**Abstract**

As we know, data is the important elements for database because database is collection of data and programs to perform operations on that data. With the development of database, data security is becoming one of the most urgent challenges. While protecting the privacy of individual recorders, there are more and more interests in securing aggregate data. Since data encryption is the basic technique used for guarding specific sensitive data-items or objects, it's easy to think that this technique is applied to databases to replace building walls around servers or hard drivers. This can prevent not only outside attacks which come from hackers, but also inside attacks which are from employees who accidentally delete sensitive information. In this paper, we will overview basic encryption terminology in database management system (DBMS) and several factors to affect encryption. Then, we will focus on different encryption implementations in three popular database manufacturers, Oracle, MySQL, Microsoft SQL server and the third-party vendors. Through comparing these applications, the final goal is to offer different database encryption options — showing how each variant affects application performance, manageability, and security, then to help a user to develop an own database encryption strategy that will meet its individual needs.

**1. Introduction: Importance for Encryption**

Nowadays, data seems to be kept on everything – from the websites we visit, to how many inventories for one product, to what kind of clothing they are buying, and how many customers this company has. Most important thing is data helps us to extract information and make various decisions. Then, we stored data in database so that retrieving and maintaining it becomes easy and manageable. However, the effect for a database’s large-scale leakage is much worse than a leaked document. For example, in 2012, the famous social networking website *LinkedIn* was hacked by Russian hackers, *Russian cybercriminals*. More than 6 million users could not log in their accounts1. In the same year, *Dangdang.com*, one of China's biggest e-commerce websites declared their database was hacked too. Form October 2011 to March 2012, More than 12 million users’ information was leaked. Some users deposited some e-money in their accounts and hackers had taken the money out2. These cases happened every day and they tell us how important to secure data in database.

To achieve a safe database environment, Database security emphasizes three main properties: confidentiality, integrity and availability3. Roughly speaking, the confidentiality property prevents unauthorized persons to access the protected data. The integrity property guarantees that the data cannot be corrupted in an invisible way. The third property, availability ensures timely and reliable access to the database. The best way to protect the data is combining these three properties and applying them together.

Figure 1. A Security Mode for Database Security: C.I.A. triangle3

Over the last two decades, database security generates two main methods, access controls and encryption to protect data4. Usually, access controls to gate that should or should not be allowed access to the database, and encryption to protect data at rest. They can meet the requirements for those aforementioned properties. Confidentiality connects with access controls. Integrity points to encryption. Availability means these two methods should not hinder users’ operation. In the real world, these two methods are complementary each other. Sometimes, access controls could not block some unauthorized users. For example, an intruder can infiltrate the information system and try to mine the database footprint on disk. Access controls breaks down. Then, it is the turn for encryption. The purpose of database encryption is to ensure the database opacity by keeping the information hidden to these unauthorized users. Even if attackers get though the firewall and bypass access control policies, they still need the encryption keys to decrypt data.

**2. Database Encryption Basics: What needs to be known?**

Before we discuss different applications for encrypting and decrypting databases, we need to understand something: what is decryption? What is database decryption? How it works and how data flows in the application?

* **Definition**

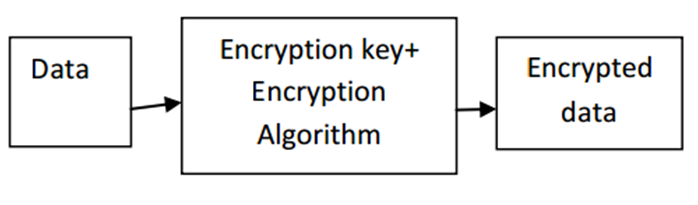


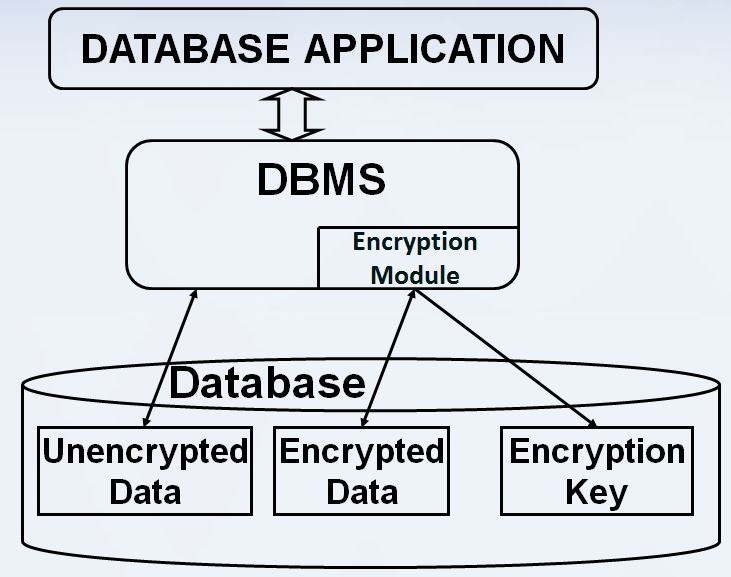
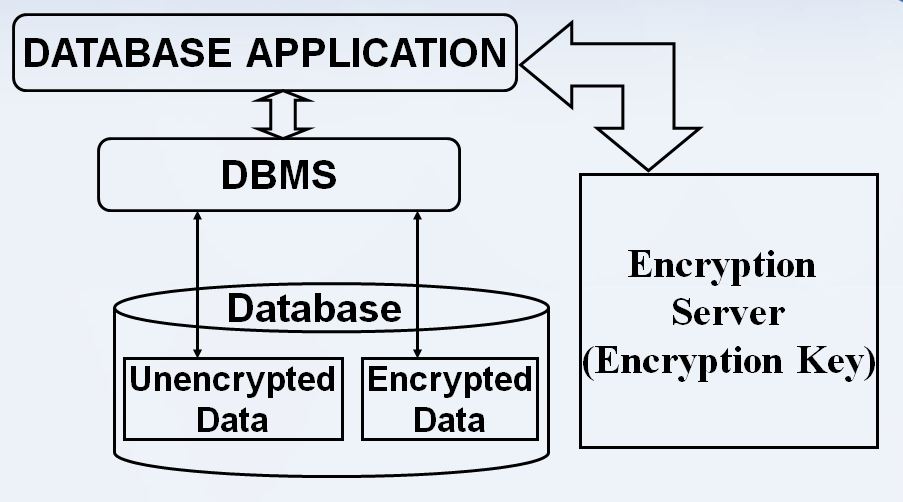
Figure 2. Basic Encryption Processes

