Why XP_CRYPT & SQL Shield?

A Project Manager's Perspective.

PART I: DEFINING THE NEED. Where does SQL Server Lack Protection?

Protecting Field Level Data

Surprisingly, SQL Server does not encrypt data at the field level. Access to data is granted by logging into the database. But the fact is that if anyone has access to the file system, they can merely copy the database files, paste them onto a SQL Server that they have System Administrator permissions on, and then they have complete access to your data.

So the truth regarding security of SQL Server is that it is strong, as long as someone cannot get into your file system. With the many exploits and hackers out there, that is a lot of faith to have when you are responsible for protecting particularly sensitive data (such as credit card numbers, health information, etc).

Protecting Store Procedures and Scripts

SQL Server allows the developer to code logic into the database. This logic is stored as stored procedures, triggers, and user-defined functions. There are a couple reasons why you want to encrypt this logic:

First, there are intellectual property concerns. If someone can see your scripting logic, that is the same as seeing your source code. That means they can understand the "secret" inner workings of your project, and that makes reverse-engineering much easier.

Second, if someone can look into your stored procedures, they can easily edit them. That means they can rewrite your stored procedures and put special logic in there that will affect your database. What could this mean? Possibilities could include their deleting data, or breaking your database, to more sinister acts such as stealing. I.e. having secret medical data written or retrieved if a special token is "submitted", or perhaps in an e-commerce application, crediting a particular person's account with money rather than debiting them every time they bought something.

Many developers would argue that many items "should" be encrypted at the database field level. These items may include credit card numbers, Social Security Numbers, and other private data, such as medical information.

Sounds scary huh? Well luckily there is a solution to both these problems…

PART II: CODE SOLUTION – How To Successfully Protect SQL Code

Protecting Scripts…

Open up your help file in SQL Server, and you will quickly learn that SQL Server DOES offer encryption for Stored Procedures and Scripts. But before you sigh a relief – here is what the help file doesn't tell you – in 5 minutes you can surf the web and download for free one of many programs that can decrypt your "Microsoft encrypted" stored procedures in an instant. That means that any hacker worth a grain of salt has the ability to get into your SQL code and do what they want even though you have encrypted it using SQL Servers "native" encryption.

What can you do? Well, luckily the SQL Shield offers encryption of your stored procedures that no known hacking program can decrypt. That means that when the hacker sees that your scripts are encrypted, no matter how many times he tries to use these hacking toolkits that are available, he will not be able to decrypt your SQL scripting code, so that makes you safe.

To that end, it is becoming ever more important to protect your data.

PART III: DATA SOLUTION – How To Increase Protection of SQL Data

There are many different algorithms that you can use to encrypt your data. XP_CRYPT includes RSA (asymmetric algorithm), AES, Triple DES, DESX and RC4 (symmetric algorithms). You can choose an algorithm depending on your needs.

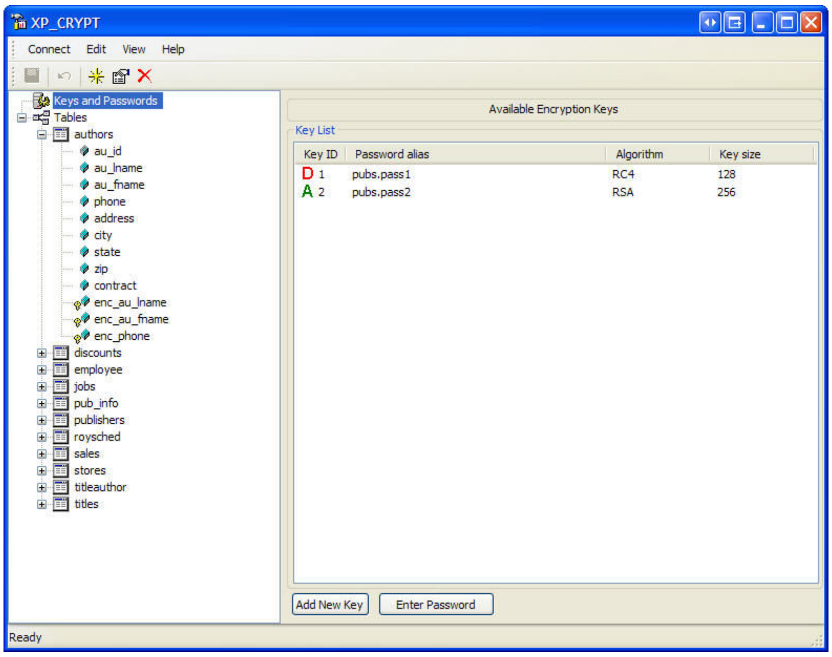
However note that asymmetric algorithms are relatively slow encryption comparing to symmetric algorithms.

