Securing Data at Rest: Developing a Database Encryption Strategy

A White Paper for Developers, e-Business Managers and IT

Most security initiatives are defensive strategies — aimed at protecting the perimeter of the network. But these efforts may ignore a crucial vulnerability — sensitive data stored on networked servers are at risk from attackers who only need to find one way inside the network to access this confidential information. Additionally, perimeter defenses like firewalls cannot protect stored sensitive data from the internal threat — employees with the means to access and exploit this data.

Encryption can provide strong security for data at rest, but developing a database encryption strategy must take many factors into consideration. Where should the encryption 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 your company develop a strategy that will meet your individual needs.

RSA SECURITY®

I. Introduction

The networked database is the heart of the enterprise. It is where your most valuable assets reside — the information that is the foundation of your business, transaction records, financial data, customer information. Protecting this data is increasingly important — and increasingly difficult.

This data's critical business value makes it an obvious target for attackers. Successful attacks can wreak massive damage to company finances and corporate image. The media spotlight falls most often on high-profile cases involving consumer transactions and credit card numbers. Public concerns, fueled by adverse news coverage, are giving rise to new regulations and legislation on data management and privacy.

But public-relations problems are not the only risk. Revelations of data gathered from Web-based transactions can damage a company's credibility and customer relationships. Database attacks can have direct — and severe — economic consequences. Database attacks are rising and they can result in the loss or compromise of information critical to running your business day-to-day, from inventory and billing data to Customer Relationship Management (CRM) applications and human-resources information. Consequently, databases are also likely to be holding increasing amounts of sensitive information on behalf of your customers — financial records, healthcare histories, order histories, credit card and Social Security numbers. Such a loss could be an operational and customer relationship disaster as well as a financial one — last year one company reported a $50 million dollar loss when its databases were exploited (CSI/FBI 2001 Computer Crime Survey).

To protect your company's database assets, there are security measures you should take today. These include encrypting data as it moves across your enterprise networks and as it sits at rest, in storage on database systems. Extra steps and precautions should be taken to carefully control access this data. This paper will focus on how to protect data at rest.

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 defenses 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.

Legal Liability for Disclosure Is Increasing

Two recent federal laws have set new standards for the protection of customer information. Regulations required by the Health Insurance Portability and Accountability Act (HIPAA) set standards for the security of medical records and other individually identifiable health information. The Gramm-Leach-Bliley Act (GLBA, Public Law 106-102) sets new requirements on financial institutions regarding the privacy and security of customers' personal financial information.

Congress's interest in privacy and security isn't surprising. In a recent poll by the Information Technology Association of America, 75 percent of the Americans surveyed feared having their personal information misused. The problem is very real: over 700,000 cases of identity theft were reported last year according to government and privacy advocacy groups. Worse, in 2001 credit card fraud cost the credit industry billions of dollars.

Congress clearly intends to make business liable for the security of customer data and HIPAA and GBLA are just the beginning. HIPAA regulations provide civil and criminal penalties for non-compliance due to willful neglect — fines of up to $50,000 and one year in prison per violation. Congress is also considering the Financial Institution Privacy Protection Act, which would stiffen the Gramm-Leach-Bliley Act to make company officers and directors liable for up to $10,000 for each privacy violation.

Protecting Data with Encryption

While laws and regulations interpret "protecting privacy" in a number of ways, any enterprise solution for protecting data — especially data at rest — must involve two things: secure encryption technology to protect confidential data and careful management of access to the cryptography keys that unlock the encrypted data. Only then has your company done due diligence to protect the privacy of its customers.

There are implementation decisions to be made as well. Where will you perform the data encryption — inside or outside of the database? Your answer can affect the data's security. How do you create a system that minimizes the number of people who have access to the keys? Storing the encryption keys separately from the data they encrypt renders information useless if an attacker found a way into the database through a backdoor in an application. In addition, separating the ability of administers to access or manage encryption keys builds higher layers of trust and control over your confidential information infrastructure. There should be limited access to the means to decrypt sensitive information – and this access should be locked down and monitored with suspicious activity logged.

