Securing Data at Rest:
Database Encryption Solution using Empress RDBMS

By: Srdjan Holovac

January 2006 *White Paper*

Contents

Introduction

Importance of security

Enforcing data security is a top priority for both governments and businesses worldwide. Recent legislation set new standards for protecting customer information, such as standards for the security of medical records and standards for the financial industry regarding the privacy and security of customers' personal financial information. How can this confidential data be protected? Organizations may protect data through data security approaches that account for host, application or network security, to mention only a few.

New technology advances, including encryption, offer significant security value for this purpose.

Protecting data at rest

This paper focuses on a security solution for protection of data at rest, specifically protection of data that resides in databases. Most databases are deployed and stored in some kind of a persistent storage device such as a disk. Periodically, even in-memory databases have the need to backup data, hence data could end up in a persistent storage device in plaintext. Many embedded devices that contain an embedded database hold sensitive data that must be protected.

Encryption, the process of disguising data in such a way to hide its substance, is a very effective way to achieve security for data at rest. Implementation of a database encryption strategy raises several important factors that must be taken into consideration including: Should the encryption be performed inside the database engine, in the application where the data is generated or in a hardware device? Should encryption keys be kept inside the database or somewhere else where it is more secure? Should the granularity of encrypting data be applied to a database, a table or a column level? What are performance and the size considerations of different approaches?

This paper describes the implementation of the encryption solution using Empress RDBMS, summarizes the benefits of this solution for users, and discusses the issues mentioned above.

Terminology

The definitions of the basic terms are simplified for the usage in this paper. The consulted terminology sources were: **[PKCS], [FIPS]** and **[BRUCE**] (see Bibliography).

Encryption: The process of disguising data in such a way to hide its substance.

Cipher (Cryptographic Algorithm): The mathematical function used for encryption and decryption.

Cipher Key (Cryptographic Key, Encryption, Key): A parameter used in conjunction with a cipher that determines:

- the transformation of plaintext data into ciphertext data,
- the transformation of ciphertext data into plaintext data.

Plaintext (Cleartext): A block of data that has not been encrypted.

Ciphertext: A block of data that has been encrypted.

Decryption: The process of transforming **ciphertext** back into **plaintext**.

Padding: A string, typically added when the plaintext block is short. For example, if the block length is 4 bytes and the cipher requires 16 bytes, then 12 bytes of padding must be added. The padding string may contain zeros, alternating zeros and ones, or some other pattern.

