SANS ANALYST PROGRAM

Sponsored by Oracle

Transparent Data Encryption: New Technologies and Best Practices for Database Encryption

A SANS Whitepaper – April 2010

Written by Tanya Baccam, SANS senior instructor
and course author for SEC509: Oracle Database Security

Encryption 101

Data Encryption Architectures

Introduction

Encryption is a key control that receives a lot of attention in organizations today. Organizations know they need to encrypt sensitive and regulated data. Encryption can help prevent data loss or theft, as well as prevent fraud within an organization. In some cases, encryption is also used to meet regulatory requirements for consumer data protection.

Many organizations have felt the pain when encryption controls haven't been implemented the way they should be within the organization, and data has been lost. There have been many situations over the years in which backup tapes have gone missing, either lost of stolen, and their sensitive data was not encrypted. A server can also be compromised, resulting in information being leaked. In any of these cases, there may be regulatory requirements to report the data leaked. To this end, more and more organizations are seeking to protect data, just in case it is leaked, by encrypting data within the environment (see Figure 1).

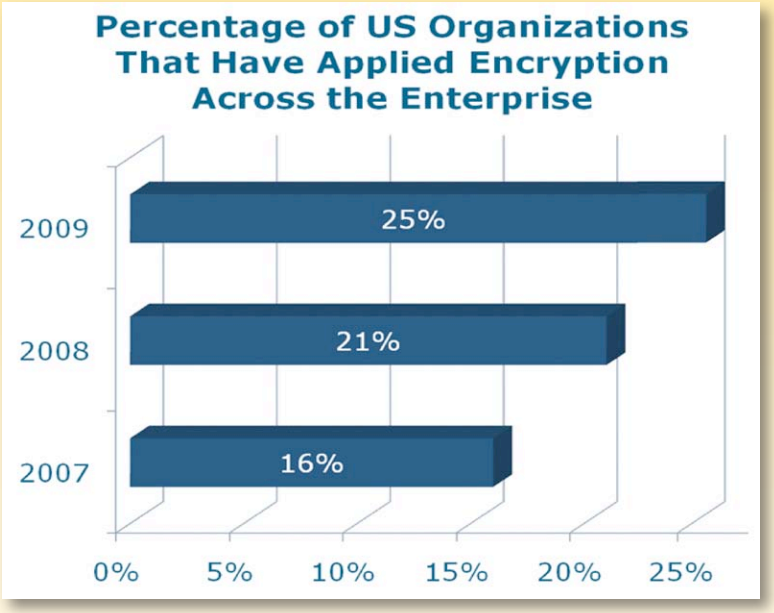

Figure 1: Encryption Usage[1]

When it comes to encrypting data at the database level, there are many areas in which encryption is applied and managed. Throughout this paper, we will look at the basics of encryption and discuss the pros and cons of leading encryption architectures available today.

Encryption is not a magical solution and cannot solve all problems, but it can mitigate many of the security risks organizations face. It is critical to understand what encryption does and does not provide within an organization so we can properly manage risk to our sensitive and regulated data.

[1] Data taken from the US 2009 Annual Study posted at www.encryptionreports.com/encryptiontrends.html

Encryption 101

A prerequisite to proper implementation of any encryption solution mandates an understanding of how encryption works. Key components related to encryption that security professionals need to understand include data at rest versus data in transit, algorithms and Key management

Data at rest and in transit. Data needs to be protected in two states. Data can exist either at rest or in transit. For example, for data at rest, the data may be stored in a database or on a backup tape. For data in transit, the data is traveling across the network, which dictates different encryption solutions for the data in transit. Database encryption can solve some of the issues related to data at rest. However, for data in transit, you might need to leverage a solution such as SSL/TLS .

Algorithms. Cryptographic algorithms can generally be grouped into two different categories:

1. Symmetric key cryptosystems, which use the same key to both encrypt and decrypt the communication
2. Asymmetric cryptosystems, which use two different keys instead of a single key — one key to encrypt the communication and another to decrypt the communication

There are advantages to using each type of algorithm. The advantage of symmetric algorithms is that they tend to be faster than asymmetric algorithms. However, the disadvantage is that key management can be more difficult. Because the same key is used to encrypt and decrypt the data, anyone who has the key for encryption can use the same key to decrypt any of the data that has been encrypted. Typical examples of symmetric algorithms in use today include Triple DES and AES (Advanced Encryption Standard).