Encryption is “*the transformation of data by using sophisticated algorithms in an attempt to make the data unrecognizable*3.” Figure 2 illustrates the basic process of encryption. Data are encrypted using encryption keys and encryption algorithms. Encrypted data are then stored in the database and decrypted when need to be used for processing purpose.

Database encryption is to use encryption techniques to transform a plain text database into a whole or partially encrypted database, thus making it unreadable to anyone except those users who hold the encryption keys.

* **Classification**

While performing database encryption, we can encrypt data inside or outside the database. Different process has different impact to database security. Of course, they have distinct advantages and disadvantages.

  Figure 3 – a. Encryption within Database Figure 3 – b. Encryption outside Database

If encryption is performed within the database, then the process of encryption happens in the database and there is less impact on application environment. Therefore, encryption becomes one role of the DBMS and is added to your database seamlessly. It can protect against a wide range of threats, including storage media theft, well known storage attacks, database-level attacks, and malicious database administrators (DBA). The drawback of this approach is encryption keys also are stored in the same database and uses that can enter the dataset have rights to use the secure keys too. To solve this problem, the keys can be stored outside the database. An additional weakness is encryption algorithm is designed by the database manufactures and users have not any more choices.

Another way to implement encryption in database is performing it on separate encryption servers. Encryption and decryption computations are performed encryption server. So here overhead of encryption is removed from DBMS and moved on to separate encryption servers to maintain the performance of DBMS. Encryption keys and data can also be separated. This approach is usually followed while encrypting database5. The only shortcoming is the function of DBMS will be limited after encryption, such as index.

**3. Different Factors for Database Encryption**

Independently of the encryption strategy, the security of the encrypted data depends on several factors like what granularity of the data needs to be encrypted or decrypted, what algorithm is used, what is the key size and how to protect the key.

* **Encryption granularity**

Unlike file encryption, database encryption can be performed at various levels of granularity. The apparent encryption granularity choices are as follows6.

(1) Page/block granularity. Whenever a page/block of sensitive data is stored on disk, the entire block is encrypted. It is a small achievable granularity, each page is encrypted separately. A typical page in SQL Server is 8 Kbytes and might contain one or multiple records.

(2) Attribute/column granularity. In this kind of encryption, the user chooses to encrypt only certain sensitive attributes in a table.

(3) Record/row granularity. Each row in a table is encrypted separately. This allows one to retrieve individual rows without decrypting the whole table.

(4) Field granularity. It is the smallest achievable granularity based on the field of a record; each attribute value of a tuple is encrypted separately.

In general, finer encryption granularity affords better performance in DBMS. Row or page level granularity may lead to encrypting large amount of data which can be overhead on the system. So generally field level encryption of only sensitive data is the suggestion to use.

* **Encryption Algorithm**

The second factor is the choice of encryption algorithm which is suitable for encrypting given data in database. Encryption algorithm is a mathematical procedure. “*Through the use of an algorithm, information is made into meaningless cipher text and requires the use of a key to transform the data back into its original form*”7.

The algorithms which are generally used for database encryption and often supported by database management system are DES, Triple DES and AES. Different algorithm has different encryption effect.

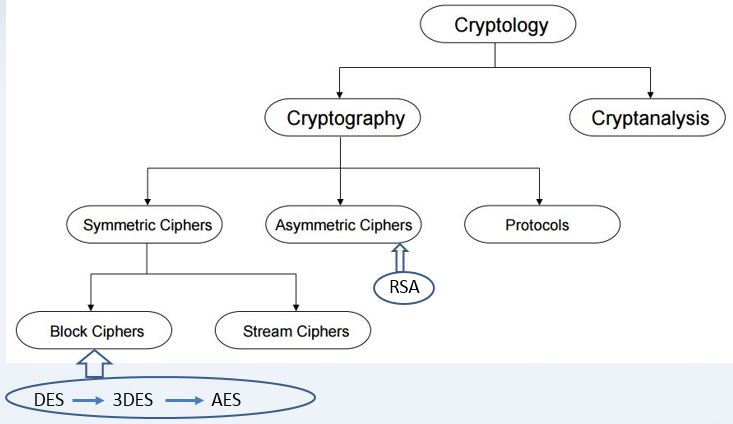


Figure 4. Classification in the Field of Cryptology8

As Figure 4 showing, there are two main types of cryptography in use today – symmetric or secret key cryptography and asymmetric or public key cryptography8. Digital Encryption Standard (DES) was the dominant symmetric encryption algorithm from the mid-1970s to the mid-1990s. It encrypts blocks of size 64bit.

DES encrypts blocks of size 64bit and is developed by IBM based on the cipher Lucifer under influence of the National Security Agency (NSA). It was introduced by the American National Standards Institute (ANSI) in 1977. By far, DES is the most popular block cipher for most of the last 30 years. It is applied in many fields, particularly in the financial services industry. Nowadays, with the development of technology, the standard DES with 56-bit key length can be broken relatively easily through an exhaustive key search. By encrypting with DES three times in a row, triple DES (3DES) is created. Then, triple DES extends the effective key length of DES to 112.