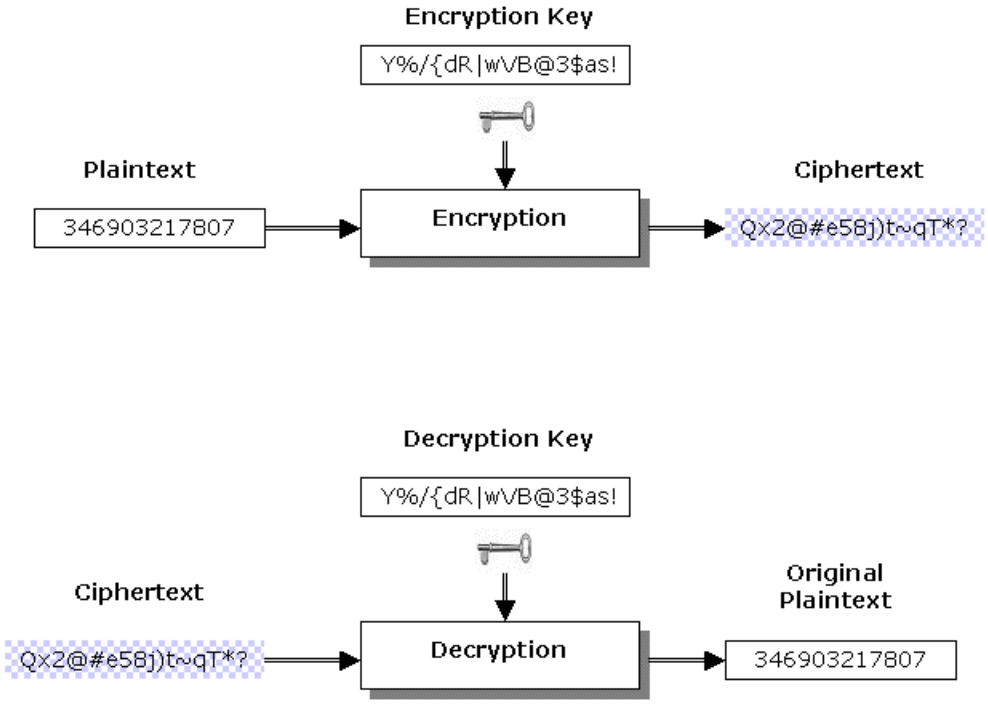

Figure 1. Encryption and Decryption

Figure 1 describes the process of encrypting and decrypting data. There are two general types of key-based algorithms: public-key and symmetric. In the case of most symmetric ciphers, the encryption and decryption keys are the same (as shown in **Figure 1**). Empress utilizes symmetric ciphers for its encryption solution.

Different Concepts of Empress Database Encryption Solution

The Empress database encryption solution is based on performing the encryption/decryption process either:

A) Outside the RDBMS by using a hardware device/appliance (**Figure 2**); Empress RDBMS can be configured with DataSecure appliances, dedicated hardware systems provided by Ingrian Networks **[INGR]**);

or

B) Inside the RDBMS (pure software solution) using a Security Library that contains an encryption algorithm (**Figure 3**); Empress RDBMS can be configured with **libgcrypt**, a general purpose library that contains various cryptographic algorithms **[GNUPG]**.

Implementing Encryption outside Empress RDBMS

Empress RDBMS incorporates C API calls from the Public Key Cryptography Standard (PKCS #11) that interface with the Security Adapter inside its Database Engine so that the encryption/decryption process is hidden from users **[PKCS]**.

PKCS is a suite of specifications developed by RSA Security in conjunction with industry, academic, and government representatives. PKCS #11 is the specification for the cryptographic token interface standard, defining a technology-independent programming interface for cryptographic applications.

Using the PKCS #11 standardized approach Empress RDBMS is effectively integrated with the Ingrain Security Adapter – NAE PKCS #11 Provider **[NAE]**. Ingrian NAE Provider initiates encrypt and decrypt operations inside the Ingrain Security Device/Appliance, such as DataSecure. Both cipher key and cipher are contained inside the Security Device. DataSecure offers support for leading, standards-based ciphers: AES, 3DES, RSA and others.

Figure 2 describes the overall concept and placement of the Security Adapter in different Empress application scenarios. Empress standalone applications and Empress utilities, such as empsql or empclean, are directly linked to the Security Adapter. In the client-server scenario, such at the ODBC and JDBC applications in **Figure 2**, the Security Adapter is linked to the Empress Connectivity Server.

