ASTES

Advances in Science, Technology and Engineering Systems Journal Vol. 5, No. 2, 87-94 (2020)

www.astesj.com

ASTESJ

ISSN: 2415-6698

New Solution Implementation to Protect Encryption Keys Inside the Database Management System

Karim El bouchti*, Soumia Ziti, Fouzia Omary, Nassim Kharmoum

Faculty of Sciences, Mohammed V University in Rabat, Intelligent Processing Systems & Security (IPSS) Team, Morocco

ARTICLE INFO

Article history:
Received: 31 December, 2019
Accepted: 13 February, 2020
Online: 09 March, 2020

Keywords:
Encryption keys protection
Database encryption
Database security
keys protection in Databases

ABSTRACT

Due to the attacks' growth on sensitive databases by deploying advanced tools, beyond access control and authentication mechanisms, the database encryption remains a useful and effective way to ensure robust security of data stored within it. Any database encryption solution is based on a specific encryption model that determines how data is encrypted inside it. A relevant database encryption model must necessarily adopt a strong security policy of data encryption keys. It determines how these keys are generated, stored, and protected. In this work, we will implement an original solution that protects the encryption keys when encrypting data occurred at the Database Management System level. Our solution suggests protecting the keys by their encryption with other ones named Master Keys, which are generated according to the encryption granularity defined by the database encryption model. The proposed solution protects the keys of two database encryption models: at the level of the columns and the level of tables.

1. Introduction

Database (DB) level encryption is a way to encrypt and decrypt data within the Database Management System (DBMS) using keys held by the DB server [1]. In fact, this encryption model offers major advantages, in particular, those related to the security of the encryption keys against external attacks performed outside the information system [2, 3]. Though the probability of the administrator attack is not negligible, he owns large privileges on DB. Hence, he can attack the DB directly, via a collaboration with a malicious external attacker (disclosure of the administrator account for example), or with an internal attacker such as a legitimate user [4, 5, 6]. Obviously, the DB administrator has the ability to perform all these attacks without leaving any traces [7, 8].

Data encryption at the DBMS level is based fundamentally on its specific encryption model. The management of encryption keys within this model is a crucial point, it defines the ways how keys are generated, stored, and protected [9, 10]. Actually, keys values, how users access them, and where they are stored are the ultimate goal of any attacker. Therefore, it should be necessary to establish a protection policy for encryption keys to minimizing their exposure face of malicious attackers. Indeed, many solutions have been developed in order to resolve this problem. For instance, the keys protection solutions implemented at DBMSs such as Oracle, Ms SQL Server and My SQL are mainly based on the use of "Wallets", Hardware Security Module (HSM) and "Security server" [11-14]. In fact, each one of these solutions has advantages and major limits, as have already explained in more detail in our previous work [8, 15].

Several studies proposed several solutions to protect DB encryption keys either in DB encryption models or separate solutions of keys protection [16, 17]. The authors of [16] proposed a model to protect encryption keys based on a concept of distributed keys representation. The first key part stored in the DB, and the other part is obtained by converting the user password. Elovici et al. presented a special DB encryption model that permits to protect keys using "Wallet" mechanism [17], although Itimar et al. used asymmetric key encryption [18]. The authors of [19] suggested a solution called "Server-HSM" which merges an HSM and a security server into a module called "HW Security Module" that might be integrated into a DB server. This module manages user privileges and protects keys with their encryption. El bouchti et al. presented a full package of encryption keys protection models inside the DBMS. Their method allows encrypting keys with master keys generated according to encryption granularity adopted by the DBMS encryption model [15]. Sesay et al. in their proposed

* Karim El bouchti, Email: elbouchtikarim@gmail.com

DB encryption model suggested protecting keys by the use of a particular way. In fact, they generate the encryption keys from a unique master key (Km) generated and stored in a tamper-proof controller [20].

Notwithstanding several DB security studies for improving the concept of encryption keys protection within DBMS, more investigations are required to develop, improve, and provide more security and flexibility to keys protection. Then, as discussed in our previous work presented in [7, 10, 15], most of the proposed solutions have their advantages and disadvantages. However, to our knowledge, a trusted and simple solution concretized by a real implementation, and adapted to more than one DB encryption model has not yet been proposed.

