International Conference on Information & Communication Technologies: From Theory to Applications, 19-23 Apr. 2004, Damascus, Syria

A New Image-Database Encryption Based On A Hybrid Approach of Data-at-Rest and Data-in-Motion Encryption Protocol[1]

Ooi Bee Sien, Azman Samsuddin, and Rahmat Budiarto
School of Computer Science, Universiti Sains Malaysia
11800, Penang, Malaysia
bsien@yahoo.com, azman@cs.usm.my, rahmat@cs.usm.my

Abstract

Conventional database encryption is built based on the idea to encrypt data-at-rest and data-in-motion. These solutions have successfully maintained the safety of database, but cause a certain stage of performance degradation at the meantime. If encryption of sensitive data were activated in a database server that implementing conventional approach, the server may need to do extra works, besides its general handling on records manipulation. This will degrade the performance of the database server itself, especially while it is dealing with huge data (e.g. image data) or millions of records. This paper proposes a new option to cope with the security weaknesses and performance degradation problem caused by the above-said weaknesses. A hybrid protocol (of combining the two database encryption categories) is introduced here and it is believed that this option should offer a better solution in security, as well as the effort to boost up the overall performance by using XOR during the encryption process..

Keyword: *Database Encryption, Image Encryption, and Cryptography*

1. Introduction

Since the evolution of electronic commerce, the usage of database is no longer restricted to store business data within an organization. Today, database is fully utilized in business applications to keep track with business and transaction records, customer information, financial information, and other sensitive data. A database contains data ranging from different degree of confidentiality, and is widely accessed by variety of users. Hence, it is also at risk to disclosure of wide-ranging users. As the importance of database become more and more vital in business, database security turns up to be a non-negligible issue in order to protect data from its vulnerability to potential attackers and cryptanalysts.

Encryption adds an additional layer of security to make data unusable if, despite all efforts, someone does get unauthorized access to the raw data. Aware of encryption is being shown as the strongest security alternative for data protection, previous researchers had done a lot of efforts in database encryption, including encrypt "data-at-rest" and "data-in-motion".

1.1. Encrypting Data-At-Rest

The former category refers to encrypt data when it is statically stored in a database server. The main objective to implement this is to prevent unauthorized party from reading through the database, or eavesdrop the information that should be kept secret. It is claimed that most attacks onto database occur when the data is sit at rest. According to [1], data stay for longer period of time inside a database server, compared to while it is being transferred between the server and clients. Hence, a lot of efforts have been put in this area. The operation of securing data-at-rest involves transforming sensitive data into unintelligible forms, so that it is only readable by authorized parties. Sensitive data are encrypted as soon as it is stored in the database. Upon leaving the database, however, the data will be transformed back into plain text [2]. As such, the data are always at risk of disclosure while in transfer, excepting if a secure communication channel is set between the client application and the database server.

1.2. Encrypting Data-In-Motion

Encrypting data-in-motion solve the problem raises up by the former database securing category. It plays an important role to protect data while they are being transmitted through communication channels. Critical information is protected through a secure connection established by the two communicating end points. To securely transmit data-in-motion, there are a few options applicable, such as *Secure Internet Protocol* (IPSec). The most common standard that database vendors adopted to is *Secure Sockets Layer* (SSL), or the follow

[1] The authors acknowledge the research grant provided by Universiti Sains Malaysia that has resulted in this article.

on Internet standard known as *Transport Layer Security* (TLS).