The encryption algorithm of 3DES is:

*ciphertext = EK3(DK2(EK1(plaintext)))*

Decryption is the reverse:

*plaintext = DK1(EK2(DK3(ciphertext)))*

Let us do a calculation. DES uses a 64-bit key to encrypt the data. So the hacker has to try 256, or 72,057,594,037,927,936 combinations to guess the encrypted data. Then, 3DES encrypts the DES encrypted data to make it more difficult to crack. It can pass the DES encrypted data up to three times through its encryption routine. Therefore, the hacker must try 2112 times for a two-pass scheme, or 2168 times for a three pass one.

Since 56-bit keys are no longer secure, Advanced Encryption Standard (AES) was created. In November 2001, AES was approved by the ANSI to replace the DES. AES is a modern block cipher which supports three key lengths of 128, 192 and 256 bit. 128 bits is quite strong. Assuming one hacker can break DES in 1 second. That means he can try 256 keys in one second. Then, he need to try 2128 keys which requires 149 Trillion years. It provides excellent long-term security against brute-force attacks and is efficient in software and hardware.

The name of RSA comes from its inventor, Ronald Rivest, Adi Shamir and Leonard Adleman. They proposed the asymmetric RSA cryptosystem in 1977. Until now, RSA is the most widely use asymmetric cryptosystem. It is mainly used for key transport and digital signatures. Just like we thought, RSA has two encryption keys, one is public key which is used to encrypt and another is private key which is used to decrypt. The public key can be a short integer but the private key needs to have the full length of the modulus. Its process to encrypt is more complicate than symmetric cryptosystem. Hence, due to the use of very long numbers, RSA is orders of magnitude slower than symmetric schemes, e.g., DES, AES.

Just a reminder, even having adopted strong algorithms, the cipher text could still disclose plain text information if an inappropriate mode is chosen. For example, if encryption algorithm is used in electronic codebook mode, identical plaintext blocks are encrypted into identical cipher text blocks, thus disclosing repetitive patterns. In database context, repetitive pattern are common because many records could have same attribute values, so much care should be taken when choosing the encryption mode.

* **Encryption Key**  

Figure 5. Encryption Algorithms: Asymmetric (left) and Symmetric (right)

The third factor is encryption keys. It is obvious that encryption typically uses a specified parameter or key to perform the data transformation. As we have mentioned in encryption algorithms and shown in Figure 4, symmetric cryptography and asymmetric cryptography are two main types of cryptography. According to these, we will know that symmetric keys and asymmetric keys are two basic types of encryption keys commonly used. Symmetric key is sometimes called private-key. This kind of encryption is the type where a single secret key is used for both encryption and decryption. Asymmetric key has another name, public-key. It is the type of encryption where a pair of secret keys is used. One of the keys is used for encryption and the other used for decryption.

* **Key Management**

Although the data is encrypted, the encryption key is not. Hence, the management for the key becomes a serious problem. How to secure keys from attacker of the system? How to give administrative rights of manipulating data using keys? And How to provide limited access for keys? It is also important to provide proper authentication mechanisms because without them, it is easy to get access to keys using social engineering techniques 5.

1. **Cryptographic Access Control**

Encryption key management is often a difficult problem to solve. Encrypting the whole database using the same key, even if access control mechanisms are used is not enough. For example, an insider has the encryption key and bypasses the access control mechanism. Then, the user can access data that are beyond his security group. If the key is stole by a hacker, the effect will be worse.

The suggestion is encrypting objects from different security groups using different keys ensures that a user who owns a specific key can decrypt only those objects within his security group9.

1. **Secure Key Storage**

The second part of managing keys is deciding where to store them. One easy solution is to store the keys in a restricted database table or file, potentially encrypted by a master key (itself stored somewhere on the database server). Then, all administrators with privileged access could also access these keys and decrypt any data within the system without ever being detected.

Recommended approach for storing the keys is, separate the keys and data residing in the database. First option is to store the keys in hardware like hardware security module (HSM)10, 11. Usually, the encryption keys are stored on the server encrypted by a master key which is stored in the HSM. At encryption/decryption time, using the master key, encrypted keys are dynamically decrypted by the HSM. Then, the keys remove from the server memory as soon as the cryptographic operations are performed, as shown in Figure 6a.