In this context, the present work aims to implement two solutions proposed in our previous work [15] to protect DB encryption keys. Our solutions protect encryption keys when encryption granularity adopted by the DBMS is fixed at the column level or the entire table level. They consist of encrypting the DB encryption keys of tables and columns using Km, generated by deploying two models. We consider that the main original features of our solution are its reliability and its principle. It does not require any control or management of key protection by a DB or security administrator as well as it resists strongly against attacks performed by administrators.

The present work will be structured as follows: section two presents the proposed models of Km and the common objects of the implementation of each model. It also explains how our Km models work with DB encryption models that we will implement. Section three explains the implementation and discusses the provided results. Finally, our article ends with a conclusion.

2. Definition of the proposed solution

In this section, we will define two models (1 and 2) of the Km generation that protects, respectively, the columns and tables keys. We will also describe the role of the common objects that have used between the two models' implementation and how those models work in an encryption/decryption process.

2.1. Proposed models of Km

The generation of the Km follows the two different models below:

- The Km (C) key used to protect the encryption keys of the DB columns. It is generated by the DBMS according to the model defined below:

  $$Km\ (C) = H\ (Table\ \_name \mid\mid Column\_name) \quad (1)$$

- The Km (T) key utilized to protect the encryption keys of the DB tables. It is generated by the DBMS according to the model defined below:

  $$Km\ (T) = H\ (Table\_name \mid\mid Database\_name) \quad (2)$$

In order to concretize the functioning and the response of the models (1) and (2), we have designed and implemented two models of DB encryption (A) and (B) having, respectively, two encryption levels: column and table. The functioning of each Km generation model is associated with the execution of a DB encryption model, as shown in Table 1.

Table 1: The generation model of Km and the DB encryption model associated

Km generation model	Associated DB encryption model
Model (1): Km generation model for protecting encryption keys of columns	Model (A): DB encryption model at the column level
Model (2): Km generation model for protecting encryption keys of tables	Model (B): DB encryption model at table level

2.2. The common objects of each model implementation

The creation of elements below is common for the implementation of the models (1) and (2).

MYTABLE_ENCRYPT_OBJET: This DB table contains records of : i) the names of the objects on which encryption has defined, ii) the used encrypt algorithms, and iii) the encryption of the encryption keys using Km. It has the following structure:

MYTABLE_ENCRYPT_OBJET (K_OBJECT_NAME, K_ENCRYPT_ALGO, K_KEY)

TEST_MANAGEMENT: This DB table contains the records of : i) the names of the objects on which encryption has defined, ii) the used encrypt algorithms, iii) the object encryption keys, and iv) the Km of each object name. In fact, TEST_MANAGEMENT table is not implemented in the real working case of our solution. Nevertheless, its main role is to illustrate the results of the creation of both encryption keys and the Km. The table has the following structure:

TEST_MANAGEMENT (C1_OBJECT, C2_ALGO, C3_KEY, C4_MASTERKEY)

where:

C1_OBJECT, K_OBJECT_NAME: the object on which we have defined encryption;

C2_ALGO, K_ENCRYPT_ALGO: the algorithm used for encryption;

C3_KEY: the encryption key;

C4_MASTERKEY: the generated Km;

K_KEY: the encrypting result of encryption keys using Km.

The Md5 hash function: it is used to create the encryption keys and the Km while implementing all models.

The AES256 algorithm: it is the algorithm used to encrypt / decrypt the data.

Note: The term "Objet" signifies the column and table names.

2.3. Principle of the encryption/decryption process

An encryption/decryption process in the model (A) follows the defined steps below:

When a user sends a query to be executed, the DBMS generates the Km for each column on which encryption is defined. The DBMS decrypts, using the generated Km, a value from the K_KEY column belonging to MYTABLE_ENCRYPT_OBJET table and which corresponds to the column desired to be encrypted or decrypted. This process generates the real encryption key of column that will encrypt (in the case of an inserting or updating query) or decrypt (in the case of a consulting query) the data.

The encryption/decryption process in the model (B) follows similar operations. In this case, the DBMS generates a single Km in response to the user's query since the encryption is defined on the entire table.

