



DB2 Data Protection: Options and Essentials

[System*i*NEWS Magazine](#)

[Kent Milligan](#) [Terry Ford](#)

Kent Milligan

Sat, 11/01/2008 (All day)

Today, we all regularly hear about data breaches in the news. In fact, [almost 250 million data records have been compromised in 1,000 incidents since 2005](#). Forrester Research estimates that [the cost to remediate a data breach is \\$90 to \\$305 per exposed record](#). Clearly, such breaches are costly both to the individuals and to the companies involved. Even the fortress we all know as DB2 is vulnerable, so let's explore the technologies and techniques available to secure your databases on IBM i.

Data = Asset

Many people view "data" as just the input and output for programs or the information used to generate reports. If there is a cost to remediate a data breach, however, data must have value and thus is no longer just information but an asset. Indeed, the IBM Data Governance Council recently predicted that [data will become an asset on the balance sheet and that data governance will become a statutory requirement for companies](#). If this is true, the way IT thinks of data security needs to change. Companies must no longer trivialize the effort to secure assets, but rather recognize that securing information assets is a cost of doing business.

Security Fundamentals

Before we review database security techniques, we need to discuss two fundamental steps in securing information assets. First and most important is the definition of a company's security policy. Without a policy, there is no definition of what is acceptable practice for using, accessing, and storing information by who, what, when, where, and how. A policy should minimally address three things: confidentiality, integrity, and availability. The monitoring and assessment of adherence to the security policy determines whether your security strategy is working. Often, IBM consultants are asked to perform security assessments for companies without regard to policy. Although such an assessment can be useful for observing how the system is currently defined and how data is being accessed, it cannot determine the level of security, because without a policy, it really isn't an assessment so much as a baseline for monitoring and capturing the changes in the security settings. A security policy is what defines whether the system and its settings are secure (or not). A word of caution: The Health Insurance Portability and Accountability Act of 1996 (HIPAA), the Payment Card Industry Data Security Standard (PCI DSS), and other industry compliance requirements may require controls that override a company's security policy. You should therefore remain aware of industry and government regulations and adjust your policy accordingly.

The second fundamental in securing data assets is the use of resource security. Properly implemented resource security prevents data breaches from both internal and external intrusions. Resource security controls are closely tied to the part of the security policy that defines who should have access to what information resources. Hackers may be good enough to get through your company's firewalls and sift their way through to your system, but if they don't have explicit access to your database, they can't compromise your information assets.

Your eyes are now open to the importance of securing information assets. Let's examine the methods available for securing database resources on IBM i.

Current State of i Security — the Default Is Insufficient

Because of IBM i's inherently secure nature, many customers rely on the default system settings to "protect" their business data stored in DB2 for i. In most cases, this means no data protection because all users have *CHANGE authority to the data. Even more disturbing is that many IBM i customers remain in this state, despite the news headlines and the significant costs involved with databases being compromised. This default security configuration makes implementing basic security policies challenging. A tighter implementation is

required if you truly want to protect one of your company's most valuable assets — its data.

Traditionally, IBM i applications have employed menu-based security to counteract this default configuration that gives all users access to the data. The theory is that data is protected by the menu options that control which database operations the user can perform. This approach is ineffective — even if the user profile has been restricted from executing interactive commands. The reason is that in today's connected world, there are a multitude of interfaces into the system, from browsers to PC clients, that completely bypass application menus. If no object-level controls exist, users of these newer interfaces have an open door to your data.

Some customers who use this default configuration have toughened their database security with exit point solutions from third-party vendors. IBM i exit points allow a user-written program to be called every time a particular interface (e.g., FTP) is used or a specific event occurs (e.g., profile created). Security tools based on these exit points increase the level of security on a system by locking down interfaces not under the control of menu-based or application authority. In addition, exit point solutions let customers implement more granular security controls, such as allowing users access to the database only during certain hours of the day.