More than Encryption

Encrypting the sensitive data is only the first step. Without strong authentication technologies applied at the application level, it is easy for imposters to get access to the keys that decrypt sensitive information. Often too many employees — from database administrators to application developers — are well aware of how to gain access to this information. Disgruntled employees therefore have it easy to find a way in — from social engineering, to stealing mismanaged passwords. Many employees who have compromised their company's database have leaked sensitive information like co-workers salary, social security numbers, and CEOs' private phone numbers to the press or co-workers. Worse, these employees will cover their tracks and make it look like the innocent co-worker with access to the database made off with the crown jewels. Protecting your database not only secures sensitive data, but it protects your employees from this type of exploitation.

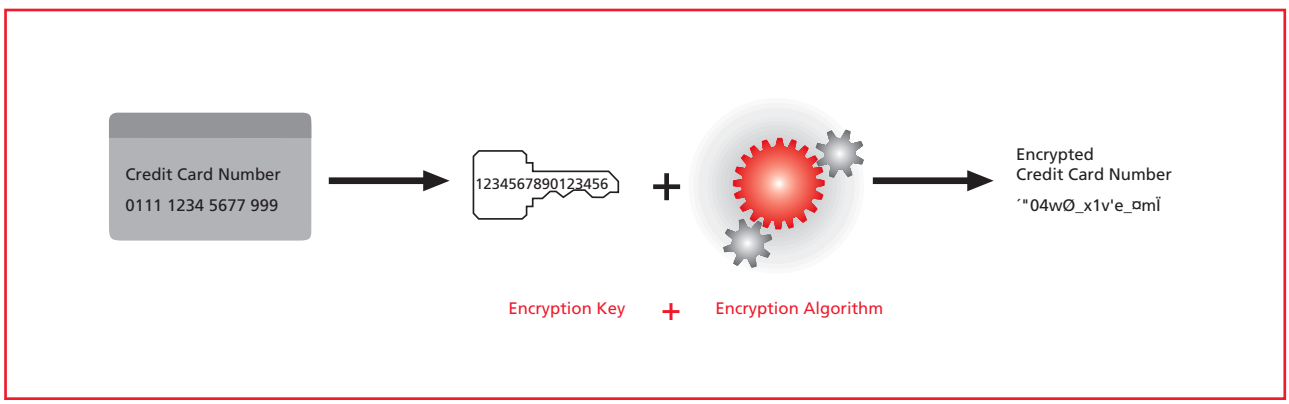
Many forward thinking organizations are beginning to develop security strategies aimed at the protection of sensitive stored data — strategies that include encryption, access management, security auditing and event logging. By using these strategies you can reap the benefits of an industry best practice — encryption technology — to provide the maximum protection for your company's greatest asset, confidential databases and maintain higher levels of trust from employees and customers.

II. 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.

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.

FIGURE 1 : HOW ENCRYPTION WORKS



Encryption Basics: What You Need to Know

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 make 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 RSA Security supports in its cryptography products. And finally, you should consider using encryption that is industry supported and supplied by a reputable cryptography provider who has a stake in providing the highest quality cryptography solutions.

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.

For more information on specific industry supported cryptographic ciphers and how encryption works, see Appendix A.