2.4. Case study: the dosimetric monitoring of agents

A real case study has chosen to illustrate the models' implementation results.

Let have a database named "ORCL10G" of a nuclear power plant intended to manage the dosimetric monitoring of agents working under ionizing radiation. The "agent" table stores the accumulation of the different types of doses received by each agent during the period of his work within the plant. We assume that all the data in the "agent" table are sensitive since the dose values are considered in the nuclear field as medical secret. The table "agent" has the following structure:

agent (idf_agent, name_agt, Dose_interne_agt,

Dose_superficielle_agt, Dose_profonde_agt, Catégorie_agt)

3. Implementation of the proposed models

In this section, we will present the implementation of the models (1) and (2) generating Km as well as the corresponding DB encryption models (A) and (B).

3.1. Implementation of the model (1)

In order to create the "agent" table and defining encryption on all its sensitive columns, we have used the following SQL syntax:

Create table agent (idf_agent varchar2(100) encrypt using AES256, name_agt varchar2(100) encrypt using AES256, Dose_interne_agt varchar2(100) encrypt using AES256, Dose_superficielle_agt varchar2(100) encrypt using AES256, Dose_profonde_agt varchar2(100) encrypt using AES256, Catégorie_agt varchar2(100) encrypt using AES256);

The "Algo1" algorithm supports the execution task of this statement; it creates the "agent" table and defines encryption on its columns using the AES256 algorithm. In fact, the data column will be encrypted/decrypted with six keys K_C which will be protected by six masters key Km (C). In addition, the "Algo1" generates and stores the encryption keys of the columns (Kc) and the master keys (Km (C)) within MYTABLE_ENCRYPT_OBJET and TEST_MANAGEMENT according to the models defined below:

/* The model of the encryption key used in the model (A)*/
Kc = H (Column_name)
/* The Km generation model of a column */
Km (C) = H (Table_name || Column_name)

Algorithm1: Process managed by Algo1.

Algo1

```
Input: Sensitive_column _query
Output:Created _sensitive_column_ query

Begin
Loop
Decompose (Sensitive_column _query);
Sensitive_column_name :=  Extract (Sensitive_column
_query);
Kc:= Kc_Generator (Sensitive_column_name);
Km (C):= Km_Generator (Sensitive_column_name, Table
_name) ;
End loop;
Insert into TEST_MANAGEMENT values
(Sensitive_column_name , Used_algo, Kc, Km (C));
 Insert into MYTABLE_ENCRYPT_OBJET values
(Sensitive_column_name , Used_algo, Encrypt_AES256 (Km
(C), Kc ));
 Execute (Sensitive_column_query) ;
End;
```

The execution of the "Algo1"algorithm generates the following records:

Table 2: Records created in the TEST_MANAGEMENT table in the model (1).

C1_OBJECT	2_ALGO	C3_KEY	C4_MASTERKEY
idf_agent	AES256	46D18AC7BD06518B5A33C650CA760D9C	36EE05BA4ACDEC7D1C07D16FCDCC9CBF
name_agt	AES256	4BF0A8B1EB8CA12C2912ED25E4D4BDC5	F8422E273F1957702DE160662923C0EA
Dose_interne_agt	AES256	C8FE8472457B1A9436EE90E6D022178F	452C55A53935CF7056CE1C293FE8D4FC
Dose_superficielle_agt	AES256	1FD82F97BE7BA2565D7FDD4BD6A91179	44BB0C031437DB46B641257337C7C458
Dose_profonde_agt	AES256	F67D1D0A822D37E3AC03D69C94A38994	8D71448C964377D9F0E539B3FB230133
Catégorie_agt	AES256	822E1CDA2CFA7B1EF36B90523E337682	FE30063A2D96A6DB6059CE708103BD5F

Table 3: Records created in the MYTABLE_ENCRYPT_OBJET table in the model (1).