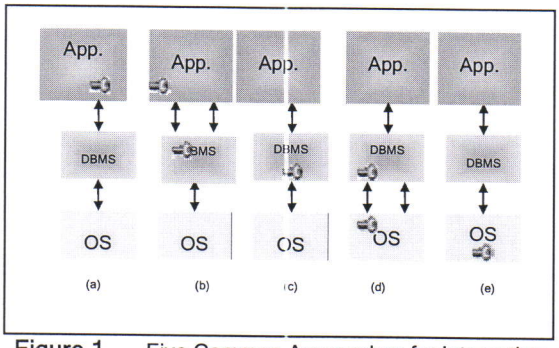

Figure 1. Five Common Approaches for Integrating Encryption Into a DBMS [3]

1.3. Motivation

Implementing either one of the above-said strategies is not sufficient to keep data safe from exposure. And hence, combining the two approaches is necessary and emerges as a better choice. However, merely joining the two methods does not yield advantage especially while dealing with the overall performance of the application. There are two times of encryption or decryption involves per call. When client is accessing data from database, the DBMS has to perform decryption of the encrypted data before sending it out. The data will then being encrypted again while pushed into the communication channel, and decrypted back upon arriving to the other end point.

2. Related works

2.1. Integrating Encryption Into DBMS Context

Previous developer classifies database encryption into five major categories, which are architectural different to each other. Common database encryption approaches consist of application-integrated (Figure 1(a)), DBMS-based (Figure 1(b)), DBMS-integrated (Figure 1(c)), OS-based (Figure 1(d)), and OS-integrated approach (Figure 1(e)).

Application-integrated approach is based on the idea that all encryption facilities are performed at the application program, rather than at the database engine. This approach is suitable for user-data encryption but not other database-generated data such as logs, metadata, and index information.

With a DBMS-based approach, cryptographic tasks are shared between the DBMS and application program. Encryption and decryption are under control of application program. The DBMS is just responsible for automating parts of the encryption process, for instant, use trigger for encrypting data during insertion or updating, which are transparent to the application program. Similar to the previous approach, it is not suitable for database-generated data.

Unlike the other two approaches discussed earlier, DBMS-integrated approach includes all encryption and decryption operations to the DBMS, making the process transparent to application program. This approach enables encryption of database-related or database-generated data, besides user data.

On the other hand, an OS-based approach is possible only on an operating system platform that supports secure persistence storage of data. This functionality is exposes through API. Encryption and decryption as well as key management is performed inside the operating system. Here, the DBMS accesses storage objects differently depending to their encryption state, which meaning those two different types of queries are necessary for accessing encrypted and unencrypted data.

The last category of database encryption is an OS-integrated approach, which based on mechanisms for secure data storage that is completely integrated into the operating system. In this case, all encryption and decryption processes are hidden from application as well as the DBMS. The processes are either performed by the operating system or with the existence of an additional hardware or device.

For a better understanding of database encryption methodology applied by current database system, we have take into account two most popular database system: the Oracle 8i/9i and the Microsoft SQL Server 2000. The encryption solution of the former database system is based on a DBMS-based approach; in which encryption and decryption are performed within the DBMS address space. On the other hand, Microsoft SQL Server 2000 support encryption of network connection and metadata, such as stored procedures, definition of triggers, and so on. Encryption of stored data is available with the assistant of *Encrypting File System* (EFS), which is performed at file level.

2.2. Related Findings On Database Encryption

Understanding the importance of database secrecy has brought to the accelerate emergence of various solution in database protection. Here are some of the solutions available with a brief overview of the features provided by these solutions:

- nCipher's database encryption solution – offers some useful safety functionality in order to ensure data security, which includes hardware key management, authorized key usage, secure audit trail, and targeted encryption. [4]
- Eruces tricryption – which is a combination of three encryption processes: i) encrypt sensitive data with unique, variable lifetime keys; ii) encrypt and store the keys in a protected database that located away from the encrypted data; and iii) encrypt the links between the encrypted data and corresponding keys. [5]

- Secure.Data for SQL Server 2000 – which is transparent to application and brings together some enormous functionality such as its key management capabilities for selectively encrypting, securing and controlling access to database information. [6]
- The RSA security solution –offers a full range of security services ranging from providing strong user authentication, to delivering Web-based access control. In addition, it is also a high-performing and simplified encryption application [2].