Encrypting data fields is a snap with XP_CRYPT, using the XP_CRYPT GUI, which is basically a program that will inject code routines into your database easily.
The XP_CRYPT GUI automates a significant amount of work. It adds all the interface and support code and applies it to your database. You can add multiple algorithms easily, each with their own keys. Due to the extra overhead from some of the more powerful algorithms, it may indeed be prudent to offer different types of encryption for different types of fields based on the length and type of the field, and the security importance.

In less than 5 minutes you can encrypt fields in your database. The program will take care of the following:

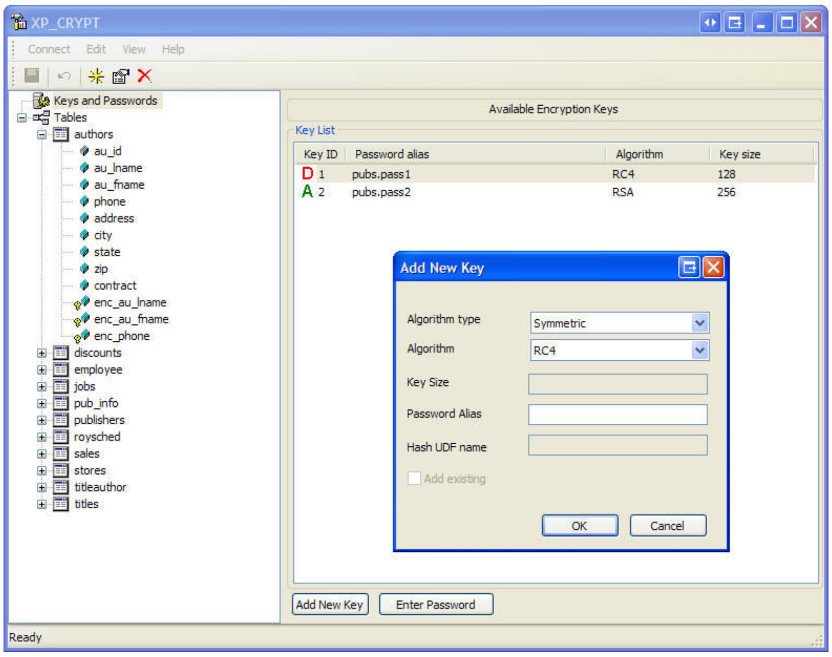
1) Create tables to manage your passwords
2) Create fields that are the encrypted representation of the fields that you choose (you delete the clear-text fields by hand).
3) Create a view that will decrypt the fields that you have encrypted
4) A procedure to activate and keep track of the decryption key for the user session.

The images below illustrate some of the screens that are used to add field level encryption to your database.



