DEVELOPING A DATABASE ENCRYPTION STRATEGY: BOTH INSIDE AND OUTSIDE THE DBMS

Beena Sachan

Beena Sachan, Senior Lecturer, Meerut Institute of Engineering & Technology, Meerut

ABSTRACT

Security is becoming one of the most urgent challenges in database research and industry, and there has also been increasing interest in the problem of building accurate data mining models over aggregate data, while protecting privacy at the level of individual records. Instead of building walls around servers or hard drives, a protective layer of encryption is provided around specific sensitive data-items or objects. This prevents outside attacks as well as infiltration from within the server itself. This also allows the security administrator to define which data stored in databases are sensitive and thereby focusing the protection only on the sensitive data, which in turn minimizes the delays or burdens on the system that may occur from other bulk encryption methods. Encryption can provide strong security for data at rest, but developing a database encryption strategy must take many factors into consideration. Where the encryption should be performed, for example — in the database, or in the application where the data originates? Who should have access to the encryption keys? How much data must be encrypted to provide security? What's an acceptable trade-off between data security and application performance? This paper examines the issues of implementing database encryption and makes recommendations that will help a company to develop a strategy that will meet its individual needs.

1 INTRODUCTION

Critical business data in databases is an obvious target for attackers. Although access control has been deployed as a security mechanism almost since the birth of large database systems, for a long time security of a DB was considered an additional problem to be addressed when the need arose, and after threats to the secrecy and integrity of data had occurred. Now many major database companies are adopting the loose coupling approach and adding optional security support to their products. You can use the encryption features of your Database Management System (DBMS), or perform encryption and decryption outside the database. Each of these approaches has its advantages and disadvantages. The approach of adding security support as an optional feature is not very satisfactory, since it would always penalize the system performance, and more importantly, it is likely to open new security holes.

Hackers Are Not the Only Threat — or Even the Most Dangerous

Threats to your databases can come from hackers, attackers external to your network, working outside of the enterprise firewall. While firewalls are indispensable protection for the network at keeping people out, today's focus on e-business applications is more about letting the right people inside your network. Consequently, as databases become networked in more complex three tier e-business applications, their vulnerability to external attack grows. Evans Data estimates that one out of ten networked databases were attacked in 2001 (Developer Database Survey 2002, Volume 1). Without extra precautions taken to secure the confidential data in databases, your company's privacy is at risk. Here, taking the right security approach enables your e-business but protects your critical data infrastructure.

But hackers are not the only threat to enterprise databases. Attacks by employees who have access to sensitive information are often even more devastating — and cannot be prevented by perimeter defences like firewalls. "While viruses, Web defacements and stolen credit card databases are the stuff of news headlines, less publicized incidents such as data theft or destruction by disgruntled former employees can result in far more actual damage," noted Information Security magazine in its October 2001 industry survey. "In a layoff economy you are tempting fate with poor security," the article quoted one survey respondent.

Do you know how many employees have access to your databases? If you are using passwords for administrators, how are passwords being stored? Do you have security policies in place that include auditing your database security and monitoring for suspicious activity? What is your security plan if your database security is breached? While preventive security mechanisms like encryption, access control and strong user identification technologies are readily available to protect databases from both types of attack they are often not implemented to secure confidential information in databases from threats.

2 PLANNING A DATABASE ENCRYPTION STRATEGY

Before you can begin to design a database encryption strategy that is secure, you need to understand three things: how encryption works, how data flows in your application, and how database protection fits into your company's overall security policy.