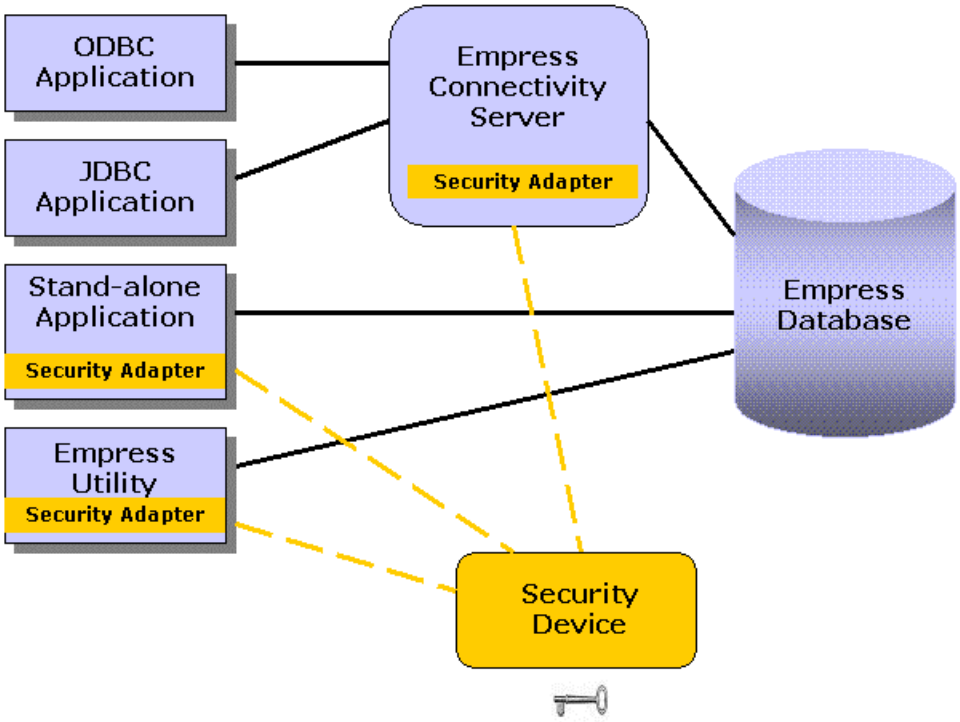

Figure 2. Concept A: Database Encryption Concept with a Security Device

Implementing Encryption inside Empress RDBMS

Figure 3 describes the second concept of adding the encryption capability to the Empress RDBMS. This is a pure software solution that involves a Security Library, which contains a cipher. Empress RDBMS is effectively integrated with **libgcrypt,** which performs data encryption and decryption. A cipher key is held either in a protected place in the file system, application/process environment or with a user.

Empress standalone applications and Empress utilities, such as empsql or empclean, are directly linked to the Security Library. In the client-server scenario, such as the ODBC and JDBC applications in **Figure 3**, the Security Library is linked to the Empress Connectivity Server.

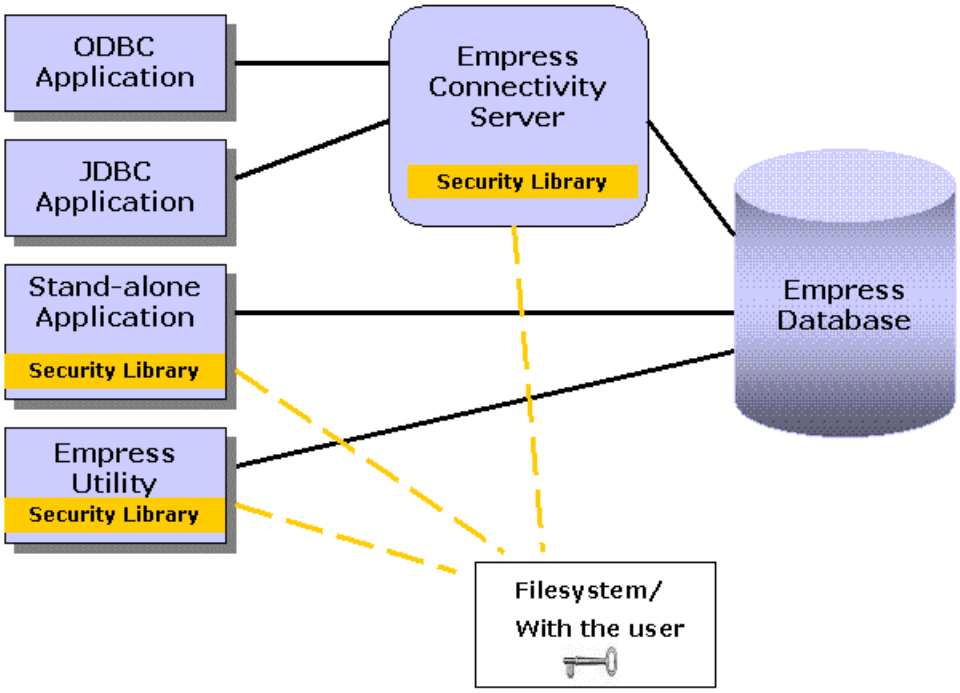

Figure 3. Concept B: Database Encryption Concept with a Security Library

How It Works

The following are the basic postulates regarding adding encryption to the Empress RDBMS. They apply to both concepts described above.

