Data privacy and security continue to be hot buttons for consumers. That's increasingly true with the growing number of Web-based applications that gather critical information for an increasing number of users. This article describes how to use new functions available in IBM DB2 UDB Version 7.2 to easily integrate data encryption into database applications.

© 2001 International Business Machines Corporation. All rights reserved.

Introduction
This paper describes how to use new functions available in IBM DB2 Universal Database Version 7.2 (Unix & Windows) to easily integrate data encryption into database applications. For years, databases have been able to keep unauthorized persons from being able to see the data. This is generally covered by privileges and authorities within the database manager. In today's environments, there is an increasing need for privacy of stored data. This means that even though a DBA may have complete access to the data in a table, there is information that the owner of the data would not want anyone else to see. This has surfaced in particular with web-based applications where the user has entered data (such as credit card numbers) that is to be kept for subsequent uses of the application by the same user. People want assurance that nobody else can access this data.

This functionality has been added to DB2 v7.2 by implementing SQL built-in functions that allow the application to encrypt and decrypt data. When data is inserted into the database it can be encrypted using an encryption password supplied by the user. When the data is retrieved, the same password must be supplied to decrypt the data. For situations where the same password is going to be used several times, the ENCRYPTION PASSWORD value can be set using an assignment statement and is valid for the length of a connection.

This paper describes the SQL functions and gives some examples of how the encryption functions could be used. We also discuss the design and performance implications of having encrypted data in a relational database.

Implementation
The signatures of the new SQL functions are shown below. More detailed documentation is available in the SQL Reference section of the DB2 documentation. (To insure you are using correct
data types and lengths for encrypted data be sure to read the "Table Column Definition" section under the ENCRYPT function in the SQL Reference.)

Did you get a SQL0440N?
If your first attempt to use the Encrypt function fails with error message SQL0440N and you are using a database that was created prior to Version 7.2 (version 7.1 Fixpak 3), ensure that a user with SYSADM authority has executed the system command `db2updv7` against your database.

The algorithm used to encrypt the data is an RC2 block cipher with a 128 bit secret key. The 128 bit secret key is derived from the password using a message digest. The encryption password is not tied to DB2 authentication, and is used for data encryption and decryption only.

An optional parameter, PasswordHint, can be provided and is a string that would help a user remember the PasswordOrPhrase that is used to encrypt the data. (For example, 'George' as a hint to remember 'Washington'.)

Column Level Encryption

Column level encryption means that all values in a given column are encrypted with the same password. This type of encryption can be used in views, and when one common password is used. When the same key is used for all of the rows in a table or tables, the ENCRYPTION PASSWORD special register can be quite useful.

Example 1: This example uses the ENCRYPTION PASSWORD value to hold the encryption password. An employee social security number is encrypted and stored in the EMP table in encrypted form.

```sql
create table emp (ssn varchar(124) for bit data);
set encryption password = 'Ben123';
insert into emp (ssn) values(encrypt('289-46-8832'));
insert into emp (ssn) values(encrypt('222-46-1904'));
insert into emp (ssn) values(encrypt('765-23-3221'));
select decrypt_char(ssn) from emp;
```

Example 2: This example uses the ENCRYPTION PASSWORD value to hold the encryption password in combination with views. The following statement declares a view on the emp table:

```sql
create view clear_ssn (ssn) as select decrypt_char(ssn) from emp;
```

In the application code we set the ENCRYPTION PASSWORD to 'Ben123' and can now use the clear_ssn view.

```sql
set encryption password = 'Ben123';
select ssn from clear_ssn;
```
Row-Column (Cell) or Set-Column Level Encryption

Row-Column (cell) or Set-Column level encryption means that within a column of encrypted data many different passwords are used. For example, a web site may need to keep customer credit card numbers (ccn). In this database each customer could use his own password or phrase used to encrypt the ccn.