K_OBJECT_NAME	K_ENCRYPT_ALGO	K_KEY
idf_agent	AES256	12BB4076B53A907324D42AFB9B0501006A93E1F347EC05536A7115E4554 CD8D06662D1EEEAC1CA8D71BC2A33EFD7810B
name_agt	AES256	90F4D05F7C5A11A5752E8AB1354BE82D797BFE16051F78DD7018A60856 49A709515637DB88472F14509DB73FC76E6B0C
Dose_interne_agt	AES256	6B40D18BF99C712F4F4034E8A11AF73FC8B422CCC218AF595FD06EB4 E4AE4BB3E8D5D0CCA9BE57434FCD690A577D05D0
Dose_superficielle_agt	AES256	5F03CF21D034BFFBA1FEA4BD5CEA43A15A32E8C9F4FDE591AF842995B BD30FF53CC25849363BCF54B8FC2E28FD7FF7A7
Dose_profonde_agt	AES256	3A468DD545A12883543199A45202E4CE3AAA4D006CE4453BFC380377423 C24C3FA85A1BF68F31F6E011C062C5E19D2AE
Catégorie_agt	AES256	34FEFD552E0D63C4A99281FC1B0A8189D5B527EDC303E9AB760B35C1A2 28C1F7B0EFDAAF26B7E3F507297D351E64FFE9

Table 4: The "agent" table before encryption.

idf_agent	name_agt	Dose_interne_agt	Dose_superficielle_agt	Dose_profonde_agt	Catégorie_agt
1000	Azzaoui	10	15	25	A
1001	Rachidi	8	11	12	A
1002	Kharmoum	16	05	14	A
1003	Sajid	14	66	65	B

Table 5: The "agent" table after encryption.

idf_agent	name_agt	Dose_interne_agt	Dose_superficielle_ agt	Dose_profonde_agt	Catégorie_agt
5754957F70316A9C02 01002A8CBFE412	745A2F1FF2BDAD322 6910989994A7287	B5063DD0A036503D0 2F11B63DF1B12B1	FDBD02C08C389AE7 583F8211E428B2A7	D694729051643896225 89DA79E580A60	D1D88EDED1CF67BF 3AB34261052FF335
A6776D65772FBF4992 D8AFC2A2B387C9	1C65506D1F8AA015B 5B98765133E9782	5C6ABEA8480996813 19175A05C4B1AB5	BD706EE5E149EBDC E6796457E944FB81	BECBC7C7B6BD0B0E 12CF1AD549E22682	D1D88EDED1CF67BF 3AB34261052FF335
AFBFEA3BC5FCD283 96E974016169DF5D	67BC01E3BF672D7BF 568ACC3A582ECB8	B8C892229BBB1E395 1290107C524B12A	125180FAB503F868E DC55F42D0BB34CB	A8A68F1B4D1D3DD0F 398AFED4A54A0A5	D1D88EDED1CF67BF 3AB34261052FF335
14DB0A7462832F494E AC7C8D12A09FC6	7A1FBFCD0E02BC191 0055CF76279F21A	23BF97F16393EC6A8 4B93FF4F9EB74AA	293A72927BD5C9760 450FFFF14CEC693	A2B800E15CAFD3AFA 9A22FB0A4C257B8	5315BF4B7B530AA72 9AC1CDCB53F4C8B

In order to test model (1), the "Algo2" algorithm represents the functioning of Model (A). It supports the data encryption inserted by a user. Tables 4 and 5 show the result of inserting four lines in the "agent" table before and after the encryption.

Algorithm 2: The DB encryption using the model (1) and (A).

```
Algo2
CREATE OR REPLACE TRIGGER Insert_Model_A
BEFORE INSERT ON agent
FOR EACH ROW
DECLARE
Kc1 varchar2(100);
Kc2 varchar2(100);
Kc3 varchar2(100);
Kc4 varchar2(100);
Kc5 varchar2(100);
Kc6 varchar2(100);

BEGIN

/*Generating Km and seeking Kc for each column from
MYTABLE_ENCRYPT_OBJET*/
```

```
Kc1:= Search_Encrypt_Key ('idf_agent');
Kc2:= Search_Encrypt_Key ('name_agt');
Kc3:= Search_Encrypt_Key ('Dose_interne_agt');
Kc4:= Search_Encrypt_Key ('Dose_superficielle_agt');
Kc5:= Search_Encrypt_Key ('Dose_profonde_agt');
Kc6:= Search_Encrypt_Key ('Catégorie_agt');

                /* Inserting in table "agent"*/

INSERT INTO agent VALUES (Encrypt_AES256
(Kc1, :new.idf_agent), Encrypt_AES256
(Kc2, :new.name_agt, Encrypt_AES256
(Kc3, :new.Dose_interne_agt), Encrypt_AES256
(Kc4, :new.Dose_superficielle_agt), Encrypt_AES256
(Kc5, :new.Dose_profonde_agt), Encrypt_AES256
(Kc6, :new.Catégorie_agt));
End ;
```