The encryption is done on a column level. Users have the capability to define which columns are to be encrypted.

Lets assume a scenario where the database table **customer** has four columns **cust_no**, **name**, **ssn** and **address,** where customer number **cust_no** and social security number **ssn** have to be encrypted. To create such a table in an Empress database one would use the SQL CREATE TABLE command:

```
CREATE TABLE customer (
        cust_no LONGINTEGER NOT NULL ENCRYPTED,
        name CHAR(20),
        ssn CHAR(9) ENCRYPTED,
        address TEXT);
```

Since the column **cust_no** requires being searchable, an index has to be created too:

```
CREATE UNIQUE INDEX customer_index ON customer(cust_no);
```

Empress RDBMS will encrypt data for the columns **cust_no** and **ssn** that need encryption and decrypt data from those columns when the application needs it.

User applications that access table **customer** need NO changing. The same scenario works for all interfaces that Empress offers and also for Empress utilities.

Furthermore, users are given the ability to toggle between an encrypted and an unencrypted form by altering database schema, changing the column that needs to be or not to be encrypted. For example:

```
ALTER TABLE customer CHANGE ssn NOT ENCRYPTED;
```

Or to define the encryption on the column again:

```
ALTER TABLE customer CHANGE ssn ENCRYPTED;
```

One ALTER command can be issued in order to define encryption on multiple columns at once.

Users do NOT have to change the data type or the size of the encrypted column.

Unit of Encryption

How much data must be encrypted to provide security? Empress encryption provides flexibility to accommodate for the varying levels of encryption granularity. Users do not have to encrypt the whole database or the whole table.

Encryption is done on a column level in the database table.

Empress groups all encrypted columns in a table record.

Padding is done on the group of encrypted columns instead of padding on every encrypted column.

An Empress database is a directory in a file system. Users can have multiple databases on the same machine and switch databases as needed.

The unit of encryption varies depending on the type of file. Types of files that contain table column data in an Empress database are: main data (.rel) files, overflow data (.dtf) files, index data (.ix) files, transaction logs, recovery logs, backup files and temporary files.

File headers are not encrypted, as they contain no user data.

Figure 4 illustrates an example scenario that describes encrypted parts and non-encrypted parts of the files related to the table **customer** in the Empress database.

