**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 applications which include three popular database manufacturers, Oracle, MySQL, Microsoft SQL server, and some third party vendors, Protegrity, RSA BSAFE, SafeNet. 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. Hence, 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.

Over the last two decades, database security generates two main methods, access controls and encryption to protect data. 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**

Encryption is “*the transformation of data by using sophisticated algorithms in an attempt to make the data unrecognizable*3.” Figure 1 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.

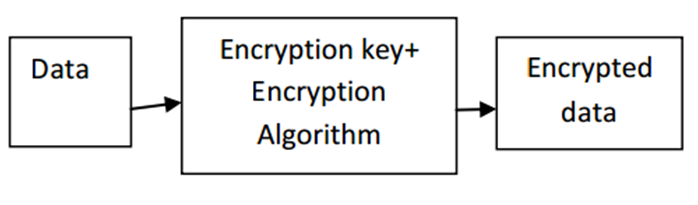


Figure 1: Basic Encryption Processes

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 different advantages and disadvantages.

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.

|  |  |  |
| --- | --- | --- |
|  | Data Encryption  Within Database | Data Encryption  Outside Database |
| Performer | DBMS | Specialized Encryption Server |
| Transparent Data encryption | Yes | No |
| Effect for Server’s Function | Affect | No |
| Encryption Key Management | Stored in database  Big Risk | Special protection  Small Risk |
| Effect for DBMS Function | No | Affect some functions  (e.g. index) |
| Encryption Algorithm | Limit | Many |

Table 1: Comparing Data Encryption within and outside Database

**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 (1) table granularity, each table is encrypted separately, (2) page granularity, a smaller achievable granularity, each page is encrypted separately, a typical page in SQL Server is 8 Kbytes and might contain one or multiple records, (3) row granularity, each row in a table is encrypted separately, and (4) field granularity, the smallest achievable granularity based on the field of a record8.

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 performed.

* **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*”4. The algorithms which are generally used for database encryption and often supported by database management system are DES, Triple DES, RC2, RC4, DESX and AES. Different algorithm has different encryption effect.

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**

The third one is encryption keys. It is obvious that encryption typically uses a specified parameter or key to perform the data transformation. There 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.



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

* **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 2.a.

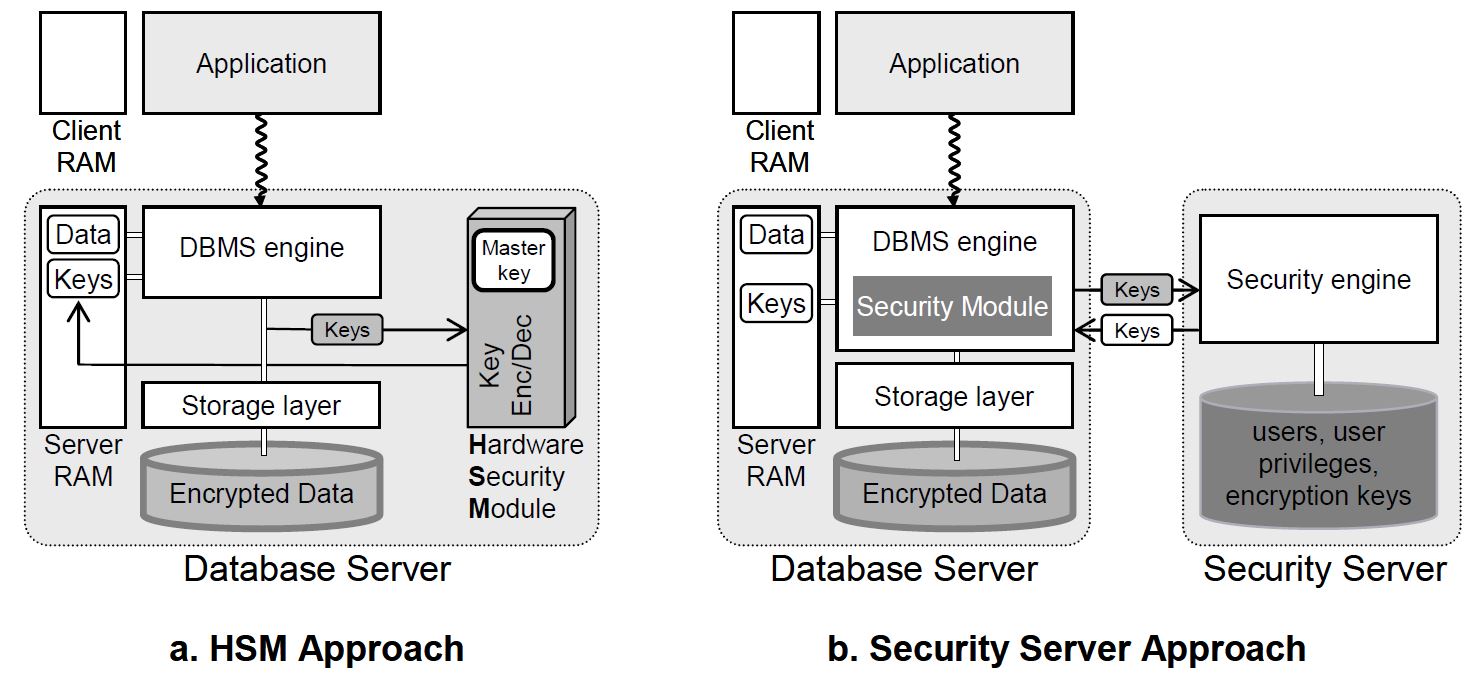


Figure 2: Key Management Approaches

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 2.b. 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**

There are so many applications to encrypt the database. According to the aforementioned classification, some applications perform the encryption within the database, such as DBMS manufactures. Almost every manufacture can provide its own native encryption capability. Next, we will first introduce three of them, Oracle, MySQL and Microsoft SQL server.