Although exit point solutions can provide great benefits, they are not an alternative to object-level control of your databases. Exit point solutions help secure interfaces, but they don't completely protect the data stored in your DB2 objects. Exit points do not exist for every data access interface on the system. Thus, if an application starts using an unprotected interface, the only thing protecting your data is object-level access control. When your security implementation totally relies on exit points, you must also track new data interfaces that pop up as IBM delivers new releases and products. By doing so, you can ensure that your exit point solution provides coverage for the new interfaces.

An exit point solution is a good option for databases with security holes caused by a reliance on the default security setup or menu-based control. However, your security work shouldn't stop there. Instead, you need to continue to work on a complete database security solution by controlling data access at the object level.

DB2 Security Approaches

Now that you understand why object-level controls are needed, let's look at the approaches commonly used to implement a security policy. In the IBM i camp, the main methodologies are to use the program-adopted authority model or define private authorities on each object.

The Adoption Approach to DB2 Security

Program adopted authority can simplify your security implementation by reducing the number of user profiles that must be granted access to DB2 objects. Because of the low number of individual authorities, this security model is the most efficient for the IBM i built-in security manager to enforce. With this method, public access to the DB2 objects is turned off by specifying *EXCLUDE for PUBLIC authority, and then the only user with access rights to the object is the user profile that owns the program.

When a program is running on the system and references DB2 objects, the default behavior is for the job's user profile to be validated against the security controls in place for the referenced object. For example, if APPUSER1 is the user profile running program WORKAPP, any time WORKAPP tries to access a DB2 object, the security manager checks whether the APPUSER1 user profile has the necessary authority to perform the requested operation on that DB2 object.

With program adopted authority, you can create (or change) the program to use the authority of the owner of the program. To do so, specify a parameter of USRPRF(*OWNER) on the Create Program (CRTPGM) or Change Program (CHGPGM) commands, as [Figure 1](#) demonstrates.

In the example in [Figure 1](#), let's assume that PRFWRKAPP is the profile that owns the program object. Now, APPUSER1 starts a session and runs the WORKAPP program. With adopted authority in place, every time WORKAPP references a DB2 object, the privileges for both the APPUSER1 and PRFWRKAPP user profiles are checked to determine whether the program is authorized to perform the specified database operation.

In addition to changing the program to adopt the owner authority, some data authorities need to be given to the program owner. With program adopted authority, typically only the object owner user profile is given access to the DB2 objects, and public access is restricted with *EXCLUDE authority. Furthermore, the application users need *EXECUTE authority on the program object. [Figure 2](#) contains an example of commands that you would

need to combine with the commands in [Figure 1](#) to complete the program adopted authority setup. This example assumes that WORKTAB is the only DB2 object that the application accesses.

This adopted authority example should demonstrate how using the adopted authority model can simplify your database security setup. Instead of granting access to the WORKTAB table to every user of the application, only the program owner (PRFWRKAPP) needs privileges to change and read the WORKTAB table.

Program adopted authority sounds as if it's tied to legacy AS/400 technology, so you may be wondering whether you can use this approach with newer technologies, such as Java and SQL. The answer is yes. When SQL is embedded in programs, the SQL precompilers let you specify *OWNER for the USRPRF (User Profile) and DYNUSRPRF (Dynamic User Profile) parameters. The same options exist on the SET OPTION clause for SQL procedures, triggers, and functions.

Java applications aren't associated with IBM i program objects, so of course there's no program object that you can alter. For these types of situations, applications can use the swap profile API set to emulate the adopted authority model. The swap profile API set consists of Get Profile Handle (QSYGETPH), Set Profile (QWTSETP), and Release Profile Handle (QSYRLSPH). The swap profile API set is executed at the beginning of the application to switch from the user profile running the application to a user profile that has the database authorizations the application requires.

When the application finishes running, the swap profile API set is used to release the swapped user profile and switch back to the original user profile. This technique is similar to the processing of program adopted authority: After the program call completes, the user running the application reverts to the authorizations rights that his or her user profile holds.