Asymmetric cryptography is also known as *public key cryptography* and relies on the use of two unique keys—the public key and the private key. The public key is used to encrypt data and cannot be used to decrypt data. Only the private key can decrypt the data. Therefore, the keys work as a pair and are often referred to as a *key pair*. The public key can be given to anyone who wants to encrypt data, but the private key must be kept confidential because it provides the capability to decrypt the data. Asymmetric algorithms rely on extremely complicated algorithms and, therefore, are generally slower than symmetric algorithms. However, with asymmetric cryptography, key management can be easier to administer. Because different keys are being used to encrypt and decrypt data, the encryption key can be provided to anyone without a risk of them being able to decrypt the communication.

Many times, today, we use asymmetric algorithms to encrypt the symmetric key. Additionally, we often implement digital signatures using asymmetric algorithms because the private key is only available to the owner of the keys. Examples of asymmetric algorithms in use today include RSA and El Gamal. Figure 2 shows how these algorithms work similarly.

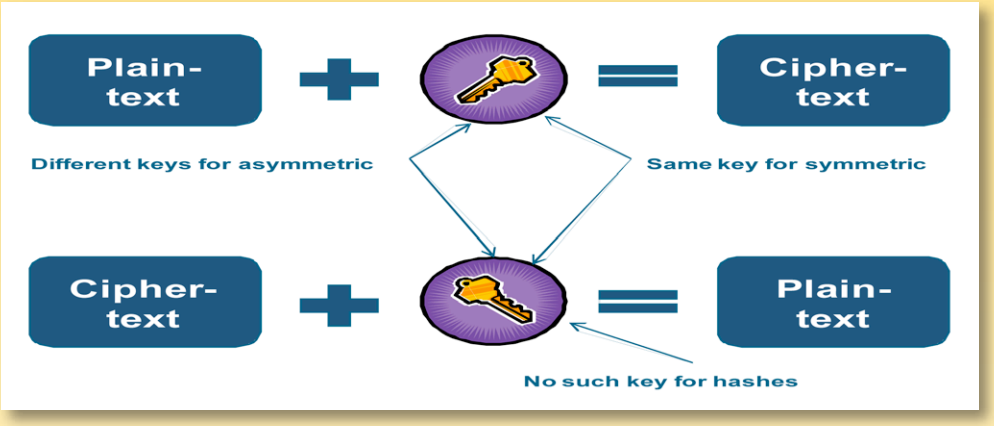

Figure 2: Encryption Algorithms: Asymmetric (left) and Symmetric (right)

When it comes to selecting algorithms for encryption purposes, leverage the algorithms that are commonly accepted and utilized within the industry.

Key management. Key management is a big concern with encryption, because the effectiveness of the solution ultimately depends on protecting the key. If the key is exposed, the data being protected with the key is, essentially, exposed. Wherever the key is stored, it must be protected, and it should be changed on occasion. For example, if an administrator with access to a key leaves an organization, the key should be changed.

Data Encryption Architectures

For data at rest within the database, there are multiple encryption options, each with its advantages and disadvantages. In this section, we look at the following architectural solutions:

- Application encryption
- File/Disk encryption
- Database encryption

Application Encryption

You can task a given application with encrypting its own data. This encryption capability is designed into the application itself, and organizations will not have to add another solution for encrypting data across the network. By the time the database receives the data, it has already been encrypted and then stored in the database in this encrypted state. As the traffic travels from the application to the database, the data can also be encrypted across the network (see Figure 3). Whether these benefits are realized depends on whether the solution has been implemented at the application layer or the database layer via an API. Communication from the client to the application needs an additional solution for encryption purposes.

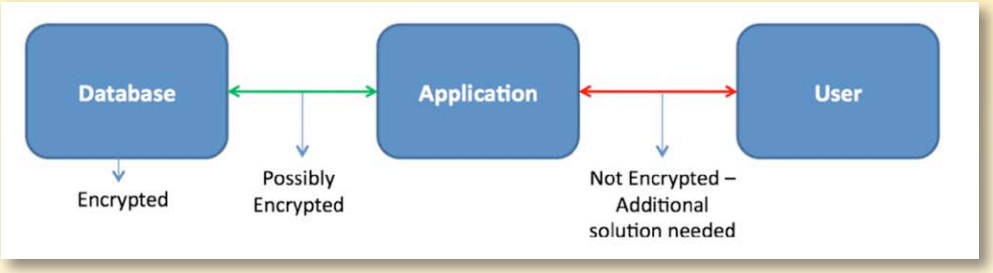

Figure 3: Application Encryption

It should be noted that the user-to-application communication channel needs additional encryption implemented, as well.

A benefit to application encryption is that the data is only accessible to authenticated, authorized application users. If an attacker, whether an insider or outsider, tried to access the data directly within the database without going through the application, the data would be encrypted and inaccessible.