Understand 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

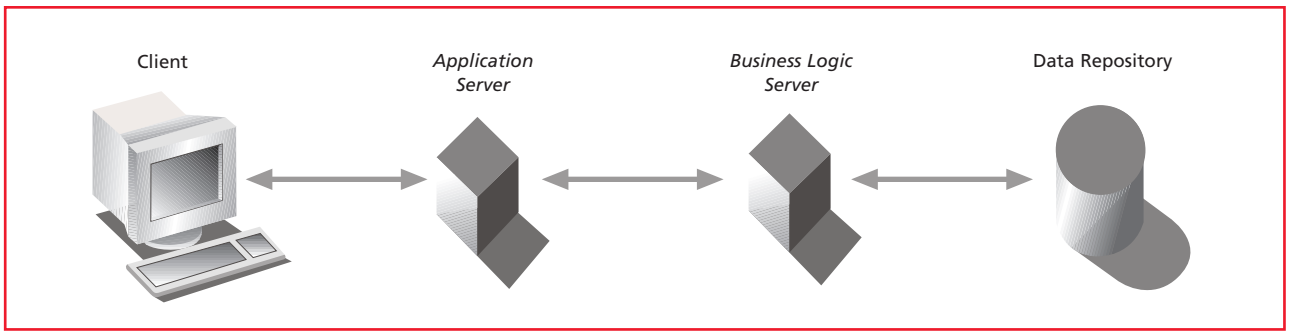


Figure 1 shows a typical three tier application architecture. The data may be at risk of exposure in any of the three tiers, or as it travels between components. Encryption performed by the DBMS can protect data at rest, but you must decide if you also require protection for data while it's moving between 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 factors: 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 call 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. Some of the more popular authentication methods are outlined in Appendix B.

III. 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 off-loaded 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 your 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 (See Appendix A for more information on encryption).

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 means to decrypt it. Nor does this solution provide adequate tracking or monitoring of suspicious activities.

Many enterprise 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.

Off-loading Encryption Outside of the Database

RSA Security recommends that companies, especially those that need to comply with Gramm-Leach-Bliley or HIPAA, consider database architectures that off-load encryption processing and secure key management to a separate, centralized Encryption Server. The Encryption Server will calculate the computation required by encryption or decryption. This has two benefits. It removes the computational overhead of cryptography from the DBMS or application servers. And perhaps even more importantly, it allows separation of encrypted data from encryption keys. The keys in this architecture never leave the encryption server. Locking down access and monitoring the Encryption Server is important in this scenario as well, but easily achievable.

Let's review each solution in more detail.

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.

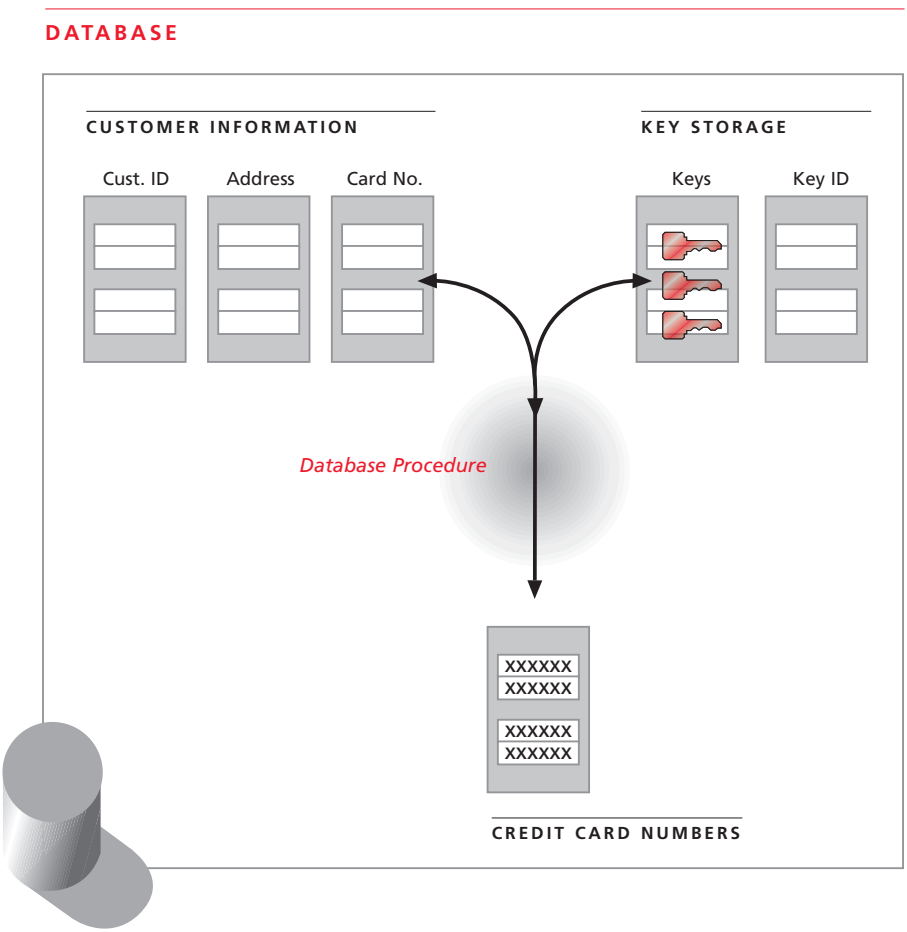
Encryption generally is implemented within the database through a "database procedure call" (the terminology varies by vendor). Some vendors support limited encryption capabilities through database add-ons. Other vendors may only provide all-or-nothing support for encryption — either the entire database 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.