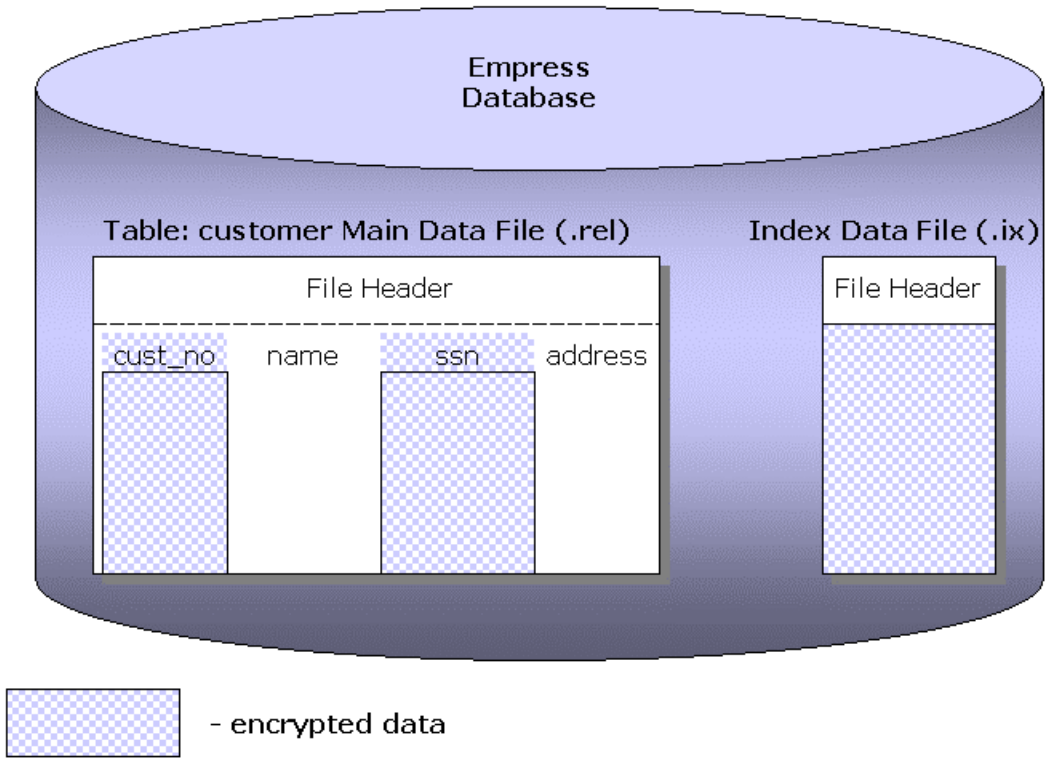

Figure 4. Encrypted and Unencrypted Parts of the Empress Database Table **customer**

There will be NO data from encrypted columns stored on the disk in plaintext.

Empress allows encryption on any column data type available in Empress RDBMS (e.g. CHAR, TEXT, BULK, INTEGER, LONGINTEGER, REAL, DOUBLE PRECISION, DECIMAL, DATE, TIME, MICROSECONDTIMESTAMP, etc.).

Encrypting Binary Large Objects/ Character Large Objects

Particular data types require more work to encrypt or decrypt data. Those kinds of data types are Empress BULK (for storing Binary Large Objects – BLOBs) and Empress TEXT and NLSTEXT (for storing Character Large Objects). The amount of data stored in the columns of these data types is expected to be large. Empress RDBMS does not impose limits on the size of those objects, eg: forcing users to split data into chunks when dealing with large volumes of binary or text data.

Encrypting Indexed Data

Users have the ability to create indexes on the encrypted columns of any Empress data type that can be normally indexed. Empress has designed encryption in its indexes in such a way to make indexes usable for all kinds of searches, not only for equality searches. Hence, there is no difference in the usability of indexes for searches on encrypted or unencrypted columns.

Type of Encryption

For encrypting and decrypting data, symmetric cipher keys are used.

Various public domain ciphers, including AES are standard with Empress and are user-selectable.

Empress also provides the means to embed user-defined ciphers.

The recommended cryptographic algorithm is Advanced Encryption Standard (**AES**). AES uses one of the 3 key lengths: 128, 192 and 256. The larger the key length the more computation it requires, and the greater security it provides.

All data in encrypted columns in a database is encrypted with the same cipher.

All the encrypted columns in a database are encrypted with one cipher key

Encryption is specified for a database when it is created. The cipher is selected at the time of creation and the selection is stored in the database. The cipher and the key should not be changed thereafter except by using the special Empress Key Rotation (Key Change) Utility.

Key Management

The Empress RDBMS engine has the basic need to obtain a cipher key for a given database. One of the essential key management questions is: Where should a cipher key reside? Some options are to store a cipher key: in a database, outside of a database in persistent storage (e.g. access-protected files), in memory, or prompt for it each time encryption is needed.

An additional option for exceptionally strong protection is to store a cipher key in the hardware security device (see **Concept A)**. All cipher keys are stored in the Ingrian DataSecure Appliance. In order to encrypt/decrypt data in the Empress database, Empress RDBMS must obtain **key info** (i.e. user name, password, key name) to pass to the Ingrian

DataSecure Appliance in order to authenticate itself and gain access to the requested cipher key.

In the **Concept B,** Empress RDBMS must obtain a cipher key to pass to a Security Library.

The following options address the need for both concepts.

1) The MS (Empress) variable MSCIPHERKEYINFO can be set to <database, key info> pairs (**Concept A**) or to <database, key> pairs (**Concept B**). Hence, key info or key resides in memory (i.e. environment of the Empress Application/Utility/Server).