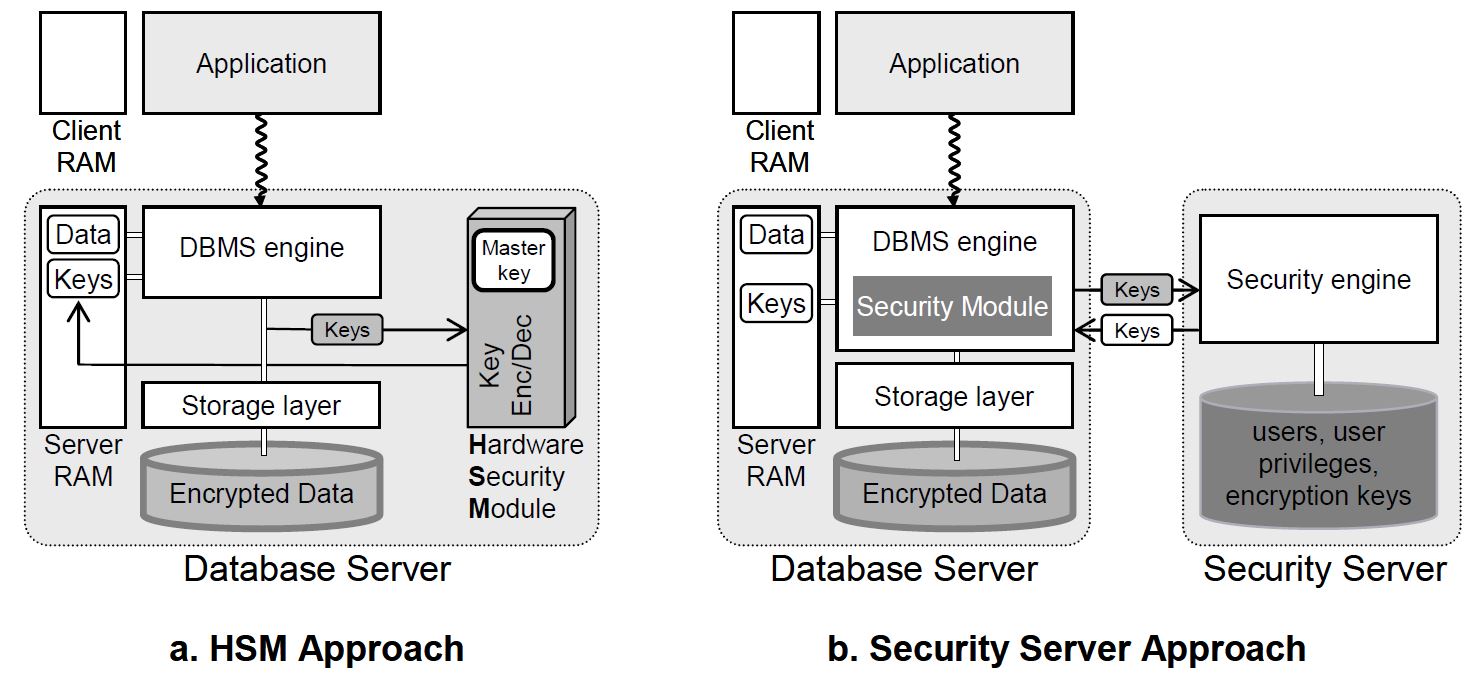


Figure 6. Key Management Approaches

An alternative solution is to move security-related tasks to distinct software running on a physically separate server that is called security server, as shown in Figure 6b. The security server then manages users, roles, privileges, encryption policies and encryption keys. Within the DBMS, a security module communicates with the security server to authenticate users, check privileges and encrypt or decrypt data. Encryption keys can then be linked to user or to user’s privileges. A clear distinction is also made between the role of the DBA, administering the database resources, and the role of the security administrator (SA), administering security parameters. The benefit is that an attack becomes more difficult because it requires a conspiracy between the DBA and SA.

**4. Applications**

* **Data Encryption within Database (Native Database Encryption)**

There are so many applications to encrypt the database. According to the aforementioned classification, some applications perform the encryption within the database and some perform it outside the database. The good examples which use encryption in the database are DBMS manufactures. Until now, almost every manufacture can provide its own native encryption capability. The forms of encryptions include encryption toolkits or packages, extension of SQL, or functions that can be embedded in SQL statements. To limit performance overhead, selective encryption can be generally done at the column level but may involve changing the database schema, such as Transparent Data Encryption (TDE). Next, we will introduce three most popular databases, Oracle, Microsoft SQL server, MySQL, and explain their encryption functions.

1. **Oracle**

Like other database, Oracle provides three encryption routines for the users to choose. They are Dbms\_Obfuscation\_Toolkit package, Dbms\_Crypto package and TDE11. In these routines, the concept is fundamentally simple. The data is fed along with the encryption key, and out pops the encrypted data. When the need comes to decipher the data, the encrypted information and the key both are fed into the routine, and the raw information comes out.

1. **Dbms\_Obfuscation\_Toolkit Package**

In Oracle 8i, the dbms\_obfuscation\_toolkit package was introduced to manually encrypt the data, especially sensitive data. Its algorithm is DES3.

Here is syntax for the Dbms\_Obfuscation\_Toolkit which it often uses DES3 to do encryption.

*DBMS\_OBFUSCATION\_TOOLKIT.DES3Encrypt(*

*input\_string IN VARCHAR2,*

*key\_string IN VARCHAR2,*

*which IN PLS\_INTEGER DEFAULT TwoKeyMode*

*iv\_string IN VARCHAR2 DEFAULT NULL) RETURN VARCHAR2;*

*DBMS\_OBFUSCATION\_TOOLKIT.DES3DECRYPT(*

*input\_string IN VARCHAR2,*

*key\_string IN VARCHAR2,*

*which IN PLS\_INTEGER DEFAULT TwoKeyMode*

*iv\_string IN VARCHAR2 DEFAULT NULL) RETURN VARCHAR2;*

1. **Dbms\_Crypto Package**

Since Oracle 10g, the Dbms\_Crypto package has been the replacement for the Dbms\_Obfuscation\_Toolkit package because this new one is easier to use and supports several industry-standard encryption and hashing algorithms, including the AES encryption algorithm. Additional, it can provide automatic padding and deprecate Dbms\_Obfuscation\_Toolkit.

|  |  |
| --- | --- |
| **Feature** | **Dbms\_Crypto Supported Functionality** |
| Cryptographic algorithms | DES, 3DES, AES, RC4,sDES\_2KEY |
| Padding forms | PKCS5, zeroes |
| Block cipher chaining modes | CBC, CFB, ECB, OFB |
| Cryptographic hash algorithms | SHA-1, SHA-2, MD4, MD5, HASH\_SH256, HASH\_SH384, HASH\_SH512 |
| Keyed hash(MAC) algorithms | HMAC\_MD5, HMAC\_SH1, HMAC\_SH256, HMAC\_SH384, HMAC\_SH512 |
| Cryptographic pseudo-random number generator | RAW, NUMBER, BINARY\_INTEGER |
| Database types | RAW, CLOB, BLOB |

Table 1. Dbms\_Crypto Package Feature Summary

Dbms\_Crypto could work on most common Oracle datatypes. The only one exception is the datatype of VARCHAR2. Before we use this package, we must use the UTL\_I18N.STRING\_TO\_RAW function to convert VARCHAR2 to the uniform database character set AL32UTF8, and then convert it to the RAW datatype. Here is the syntax example.

*UTL\_I18N.STRING\_TO\_RAW (string, 'AL32UTF8');*

When we convert RAW to VARCHAR2, these same steps must be taken in reverse order. Here is the syntax example.