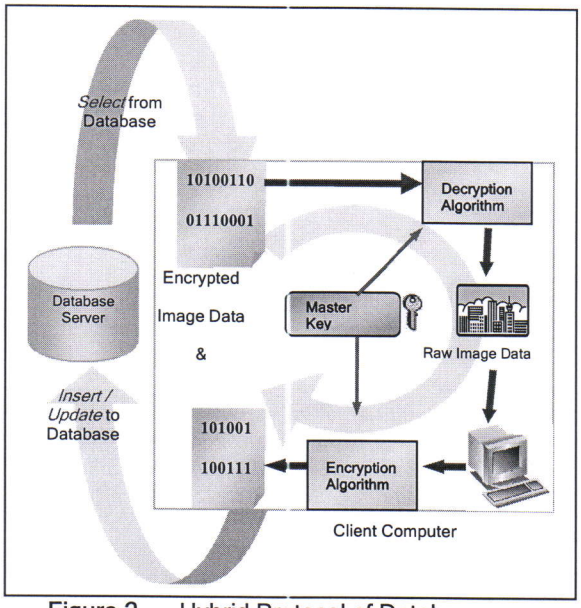

Figure 2. Hybrid Protocol of Database Encryption

3. Design Methodology

3.1. Hybrid Protocol For Image Data

The main objective of designing this hybrid solution is to boost up the performance of a database engine when image data encryption is requisite. As discussed earlier, the hybrid protocol that we are going to propose here is actually a combination of encrypting data-at-rest and data-in-motion. Image data is protected in both cases – while statically staying at rest, and being transmitted.

As illustrated in Figure 2, the methodology that we propose is application-integrated: all cryptographic tasks are neither performed at the database server nor additional cryptography server, but at the authenticated clients that intend to access image data from the protected database. Image data are being encrypted at client side before storing into database. In other words, the responsibility of the database server is just to store those unintelligible image data that have been encrypted at client side.

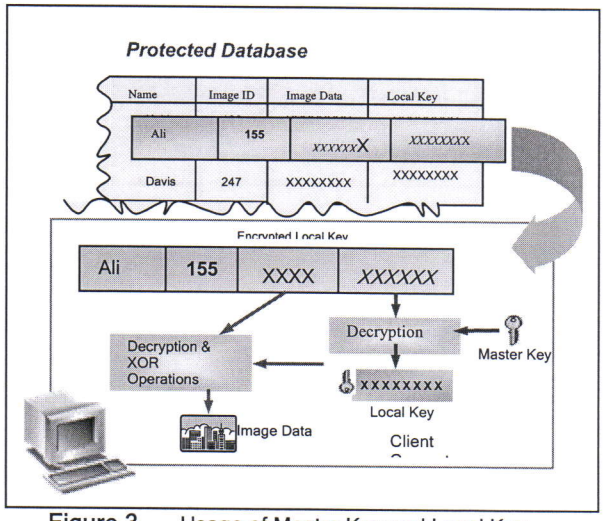

Figure 3. Usage of Master Key and Local Key

With image data encrypted at client side, the requirement for having a secure SSL connection could be eliminated. This is because image data is already in an encrypted form before sending out from the client. Hence, even if the image data is eavesdropped during its transmission to or from the database server, it is still unreadable and meaningless to other party without knowing local and master key of that particular image data.

Each client that connected to the database server is allocated with a *"Master Key"*, which is a public key shared among a group of clients. Besides, there is a randomly generated *"Local Key"* (by client who first insert the image data) for each sensitive record. Client uses local keys to perform encryption and decryption for the corresponding image data. The local key will then encrypted by master key and stored at database server along with the sensitive data. For safety reason, master key is stored at client side, separated away from the encrypted image data and local keys that are stored at database server.

Figure 3 depicts the storage pattern of local key with their corresponding image data in a protected database. The relationship of local key and master key along with their usage is shown in Figure 3 as well. As we can see from the figure, the responsibility of master key is to encrypt or decrypt the local key of each image data.