2) The MS variable MSCIPHERKEYINFOFILE can be set to the name of a file containing either key info (**Concept A**) or cipher key (**Concept B**). The file is called **credentials** file.

3) An additional very secure option is for a user to input key info or a key itself each time Empress application/utility is restarted.

Cipher Key Verification

A means of verifying that a cipher key is correct for a database is highly desirable so that meaningless data is not produced with an incorrect cipher key.

Concept A

During database creation, key info is encrypted with the cipher key it relates to and stored in the database. Thereafter, a given key info is encrypted with the cipher key it relates to and compared to the copy in the database to verify that it is correct for that database. If the key info is incorrect, a sleep period of several seconds is enforced before reporting the error.

Concept B

During a database creation, the cipher key is encrypted with itself and stored in the database. Thereafter, a given cipher key is encrypted with itself and compared to the copy in the database to verify that it is correct for that database. If the key is incorrect, a sleep period of several seconds is enforced before reporting the error.

Key Rotation/ Key Change

Empress provides a utility for off-line key rotation. The utility is callable via a stand-alone interface and may be used for key rotation, either periodic or as needed.

The assumption is made that the database needs to be off-line in order to re-encrypt all of the sensitive data within the database with a new cipher key. The following steps occur:

1. A second key is generated in the Security Device or in the **credentials** file.
2. A utility reads the encrypted data from the database, decrypts the information using the first key, encrypts with the second key, and writes the new encrypted data back to the database.
3. The key parameters are updated in the database to reflect the usage of the second key.

One Key vs. Multiple Key Consideration

"One Key"
Pros:

- Uses less C API calls when communicating with the Security Device
- Efficient solution. Requires padding only once on a group of encrypted columns

Cons:

- Less flexible solution

"Multiple Key"
Pros:

- Provides flexibility
- Users need the ability to associate a different key for different columns in database tables

Cons:

- Introduces additional complexity
- Requires padding on every encrypted column
- Greater increase of the size of a database table

Performance and Size Considerations

What is an acceptable trade-off between data security and application performance?

The Empress encryption strategy includes multiple design and optimization consideration in order to ease the trade-off between data security and application performance.

Empress encryption initiates encrypt/decrypt API calls inside the Empress Engine. This preferable design offers a more efficient solution and requires much less overhead than other design alternatives such as via stored procedures/triggers.

Size Considerations

Furthermore, as an additional optimization, Empress groups columns that are to be encrypted at the record level to optimize the encryption process, thus decreasing the number of times it has to encrypt/decrypt data. See **Figure 5** for sample Internal Empress Record Layout Optimization.