*UTL\_I18N.RAW\_TO\_CHAR (data, 'AL32UTF8');*

|  |  |
| --- | --- |
| **Subprogram** | **Description** |
| Decrypt Function | Decrypts **RAW** data using a stream or block cipher with a user supplied key and optional IV (initialization vector) |
| Decrypt Procedures | Decrypts **LOB** data using a stream or block cipher with a user supplied key and optional IV |
| Encrypt Function | Encrypts **RAW** data using a stream or block cipher with a user supplied key and optional IV |
| Encrypt Procedures | Encrypts **LOB** data using a stream or block cipher with a user supplied key and optional IV |
| Hash Function | Applies one of the supported cryptographic hash algorithms (MD4, MD5, or SHA-1) to data |
| MAC function | Applies Message Authentication Code algorithms (MD5 or SHA-1) to data to provide keyed message protection |
| RandomBytes Function | Returns a **RAW** value containing a cryptographically secure pseudo-random sequence of bytes, and can be used to generate random material for encryption keys |
| RandomInteger Function | Returns a random **BINARY\_INTEGER** |
| RandomNumber Function | Returns a random 128-bit integer of the **NUMBER** datatype |

Table 2. Dbms\_Crypto Package Subprograms

This package enables encryption and decryption. Shown in table 2, Dbms\_Crypto package’s procedures are used to encrypt or decrypt LOB datatypes. In contrast, its functions are used to encrypt and decrypt RAW datatype.

Here is a simple encryption function using Dbms\_Crypto Package.

*create or replace function get\_enc\_val (*

*p\_in in varchar2,*

*p\_key in raw)*

*return raw is*

*l\_enc\_val raw (2000);*

*l\_mod number := dbms\_crypto.ENCRYPT\_AES128*

*+ dbms\_crypto.CHAIN\_CBC*

*+ dbms\_crypto.PAD\_PKCS5;*

*begin*

*l\_enc\_val := dbms\_crypto.encrypt(*

*UTL\_I18N.STRING\_TO\_RAW*

*(p\_in, 'AL32UTF8'),*

*l\_mod,*

*p\_key);*

*return l\_enc\_val;*

*end;*

Here is another simple decryption function using Dbms\_Crypto Package.

*create or replace function get\_dec\_val(*

*p\_in in raw,*

*p\_key in raw)*

*return varchar2*

*is*

*l\_ret varchar2 (2000);*

*l\_dec\_val raw (2000);*

*l\_mod number := dbms\_crypto.ENCRYPT\_AES128*

*+ dbms\_crypto.CHAIN\_CBC*

*+ dbms\_crypto.PAD\_PKCS5;*

*begin*

*l\_dec\_val := dbms\_crypto.decrypt(*

*p\_in,*

*l\_mod,*

*p\_key);*

*l\_ret:= UTL\_I18N.RAW\_TO\_CHAR(l\_dec\_val, 'AL32UTF8');*

*return l\_ret;*

*end;*

Now, both packages are complicated to use. The biggest inconvenience is users need to encrypt the data by themselves and need to think about the encryption key management. Although this package can generate random material for encryption keys, it does not provide a mechanism for maintaining them. This means that the user must find a way of storing and retrieving keys securely.

1. **Oracle TDE**

Is it possible that database encrypts the data by itself and users do not need to deal with encrypted data? Yes, it could. It is TDE. So, what is TDE? Unlike these cryptographic functions to encrypt some rows or columns, TED encrypts the whole database. The whole database is protected by a single key. TDE performs all of the cryptographic operations within the database system, and removes any need for users to create custom code to encrypt and decrypt data. It seems the data is not encrypted and users can use them freely but the fact is the data is. Meanwhile, to limit encrypt overhead, TDE can encrypt individual application table columns or entire application tablespaces.

TDE has been introduced in Oracle database 10g which is greatly enlarging the possibilities of using cryptography within the DBMS11. In Oracle 10.2, it is introduced column encryption. In Oracle 11.1, it is enhanced tablespace encryption. However, TDE is only available in Oracle Enterprise Edition.

To compare the table 1 of Dbms\_Crypto package feature summary, TDE has similar encryption algorithm as the package. For instance, TDE’s common encryption algorithm is AES, too.

|  |  |
| --- | --- |
| **Encryption algorithms** | **Hashing Algorithms (optional)** |
| AES   * Key length: 128, 192, 256 bits | SHA-1   * Digest length: 160 bits |
| 3DES   * Key length: 168 bits |  |

Table 3. Standard encryption and hashing algorithms used by TDE

Here is the implement for TDE.

* 1. Setup Wallet and Master Key
  2. Identify columns with sensitive data
  3. Review constraints
  4. Encrypt existing and new data

It is worth mentioning that TDE provides a two-tier encryption key management architecture consisting of data encryption keys and master encryption keys. Its master encryption keys can now be stored in an external file named Oracle Wallet which is encrypted using an administratively defined password. At the same time, version 12*c* introduces a new dedicated SYSKM role for optional delegation to a designated user account that may run all key management operations. Oracle Enterprise Manager provides a convenient graphical user interface for creating, rotating, and managing TDE master keys as shown in the figure below.



Figure 7. Managing and rotating TDE master keys using Oracle Enterprise Manager

A disadvantage to TDE is that the data is not protected from authenticated, authorized database users, including the DBA. A separate access control solution is necessary to protect the data from the DBA.

1. **Other tips**

Finally, as a reminder, besides encrypting the sensitive data, there are two more part must be encrypted12.

One is setting a TNS Listener password. The TNS Listener is the first component which an attacker will see. By default, the TNS Listener has no password set and can be administered remotely by anybody who can connect but after Oracle 10g this has changed. Setting a Listener password will prevent unauthorized administration of the Listener. Because this password is in clear text, and clear text passwords are not secure, it should be encrypted.

The second one is to enable database link login encryption. The SYS.LINK$ table contains credentials for remote database servers. Anybody who can select from this table will be able to view these credentials. As such it is better to have the credentials encrypted.