3.2. Implementation of the model (2)

To implement model (2), we define encryption on the "agent" table level using the following SQL syntax:

Create table agent encrypt using AES256 (idf_agent varchar2(100), name_agt varchar2(100), Dose_interne_agt varchar2(100), Dose_superficielle_agt varchar2(100), Dose_profonde_agt varchar2(100), Catégorie_agt varchar2(100));

The "Algo3" algorithm supports the execution of this instruction. It creates the "agent" table and defines encryption on all its data using the algorithm AES256. In this case, the data are encrypted /decrypted with a single key K_T, which will be protected by a single master key Km (T). The "Algo3" algorithm generates and stores K_T and Km (T) also in MYTABLE_ENCRYPT_OBJET and TEST_MANAGEMENT according to the models defined below:

/* The model of the encryption key used in the model (B)*/
K_T= H (Table_name)
/* The Km generation model of a table */
Km (T) = H (Table_name || Database_name)

Algorithm 3: Process managed by Algo3.

```
Algo3
Input: Sensitive_table _query
Output: Created _sensitive_table _query

Begin
Decompose (Sensitive_table _query) ;
Sensitive_table_name:=  Extract (Sensitive_table _query);
K_T:= K_T_Generator (Sensitive_table_name);
Km(T):= Km_Generator (Sensitive_table_name, Database
_name);
Insert into TEST_MANAGEMENT values
(Sensitive_table_name, Used_algo, K_T, Km(T));
Insert into MYTABLE_ENCRYPT_OBJET values
(Sensitive_table_name ,  Used_algo, Encrypt_AES256
(Km(T), K_T));
Execute (Sensitive_table _query);
End ;
```

The execution of the "Algo3" algorithm generates the following records:

The "Algo4" algorithm represents the functioning of the model (B). It supports the data encryption inserted by a user. The tables 8 and 9 show the result of inserting four rows in the "agent" table before and after the encryption.

Algorithm 4: The DB encryption using model (2) and (B).

```
Algo4
CREATE OR REPLACE TRIGGER Insert_Model_2
BEFORE  INSERT ON agent
FOR EACH ROW
DECLARE
K_T  varchar2(100);
BEGIN

   /* Generating Km and seeking K_T for each column from
                MYTABLE_ENCRYPT_OBJET*/

K_T :=Search_Ecrypt_Key( 'agent');

                /* Inserting in table "agent"*/

INSERT INTO agent VALUES (Encrypt_AES256
(K_T,:new.idf_agent), Encrypt_AES256 (K_T,:new.name_agt),
Encrypt_AES256 (K_T,:new.Dose_interne_agt),
Encrypt_AES256 (K_T,:new.Dose_superficielle_agt),
Encrypt_AES256 (K_T,:new.Dose_profonde_agt),
Encrypt_AES256 (K_T,:new.Catégorie_agt));
End ;
```

Table 6: Records created in the TEST_MANAGEMENT table in the model (2).

C1_OBJECT	C2_ALGO	C3_KEY	C4_MASTERKEY
agent	AES256	B33AED8F3134996703DC39F9A7C95783	697C371A913425CF202D15F143D2DAF0

Table 7: Records created in the MYTABLE_ENCRYPT_OBJET table in the model (2).