The disadvantages to application encryption include:

1. First, to implement application encryption you must make significant changes in both the application layer and the database layer. Applications accessing the data need to be modified to understand and implement encryption. This could mean changing literally hundreds of applications for some organizations. In addition, the database tables and views that reside in the database and support the application need to be changed because the values being stored will no longer match the external data type representation. For example, a nine-digit SSN could not be encrypted and stored in the same field or data type that was originally used to store the unencrypted SSN. Complicating the situation further is the fact that many organizations do not even know all the applications that may be accessing the data. Some applications, such as legacy applications, may also make it extremely difficult, if not impossible, to implement this solution.
2. Second, database performance issues may arise because external applications control the encrypted data within the database. For example, if the application layer is doing the encryption, indexes and search capabilities within the database will not work. Alternatively, a database layer encryption solution can be implemented using an API, but that requires applicable triggers and views, which also introduce additional overhead, thus affecting database performance.
3. Finally, consider key management. Because multiple applications may be doing the encryption then sending to the database, keys are stored in many locations, introducing additional exposure points. When it is time to change the key, this may mean decrypting all the data with the old key and re-encrypting all the data with the new key—a significant impact to the environment.

File/Disk Encryption

Another encryption option is file or disk encryption. Typically, file or disk encryption requires operating system support hooks to encrypt database files or the media on which the files reside. In these cases, keys typically must be managed by system or storage administrators. With file and disk encryption, there is no need to change the application(s) to accommodate encryption to the database with application encryptoin. In addition, file/disk encryption is a concept that organizations tend to be familiar with because it is similar to laptop encryption that many organizations have already implemented.

One of the disadvantages of file/disk encryption is that it typically does not support separation of duties between the system administrator and DBA (database administrator). Some vendors have implemented additional solutions to overcome this weakness. And organizations are compensating with other tools, for example using intrusion prevention systems (IPS) to monitor access to the keys themselves.

Because the encryption is occurring at the file or disk level, everyone with access to the operating system typically has access to all the data encrypted on that system. In such cases, an external key management infrastructure is often implemented; however, it is impossible to manage multiple keys at the database file or table column level, only at the whole disk level. In addition, operating system backups still require another solution because a system administrator can see the data unencrypted during backup routines.

Database Encryption

The final architectural solution is to use database encryption. Database encryption falls somewhere between application encryption and file/disk encryption. As an example, we'll discuss Oracle Advanced Security's transparent data encryption (TDE), which automatically encrypts and decrypts the data stored in the database and provides this capability without having to write additional code.

With TDE, the encryption process and associated encryption keys are created and managed by the database. This is transparent to database users who have authenticated to the database. At the operating system, however, attempts to access database files return data in an encrypted state. Therefore, for any operating system level users, the data remains inaccessible. Additionally, because the database is doing the encryption, there is no need to change the application(s), and there is a minimal performance overhead when changes occur in the database. TDE is designed into the database itself: Oracle has integrated the TDE syntax with its data definition language (DDL) so that encryption can be specified directly in the **CREATE Table**, **ALTER Table**, and **CREATE TABLESPACE** syntax. TDE can also be managed using Oracle Enterprise Manager.

In addition, existing Oracle database processes will backup data protected with TDE without any modification. In other words, if a column or tablespace is encrypted, the same data will remain encrypted when it is backed up. This is a significant side benefit of database encryption because no additional time is required during the backup process to re-encrypt the database. The only additional requirement is to make sure that the TDE master key is backed up to a secure location separate from the backup medium—a requirement that exists for all encryption scenarios.

Oracle Data Pump is also integrated with TDE such that database exports can be encrypted using either the TDE master key or a pass phrase.

With TDE, you might be able to choose to encrypt an entire tablespace or just certain columns, depending on multiple factors:

- Can you identify where the sensitive data is located? If not, tablespace encryption might be the better solution because it encrypts the whole table rather than just what currently is considered a sensitive data type.
- Are you encrypting a lot of data? If so, it might make more sense to use tablespace encryption.
- If you are only encrypting a few columns, column-level encryption might be a better choice because tablespace encryption might be overkill.
- Is sensitive column data used as a foreign key? If so, leverage tablespace encryption as it also has separate mechanisms for key encryption.
- Could the data you consider to be sensitive change over time? If so, it might be easier to use tablespace encryption and encrypt everything from the beginning.
- Are there limitations based on the database being used? Some databases only support full tablespace encryption. In other instances, if you want to be able to re-key or use external hardware security module (HSM) support, you might only be able to use column-level encryption. Be sure to take database-specific considerations into account.

A disadvantage to TDE is that the data is not protected from authenticated, authorized database users, including the DBA. A separate access control solution, such as Oracle Database Vault, is necessary to protect the data from the DBA. Similar to the other encryption types we discussed, TDE addresses data at rest, but does not address data in transit. Remember, for any of the architectural options, there is some additional data in transit that must be protected with another tool. You need layers of protection, including access controls, data masking and other techniques to keep data protected from privileged users and to encrypt data in motion, in development/test environments and in long-term storage.