Thus far, we've discussed only the advantages of the adopted authority model. Now let's look at the disadvantages of the adopted authority approach. One drawback is the exposure that exists if the program running with adopted authority presents the application user with a command line. For example, maybe the WORKAPP application provides printing capabilities and a menu option that takes the user to the Work with Spooled Files (WRKSPLF) interface. A command line is present on the WRKSPLF interface. Any operations that APPUSER1 executes from that command line interface also include the new authorizations added with the adopted authority (PRFWRKAPP). The application user is free to use these adopted authorities from the command line in any manner that he or she chooses. To minimize this risk, you can use the USEADPAUT parameter on the CHGPGM command. Clearly, however, the implementation of program adopted authority needs careful planning to eliminate the ability for the application user to use the adopted authorities outside of the application.

The program adopted authority model also requires the ability to alter the program object attributes or the program code itself, but if you're using IBM or third-party programs, you can't make these kinds of alterations. Consider the scenario in which a business analyst wants to create a report by using Query/400 or IBM DB2 Web Query for i. In this case, extra programming is necessary to make the adopted authority approach work. For example, you could write a wrapper CL program that adopts the necessary authority for the Query/400 report to work. DB2 Web Query can query a result set returned from a stored procedure call, so you could code a stored procedure that adopts authority. Does your IT staff have enough resources to create a program for every report? What if it's impossible to create a wrapper for the third-party tools such as System i Navigator? You have to research and investigate third-party software and database interfaces outside of your company's application before depending too much on adopted authority to protect your DB2 databases.

The Private Authority Approach to DB2 Security

The private authority approach really amounts to rolling up your sleeves to perform the hard work of analyzing and securing each DB2 object on the system with the proper authorities. Instead of defining access rights for each user profile on the system, you can use group profiles and authorization lists in IBM i to reduce the number of security privileges that you must grant. This private authority approach probably already sounds like too much work. However, if you and your company view the data stored in your DB2 databases as one of the organization's most valuable business assets, it's easy to justify the effort to secure them effectively.

This private authority approach has the advantage of delivering a security implementation that is difficult to breach — even as new interfaces and applications are introduced over time. The reason the security is so effective is that you're securing the database objects themselves instead of just the programs or interfaces. As we

mentioned, the challenge with this approach is the time involved in defining and managing all the private authorities for each DB2 object. A small performance degradation also occurs with this methodology because there are more individual authorities for the IBM i security manager to process and validate. The performance impact will depend on a number of factors, including the number of objects and the number of authorities.

To simplify the administration of database security policy on DB2 for i, you can use group profiles and authorization lists. First let's explain group profiles. IBM i user profiles can belong to one or more groups. With this feature, you can grant authorities to a group of users instead of having to define authorities for each user profile. For example, there may be one group profile for your sales database (GPSales) and another group profile for the payroll database (GPPay). You add individual user profiles to these groups and then grant authority on the database object to the group profiles. Group profiles are often based on department or business role (e.g., help desk, teller).

[Figure 3](#) shows an example of an implementation that uses a group profile setup. Notice in the example that there's no command to create a group profile. Instead, a user profile is implicitly transformed into a group profile when that user profile is referenced on the group profile parameters (i.e., Group Profile — GRPPRF and Supplement Group Profile — SUPGRPPRF) on the CRTUSRPRF or CHGUSRPRF commands. Typically, a group profile is created with PASSWORD(*NONE), as in this example, so that no individual user can use the group profile to sign on.

Notice in the example that APPUSER3 belongs to both group profiles. A user profile can belong to up to 16 group profiles. Group profiles can't belong to other groups. The first group must always be specified on the GRPPRF parameter and additional groups on the SUPGRPPRF parameter. The main difference between these two group parameters is that the OWNER, GRPAUT, and GRPAUTTYP parameter values for the individual user profile apply only to the group profile specified on the GRPPRF parameter. These parameters control how object ownership is assigned when a user profile belonging to a group creates an object.