1. **Microsoft SQL Server**

In SQL Server 2000 or older version, there is no encryption package or function. Users only write their own functions to encrypt the data in some rows. In SQL Server 2005, it was the first time that some cryptographic functions were introduced to encrypt data in column level. Since SQL Server 2008, TDE had been introduced to the world13. Unlike Oracle TDE, MS SQL TDE can use the operating system, Windows to do encryption besides the database. Hence, its encryption Model adds the OS level.

1. **SQL Server Encryption Model**

SQL Server had different encryption keys which have different functions. Therefore, the SQL Server encryption model is users how to build their own encryption key management according to ANSI X9.17 standard. This standard defines several layers of encryption keys that are used to encrypt other keys. Then, these other key can encrypt actual data.

|  |  |  |
| --- | --- | --- |
| **SQL Server Layer** | **ANSI X9.17 Layer** | **Description** |
| SMK | Server Master key | The SMK is the top-level key used to encrypt the DMK. The SMK is encrypted by the Windows DPAPI. |
| DMK | Database Master key | The DMK is a symmetric key that’s used to encrypt a symmetric key, asymmetric key, and certificates. Only one DMK can be defined for each database. |
| Symmetric keys, asymmetric keys, and certificates | Database encrypting key | Symmetric keys, asymmetric keys, and certificates are used to encrypt data. |

Table 4. SQL Server and ANSI X9.17 Encryption Key Layers

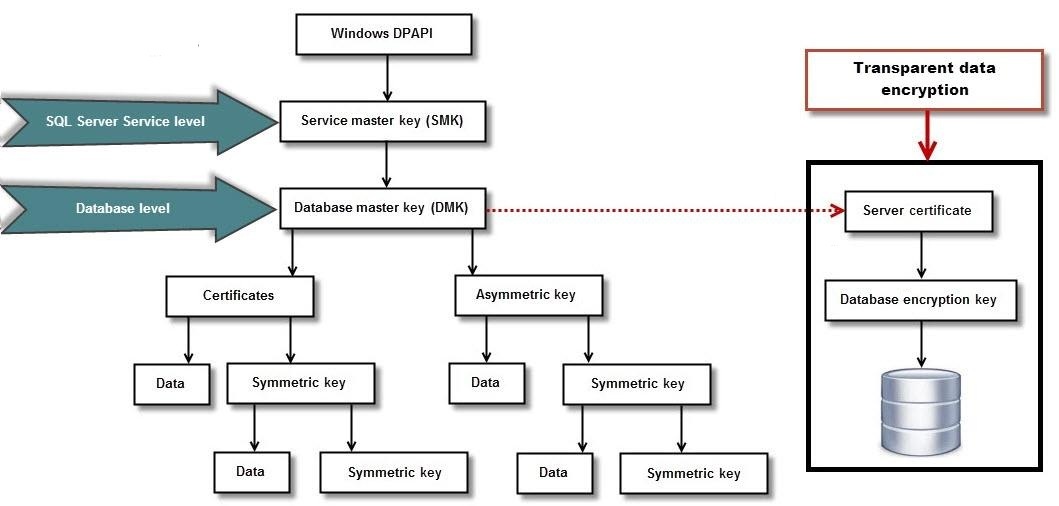


Figure 8. Encryption Key Hierarchy in SQL Server 2008 and Later14

There are three different levels of keys. The top-level key is the service master key (SMK). It is the father of all the keys in SQL Server. The SMK is an asymmetric key which is encrypted by the Windows Data Protection API (DPAPI). The SMK is automatically created the first time when you encrypt something and is tied to the SQL Server Service account. The SMK is used to encrypt the database master key (DMK).

DMK is the second layer and encrypts symmetric keys, asymmetric keys, and certificates. Each database can have only one DMK.

The third and last level of the encryption key hierarchy includes symmetric keys, asymmetric keys, and certificates. Symmetric keys are the main method to encrypt the data in the database. Microsoft recommends encrypting data with only symmetric keys. In addition, SQL Server 2008 and later version also introduce server-level certificates and database encryption keys to support TDE.

According to this hierarchy, it’s an obvious deduction that DMK is the key point. If the DMK is broken, these symmetric keys, asymmetric keys, and certificates which are built by it are all broken.

1. **Column-level Encryption**

In SQL Server 2005, the function of column-level encryption was introduced. Now, it is available in all editions of SQL Server, including the free SQL Server Express edition. Using the symmetric keys, asymmetric keys, and certificates, it can encrypt some specific columns. There are four different built-in functions for this kind of encryption and decryption, as follows.

* **EncryptByCert()** and **DecryptByCert()**, which use the public key of a certificate to encrypt and decrypt data
* **EncryptByAsymkey()** and **DecryptByAsymkey()**, which use an asymmetric key to encrypt and decrypt data
* **EncryptByKey()** and **DecryptByKey ()**, which use a symmetric key to encrypt and decrypt data
* **EncryptByPassphrase()** and **DecryptByPassphrase()**, which uses a passphrase to generate a symmetric key to encrypt and decrypt data

Besides these functions, there are two more decryption functions, as follows,

* **Decryptbykeyautoasymkey()**, which decrypts data by using a symmetric key that's automatically decrypted with an asymmetric key
* **Decryptbykeyautocert()**, which decrypts data by using a symmetric key that's automatically decrypted with a certificate

Honestly, there are advantages and disadvantages to use column-level encryption. The first benefit is it provides a more granular level of encryption which means you can encrypt a single column in a table. The second one is the data isn't decrypted until it's used. The disadvantage is that using this is a trouble thing. Firstly, the schema must be changed to varbinary. Then, it is reconverted to the desired data type. This means the application must be changed to support the encryption-decryption operation; in addition, it can affect performance. It is easy to be overhead in the system.

1. **SQL Server TDE**

SQL Server 2008 released TDE to encrypt an entire database on the hard disk. With TDE, databases can be secured without modifying existing applications, database structures, or processes. It is well suited for encrypting files or entire directories in an operating system context.