Example 3: The web application collects user information about a customer. This information includes the customer name which is stored in host variable `custname`, the credit card number which is stored in a host variable `cardnum` and the password which is stored in a host variable `userpswd`. The application performs the insert of this customer information as follows.

```sql
insert into customer (ccn, name)
values(encrypt(:cardnum, :userpswd), :custname)
```

When the application needs to re-display the credit card information for a customer, the password is entered by the customer and again stored in host variable `userpswd`. The ccn can then be retrieved as follows:

```sql
select decrypt_char(ccn, :userpswd) from customer where name = :custname;
```

Example 4: This example uses the hint to help customers remember their passwords. Using the same application as example 3, the application stores the hint into the host variable `pswdhint`. Assume the values 'Chamonix' for `userpswd` and 'Ski Holiday' for `pswdhint`.

```sql
insert into customer (ccn, name)
values(encrypt(:cardnum, :userpswd, :pswdhint), :custname)
```

If the customer requests a hint about the password used, the following query is used.

```sql
select gethint(ccn) into :pswdhint from customer where name = :custname;
```

The value for `pswdhint` is set to "Ski Holiday."

Encrypting Non-Character Values

The encryption of numeric and date/time data types is indirectly supported via casting. By casting non-character SQL types to "varchar" or "char" they can be encrypted. For more information on casting, see the "Casting Between Data Types" section in the SQL reference documentation.

Example 5: Casting functions used when encrypting and decrypting TIMESTAMP data.

```sql
-- Create a table to store our encrypted value
create table etemp (c1 varchar(124) for bit data);
set encryption password 'next password';

-- Store encrypted timestamp
insert into etemp values encrypt(char(CURRENT_TIMESTAMP));

-- Select & decrypt timestamp
select timestamp(decrypt_char(c1)) from etemp;
```
**Example 6**: Encrypt/Decrypt double data.

set encryption password 'next password';
insert into etemp values encrypt(char(1.11111002E5));
select double(decrypt_char(c1)) from etemp;

**Performance**

Encryption, by its nature, will slow down most SQL statements. If some care and discretion are used, the amount of extra overhead should be minimal. Also, encrypted data will have a significant impact on your database design. In general, you want to encrypt a few very sensitive data elements in a schema, like Social security numbers, credit card numbers, patient names, etc. Some data values are not very good candidates for encryption -- for example booleans (true and false), or other small sets like the integers 1 through 10. These values along with a column name may be easy to guess, so you want to decide whether encryption is really useful.

Creating indexes on encrypted data is a good idea in some cases. Exact matches and joins of encrypted data will use the indexes you create. Since encrypted data is essentially binary data, range checking of encrypted data would require table scans. Range checking will require decrypting all the row values for a column, so it should be avoided or at least tuned appropriately.

The following scenario illustrates our discussion. Consider a common master-detail schema where one programmer can work on many projects. We will implement column level encryption on the employee's social security number(ssn). In the master table emp and the detail table empProject the ssn will be stored in encrypted form.

```sql
-- Define Tables and Indexes for encrypted data
create table emp (ssn varchar(48) for bit data, name varchar(48) );
create unique index idxEmp on emp ( ssn ) includes (name) ;
create table empProject( ssn varchar(48) for bit data, projectName varchar(48) );
create index idxEmpPrj on empProject ( ssn );

-- Add some data
set encryption password = 'ssnPassWord';
insert into emp values (encrypt('480-93-7558'),'Super Programmer');
insert into emp values (encrypt('567-23-2678'),'Novice Programmer');
insert into empProject values (encrypt('480-93-7558'),'UDDI Project');
insert into empProject values (encrypt('567-23-2678'),'UDDI Project');
insert into empProject values (encrypt('480-93-7558'),'DB2 UDB Version 10');

-- Find the programmers working on UDDI
select a.name, decrypt_char(a.ssn)
from emp a, empProject b
where a.ssn = b.ssn
and b.project = 'UDDI Project';

-- Build a list of the projects that the programmer with ssn
-- '480-93-7558' is working on
select projectName
```
The following is an example of how **not** to write the two queries on the emp and empProject table. Although these queries return the correct answers, they also decrypt the ssn for all rows. When tables get large this problem will become significant.

```
from empProject
where ssn = encrypt('480-93-7558');
```

This would require decryption of every row of the emp table and each 'UDDI Project' row of the empProject table to perform the join.

```
select projectName
from empProject
where decrypt_char(ssn)= '480-93-7558';
```

This would require decryption of every row of the empProject table.

### Summary

In this paper we have demonstrated how the new encryption functions in IBM DB2 Universal Database Version 7.2 provide a simple way to encrypt sensitive data. These functions can be used to implement column and row-column level encryption. There are some important performance implications that developers should review during design and implementation. Data encryption adds a new tool to be used to hide private data, even from administrative staff.