The following is an illustration (see Figure 4) of how Oracle Advanced Security TDE is implemented. First, a DBA or designated security DBA must open the TDE master key by supplying a password. By default, the master key is stored in an Oracle Wallet, a **PCKS#12** file, located on the operating system. Alternatively the master key can be stored on an HSM. The master key is very important because it is used to protect additional encryption keys stored inside the Oracle database in what is known as *2-tier key architecture*.

For column encryption, TDE creates an encryption key for each table containing an encrypted column. The resulting encryption key is encrypted using the master key and stored in the Oracle data dictionary.

For tablespace encryption, TDE encrypts the underlying tablespace or files that make the database. The encryption keys for the tablespace are encrypted using the TDE master encryption key. It is critical that you backup the TDE master key after it is first initialized and whenever it is changed. If lost, the TDE master key cannot be recovered.

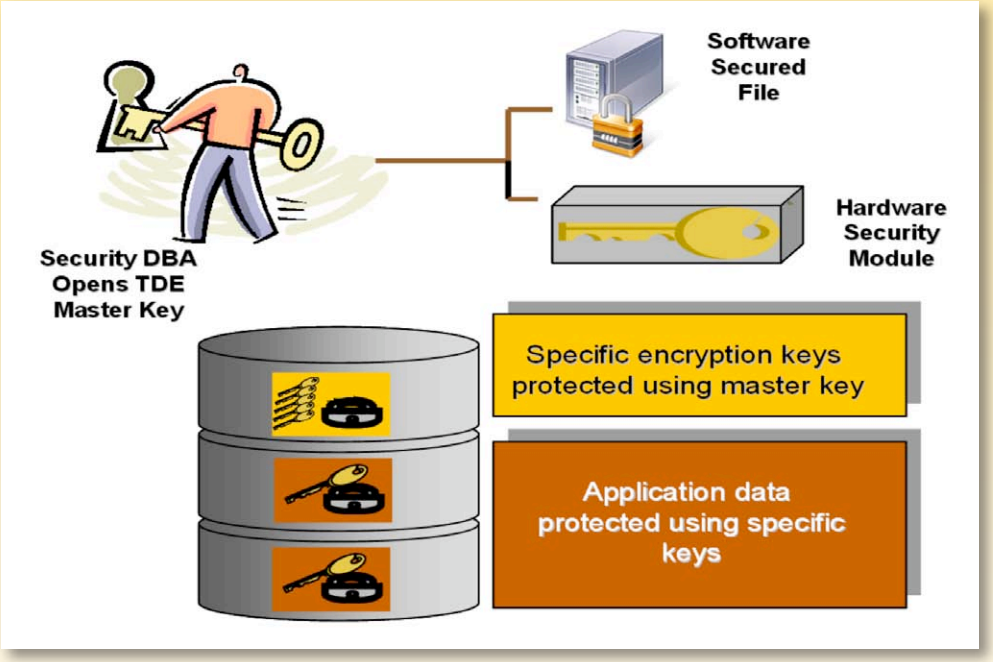

Figure 4: Oracle TDE Architecture

Conclusion

In conclusion, encryption is becoming a requirement for any organization dealing with sensitive, regulated data. Encryption can solve many of data security risks we face today, but it is not a silver bullet that will solve all of our problems. There are multiple options when it comes to encrypting data being stored in a database. It is important to understand the advantages and disadvantages of the solution you choose to implement and to compensate for any disadvantages or security risks that remain.

Make sure you consider a defense in depth approach that goes beyond encrypting data at the database level and address other security risks such as access control, masking, configuration management, and others. (See our SANS Analysts Program paper on Oracle Database Security.)[2]

Armed with the information in this paper, you should be able to ask the key questions that must be answered when implementing an encryption solution in order to ensure that the solution meets the needs of your organization.

[2] www.sans.org/reading_room/analysts_program/oraclewhitepaper_201004.pdf

About the Author

Tanya Baccam is a SANS senior instructor as well as a SANS courseware author. She is the current author for the SANS Security 509: Securing Oracle Databases course.[3] She works for Baccam Consulting, where she provides many security consulting services for clients, including system audits, vulnerability and risk assessments, database audits, and web application audits. Today much of her time is spent on the security of databases and applications within organizations. Tanya has also played an integral role in developing multiple business applications. She currently holds the CPA, GCFW, GCIH, CISSP, CISM, CISA, and OCP DBA certifications.

[3] www.sans.org/security-training/securing-oracle-74-mid

SANS would like to thank its sponsors:

ORACLE®