Like Oracle TDE, SQL Server TDE is available only in the SQL Server Enterprise edition. But, SQL Server TDE is totally different with Oracle one. It encrypts databases in real-time, as the records are written to the SQL Server database files (\*.mdf) and transaction log files (\*.ldf). Records are also encrypted in real-time during database backups and when snapshots are taken. SQL Server TDE encrypts the data before it's written to disk and decrypts it before it is retrieved. This process is totally transparent to the user or application because it's performed at the SQL Server Service layer.

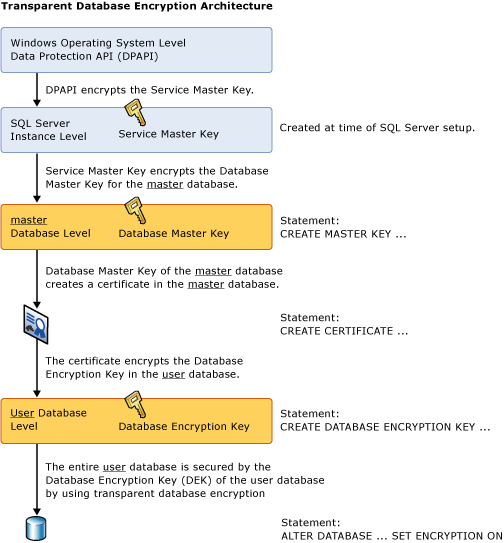


Figure 9. SQL Server TDE Architecture

With TDE, SQL Server encrypts a database using its database encryption key. This asymmetric key is stored in the database boot record, which is why it's always available during recovery.

As shown in Figure 9, the database encryption key is encrypted using the server certificate, which is encrypted using the Database Master Key of the master database. The master database's Database Master Key is encrypted using Server Master Key. The Server Master Key is an asymmetric key that's encrypted by the Windows DPAPI. The Server Master Key is automatically created the first time you encrypt something and is tied to the SQL Server Service account.

To use TDE, follow these steps.

* Create a master key
* Create or obtain a certificate protected by the master key
* Create a database encryption key and protect it by the certificate
* Set the database to use encryption

In the documentation of SQL Server Encryption, it shows an example which illustrates encrypting and decrypting the AdventureWorks2012 database using a certificate installed on the server named MyServerCert14.

*USE master;*

*GO*

*CREATE MASTER KEY ENCRYPTION BY PASSWORD = '<UseStrongPasswordHere>';*

*go*

*CREATE CERTIFICATE MyServerCert WITH SUBJECT = 'My DEK Certificate';*

*go*

*USE AdventureWorks2012;*

*GO*

*CREATE DATABASE ENCRYPTION KEY*

*WITH ALGORITHM = AES\_128*

*ENCRYPTION BY SERVER CERTIFICATE MyServerCert;*

*GO*

*ALTER DATABASE AdventureWorks2012*

*SET ENCRYPTION ON;*

*GO*

When the users enable or disable TDE, the encryption and decryption operations are scheduled on background threads by SQL Server. Users can view the status of these operations using the catalog views and dynamic management views.

As a reminder, because TDE encrypts data and decrypts it through the buffer pool, data in the buffer pool remains there in clear text format. Hence, if the users want to protect data in the buffer pool with encryption, they need to choose a different technique.

1. **MySQL**

Unlike Oracle or MS SQL Server, MySQL does not offer any encryption routines. To encrypt or decrypt the data in MySQL, the users can choose encryption functions for specific fields. The most common solution is the users use Aes\_Encrypt() to encrypt the data and Aes\_Decrypt() to decrypt it, which used the AES algorithm & encode data with a 128-bit key length. As we mentioned before, AES is much better than DES. Hence, starting with MySQL 5.7.6, the functions which includes Des\_Encrypt(), Des\_Decrypt() and Encrypt() are not available any more15.

| **Name** | **Description** |
| --- | --- |
| **Aes\_Decrypt()** | **Decrypt using AES** |
| **Aes\_Encrypt()** | **Encrypt using AES** |
| Compress() | Return result as a binary string |
| Decode() | Decodes a string encrypted using ENCODE() |
| Des\_Decrypt() | Decrypt a string (not available in MySQL 5.7.6) |
| Des\_Encrypt() | Encrypt a string (not available in MySQL 5.7.6) |
| Encode() | Encode a string |
| Encrypt() | Encrypt a string (not available in MySQL 5.7.6) |
| Md5() | Calculate MD5 checksum |
| Old\_Password() | Return the value of the pre-4.1 implementation of PASSWORD |
| Password() | Calculate and return a password string |
| Random\_Bytes() | Return a random byte vector |
| Sha1(), Sha() | Calculate an SHA-1 160-bit checksum |
| Sha2() | Calculate an SHA-2 checksum |
| Uncompress() | Uncompress a string compressed |
| Uncompressed\_Length() | Return the length of a string before compression |
| Validate\_Password\_Strength() | Determine strength of password |

Table 5. MySQL Encryption Functions

At the same time, if users need to connect the network, encrypt the connection between clients and the server using the Secure Sockets Layer (SSL) protocol because MySQL traffic is not encrypted by default. To configure this, use the --ssl option, and other associated SSL options. The best way to ensure that an SSL connection is actually being used is to specify the REQUIRE SSL clause in the GRANT statement that creates a user, or to manually set the ssl\_type field in the mysql.user table12.

MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. Another advantage is it can use combination of public, private, and symmetric keys to encrypt and decrypt data.

* **Data Encryption outside Database (Third-Party Vendors’ Encryption)**

Apart from encrypting the data inside the database, the users can choose data encryption outside database. Because the best practice for implementing security solutions involves a layered approach, an encryption solution from a third party on top of database server can provide the in-depth defense their database needs. At the same time, these third-party vendors’ products can adapt to most DBMS engine.