Because the local key is stored in database in encrypted form (for security reason), it needs to be decrypted by the master key while arrives at client. Only after then the processing of image data (encryption or decryption) can be carried out. The details of client side encryption and decryption will be further discussed in Section 3.2. Incidentally, we will not go into depth in discussion of key management throughout this paper, since it is of another research area that is out of our scope.

Theoretically, we would like to conclude that practicing our hybrid approach offers great advantage to the database server itself. Without extra processing of

encryption and decryption that are time-consuming, the database server is proficient to operate efficiently with optimal performance. In addition, even without the existence of secure network connection through SSL, encrypt data at client side prevent data at risk of enclosure while being transmit to the database server.

More to the point, the elimination of SSL connection enhances further the overall system performance besides keeping data away from unwanted threats.

3.2. Image Encryption

For better security reason, we replace the conventional encryption and decryption method with our new option, which is designed specially for image data that is normally huge in size.

Huge data always cause performance degradation while standard encryption or decryption operations like *DES* were to carry out. The operations are time-consuming, as encryption of huge data will not complete with a single round of encrypting operation. Applying block-chaining method (such as *ECB*, *CBC*, *CFB*, *OFB*, counter mode etc.), the data is split into blocks and encryption is performed onto each of these blocks.

Similarly, our solution splits huge binary data into blocks. The only different is: encryption or decryption is only executed onto first block of each data, with the data's local key. Output from the first block encryption will be *XOR*-ed to the second block. Likewise, this result will be used to *XOR*-ed to the following block. The operation continues until all of the blocks are completely worked on. Figure 4 summarizes the operations during an encryption process.

With replacement of *XOR* operations, the encryption and decryption processes involve are decreased. We believe that this solution is capable to enhance the overall system performance, while offering equal security as provided by conventional database encryption solutions.

4. Implementation

For experimental purpose, this project is implementing on a database engine called MySQL, which delivers a fast, multi-users, and robust SQL database server. The MySQL database engine that we used for this project is of version 4.0.13-max-nt. It is written in C and C++. The main reason why this database engine is chosen is the availability of its open sources, and its portability in multiple platforms, such as Solaris, Linux and Windows.

Throughout this project, implementation is carried out on Windows platform (Microsoft Windows XP). Sample results shown later in this section is obtained based on the following hardware specifications:

System	Microsoft Windows XP Home Edition (Version 2002)
Processor	Mobile Intel Pentium III 1.2 GHz
RAM	256 MB

