments (Algorithm(i)) and different values of the arguments (Key_class(i)). Of course, these derived classes must have distinct Id_class(i).

3rd *Criterion*: *Data Sensitivity Level*

In the "Full Encryption Model", sensitive data in columns can belong to one level of data sensitivity or several levels of data sensitivity. The sensitivity value is fixed in the argument (Sensitivity(i)) within classes. This concept allows encrypting data according to their level of data sensitivity using different values of arguments (Key_class(i)) and (Algorithm(i)). This feature is special and unique to our "Full Encryption Model" compared to other models mentioned in Table 3, or generally in the models of the literature.

Another function delivered using classes is the possibility to create other classes with finer granularity by adjusting the argument (Sensitivity(i)). For example if the data of two columns COL1 and COL2 have secret nature and if the level of data sensitivity of COL2 is more important than COL1, then we can define an encryption on COL1 using a Class(i) whose the argument (Sensitivity(i)) is equals to "Secret" and another encryption on COL2 using Class(j) whose the argument (Sensitivity(j)) is equals to "Top secret". Both classes must use different values of arguments (Key_class(i)) and (Algorithm(i)).

Encrypting data in columns according to their level of sensitivity in our "Full Encryption Model" brings two major advantages. First, it allows the encryption/decryption only for these columns. This will improve extremely the performance because only the DB sensitive parts are encrypted/decrypted while executing queries. Second, the protection of the only DB sensitive parts provides a perfect optimization of the DB size.

4th *Criterion*: *Protection of Encryption Keys*

Using a specific model dedicated to protect encryption keys inside the DB server is the particularity of our "Full Encryption Model". As shown in Table 4, this concept doesn't exist in the models developed by Shmueli *et al*. (2014) and by Sesay *et al*. (2005). The "Full Encryption Model" eliminates the implementation of the approaches based on External Security Module (ESM) such as (Wallets, HSM, Security Server, etc.), as implemented in the other models in Table 4. We have already given in our previous work (El bouchti *et al*., 2018) certain limits that may disturb or decrease the security when adopting these approaches. The "Full Encryption Model" via the model (8) generates the Master keys automatically when defining encryption on columns data and there are no defined storages for them, so the administrators can't access them.

As we can see in the implementation results illustrated in Table 7, each sensitive column has its encryption key ($K_{COL(i)}$) and its Master key ($Km(COL(i))$). This concept reinforces enormously the DB security and makes the operation to get encryption keys by attackers an impossible operation.

5th *Criterion*: *Encryption Granularity Level*

The encryption granularity level of our "Full Encryption Model" is fixed at cell columns. The same operation is performed in the other models in Table 4, except the "Classified" data in (Sesay *et al*., 2005) model. Encrypting DB at cell level provide strong security for data as each cell is encrypted independently and differently. As illustrated in Table 5 and Table 6, even if the values of the column "First_name" are equal, their encryption values are different. Using this technique, we reinforce the DB security against the probability of frequency analysis attacks.

6th Criterion: Protection of Database structure

This is the specificity and the uniqueness of the "Full Encryption Model" compared to other models in Table 4. The DB structure encryption is another line of defense of DB beside to the data encryption and the access control mechanism. The design of our "Full Encryption Model" based on the encryption classes allows implementing two security levels on the DB. The first level concerns the column data, while the second one concerns the name of this column (see Table 6). A DB user can't access to data without having the appropriate classes for both levels.

Conclusion

This article outlines the role that can play encryption in the protection of the sensitive DB. This aspect strengthens and multiplies the complexity for the attackers to access a DB when they bypass other security barriers such as the access control mechanism and authentication.

The originality of our work is to propose a novel database encryption model called the "Full Encryption Model". It is based on a new concept that we have developed and named "Encryption Class". Compared to other relevant models, our model is considered full in terms of data security mechanisms. It perfectly satisfies the requirements of an effective DB encryption model. This has been proven by its implementation and its comparison to two different models according to several criteria.

In forthcoming works, we aim to improve the proposed model in order to limit access for the DB's users belonging to specific categories.

Acknowledgement

I would like to express my appreciation to all my professors, who they helped and guided me to realize this work. I would like also to thank the reviewers for their reviews and comments, which have helped to improve the quality of this paper.

Author's Contributions

***Karim El Bouchti:** Worked of the most parts, method idea and implementation of the method: conception of the models and their implementation, analysis and discuss of results

Soumia Ziti: Engaged in the literature review, proposed the related work and participated in the analysis and discussion of the results

Fouzia Omary: Proposed the research methodology. Participated in the reviewing of the final version.

Nassim Kharmoum: Participated in the evaluation of results

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all the authors have read and approved the manuscript and there are no ethical issues involved.