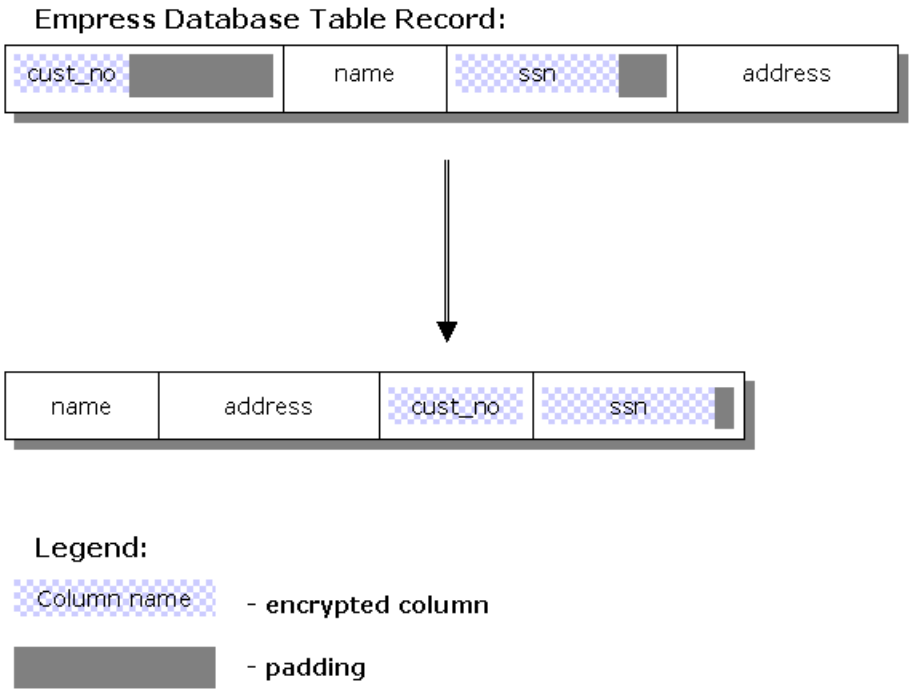

Figure 5. Internal Empress Record Layout Optimization

Empress encryption solution also saves space. Space is an issue with column encryption because, in general, encrypted columns are larger than unencrypted columns.

For example, the **cust_no** column of the data type LONGINTEGER and 4 bytes in size might need 12 bytes of padding.

The overhead in an encrypted database size compared to the unencrypted database size is largely dependant on the cipher used. Let's assume that cipher requires padding to the nearest 16 bytes. Hence, a record of 508 bytes would have to be extended to 512 bytes, making the overhead insignificant.

This internal optimization with grouping of encrypted columns is not visible to users. Hence, the original order of table columns is preserved. If for example, a user displays a table definition or retrieves data from a table, the user still get the original column order:

```
1* display customer all;
***  Table: customer  ***

  Attributes:
    cust_no                          longinteger Not Null Encrypted
    name                             character(20,1)
    ssn                              character(9,1) Encrypted
    address                          text(20,1024,1024,1)

  Creator:     srdjan
  Indices:     UNIQUE BTREE customer_index ON (cust_no)

. . .
2* select from customer;
    cust_no  name                 ssn        address
. . .
```

Future Developments

Multiple Key Approach

The current security strategy is to use a single cipher key for each database. This simplifies key management and allows for performance optimizations when Empress issues encrypt/decrypt calls.

A future solution will be implemented where a key per database column can be specified. This will allow users to tighten their overall security policy when encrypting additional data types.

On-Line Key Rotation Utility

The current Empress Key Rotation Utility is an off-line tool where the database has to be quiescent. Empress plans to upgrade this utility with the option to perform an on-line key rotation where the database can remain available to applications. The on-line process will perform a similar re-encryption process as the off-line tool, updating the new encrypted data along with the meta information that indicates which key has been used to encrypt a specific field.

In Summary

In summary, the main benefits for implementing the encryption solution using Empress RDBMS are emphasized:

- Secure all database data. A solution for protecting user data in a database including protection of all logs and backup files.

- Eliminate the potential for lost data that could be read by another party and all of the serious ramifications that accompany it, including bad press, loss of customers, government intervention.

- An efficient security solution. Insignificant performance overhead when performing the encryption/decryption process inside the RDBMS (pure software solution) and a small performance overhead when performing the encryption/decryption process outside the RDBMS by using a hardware device/appliance. Furthermore, Empress has implemented additional provisions to keep the size of the database increase minimal.

- No need for application code changes. This solution does not impact already written applications that use the data in an unencrypted Empress database.

- No need for adding external provisions in the database to accommodate encryption such as stored procedures, triggers, views, etc. The Empress solution painless for users who choose to convert their non-secure database solution to a secure one.

Literature

[BRUCE] Bruce Schneier: Applied Cryptography (Second Edition), John Wiley & Sons, 1996 ISBN 0-471-11709-9

[FIPS] FIPS PUB 140-2, Security Requirements for Cryptographic Modules, NIST, 2001.

[PKCS] PKCS #11 v2.11: Cryptographic Token Interface Standard, RSA Laboratories, Revision 1 — November 2001

[INGR] www.ingrian.com

[GNUPG] The GNU Privacy Guard, www.gnupg.org

[NAE] NAE Developer Guide for the PKCS #11 Provider, Ingrian Networks, 2005.