FIGURE 3 : ENCRYPTING INFORMATION INSIDE THE DATABASE



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 evaluate 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.

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	WEAKNESSES
Applications are unaffected	Extra processing = performance degradation
Encryption may already provided in database product	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

Solution Two: Applying Encryption Outside the Database

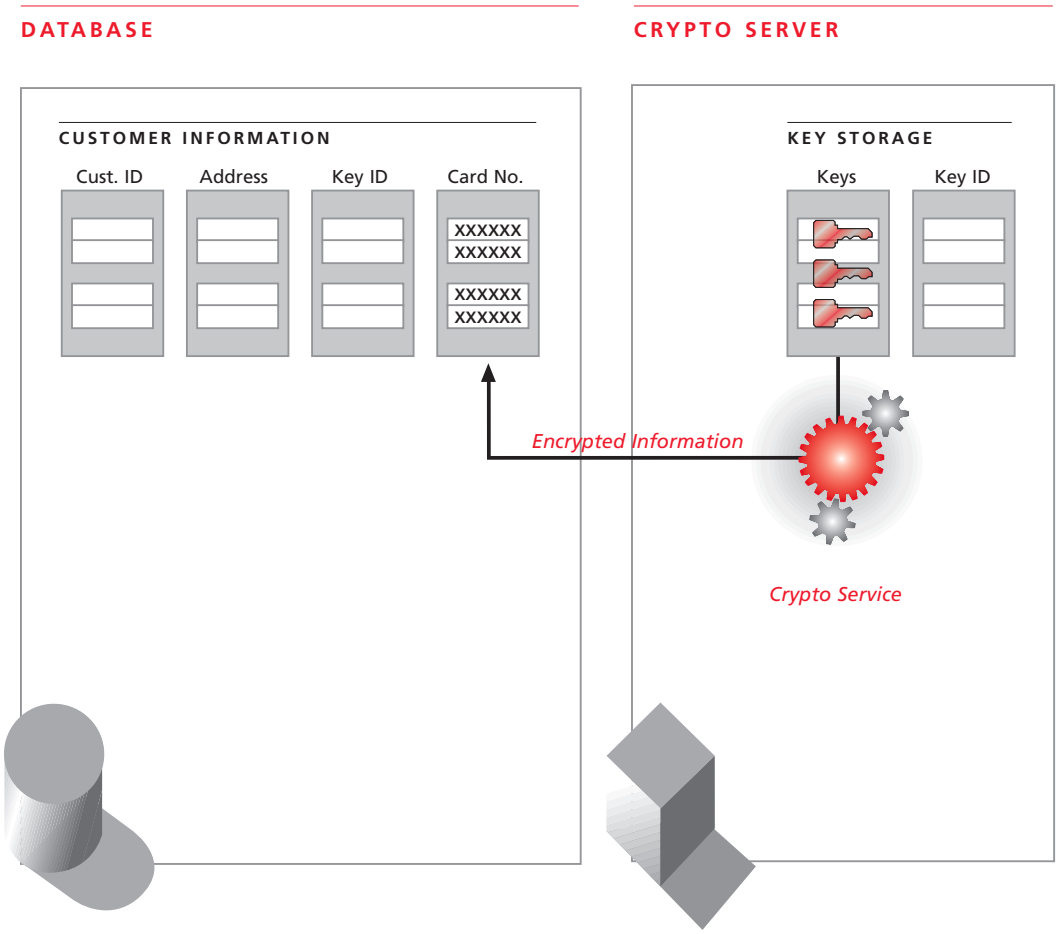
If the potential for data exposure in the database or in transit between client and server concerns you, a more secure solution is moving the encryption to the applications that generate the data.