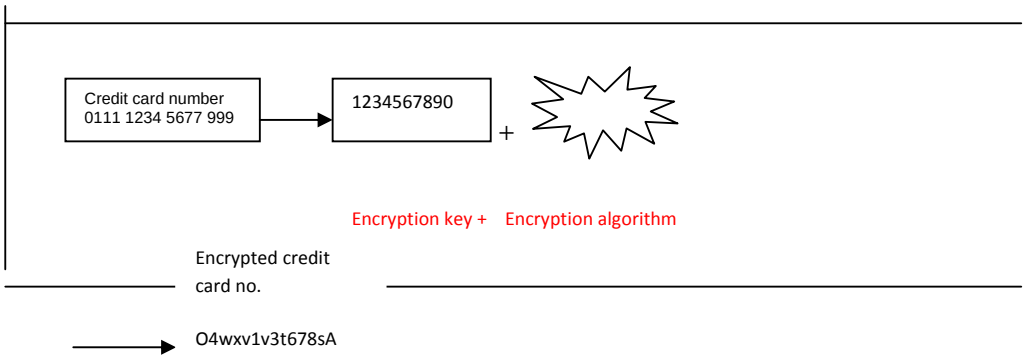


FIGURE 1: HOW ENCRYPTION WORKS

Once you've assessed the security and encryption needs of the sensitive data being gathered in your application, you will need to pick a course of action to ensure it is protected once it reaches the database. There are two strategies you can use — using encryption features of your Database Management Strategy (DBMS), or performing encryption and decryption outside the database. Each of these approaches has its advantages and disadvantages. In this section we will outline the two different strategies for encrypting stored data so you can make the decision that is best for your environment.

Encryption Basics: What Needs to be Known

To give sensitive data the highest level of security, it should be stored in encrypted form. The goal of encryption is to make data unintelligible to unauthorized readers and extremely difficult to decipher when attacked. Encryption operations are performed by using random encryption keys (see Figure 1). The randomness of keys makes encrypted data harder to attack. Keys are used to encrypt data, but they also perform decryption. Keys are often stored to allow encrypted data to be decrypted at a later date.

All encryption isn't alike. The security of encrypted data depends on several factors like what algorithm is used, what is the key size and how was the algorithm implemented in the product. For instance, many databases use DES encryption to protect sensitive fields, but DES has long been considered insecure for protecting data for any significant length of time. Additionally, different algorithms perform differently, so while DES is insecure it is faster than 3DES — another popular algorithm used in database products. In fact, 3DES is the slowest block cipher.

Besides the quality of the encryption technology, any database protection effort is only as secure as the key management strategy that supports it. There is an inevitable tension between access and security. On the one hand, anyone or any system that decrypts information must be able to access the stored encryption key so keys must be accessible. Yet on the other hand, anyone or any system that accesses keys can decrypt encrypted information.

A database encryption strategy that provides too little key security is like locking your car but leaving your key in the car door. But a strategy that makes key access too difficult may impact system performance and maintainability — and ultimately lead to lowered security as administrators and developers are forced to circumvent security measures in order to get their work done. These issues will be dealt with in more detail throughout this paper.

How does encrypted data impact database applications?

While encryption provides great security and is accepted as an industry best practice to keep data private, encryption can affect your data and your database. In particular, the impacts of encryption can increase data size and decrease performance. But, there are educated trade-

offs in application planning and design that can be made if you anticipate the affect. Knowing exactly what data needs to be protected will give you more flexibility to make better performance and data size trade-offs later. For instance, in the case of a credit card, does a company policy or regulation require the entire credit card number to be encrypted or only the last four numbers? Often the decision on how much of the data must be encrypted is the first step in determining the overall architecture of your solution.

Encryption affects data size. Often the ciphers used to encrypt blocks of text in a database produce output in fixed block sizes and require the input data to match this output size or it will be padded. Encryption operations, especially on smaller data items, may increase the size of the stored data in your database table and cause you to resize database columns.

Also, because encryption transforms character data into meaningless binary data, it can impact size if encrypted data must be translated into character-type data. Using something called Base64 encoding, encrypted data can be transformed from binary into characters but increases the data size by approximately one third.

Finally, carefully plan before encrypting information in indexed fields. Look-ups and searches in large databases may be seriously degraded by the computational overhead of decrypting the field contents each time searches are conducted. This can prove frustrating at first because most often administrators index the fields that must be encrypted – social security numbers or credit card numbers. New planning considerations will need to be made when determining what fields to index.

How Data Flows in the Application

A good starting point for developing a data protection strategy is to consider how data flows through the application and what system components process that data. You can then determine where the data may be at risk.