The IBM i security manager doesn't add together the individual and group profile authorities when determining whether the user profile is authorized to perform a database operation. The security manager examines the credentials for a user and group profile separately. When you use group profiles, it's usually best for individual user profiles to have no authorities (even *EXCLUDE) to the DB2 objects, with object authorities granted to the group user profile as in [Figure 3](#). Another tip for group profile implementation is to use the Change Object Primary Group (CHGOBJPGP) command, which [Figure 4](#) shows, to improve performance. This command improves performance by storing information about the specified group profile's authority in the table object itself.

Authorization lists are another way you can simplify the management of private authorities, and you use them similarly to the way you use group profiles. Authorization lists provide a way to group objects that have comparable security requirements. For example, a data warehouse contains several tables that include historical sales data for business analysts to explore. In this case, each business analyst using the data warehouse needs access to each table. Instead of granting private authorities on each table, you can create an authorization list that effectively provides users with the necessary authority to each table associated with the authorization list.

Think of an authorization list as containing a list of user profiles and the authority that each user has for the DB2 objects associated with the authorization list. [Figure 5](#) shows an authorization list in action. The first step is creating the authorization list object with the Create Authorization List (CRTAUTL) command. The AUTHORITY parameter on the CRTAUTL command isn't the authority that each user profile has to the DB2 objects associated with the list. If an associated DB2 object has *AUTL for its public authority setting, the AUTHORITY parameter value (*EXCLUDE in this case) from the authorization list is used as the public authority setting of the DB2 object. The next step is putting the database object under the security control of an authorization list. This is done with the GRTOBJAUT command. An IBM i object can be associated with only one authorization list at a time.

The final step involves giving users the authority to the authorization list. After this step the user profiles have the specified authority to any object the authorization list secures. You perform this step with the Add Authorization List Entry (ADDAUTLE) command — no SQL statements to do this task exist. In this example, the first two users are given *USE authority to the DB2 objects secured by the list, and the last administrator user is given *CHANGE authority. The result is that users BIZUSER1 and BIZUSER2 have *USE authority for any DB2

object associated with the authorization list, whereas DWADMIN user profile has *CHANGE capabilities.

A key concept to understand is that user authority is defined for the authorization list, not for the individual objects secured by the list. If a new DB2 object is secured by the authorization list, the users on the list automatically gain authority to the object. This behavior makes it easy to secure new objects with security requirements that are the same as existing objects in your database.

When you secure a DB2 table with an authorization list, you can change authorities even when the table is open. With group and individual user profiles, you can't grant or revoke authorities from an open object. This fact may lead you to question whether an authorization list is better than a group profile. The answer is that neither is better. The choice of which technique to use really boils down to which option best solves your problem. In fact, you can add a group profile to an authorization list in the same way that you add an individual user profile. Many shops use these two options together.

As with program adopted authority, a security implementation that uses the private authority approach should involve reducing the public authority to *EXCLUDE if at all possible. Excluding all users from accessing a DB2 object is much safer than leaving the door open to your DB2 object by default. It might also be possible to use a combination of adopted authority and private authorities to reduce the database security administration, with private authorities being employed only to address the interfaces not easily supported by adopted authority.

Although a private authority implementation requires more time to implement and manage, it delivers security defenses that protect your data across all interfaces. This protection includes interfaces that exist today and those delivered in the future. With this understanding of the two basic approaches to DB2 security, let's review more detailed security considerations for the interfaces and objects in your database.

Connection- and Interface-Level Controls

Securing your database at the interface level was an option that we touched on earlier during the discussion of exit point programs. Exit point-based solutions do let you disable or limit user database access from data interfaces such as ODBC and FTP connections. However, complete protection is impossible with these solutions because exit points don't exist for all data access interfaces.