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):

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

FIGURE 4 : ENCRYPTING INFORMATION OUTSIDE THE DATABASE



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.

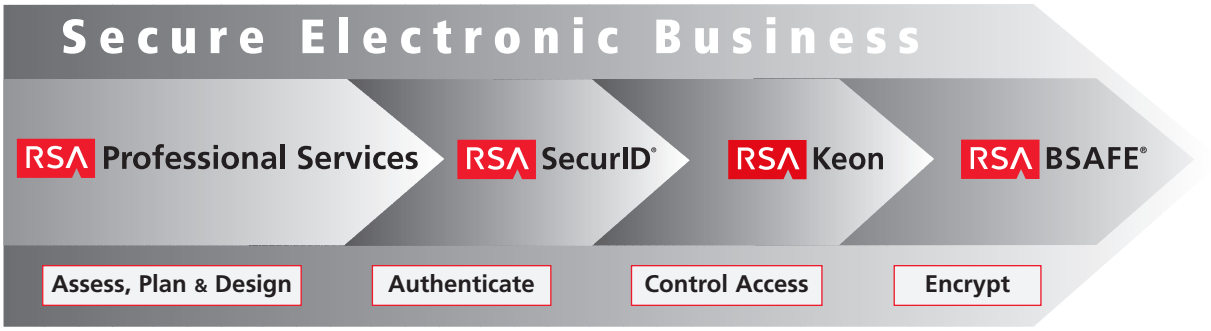
For corporations with a large number of applications that require highly secure key storage, this approach puts them in control of their data encryption and provides a standard platform for encryption and key handling for all applications.

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.

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.

STRENGTHS/BENEFITS	WEAKNESSES
Off-loads crypto processing from database server	Communications overhead
Separates encrypted data from encryption keys for secure storage	Must administer more servers
Easy to apply strong authentication solutions that work with encryption server	Must change or modify applications
Can separate administrator roles	Must harden encryption server with an authentication policy and a way to monitor and log events
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	



- Assess data flow in application
- Determine threats
- Re-design application
- Implement security

- Know who is using app.
- Make it easy for end user
- Provide easy integration
- Provide choice of products

- Audit who has access
- Ease for administrator
- Scale to 1000s users
- Scale to 1000s servers

- Protect data
- Make it easy for developer
- Be high-performing
- Deliver trusted crypto

IV. RSA Security Solutions

RSA Security offers a full range of security products and services designed to assess the database application, provide strong authentication of users, deliver Web-based access control for end users or administrators, and simplify the art of building high-performing encryption applications.

Assessment, Planning and Implementation

RSA Professional Services offers a database assessment service to review application architecture and design a database strategy. The goal is to identify key sensitive data, analyze the flow of data for potential vulnerabilities and threats, and make design recommendations. Further planning and implementation services are available for companies wishing to deploy the recommended solution. Benefits to customers include access to specialized encryption consultants and knowledge transfer or training of staff.

Authentication

RSA SecurID® tokens and smart card solutions are designed to deliver strong two-factor authentication that interoperate with a wide variety of products and applications. These products are easy for the end user interfacing with a client/server application and they positively identify the user with a higher level of trust than using a simple password. RSA SecurID authenticators are as simple to use as entering a password, but much more secure. Each end user is assigned an RSA SecurID authenticator which generates a new, unpredictable code every 60 seconds. The user combines this number with a secret PIN to log into protected resources. The authenticator is tied to a powerful algorithm generating a new code every 60 seconds in the RSA SecurID authentication server known as the RSA ACE/Server,® solution. Only it knows which number is valid at that moment in time for that user/authenticator combination.

RSA Security products are designed to provide the strong authentication required to protect access to the encryption keys stored in the database or on an Encryption Server. Using these authentication products, companies can reduce the threat of external attacks on their database.