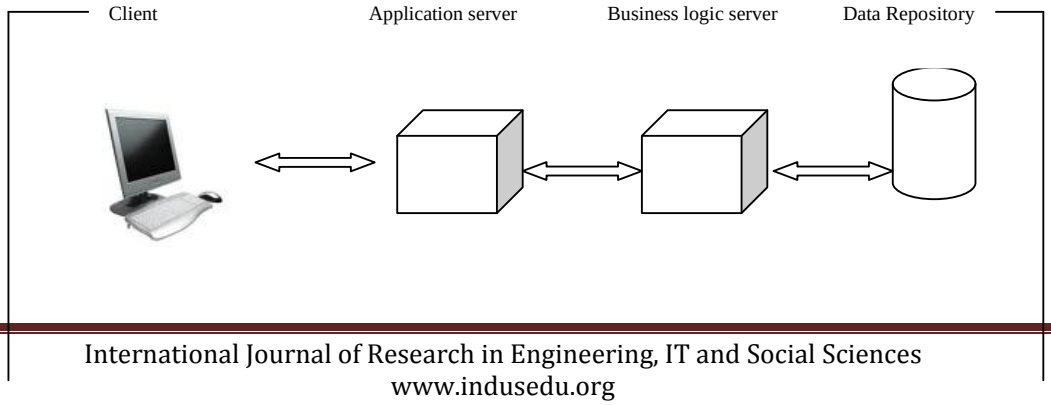


FIGURE 2: TYPICAL THREE-TIER APPLICATION ARCHITECTURE

Figure 2 shows typical three tier application architecture. The data may be at risk of exposure in any of the three tiers, or as it travels between components. Encryption performed by the DBMS can protect data at rest, but you must decide if you also require protection for data while it's moving between the applications and the database. How about while being processed in the application itself, particularly if the application may cache the data for some period? Sending sensitive information over the Internet or within your corporate network clear text, defeats the point of encrypting the text in the database to provide data privacy. Good security practice is to protect sensitive data in both cases – as it is transferred over the network (including internal networks) and at rest.

Understand How to Manage Keys

Because cryptography is based on keys that encrypt and decrypt data, your database protection solution is only as good as the protection of your keys. Security depends on two where the keys are stored and who has access to them. Secure key management is too often overlooked in database security strategies.

Some important questions to address in planning include:

• How many encryption keys will you need?
• How will you manage keys?
• Where will the keys be stored?
• How will you protect access to the encryption keys?
• How often should keys change?

How many keys do you need and how will you manage the keys?

Encryption key management is often a difficult problem to solve. Using a single key for all your cryptography applications makes implementation simpler, but it also means all your sensitive data is vulnerable if the key is stolen. Scenarios become more complicated as the number of applications and users requiring access to keys to decrypt data grows. Because

encrypted data can only be decrypted with its corresponding key, any system or application will need to know how to find that key. The more systems that know the encryption key, the higher the risk that the key will be exposed unless a strong access management system is applied.

A good rule of thumb is the fewer keys you use to encrypt information, the easier the solution is to manage, but the more critical key security becomes

Where to store keys?

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. But, all administrators with privileged access could also access these keys, decrypt any data within your system and then cover their tracks. Your database security in such a situation is based not on industry best practice, but based on an honor code with your employees. If your human resources department locks employee records in file cabinets where one person is ultimately responsible for the keys, shouldn't similar precautions be taken to protect this same information in its electronic format?

A recommendation is to consider separating the keys from the database where the encrypted data resides by storing keys in hardware. For example, the keys can be stored in access-restricted files, or for the strongest protection, in hardware storage modules. An ancillary benefit of hardware storage is since the keys need never leave the hardware device, access can be controlled so neither administrators nor intruders can penetrate the machine and steal them.

Protecting access

Secure key storage depends on how well you manage access to the secure key store. Many enterprises have learned the hard way. A recent study by @stake research found one third of their customers stored confidential data and encryption keys insecurely (see the report "The Security of Applications: Not All Are Created Equal", February 2002, at http://www.atstake.com).

Without a strategy to separate keys and store them securely with some type of access control, too many people will be able to access the keys — from developers who write the

*applications that ca*ll the database, to the administrators that manage the database — which discounts any effective accountability. Fortunately, popular authentication methods may be used to authorize who may decrypt information.