Even if the database is secured at the object level with private authorities, you must consider the business data being transmitted over the network. Once a user has been authorized to the data, database and interface security controls offer no protection to the data being transported in the clear across the network. Here are a few of the options available on IBM i to protect your data transmissions:

Transport Layer Security (TLS)/Secure Sockets Layer (SSL). TLS and its predecessor SSL are technologies that you can deploy to protect business data being transmitted over the network. TLS/SSL secures the data transmission by encrypting the data. Extra system resources are needed to encrypt the data transmissions, but IBM offers a cryptographic card to offload the TLS/SSL encryption processing from the main processors.

Secure Virtual Private Network (VPN). A Secure VPN uses cryptographic protocols such as IP Security (IPSec) to provide secure communications over unsecured networks. To provide the necessary data protection, a VPN must be designed and implemented with clearly defined security policies.

Secure Shell (SSH)/OpenSSH. SSH is a network protocol that lets you securely transfer data between systems. A wide variety of operating systems support this protocol.

Homegrown encryption. Applications can use IBM i cryptography services to encrypt the data before sending it over the network and have the target system decrypt the data. This approach requires substantial programming resources to implement the encryption/decryption as well as extra administration to ensure that utilities such as FTP are accessing only encrypted objects.

Schema-Level Controls

You may think that you can simplify management of object-level access controls by just defining authorities at the schema (library) level. A schema is a container for all DB2 objects, so controlling access to the containing object seems a logical way to control access to the objects within it. Unfortunately, this is untrue because schema-level controls offer insufficient granularities. A user profile needs a minimum of *USE authority to

access any of the DB2 objects within a schema. The data authorities defined at the table or object level are then used to determine whether a user can actually access the business data stored in a DB2 object. Hence, there are no security management shortcuts at the schema level — user profiles need to have authorities granted at both the schema and the object level.

An SQL schema and IBM i library are equivalent objects; however, the default public authority given to the respective objects is different. With the Create Library (CRTLIB) command, the CRTAUT parameter controls the default public authority. The default for this parameter is *SYSVAL, which means that the QCRTAUT system value determines the policy for public access. The system value default is *CHANGE, which is the reason for our earlier statement about IBM i shops giving all users on their system *CHANGE access to their databases. To remedy this problem, IBM recommends changing QCRTAUT to *EXCLUDE or *USE in most cases to limit public access to your database objects. You will most likely need some changes to your environment and programs to accommodate the change to the QCRTAUT system value.

The SQL Create Schema statement creates a library with a different public authority when SQL naming (*SQL) is used. The naming format parameter is available on all SQL interfaces that enable either SQL or System (*SYS) naming. If you use System naming, the public authority behavior follows the CRTLIB semantics. With SQL naming in effect, a library (and all SQL objects) is created with a public authority of *EXCLUDE. Thus, SQL naming forces you to explicitly grant public access to the object or define private authorities. Again, this behavior follows IBM's general guideline of excluding all public access to the schema by default.

Table-Level Controls

Although a user needs schema-level privileges in order to change or add data, table-level access controls are the key to securing your business data. This detail is true whether the security implementation uses private authorities or program-adopted authorities.

Object ownership is one aspect of data security that we haven't yet discussed. When a DB2 table (physical file) is created, a user profile is assigned ownership of the object. The owner of a DB2 table can perform any operation on that table. Obviously, you must carefully plan the access levels of the owners of each object in your database.

The way you assign object ownership depends on the interface you use. With non-SQL interfaces, you assign ownership to the user profile or group user profile of the job that creates the database object. With SQL interfaces, you control the object ownership behavior by using the naming format parameter — just like public authority. For DB2 tables that you create with system naming, assign object ownership in the same manner that you do with non-SQL interfaces. When you create a table with SQL naming, the owner of the table is the user profile with the same name as the schema into which the object is created. For example, if you create table TAB1 into a schema named USER1, the object owner for TAB1 is USER1. When no user profile matches the name of the schema, the owner of the object is the user or group profile of the job creating the table. Assigning object ownership to a group profile is unwise because that lets every member of the group profile inherit ownership authority of the created object.