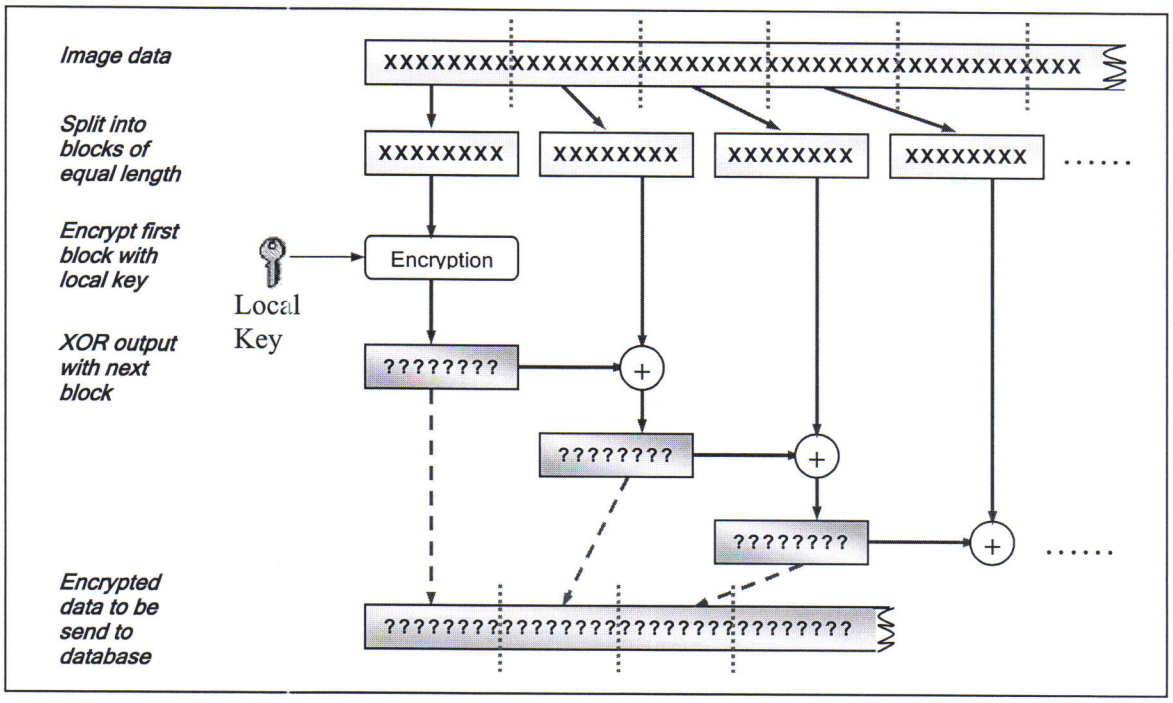

Figure 4. Hybrid Protocol: Encryption of Image Data Performed at Client Side

4.1. Encrypting Data-At-Rest

AES (Advanced Encryption Standard) is chosen as the data encryption algorithm, which is operating inside the database server. As described earlier, data are encrypted at the database server during its insertion, and decrypted at the database server upon retrieval. Because of the block-based nature of AES encryption, large image data is forced to be processed block-by-block recursively until all blocks operations are completed. Padding is used whenever a block is uneven.

We have carried out some experiments to observe how much time elapsed when encryption as well as decryption is performed inside the database server. Table 1 records the execution time for AES encryption and decryption operations with 16-bytes key. The execution time is proportional to the image size. Figure 5 depicts the comparison between encryption and decryption, where decryption of image data is relatively faster than encryption. This phenomenon happens because encryption involves a new random key generation for each data during insertion, whilst decryption just performs key retrieval from the database server.

4.2. The hybrid protocol

Next, we would like to make a simple comparison between the conventional AES encryption with our hybrid protocol performed onto same image data. Table 2 shows the results of the operations performed with different key length.

Table 1. Encryption And Decryption Of Image With Different Size

Image size (kb)	Time elapsed (sec)	
	Encryption	Decryption
18.4	0.60	0.20
58.2	0.71	0.23
102.0	1.40	0.65
191.0	2.32	0.70
254.0	5.00	1.02

Taking 16 bytes key as sample, Figure 6 makes a simple comparison between a server-side AES encryption operation and a client-side XOR operation performed onto same length of image data.

As illustrated, client-side hybrid protocol proved to be able to reduce the total execution time of encrypting data-at-rest about 10-40%, depending to the image size. The method is less time-consuming compared to the conventional database encryption approach, while providing greater security to sensitive data (able to protect data while statically stay at rest as well as while the data is being transmitted).

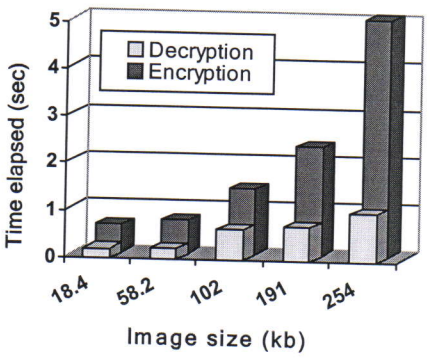

Figure 5. Time Elapsed During Encryption and Decryption of Image Data

Table 2. Hybrid Protocol Applied To Image With Different Size

Image size (kb)	Key length (bytes)		
	16	24	32
18.4	0.37	0.43	0.41
58.2	0.50	0.45	0.47
102.0	0.89	0.62	0.60
191.0	1.92	1.86	1.82
254.0	4.64	4.43	4.42