K_OBJECT_NAME	K_ENCRYPT_ALGO	K_KEY
agent	AES256	736079082547466884631FC41910AB5770A6962367465EC1CB9526A 864367916929D05016D02B96D9B55511854D3AB13

Table 8: The "agent" table before encryption.

idf_agent	name_agt	Dose_interne_agt	Dose_superficielle_agt	Dose_profonde_agt	Catégorie_agt
1000	Azzaoui	25	25	25	A
1001	Rachidi	8	11	12	A
1002	Kharmoum	16	05	14	A
1003	Sajid	14	66	65	B

Table 9: The "agent" table after encryption.

idf_agent	name_agt	Dose_interne_agt	Dose_superficielle_ag	Dose_profonde_agt	Catégorie_agt
3F843D72B4CF6AF3B FEDD0F8A73A46DB	03377847D26334CDA6 65F815335C87F9	7012CB55124F226FA2 E530E0D8133F14	7012CB55124F226FA2 E530E0D8133F14	7012CB55124F226FA2 E530E0D8133F14	91B5BD099938E4CE4 DF76529F6740B8A
2D80DCE15D394B765 94AD5E18F3405BE	0B71021170A841DAE 0CB4641C08076CD	0C04B7B8C2A594ED2 D6CE2FCD9EE91FD	52516FBF14B5DB600 CF294F47153C168	CF92FB85EF0E373795 F9C4D57D66ECF1	91B5BD099938E4CE4 DF76529F6740B8A
4CDDFEF4428E61343 AD6C823FF1860A4	DA45B069E89AC7BE4 C2B69EE232EB1A4	0F8913B9B06C772A68 8E9489B13A1124	5C9D1E8240D88EAD 7072C8B7673AB4C1	CC2CCB3C2994CE129 59B9C9B66F478A5	91B5BD099938E4CE4 DF76529F6740B8A
CA80281E0ADBD105C E371803DD3F575E	EE6099C500334279227 AFA5C27E199FB	CC2CCB3C2994CE129 59B9C9B66F478A5	9B07E13AC87C52956 2187B81CFCD6B2B	EC2F5721D81D6CACE 4C6EA7B9B49ED52	72621DFBF38C674D9 BA24509BDA41160

3.3. Results and discussion

This section introduces data discussion and analysis based on the findings obtained by implementing the two Km models. They are summarized as follow:

- The proposed solutions are more practical than the conventional ones, primarily the Wallet, HSM, and the security server, where their disadvantages have explained and revealed in [15]. Our solutions optimize perfectly additional costs to protect the keys either in terms of hardware acquisition (case of HSM and security server) or in human resources (the administrator of the security server).
- The Wallet solution used in Oracle TDE generates Km, which protects encryption keys, and stores it within the Wallet. Here, it is necessary to create the Wallet and its password as well as a secured location (such as backup systems) to store the password whenever the Km is newly created. This operation is mandatory before starting the process of data encryption/decryption inside DBMS [11]. It is worthy to mention that the backup system is a critic component of the Wallet concept. In this vein, the protection concept of encryption keys based on the proposed models (1) and (2) is similar to the Wallet solution in terms of Km generation within the DBMS. However, with our concept, the Km creation does not require any Wallet creation to store Km or secured location to store the password.
- The proposed solution does not require any protection management of the encryption keys by a security administrator, DB administrator or another trusted collaborator, as discussed in [15]. Hence, none of them knew about the generation of Km or its location. The probability of attacking keys is almost impossible, even if the attacker arrives to consult table MYTABLE_ENCRYPT_OBJET stored in the DB dictionary.
- The proposed solution does not define the place where storing Km. Km generation is performed automatically while defining encryption on a sensitive object (column or table), precisely during the creation by the DB administrator. Obtaining a value of Km by attackers (administrators, internal or external attackers) is almost an impossible operation.
- It is important to notice that the new concept we have proposed and implemented enhances the security of encryption keys. In fact, compared to the Oracle TDE Column Encryption solution that uses a single master key to protect all the column keys, the model (1) generates several Km to protect each column key. The number of Km generated is equal to the number of sensitive columns. For example, if a DB contains 20 sensitive columns to encrypt, we need 20 keys to encrypt data and 20 Km to protect them.
- Our solution can work with any DB encryption models, obviously with those implementing encryption granularities at the level of columns and tables. It is well adapted to free license DBMSs.
- In the model (1) implementation, we focused on to protect 6 encryption keys of the following columns (idf_agent, name_agt,Dose_interne_agt,Dose_superficielle_agt,Dose_ profonde_agt, Catégorie_agt). Each column key is protected by encrypting it with its own Km generated by the model (1). This protection was tested by the implementation of the model (A). Table 4 shows the encryption test results of the 6 columns of the "agent" table deploying the model (1). Likewise, in the model (2) implementation, one encryption key of the table "agent" has protected with a single Km generated using this model. Then, this protection was tested using the model (B). Table

8 represents the result obtained of encrypting table "agent" by deploying the model (2).