1. **Oracle**
2. Oracle TDE

Transparent Data Encryption (TDE) has been introduced in Oracle database 10g and the newest version is 12c which is greatly enlarging the possibilities of using cryptography within the DBMS11.

The solution is transparent to applications because data is encrypted automatically when written to storage and decrypted when read from storage. Users do not need to deal with encrypted data. To limit encrypt overhead, TDE can encrypt individual application table columns or entire application tablespaces.



Figure 3: Standard encryption and hashing algorithms used by TDE

TDE’s common encryption algorithm is AES. 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: 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. DBMS\_CRYPTO

However, TDE is only available in Oracle Enterprise Edition. As a normal user, you can use the DBMS\_CRYPTO PL/SQL package to manually encrypt data, such as sensitive data.

This package enables encryption and decryption. Its procedures are used to encrypt or decrypt LOB datatypes. In contrast, its functions are used to encrypt and decrypt RAW datatype. 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 convert it to the uniform database character set AL32UTF8, and then convert it to the RAW datatype.

Table DBMS\_CRYPTO Package Feature Summary

As we see, in the table of DBMS\_CRYPTO package feature summary, this package has the same encryption algorithm like TDE.

The DBMS\_CRYPTO package can generate random material for encryption keys, but it does not provide a mechanism for maintaining them.

Key management is programmatic. That is, the application (or caller of the function) must supply the encryption key. This means that the application developer must find a way of storing and retrieving keys securely. The relative strengths and weaknesses of various key management techniques are discussed in the sections that follow. The DES algorithm itself has an effective key length of 56-bits.

Here are two security recommendations for normal users in encryption part12.

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. To set a password, edit the listener.ora file and add the following line:

*PASSWORDS\_listenername = t1n5eLt0wn*

Stop and restart the Listener. Because this password is in clear text, and clear text passwords are not secure, it should be encrypted. To do this is, connect to the Listener using the Listener Control Utility — lsnrctl:

*LSNRCTL> set current\_listener 10.1.1.100*

*Current Listener is listener*

*LSNRCTL> change\_password*

*Old password:*

*New password:*

*Reenter new password:*

*Connecting to (DESCRIPTION=(ADDRESS=(PROTOCOL=IPC)(KEY=EXTPROC0)))*

*Password changed for listener*

*The command completed successfully*

*LSNRCTL> set password*

*Password:*

*The command completed successfully*

*LSNRCTL> save\_config*

*Connecting to (DESCRIPTION= (ADDRESS= (PROTOCOL=IPC) (KEY=EXTPROC0)))*

*Saved LISTENER configuration parameters.*

*Listener Parameter File C:\oracle\ora92\network\admin\listener.ora*

*Old Parameter File C:\oracle\ora92\network\admin\listener.bak*

*The command completed successfully*

*LSNRCTL>*

This will set the password in the listener.ora file to an encrypted password.

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. **MySQL**

First: You store your key with the application and handle all encryption at the application layer.

Next: you ensure that the MySQL instance and the application [server] are on separate machines so that a root compromise on the MySQL server doesn't allow the attacker to read the key from application source.

This approach seems excessive. Handle sensitive data properly (passwords, credit cards, etc) but encrypting everything is overkill. (And likely counter productive in the world of primary keys)

Best practices for protecting network connections

* Encrypt your connection to the server using SSH or SSL

1. **Microsoft SQL server**

SQL Server 2008 [14] introduces transparent data encryption (TDE) which is actually very similar to storage-level encryption. The whole database is protected by a single key (DEK for Database Encryption Key), itself protected by more complex means, including the possibility to use HSM. TDE performs all of the cryptographic operations at the I/O level, but within the database system, and removes any need for application developers to create custom code to encrypt and decrypt data.

* **Data Encryption outside Database**

The database-level encryption with security server approach mentioned above is proposed by IBM DB2 with the Data Encryption Expert (DEE [5]) and by third-party vendors like Protegrity [6], RSA BSAFE [17] and SafeNet [19] (appliance-based solution). The third-party vendors’ products can adapt to most DBMS engine (Oracle, IBM DB2, SQL Server and Sybase).

*Work Cite:*

1. David Goldman, “More than 6 million LinkedIn passwords stolen”, <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>
3. Alfred Basta, Melissa Zgola, “Database Security” P5-7
4. <http://www.webopedia.com/TERM/E/encryption_algorithm.html>
5. RSA Security Inc., “Securing Data at Rest: Developing a Database Encryption Strategy, A White Paper for developers, e-business managers and IT‖, Website, September 7 2012, http://www.rsa.com/products/bsafe/whitepapers/DDES\_WP\_0702.pdf
6. Ernesto Damiani, S. De Capitani Vimercati, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati, Balancing confidentiality and efficiency in untrusted relational dbms, Proceedings of the 10th ACM conference on Computer and communications security, ACM, 2003, pp. 93-102.
7. Luc Bouganim and Philippe Pucheral, Chip-secured data access: confidential data on untrusted servers, Proceedings of the 28th international conference on Very Large Data Bases. 2002, pp. 131-142.
8. Intelligence and Security Informatics: International Workshop, WISI 2006, Singapore, April 9, 2006, Proceedings. edited by Hsinchun Chen, Feiyue Wang, Christopher C. Yang, Daniel Zeng, Michael Chau, Kuiyu Chang. pp. 80
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. Encryption and Redaction in Oracle Database 12c with Oracle Advanced Security, White Paper June 2013
12. The Database Hacker's Handbook: Defending Database Servers by David Litchfield et al. John Wiley & Sons © 2005,
13. “Encrypt Decrypt”, <http://sql-tuning.com/oracle-encrypt-decrypt/>