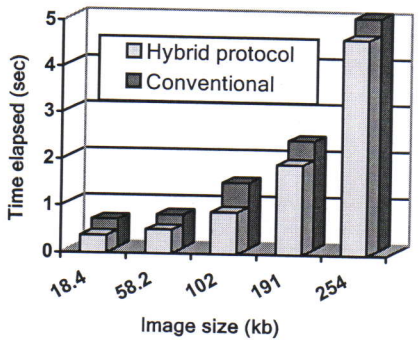

Figure 6. Comparison of Server-Side Encryption and Client-Side Hybrid

5. Discussion

Before ending this paper, we would like to emphasize that all the proposed frameworks are restricted to image data (or other binary data) encryption. The main reason for this limitation is because of the varying access pattern of a binary data and a normal field placed in database server. Consider a table containing three fields

of *Name*, *ImageID*, and *ImageData*. For instant, database user can write select statements as follow:

*Select * From Table1 Where Name = 'Sharon'*

*Select * From Table1 Where ImageID Like '10%'*

As *ImageData* is of binary datatype, retrieval of records from this table could only be done either by *Name* or by *ImageID*, but not by *ImageData*. We would like to categorize them as fields that could be included for recordset binding. These fields are not suitable to be implemented with our proposed encryption methodology. Recall back our design of storing encrypted local key at database together with the encrypted data. If *Name* is encrypted with this method, problems may occur during retrieval. Because *Name* is encrypted and unreadable, we need to decrypt it but the problem is – we do not know which local key to be used for that particular data. Consequently, select statements as shown above are no longer valid. In short, the only protectable column is the binary *ImageData*.

Another strong point we get from the experiment that have been carried out is – conventional encryption is much more time-consuming, and yet, this does not include time taken for transmitting data securely (through secure SSL connection). We can imagine how a database server is heavily loaded when a couple of data are in query for processing if securing data-at-rest and data-in-motion are intended.

6. Conclusion

Throughout the experiments that have been carried out, cryptographic tasks that are performed at client side is proved capable to reduce workloads of a database server. We would like to conclude that the proposed hybrid protocol could successfully provide an alternative security solution for image data to be stored into database.

7. References

[1] Application Security, Inc. *Encryption of Data at Rest – Database Encryption*. White Paper. (2002)

[2] RSA Security, Inc. *Securing Data at Rest: Developing a Database Encryption Strategy*. White Paper. (2002)

[3] Thomas Fanghänel. *Using Encryption for Secure Data Storage in Mobile Database Systems*. Faculty of Mathematics and Informatics, University Friedrich-Schiller: Jena, German. (September, 2002)

[4] NCIPHER, Inc. *Database Encryption: Secured by Hardware*. nCipher Solution Brief, Issue One. (December, 2001)

[5] Eruces, Inc. *Securing Data Storage: Protecting Data at Rest*. Issue Four. (2001)

[6] Protegrity, Inc. *Protegrity Secure.Data for SQL Server 2000*. (October, 2002)

[7] William Stallings. *Advanced Encryption Standard*. Chapter 5 in Cryptography and Network Security: Principles and Practices. International Third Edition. Prentice Hall: United States of America. (2003)

[8] Oracle Corporation. *Protecting Data Within the Database*. Oracle9i Security Overview, Release 2 (9.2), Part Number A96582-01. (2001) Available from: http://www.csis.gvsu.edu/GeneralInfo/Oracle/network.920/a96582/protdata.htm

[9] Jennifer Vesperman. *Introduction to Securing Data in Transit*. (2002)

[10] Peter Nilsson. *Getting up to Speed with Data Encryption*. Business Briefing: Global Info Security. (2002)