Most of third-party vendors have similar functions. Each one offers its own patented system that allows implementation of encryption down to the column level within the database. Meanwhile, they can monitor the data and give the encrypted data a dynamic mask. It sounds that they have the similar functions like the DBMS-based TDE. They are also called TDE. However, as mentioned in our paper’s classification, they definitely have some differences. Ashvin Kamaraju, a vice president of a third-party vendor, Vormetric issued an article in 2011 and compared these two kinds of TDE.16

As for DBMS-based TDE, the best benefit is it is bundled with the database solution. At first glance, it is very simple to use and offer better granularity of encryption. The big inconvenience is the user need to think about the encryption key management for the data safety. As usually, encryption keys are typically co-located with data on the same database server. The user need to install another layer of controls to separate the management of the data and the encryption keys, just like we talked about Oracle TDE and MS SQL Server TDE. Another drawback is DBMS-based TDE only apply on its own database. If a company use more than two different DBMS, they will need numerous methods of encryption and key management.

These native database TDE’s disadvantages just become third-party vendors TDE’s advantages.

* Data protection across platforms: they offer a transparent, file-based approach to encryption that can protect both the data in the database and the data outside the database such as pdf files, spreadsheets and reports.
* Separation of duties: they have centralized key management systems with secure storage, life cycle management, auditing and separation of duties between the system, database and security administrators. They ensure that keys are provisioned only to authorized personnel and applications.
* Separation of functionality: The database and encryption functionality are independent of each other so that database resources are not needed for encryption and decryption. This enables databases to operate at optimal performance while encryption remains transparent to the database as well as its users.

As we mentioned before, the disadvantages for them are affecting the database performance and cost more money to encrypt the data.

1. **Data Encryption’s limitations**

At the beginning, we discuss the reason why we need to encrypt the data. However, there are still some limitations for encryption. Sometimes, encryption does not solve all security problems, and may even make some problems worse.

Here are some misconceptions about encryption of stored data17.

* Encryption Does Not Solve Access Control Problems

Encryption means no one can read the data unless the user has the encryption key. It cannot give different user different authority to review the file. This function belongs to access control.

* Encryption Does Not Protect Against a Malicious DBA

The DBAs who have all privileges are able to see all data in the database even these files are encrypted. The correct solution is to protect the DBA account, and to change default passwords for other privileged accounts.

* Encrypting Everything Does Not Make Data Secure

As the prior discussion, encryption all data will significantly affect performance. Even using the best algorithm to encrypt the data, there still exits some possibility that it is broken. The best way to solve it is backup the data at regular intervals and move the old one out of the database.

* **Conclusion**

Encryption is the process of obfuscating data by the use of a key or password. This can make the data useless without the corresponding decryption key or password. It is one basic and important method to secure the data. The users can choose encrypt data inside or outside the database. This paper describes some applications’ encryption functions according to this classification. However, there is no perfect application. Each one has some advantages and some disadvantages. At the same time, this paper reminder users that encryption has some limitations. For instance, encryption does not solve access control problems. It is a backup technique to protect the data. When users think about the secure plan, it should be a combination which includes encryption and other fields.

*Work Cite:*

1. David Goldman, “*More than 6 million LinkedIn passwords stolen*”, [*http://money.cnn.com/2012/06/06/technology/linkedin-password-hack/*](http://money.cnn.com/2012/06/06/technology/linkedin-password-hack/)
2. An Baijie (China Daily), “*The world wide web of deception*”, [*http://usa.chinadaily.com.cn/epaper/2011-12/30/content\_14357229.htm*](http://usa.chinadaily.com.cn/epaper/2011-12/30/content_14357229.htm)
3. Alfred Basta, Melissa Zgola, “*Database Security*”
4. RSA Security Inc., “*Securing Data at Rest: Developing a Database Encryption Strategy, A White Paper for developers, e-business managers and IT*”, http://www.provision.ro/threat-management/database-security/database-encryption#pagei-1|pagep-1|
5. Beena Sachan, “*Developing A Database Encryption Strategy: Both Inside And Outside The DBMS*”, International Journal of Research in Engineering, IT and Social Sciences, Volume1, Issue 2 ISSN: 2250‐0588
6. Bijit Hore, Sharad Mehrotra, and Hakan Hacigumus, “*Handbook of Database Security Applications and Trends - Managing and Querying Encrypted Data*”, Springer Science+Business Media, LLC. 2008
7. “Term: encryption algorithm”, <http://www.webopedia.com/TERM/E/encryption_algorithm.html>
8. Wade Trappe and Lawrence C. Washington. “*Introduction to Cryptography with Coding Theory*”, 2nd Edition, Pearson Prentice Hall, 2006
9. Bertino E, Ferrari E (2002) Secure and Selective Dissemination of XML Documents. ACM Transactions on Information and System Security, 5(3), 290-331.
10. Sung Hsueh, Database Encryption in SQL Server 2008 Enterprise Edition, SQL Server Technical Article, 2008. <http://msdn.microsoft.com/enus/library/cc278098.aspx>.
11. Oracle Corporation, Encryption and Redaction in Oracle Database 12c with Oracle Advanced Security, White Paper June 2013
12. David Litchfield, “*The Database Hacker's Handbook: Defending Database Servers*”, John Wiley & Sons, 2005.
13. Books Online for SQL Server 2016: SQL Server Encryption, <https://msdn.microsoft.com/en-us/library/bb510663.aspx>
14. “SQL Server Encryption”, <http://sqlmag.com/database-security/sql-server-encryption>
15. “12.13 Encryption and Compression Functions”, https://dev.mysql.com/doc/refman/5.7/en/encryption-functions.html
16. Ashvin Kamaraju, “*Native DB Encryption Versus Third-Party Enterprise Encryption--What's the Difference*?” Database Trends & Applications;Dec2011, Vol. 25 Issue 4, p22
17. “Developing Applications Using Data Encryption”, <http://docs.oracle.com/cd/B14117_01/network.101/b10773/apdvncrp.htm>