Column- and Row-Level Controls

Some database security implementations give users no direct access to the table. Instead, users have access only to views (i.e., logical files) defined over the tables. Because views can contain a subset of the columns (i.e., fields) in a table, views can strengthen database security by letting columns containing sensitive data be hidden from the user.

The example in [Figure 6](#) uses SQL to demonstrate this security technique in action. You can also implement the technique by using logical files and the CL security commands. In this example, EMP_TAB is the DB2 table that contains data on each employee. The first step is defining an SQL view, EMPVIEW, that omits the sensitive salary column from its definition. Access to the underlying table is then taken away with the REVOKE statement, and "normal" users are given only data authorities to the view with the GRANT statement. The result is that USER1 is authorized to access just the employee ID and name because those are the only columns defined in the view.

We create a second view for those users in human resources (HR) who require access to the salary column. Notice that even the HR user profile has no privileges to access the table — access is again limited to the view, empview_hr. As new columns are added to the underlying table, view users have to be explicitly granted access

to the data in the new columns instead of automatically giving all authorized table users access to the new column.

Instead of hiding sensitive columns, you could also use views to mask data in sensitive columns, as in the following example:

```
CREATE VIEW empview_mask AS
```

In addition, if you need to restrict users to a subset of rows within a table, you could add selection predicates to the view definition, as the following example demonstrates.

```
CREATE VIEW empview_hrsup AS
```

```
SELECT empid, empname, empsalary
```

```
FROM emp_tab
```

```
WHERE empsalary < 1000000
```

The main downside to the view approach is that creating additional views means that you have additional DB2 objects to manage and administer. If you're using only SQL views and nonkeyed logical files, there's no performance impact to the applications, because these DB2 objects don't contain any data that has to be maintained as the underlying table changes. If you use a keyed logical file for this purpose, a slight performance overhead will exist because DB2 has to update the logical file each time the underlying table changes.

DB2 for i also supports the capability to define data authorities at the column level. However, people use this support infrequently because they can control only update operations at the column level — no column-level support exists for read operations. Being able to control updates at the column level can help in situations in which you need to let users update some of the columns in a table, but not all of them. For example, in [Figure 7](#), the HRADMIN user profile is granted access to perform any data operation on any column in the emp_tab table. In contrast, the HRUSER profile is limited to updating the empid and empname columns and reading rows from the table. Column-level security is supported only by the SQL GRANT and REVOKE statements.

Data Protection with Encryption

Curious about why we haven't mentioned data encryption yet? The reason is that the encryption offers minimal protection if a database has not been secured first with object-level access controls. This is especially true if you're using a solution in which the data is automatically encrypted on the way in and automatically decrypted by the application each time the data is accessed — all authorized users are able to view the clear-text version of the data. Your best security investment is getting your database secured at the object-level before even considering encryption, in part because a good encryption implementation also requires an access control scheme, which we discuss later in this article.

Because government legislation and auditors mandate that sensitive data stored on disk be encrypted, many IBM i customers are looking for "automatic" encryption schemes. This customer demand resulted in IBM delivering auxiliary storage pool (ASP) encryption in IBM i 6.1. This support enables IBM i to automatically encrypt data as it's written to a basic ASP or an independent ASP (IASP). When the operating system or an application reads data read from an encrypted ASP, the data is automatically decrypted. This type of encryption approach offers several advantages:

- a quick implementation method that satisfies the requirements of many security auditors
- data protection in the case of a drive theft
- data protection when a bad drive has to be replaced and returned to the vendor

The downside of automatic encryption is that the data is presented to all authorized system users in cleartext form. Thus, it's not reducing the number of application users who can view your credit card number or Social Security number. In addition, this approach requires the creation of an ASP as well as moving existing data into the encrypted ASP. To implement ASP-level encryption on your system, you must purchase and install the IBM i licensed program product 5761SS1 Option 45 - Encrypted ASP Enablement.