Access Management

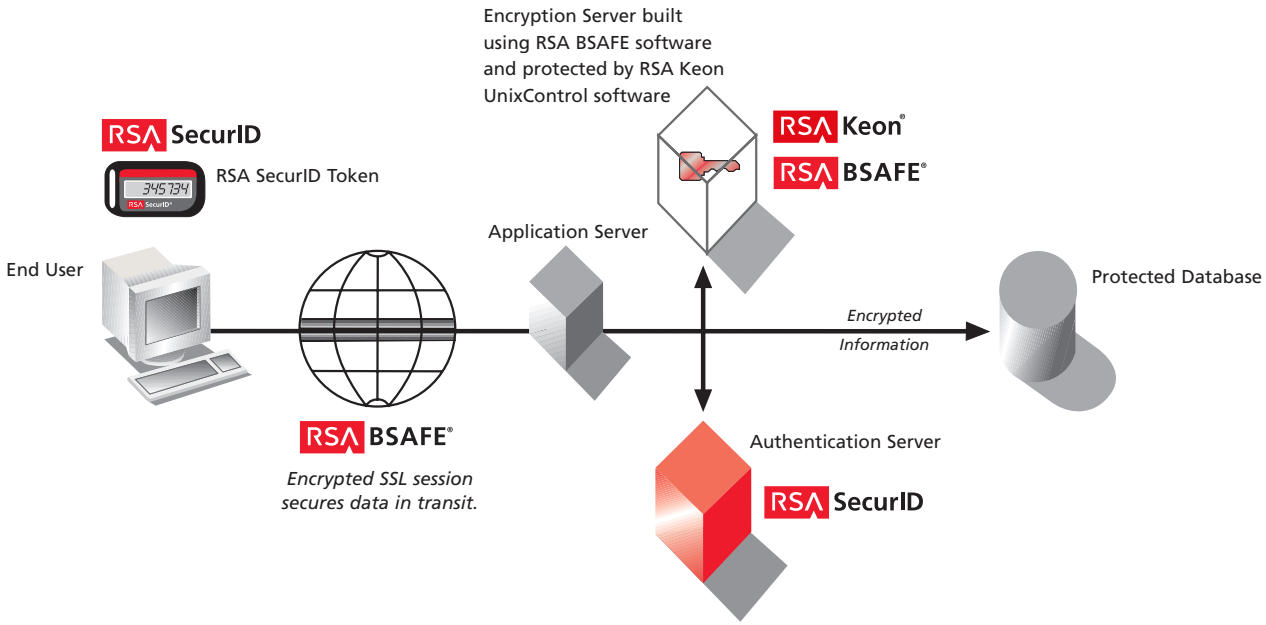
For organizations seeking to control access and simplify administration of their Unix environment, RSA Keon® UnixControl software is a simple way to help lock down access to these servers. This is especially important for organizations who want to deploy a Unix Encryption Server in front of their database. RSA Keon UnixControl software employs a unique, non-intrusive architecture to help centrally manage a UNIX environment. RSA Keon UnixControl software is designed to perform the following functions. It hardens the Encryption Server against intrusion. It can centrally manage the identification and authentication of users (it includes out of the box support for RSA SecurID strong authentication products), and establish access control while also providing data privacy and integrity monitoring of the entire process. It simplifies administration by propagating changes such as deleted or added user accounts across the entire UNIX enterprise in a single step. In short, RSA Keon UnixControl software helps simplify UNIX management while enforcing a strict enterprise-wide security policy without limiting productivity. Other features include file integrity checking and operating system vulnerability testing. In addition, RSA Keon UnixControl software is designed to proactively log changes to security parameters, access attempts, and administrative activities to create a complete audit trail.

Encryption

RSA BSAFE® encryption software is designed to allow developers to easily build a solution that would off-load encryption processing from the database to an encryption server for better performance. Alternatively, RSA BSAFE cryptography could be called from a stored database call to encrypt information inside the database.