3 IMPLEMENTING A DATABASE ENCRYPTION STRATEGY

To effectively secure your databases using encryption, three issues are of primary importance: where to perform the encryption, where to store encryption keys and who has access to encryption keys. The process of encryption can be performed either 1) within the database, if your DBMS supports the encryption features you need, or 2) outside the DBMS, where encryption processing and key storage is offloaded to centralized Encryption Servers. These two strategies will be covered in more detail below, but first some general comments:

DBMS Features and Limitations

While encrypting inside the database may be beneficial because it has the least impact on your application environment, there are performance trade-offs and security implications to consider. Depending on the algorithms used and their implementation, some encryption can degrade DBMS performance. If you're DBMS includes encryption, it is important to understand what algorithms it uses, the performance and strength of those algorithms, and how much flexibility you have in selecting what data you encrypt.

Some general guidelines are DES is insecure, 3DES is slow and any symmetric ciphers should use 128-bit keys at a minimum.

An inherent vulnerability of DBMS-based encryption is the encryption key used to encrypt data likely will be stored in a database table inside the database, protected by native DBMS access controls. Frequently, the users who can have access rights to the encrypted data also have access rights to the encryption key. This can create a security vulnerability because the encrypted text is not separated from the mean to decrypt it. Nor does this solution provide adequate tracking or monitoring of suspicious activities.

Many enterprises IT managers have found the out-of-the-box encryption features offered by their DBMSs have weaknesses of performance and key management sufficiently severe that they decide not to use them.

SOLUTION ONE: IMPLEMENTING ENCRYPTION INSIDE THE DBMS

If encryption features are available within your DBMS product, you can encrypt and decrypt data within the database and the process will be transparent to your applications. The data is encrypted as soon as it is stored in the database. Any data that enters or leaves the database, though, will be transported as clear text. This is one of the simplest database encryption strategies, but it presents performance trade-offs and security considerations that must be evaluated.

Encryption generally is implemented within the database through a "database procedure call" (the terminology varies by vendor). Some vendors support limited encryption capabilities through database add-ons. Other vendors may only provide all-or-nothing support for encryption — either the entire database is encrypted, or nothing is. While this may make sense for protecting your backup copies, encryption of the entire database means additional processing is expended on non-sensitive data — an overkill situation resulting in unnecessary performance degradation.

A major drawback to encrypting inside the database is the extra processing load. Because encryption and decryption are performed within the database, the DBMS is asked to perform additional processing – not only when the data is stored, but each time it is accessed. This additional processing can add up.

Encrypting data when it is stored in the database using a database procedure call is shown in the diagram below. The procedure has to locate the stored encryption key (typically encryption keys are stored in a restricted table in the database) and query it. The DBMS must verify the procedure can access the key. The database procedure then uses the key in the encryption algorithm and returns the encrypted result.

Reading the data requires the same procedure in reverse. Consider, for example, an application that does a sorted report based on credit card data and accesses a database Containing encrypted card numbers. The database procedure for decrypting an item is executed against each encrypted data item. If it's a large report, that can add up to a lot of extra processing. On the other hand, applications that depend on indexes built on encrypted

data make the process even slower. For performance, it is advisable to architect the data so that encrypted data is not indexed. But, if you must encrypt indexed data, encrypt the search value before performing the search. This means that the search procedure must be changed, and will require access to the encryption function as well as the encryption key.

The strongest argument in favor of encrypting data within the DBMS is that applications are unaffected by the encryption. You can implement DBMS-based encryption without making any changes in legacy applications, e-commerce applications, or any other applications that use the data. However, this solution results in some equally compelling negatives: unless you use encrypted communications between the database and your applications, the data will be at risk of exposure while in transit. Also, if encryption keys are stored within the database, or even in other databases managed by the DBMS, the database administrators may have access to them — and thus to any of your encrypted data.

When evaluating database products, make sure you understand the performance of the encryption ciphers and strength of cipher based on key size. Many databases offer only the DES or 3DES algorithms which are generally regarded as slow performing. Another cipher, AES (which will replace DES) is preferable from a security perspective, or for higher performance and security evaluates the RC5® block cipher.

Encryption keys are based on pseudo random number generation. Thus the security of your data depends on how truly random the base numbers are. You should understand how random keys are generated in your DBMS. What type of pseudo random number generation is used? It may help to talk to outside security experts about random number generation in database products before making a purchase decision. For example, RSA Security's cryptography products are designed to provide random number generation in both software and hardware.