A more secure encryption solution involves selectively decrypting data. With this method, you must enhance application solutions to examine who is attempting to view a column that contains encrypted data and determine whether that user is authorized to see the decrypted version of the data. The method is similar to the access controls necessary for securing databases at the object level. Ideally, applications would require a form of user authentication, such as a password or a biometric reading, to determine whether the user has the credentials needed to access encrypted data. In some cases, it may be more practical to remove sensitive data from reports instead of worrying about how the data will be decrypted. At this point, it should be obvious that a proper implementation will require a fair amount of coding changes to your application and interfaces. In addition, a system upgrade may be required to handle the increased CPU demand that encryption algorithms impose.

Application-based encryption also needs to securely handle the management of and access to encryption keys (for some good tips, see "Best Practices for Key Management" on page ProVIP 3). Once the data is encrypted, the encryption key is as valuable as the decrypted version of the data. The encryption key would preferably be stored in a separate hardware device and retrieved programmatically to provide the most protection. IBM delivered a set of key management APIs in V5R4 to help programmers protect and manage encryption keys. (For more information about key management APIs, as well as some downloadable utilities that help you put them to work, see "APIs by Example: Cryptographic Key Management," below.)

Keeping the encryption key separate from the encrypted data is an important design point for application-based encryption. This point is also true for encrypted backups of your database. To truly protect your backups with encryption, be sure to store the encryption key on different media from the encrypted data.

Even though encryption can add another layer of protection to your database, you can see that a large amount of work is involved in setting up a complete encryption solution for your applications and databases. The Redbook [Protecting i5/OS Data with Encryption](#) provides further details and considerations.

An Ongoing Endeavor

You should now have a good understanding of the technologies and techniques available to secure your databases on IBM i. As you've seen, no single tool or single implementation of technology can make a business's information assets secure. You must attend to these data assets regularly and with diligence. Securing information assets is an ongoing task of planning, monitoring, and implementing — if you're committed to keeping your data and your company out of the news.

***Terry Ford** started his IT career in 1982 as a programmer in local Illinois government designing municipal and court applications. He started with IBM in 1988 as an SE in the field and later moved to the Rochester Lab. He has spent the last 15 years in competitive analysis, in the Customer Benchmark Center, and in his current position as security practice leader for STG Lab Services. Terry has authored several articles and white papers and holds a patent on workload characterization.*

Kent Milligan is a senior DB2 for i consultant on IBM's ISV Enablement team for IBM i. Kent spent the first eight years of his IBM career as a member of the DB2 development group in Rochester. He speaks and writes regularly about relational database topics.

APIs by Example: Cryptographic Key Management

The following articles written by Carsten Flensburg and published in the *System iNetwork Programming Tips* email newsletter discuss and demonstrate key management with the Cryptographic Services APIs. They are part of Flensburg's long-running and well-received "APIs by Example" series.

- "APIs by Example: AES Encryption to Actual Field Length" (August 28, 2008, article ID [57114](#))
- "APIs by Example: Cryptographic Key Management - Encrypt/Decrypt with Key Hierarchy" (April 24, 2008 article ID [56586](#))
- "APIs by Example: Cryptographic Key Management - Creating Data Key Stores and More" (March 27, 2008, article ID [56462](#))
- "APIs by Example: Cryptographic Key Management - Creating, Displaying, and Deleting Key Records" (February 28, 2008, article ID [56351](#))
- "APIs by Example: Cryptographic Key Management - Creating and Translating Key Stores" (January 24,

2008, article ID [56187](#))

- "APIs by Example: Cryptographic Key Management - Testing and Clearing Master Keys" (December 13, 2007, article ID [56035](#))
- "APIs by Example: Cryptographic Key Management - Loading and Setting Master Keys" (November 8, 2007, article ID [55862](#))

Source URL: <http://iprodeveloper.com/security/db2-data-protection-options-and-essentials>