RSA BSAFE Crypto-C or RSA BSAFE Crypto-J (Java) developer tools are designed to provide a wide range of high-performing algorithms that are shipped with award-winning documentation and comprehensive sample code to ease development. In addition, RSA BSAFE SSL-C or SSL-J software helps developers build a Secure Sockets Layer (SSL) connection to encrypt and protect the privacy of information as it travels between the Encryption Server and the end users desktop. Today, RSA BSAFE software is pervasively used on the Internet, helping to secure over 1 billion applications worldwide.

FIGURE 5 : THE RSA SECURITY DATABASE ENCRYPTION SOLUTION



V. Conclusion

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, RSA Security recommends:

- 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.

Appendix A: Applying Encryption Technology

To give sensitive data the highest level of security, it should be stored "at rest" in encrypted form. This is best achieved by using tested and industry-supported encryption algorithms. To be successful, a database encryption strategy must address a number of considerations that will affect the security of the application and how the application performs. These include:

- What is the best type of encryption for my data?
- What data needs to be protected using encryption?
- How strong must the cipher be?

Solution Two: Applying Eencryption Outside the Database

The goal of encryption is to make data unintelligible to unauthorized readers and extremely difficult to decipher when attacked. There are three general categories of encryption algorithms: symmetric ciphers, hashing algorithms and asymmetric ciphers.

Symmetric-key Ciphers

If the data is being encrypted for storage, symmetric encryption is most commonly used. Symmetric-key ciphers use the same key to encrypt and decrypt the data. There are two types of symmetric ciphers: block ciphers and stream ciphers. Stream ciphers are generally twice as fast as block ciphers but they require the use of unique keys. Block ciphers on the other hand, allow keys to be reused. Most DBMSs include encryption features that use some form of block cipher technology.

The recommended minimum key length for all symmetric ciphers is 128-bits. Key length is extremely important to data security because there are two main ways to attack encrypted data. The "guess and check" attack is the first type and is usually successful when an algorithm has been implemented incorrectly or is based on bad random number generation. Here the attacker is able to guess at a key, and then tries it on a sample of the data. Second, the brute-force guesswork approach is becoming more successful as increasingly powerful computers makes it possible to create and test a numbers of keys in relatively short periods of time. For example, DES ciphers using 56-bit keys have fallen to brute-force attacks in less than 24 hours. If you are using a database with a symmetric cipher, look for at least a 128-bit key length. Anything less puts your data in an unnecessary risk.

There are many popular symmetric key ciphers used for database encryption and they include AES, DES, 3DES and RC5 — all block ciphers. The RC4,® algorithm — a stream cipher — could also be used but each time a data input is encrypted, a unique key must be used which makes key management more complex. One main advantage of using block ciphers is that they encrypt small blocks of data well, and they can reuse keys within certain security parameters before a new key must be used. One important block cipher, DES, has been a government-recommended standard and is widely used, but is also somewhat slow and generally viewed as insecure. 3DES is an acceptable near-term alternative, with AES also recommended as a strong replacement alternative.

Asymmetric-key Ciphers — a.k.a. Public Key Cryptography

If the data is being encrypted for storage, symmetric encryption is most commonly used. Symmetric-key ciphers use the same key to encrypt and decrypt the data. There are two types of symmetric ciphers: block ciphers and stream ciphers.

Asymmetric algorithms use a pair of keys. The key pairs are created using mathematical principles that allow any data encrypted by one key to be decrypted only by the other. Asymmetric-key ciphers are most commonly used to protect data during transmission in a system that makes one key public and keeps the other key secret. These algorithms are more popularly known as public key cryptography and they make up the foundation of a "public key infrastructure" (PKI) which is based on a hierarchy of digital certificates that contain a user's public key and are "signed" as testament that the holder of one key of a pair can trust any data that key decrypts was encrypted by the matching key. "PKI" sometimes refers simply to a trust hierarchy based on public-key certificates, and in other contexts embraces the actual application of public key cryptography like digital signature services that can be provided to end users. The most widely used algorithms for digital signatures are the RSA algorithm, the U.S. government's Digital Signature Algorithm (DSA), and elliptic curve algorithms.