If you do not want to store your keys in a table in the database, plan how you will store keys separately. The strongest key protection is with separate hardware that interoperates with the database. Depending on the level of security required this often means purchasing a hardware security module (HSM), a device that provides secure storage for encryption keys and, depending on the device, additional features such as a co-processor to perform cryptographic functions and hardware acceleration. HSMs are also a great way to back up encryption keys.

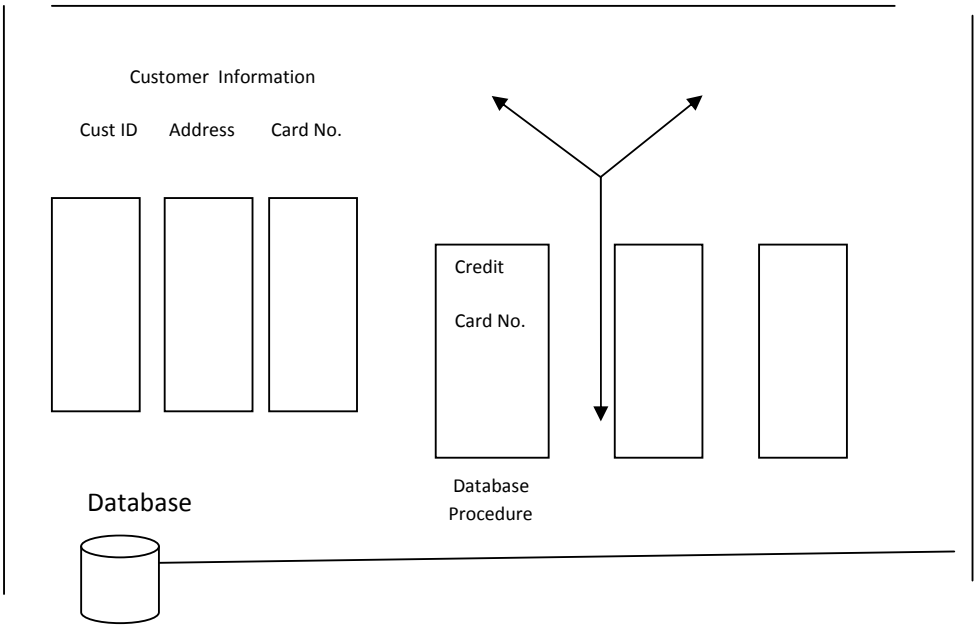


FIGURE 3: ENCRYPTING INFORMATION INSIDE THE DATABASE

Bottom line: This solution provides encryption, but does not offer either secure key management or high performance. Also, your DBMS vendor may only offer a few ciphers from which to choose. Here's a summary table of the pros and cons of using DBMS-based encryption:

STRENGTHS/BENEFITS

Applications are unaffected Encryption may already provided in database product

WEAKNESSES

Extra processing = performance degradation

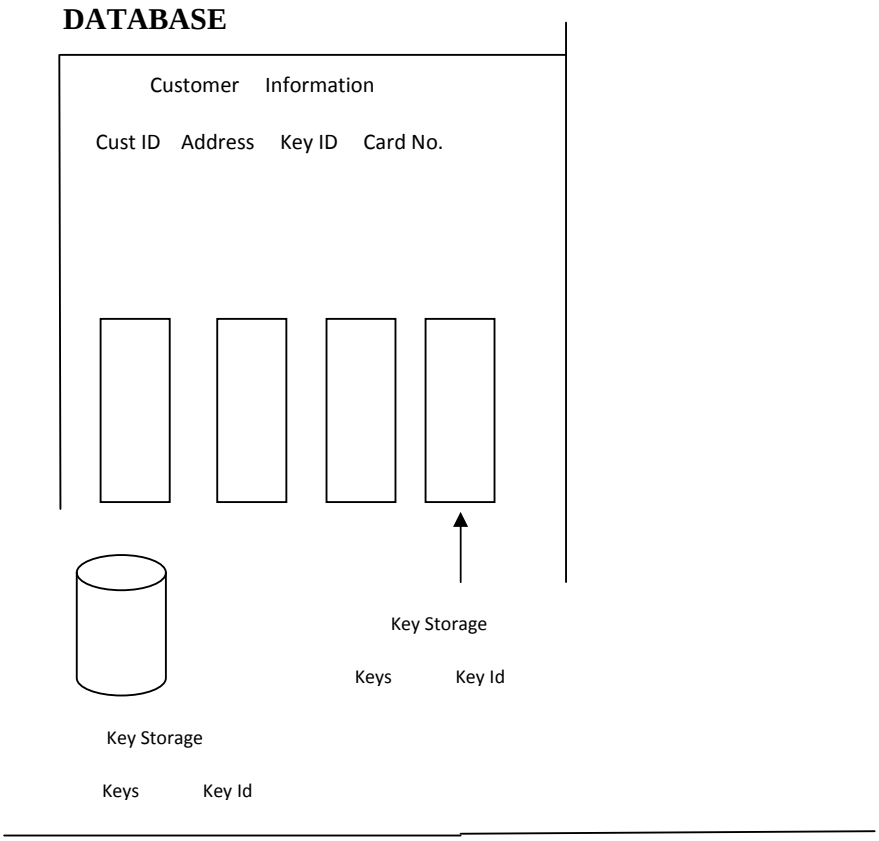
Data at risk outside the database. Encryption keys are stored in a database table with encrypted text, no separation of keys from text

To separate keys, additional hardware is required — like HSMs

If protection of keys is based on passwords, difficult to manage and insecure

Limited choice in algorithms supported

FIGURE 4: ENCRYPTING INFORMATION OUTSIDE THE DATABASE

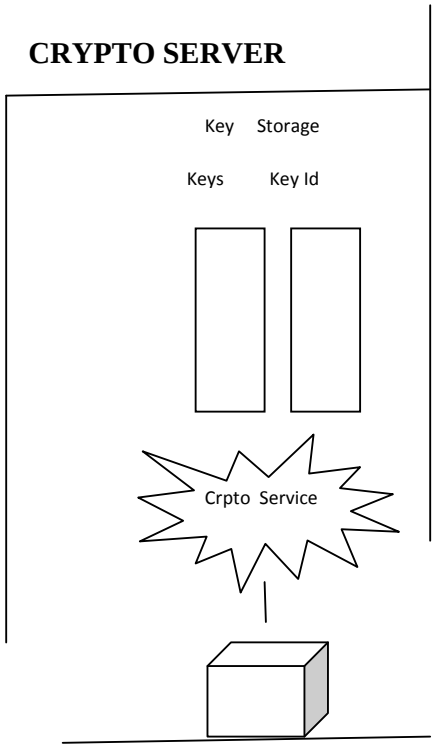


Solution Two: Applying Encryption outside the Database

If the potential for data exposure in the database or in transit between client and server concerns you, a more secure solution is moving the encryption to the applications that generate the data.

When you use client/server application security protocols like SSL, sensitive data is in clear text form for the shortest possible time. Encryption is performed within the application that introduces the data into the system; it travels encrypted and can be stored encrypted at its final destination. This approach can provide good end-to-end data protection, but may require changes to your applications to add or modify encryption and decryption capabilities.

One way to achieve this type of a solution and optimize your investment is to build an Encryption Server to provide centralized encryption services for your entire database environment. This simplifies management and provides more control in a multi-application environment using many databases. This server can be optimized to perform cryptographic operations requested by your applications, giving you the flexibility to allow applications to make multiple requests for cryptographic operations, while consolidating and implementing the cryptography in a consistent way. Here is a diagram of an
Encryption process that includes an encryption server to provide cryptography processing and key storage (as illustrated in Figure 4)



Encrypted Information

One great benefit of this solution is it offers one of the best secure key management strategies. This solution separates encryption keys from the encrypted data stored in the database (the encrypted data is injected into the database, but the keys never leave the Encryption Server) providing another layer of protection for the database. By contrast, Scenario One stores keys in the database with the encrypted data allowing an attacker easy access to both the keys and encrypted data. In Scenario Two outlined by the diagram above, the Encryption Server adds another layer of protection between the database and the attacker. The keys in the Encryption Server must be found before the hacker can decrypt data. The goal is to harden the Encryption Server against intrusion so that if anyone gained access to sensitive data in the database, they would find this text encrypted.