References

Bouganim, L. and Y. Guo, 2009. Database encryption. In: Encyclopedia of Cryptography and Security, S. Jajodia and H. Van Tilborg, (Eds.), Springer, pp: 1-9. ISBN-10: 978-1-4419-5905-8

Chen, B.H., P.Y. Cheung, P.Y. Cheung and Y.K. Kwok, 2018. Cypherdb: A novel architecture for outsourcing secure database processing. IEEE Trans. Cloud Comput., 6: 372-386. DOI: 10.1109/TCC.2015.2511730

DECJ, 2013. Directive No. 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the Legal Protection of Databases, Art. 7(1) and (5) – Innoweb BV v. Wegener ICT Media BV and Wegener Mediaventions BV. IIC - International Review of Intellectual Property and Competition Law.

Dixit, P., A.K. Gupta, M.C. Trivedi and V.K Yadav, 2018. Traditional and hybrid encryption techniques: A Survey. In Networking Communication Data Knowledge Engineering, G.M. Perez, (Ed.), Springer, pp: 239-248.

El Bouchti, K., N. Kharmoum, S. Ziti and F. Omary, 2019. A new approach to prevent internal attacks on Database encryption keys. Proceedings of the International Conference Scientific Days Applied Sciences, Feb. 15-16, Morocco.

El Bouchti, K., S. Ziti and F. Omary, 2018. A new approach to protect encryption keys in database management system. Proceedings of the International Conference Modern Intelligent Systems Concepts, Dec. 12-13, Morocco.

El Bouchti, K., S. Ziti, Y. Ghazali and, N. Kharmoum, 2018. Sécurité des bases de donnees: Menaces principales et solution de chiffrement existantes. Proceedings of the JDSIRT Conference Information Systems, Networks Telecommunications, Nov. 28-29, Morocco, pp: 13.

Elovici, Y., R. Vaisenberg and E. Shmueli, 2018. U.S. Patent No. 9,934,388. Washington, DC: U.S. Patent and Trademark Office.

Galushka, V.V., A.R. Aydinyan, O.L. Tsvetkova, V.A. Fathi and D.V. Fathi, 2018. System of end-to-end symmetric database encryption. J. Phys.: Conf. Series, 1015: 042003. DOI: 10.1088/1742-6596/1015/4/042003

Itamar, E. and A. Rotem, 2018. U.S. patent application no. 15: 570-775.

Jacob, S., 2012. Protection cryptographique des bases de données: Conception et cryptanalyse. Unpublished dissertation in partial fulfillment of the requirements for the degree of Doctor of Philosophy, Pierre et Marie Curie-Paris VI University, Paris, France.

Lee, I., S. Jeong, S. Yeo and J. Moon, 2012. A novel method for SQL injection attack detection based on removing SQL query attribute values. Math. Comput. Modelling, 55: 58-68.
DOI: ORG/10.1016/J.MCM.2011.01.050

Liu, L. and J. Gai, 2008. A new lightweight database encryption scheme transparent to applications. Proceedings of the International Conference Industrial Informatics, Jul. 13-16, IEEE Xplore press, South Korea, pp: 135-140.

Ma, S., Y. Mu and W. Susilo, 2018. A generic scheme of plaintext-checkable database encryption. Inform. Sci., 429: 88-101.
DOI: RG/10.1016/J.INS.2017.11.010

Mattsson, U.T., 2005. A practical implementation of transparent encryption and separation of duties in enterprise databases: Protection against external and internal attacks on databases. Proceedings of the International Conference E-Commerce Technology, Jul. 19-22, IEEE Xplore press, Germany, pp: 559-565. DOI: 10.1109/ICECT.2005.82

Priebe, C., K. Vaswani and M. Costa, 2018. Enclavedb: A secure database using sgx. Proceedings of the Symposium Security Privacy, May. 21-23, IEEE, USA, pp: 264-278.

Sallam, A.I., E.S. El-Rabaie and O.S. Faragallah, 2012. Encryption-based multilevel model for DBMS. Comput. Security, 31: 437-446.
DOI: ORG/10.1016/J.COSE.2012.02.008

Sekhar, J.R. and G. Sivaranjani, 2018, Database encryption using TSFS algorithm. Int. J. Scientific Res. Comput. Sci. Eng. Inform. Technol., 4: 494-500.

Sesay, S., Z. Yang. J. Chen and D. Xu, 2005. A secure database encryption scheme. Proceedings of the Consumer Communications Networking Conference, Jan. 3, IEEE Xplore Press, pp: 49-53.

Shmueli, E., R. Vaisenberg, Y. Elovici and C. Glezer, 2010. Database encryption: an overview of contemporary challenges and design considerations. ACM SIGMOD, Dec. 15, pp: 29-34. DOI: 10.1145/1815933.1815940.

Shmueli, E., R. Vaisenberg, E. Gudes and Y. Elovici, 2014. Implementing a database encryption solution, design and implementation issues. Comput. security, 44: 33-50.
DOI: ORG/10.1016/J.COSE.2014.03.011

Zabihimayvan, M. and D. Doran, 2019. Fuzzy rough set feature selection to enhance phishing attack detection. Proceedings of the International Conference Fuzzy Systems, Jun. 23-26, IEEE, USA, ARXIV preprint ARXIV:1903.05675.