Asymmetric algorithms are generally used for authentication and digital signatures rather than encrypting data. Asymmetric operations are considerably slower than symmetric key algorithms, making them not well-suited for a database encryption strategy, because their performance could adversely impact database performance. They may be required, however, if you require strong integrity controls over your data, such as to meet HIPAA or FDA 21 CFR requirements. Digital signatures can provide this integrity proof. If you want to apply signatures or to encrypt small data items, you will want to off-load these operations from your DBMS. One way to achieve this is with a procedure that communicates with a separate cryptographic processor.

Hashing Algorithms

Hashing algorithms do not encrypt data, but provide a one-way transformation used to store data securely as well as to verify data integrity. Put simply, hashing data for integrity is like taking a fingerprint. Hashing a block of data yields a "hash value" of a fixed byte length that is a unique digest based on the data. When new data arrives that should match the original, the new data is hashed and the two hash values compared. If the data has not been corrupted or altered the values will be identical. If the data has been changed, the hash values will be different. Hashing is commonly used to store password data. While the original data can be hashed before storage or transmission, hashing is not reversible and therefore should not be used as an alternative to encrypting data.

Popular hashing algorithms include MD5 and SHA-1. Hashing algorithms do not require a key, so key length is not a consideration. Hashing operations have a fairly low computational overhead: the impact of calculating hashes within your DBMS will be roughly comparable to symmetric encryption, without the key management issues. The size of the value output by the hashing process is fixed — 20 bytes for SHA-1, 16 bytes for MD5.

Appendix B: Authentication Methods to Control Access to Encryption Keys

Password-based Encryption

One solution is to encrypt or hash the encryption keys using Password Based Encryption (PBE is supported by the PKCS#5 standard), so encryption keys are not stored in clear text. To unlock the encryption key the user must know the password. This solution, however, is just as strong as your password policy. How strong is the password — will it stand up to a dictionary attack? How many people know the password? How far do you trust them? Another drawback, of course, is if too many passwords are required, the system may be too complex to be used properly. Also if a password is lost, there is no way to decrypt the data. But, for some less complex implementations, PBE could be worth considering.

Smart Cards or Cryptography Tokens

Smart cards or similar cryptographic tokens (based on public key algorithms or time-synchronous technology) can offer a range of strong authentication options. Smart cards provide two-factor authentication, stronger than single-factor password-based authentication because it is based on a thing the user has (a smart card or token) as well as a thing the users knows (a shared secret PIN or password).

Smart cards are easy for the end user. In a complex application environment where many users with different roles and responsibilities will need access to keys to decrypt sensitive information, the users need only to insert their smart card into the client device and the application environment does all of the work. Audit trails can track who accessed what and when.

Smart cards also may be used in solutions that require multiple passwords because they can store username-password combinations. In such a system, access to encryption keys could require an authenticated user and an authenticated smart card. The card could be controlled by a security officer who would not have to be present when the key is accessed. He or she would manage access to the keys by managing possession of the physical card and audit trails would provide documentation.

Biometrics

Biometric systems are based on something the user is — a physical characteristic like a unique attribute (for example, a fingerprint, voice or a retinal scan). Biometric systems provide good authentication for solutions which can be combined with passwords and/or smart cards to provide simple two-factor or even three-factor authentication of individual users.

About RSA Security

RSA Security, the most trusted name in e-security,® helps organizations build trusted e-business processes through its RSA SecurID® two-factor authentication, RSA ClearTrust® Web access management, RSA BSAFE encryption and RSA Keon® digital certificate management product families. With approximately one billion RSA BSAFE-enabled applications in use worldwide, more than twelve million RSA SecurID authentication devices deployed and almost 20 years of industry experience, RSA Security has the proven leadership and innovative technology to address the changing security needs of e-business and bring trust to the online economy. RSA Security can be reached at www.rsasecurity.com.

DDES WP 0702

RSA SECURITY®

RSA Security Inc.
www.rsasecurity.com

RSA Security Ireland Limited
www.rsasecurity.ie