- The results presented in tables 2 and 3 shows the keys generation when the "agent" table is created by the administrator. In table 2, the columns C3_KEY and C4_MASTERKEY represents, respectively, the encryption key generated of each column and its associated Km. In table 3, the column K_KEY, belongs to MYTABLE_ENCRYPT_OBJET table, represents the encryption of each column key of the table "agent" using its associate Km.
- In table 6, the columns C3_KEY and C4_MASTERKEY represents, respectively, the key generated to encrypt the entire table and its Km. The value of the column K_KEY in table 7 represents the encryption of the encryption key using its associate Km. Both tables show the keys generation when the "agent" table is created by the administrator.
- As described earlier, tables 4, 5, 8, and 9 show the result of inserting rows in the "agent" table before and after the encryption. The encryption results showed that our solution works perfectly, either when encrypting or decrypting data. Actually, the encryption process via model (A) or (B) requires the generation of Km through model (1) or (2), respectively. Each value of generated Km is used to extract, from K_KEY column, the real encryption /decryption key. The process of data decryption follows the same operations.
- Finally, both proposed model works inside DBMS are summarized according to the algorithm flowchart described below:

Let's consider a sensitive table A

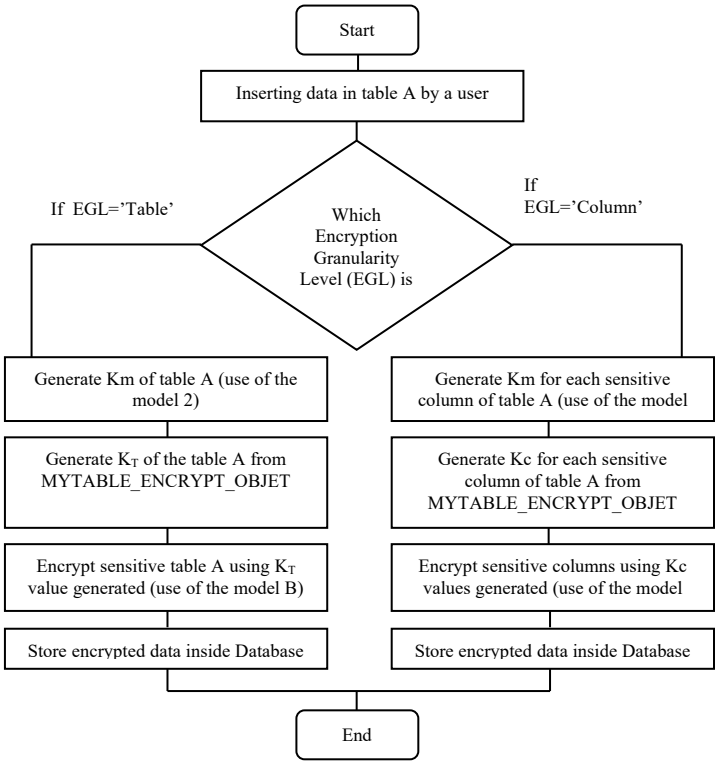

Figure 1: Algorithm flowchart specifying the both proposed models.

Conclusion:

Besides the conventional mechanisms deployed to secure sensitive DB (network protection, authentification, and access control), data encryption at DBMS level is a strong way that reinforces the defense in depth of the sensitive data. This process is strongly linked to the protection of the encryption keys on which they depend on two main factors: the location where the keys are stored and users who have access to them.

Our contribution in this article is to implement two solutions that secure encryption keys within the DBMS. These solutions are original and well adapted to any encryption model inside a DBMS. The solution's purpose is to protect DB keys by their encryption using a master key generated when defining encryption on a table or column. In forthcoming works, we aim to develop a solution that covers the protection of encryption keys when encryption is done on Tablespaces.

Conflict of Interest

This manuscript has not been published and is not under consideration for publication elsewhere. We have no conflicts of interest to disclose.