The Second Scenario is more secure because:

• Encryption keys are stored in hardware, separately from the encrypted text.

• End-to-end encryption is applied between the client and Encryption Server. Encrypted information is simply injected into the database.

The solution requires:

• Protecting the application and Encryption Server with an authentication strategy, preferably strong authentication, allows only authorized users to decrypt sensitive information by accessing keys stored in the Encryption Server.

• Hardening the Encryption Server against intrusion by monitoring it for suspicious activity and auditing event logs regularly.

For corporations with a large number of applications that require highly secure key storage, this approach puts them in control of their data encryption and provides a standard platform for encryption and key handling for all applications.

Other benefits include performance improvements gained from off-loading the encryption from DBMSs, and the ability to scale the encryption function on demand.

Building an encryption server and customizing applications to work with it may seem like more work and expense in the beginning. However, it offers greater security, better performance and, ultimately, a lower cost of implementation — you can maximize your investment by leveraging one secure encryption solution across multiple applications and lock down access to encryption keys across the enterprise.

STRENGTHS / BENEFITS

Off-loads crypto processing from database server

Separates encrypted data from encryption keys for secure storage

Easy to apply strong authentication solutions that work with encryption server

Can separate administrator roles

More control over who accesses data

Scalable – can scale to handle encryption from many applications and many databases

If database does not handle encryption functionality, no need to buy a new database and migrate data

Provides end-to-end encryption from client to encryption server

Bottom line: This solution is one of the best for key management and often offers more choices in algorithms which could lead to higher performance. Importantly, this is a scalable solution: it will be of particular long-term value to enterprises with many databases and many users who need to gain authorized access to encrypted data. Application based encryption will require custom development, perhaps combined with consulting services and the acquisition of commercial security products or services — you have to understand the

investment you are making and the time required to deliver. Here's a summary table of the pros and cons.

WEAKNESSES

Communications overhead

Must administer more servers

Must change or modify applications

Must harden encryption server with an authentication policy and a way to monitor and log events

4 CONCLUSIONS

Database attacks are on the rise even as the risks of data disclosure are increasing. Already the financial services and health care industries must deal with legislation and regulation on data privacy. Consumer concerns about data disclosure and misuse will inevitably expand the responsibility of your enterprise to secure customer information. Failure could expose you to legal liability, negative publicity, lost public trust, as well as cost you money and lost productivity.

In this environment, your security planning must include a strategy for protecting sensitive databases against attack or misuse by encrypting key data elements. Whether you decide to implement encryption inside or outside the database, it is recommended:

• Encrypted information should be stored separately from encryption keys.

• Strong authentication should be used to identify users before they decrypt sensitive information.

• Access to keys should be monitored, audited and logged.

• Sensitive data should be encrypted end-to-end — while in transit in the application and while in storage in enterprise databases.

REFERENCES

[1] R. Agrawal and J. Kiernan. Watermarking relational databases. In 28th Int'l Conference on Very Large Databases, Hong Kong, China, August 2002.

[2]. E. Denning. Cryptography and Data Security. Addison-Wesley Publishing Company, Inc., 1982.

[3] S. Garnkel and G. Spa ord. Web Security & Commerce. O'Reilly & Associates, Inc., 1997.

[4] S. B. Guthery and T. M. Jurgensen. Smart Card Developer's Kit. Macmillan Technical Publishing, 1998.

[5] G. Koch and K. Loney. Oracle8: The Complete Reference. Osborne/McGraw-Hill, 1997.

[6] J. C. Lagarias. Pseudo-random number generators in cryptography and number theory. InCryptology and Computational Number Theory, pages 115{143. American Mathematical Society, 1990.

[7] T. F. Lunt. A survey of intrusion detection techniques. Computer & Security, 12(4), 1993.

[8] National Bureau of Standards FIPS Publication 46. Data Encryption Standard (DES), 1977.

[9] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signature and public key cryptosystems. Communications of the ACM, 21:120{126, February 1978.

[10] D. R. Stinson. Cryptography; Theory and Practice. CRC Press, Inc., 1995.