Acknowledgment

I would like to express my appreciation to all my professors, whom they helped and guided me to realize this work.

References

[1] E. Shmueli, R. Vaisenberg, Y. Elovici, C. Glezer, "Database encryption: an overview of contemporary challenges and design considerations" ACM SIGMOD Record, New York, NY, USA, 2010. DOI: 10.1145/1815933.1815940

[2] Hashim, Hassan B, "Challenges and Security Vulnerabilities to Impact on Database Systems" Al-Mustansiriyah Journal of Science 29(2), 117-125,2018. DOI: http://doi.org/10.23851/mjs.v29i2.332

[3] S. Jacob, "Protection cryptographique des bases de données: conception et cryptanalyse," Ph.DThesis, Université Pierre et Marie Curie-Paris VI, 2005.

[4] A.M.Mostafa, F.A. Almutairi, M.M. Hassan, "False alarm reduction scheme for database intrusion detection system" Journal of Theoretical & Applied Information Technology., 96(10), 2816-2825. ISSN: 1992-8645

[5] I. Homoliak, J.Guarnizo, Y.Elovici, M.Ochoa, "Insight into insiders and it: A survey of insider threat taxonomies, analysis, modeling, and countermeasures" ACM Computing Surveys, New York, NY, USA 2019. https://doi.org/0000001.0000001

[6] Deepicata. N. Soni, "Database Security: Threats and Security Techniques" International Journal of Advanced Research in Computer Science and Software Engineering., 5(5), 621-624, 2015. ISSN: 2277 128X

[7] K. El bouchti, S.Ziti, Y.Ghazali, N.Kharmoum, "Sécurité des Bases de Données : Menaces principales et solution de chiffrement existantes", in Proceedings of the JDSIRT Conference Information Systems, Networks Telecommunications, Meckness, Morocco, 2018.

[8] K. El bouchti, N. Kharmoum, S. Ziti, F. Omary, "A new approach to prevent internal attacks on Database encryption keys" Proceedings of the International Conference Scientific Days Applied Sciences, Larache, Morocco, 2019

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

[10] K. El Bouchti, S. Ziti, F. Omary, N. Kharmoum, "A New Database Encryption Model Based on Encryption Classes" Journal of Computer Science., 15(6), 844.854, 2019. DOI: 10.3844/jcssp.2019.844.854

[11] Oracle (2016), Oracle® Database Advanced Security Administrator's Guide 11g Release 2 (11.2)[online] Technical Document:

[12] S. Mukherjee, "Popular SQL Server Database Encryption Choices" International Journal of Computer Science and Engineering, arXiv preprint arXiv: 1901.03179, 2018. ISSN: 2231 – 2803

[13] MySQL Server Documentation. MySQL 5.7 Reference Manual Online

[14] A. K. Maurya , A.Singh, U.Dubey, S.Pandey, U. N.Tripathi, "Protection of Data Stored in Transparent Database System using Encryption" Journal of

Computer and Mathematical Sciences., 10(1), 190-196, 2019. ISSN 2319-8133.
[15] K. El bouchti, S.ZITI, F.OMARY, "A new approach to protect encryption keys in Database Management System", Proceedings of the International Conference Modern Intelligent Systems Concepts, Rabat, Morocco, 2018.
[16] V.V.Galushka, A.R.Aydinyan, O.L.Tsvetkova, V.A.Fathi, D.V.Fathi, "System of end-to-end symmetric database encryption" In International Conference Information Technologies in Business and Industry, 2018. Doi :10.1088/1742-6596/1015/4/042003.
[17] Elovici, Y., Vaisenberg, R., & Shmueli, E. (2018). U.S. Patent No. 9,934,388. Washington, DC: U.S. Patent and Trademark Office.
[18] Itamar, E., & Rotem, A. (2018). U.S. Patent Application No. 15/570,775.
[19] L.Bouganim, Y.Guo, Database encryption. In Encyclopedia of Cryptography and Security, Springer US, 2011
[20] S.Sesay, Z. Yang, J.Chen, D. Xu, "A secure database encryption scheme" In Consumer Communications and Networking Conference, CCNC. 2005 Second IEEE, Las Vegas, NV, USA, 2015. DOI: 10.1109/CCNC.2005.1405142