Key ID	Password alias	Algorithm	Key size
D 1	pubs.pass1	RC4	128
A 2	pubs.pass2	RSA	256

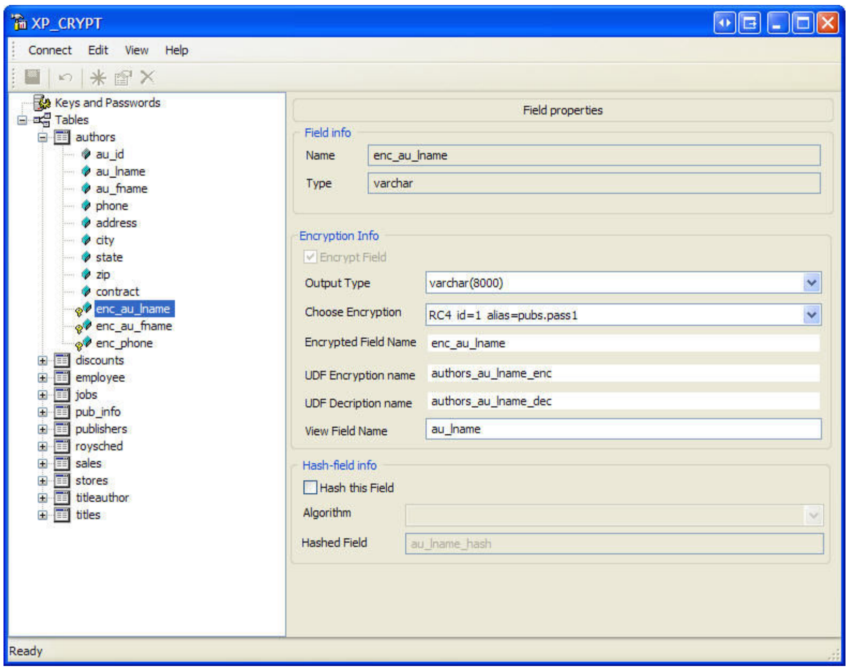
Figure 1 – Here we can add multiple encryption types to the database using the XP_CRYPT GUI



Key ID	Password alias	Algorithm	Key size
D 1	pubs.pass1	RC4	128
A 2	pubs.pass2	RSA	256

Algorithm type	Symmetric
Algorithm	RC4
Key Size	
Password Alias	
Hash UDF name	

Figure 2 – Here we can set attributes of a given encryption key. Different algorithms are supported, including both symmetric and asymmetric encryptions.



Name	enc_au_lname
Type	varchar
Encrypt Field	✓
Output Type	varchar(8000)
Choose Encryption	RC4 id=1 alias=pubs.pass1
Encrypted Field Name	enc_au_lname
UDF Encryption name	authors_au_lname_enc
UDF Decription name	authors_au_lname_dec
View Field Name	au_lname
Hash this Field	
Algorithm	
Hashed Field	au_lname_hash

Figure 3 – Here we can apply the encryption algorithms to field(s) and chose the output type, encryption type, and the field names for the procedures that the XP_CRYPT GUI makes.

PART IV: MAKING THE CASE – Buy Versus Build.

There comes a time when you evaluate a toolkit, when you need to ask the question of buy versus build.

1) Extended stored procedures **need to be bug free**. A bad extended stored procedure that is called often by your database can hang or corrupt your database. The fact that XP_CRYPT is a commercial product that has been used by many users reduces this risk.
2) An advantage of building your own solution is that you get source code. Fortunately, you may purchase the XP_CRYPT version **with source code**.
3) **Time is money**. Building your own is going to require a lot of money, if you wish to have multiple algorithms supported and tested well. Also the nice XP_CRYPT GUI makes it very easy to get going fast.
4) How much is your data worth? Or better yet, what is the worst that can happen if your data looses its integrity, or you cant decrypt it? Problems like that can happen if bugs do occur. Spending several thousand dollars for the peace of mind of a proven solution is a lot better than having to wonder if the bug lies in a stored procedure, or your self-written encryption code. If the **data is very valuable**, ask yourself the costs if things don't go right in your database.
5) This is **a proven solution**. XP_CRYPT is already being used by USA government organizations, financial companies, medical organizations, and universities all over the world.
6) Licensing - XP_CRYPT offers multiple licensing options. The **pricing is very reasonable**, as XP_CRYPT pricing is a fraction of the price of competitors. License options include:

    a. **Single** licenses are for one server
    b. **Site** license can be purchased that will cover all servers in one company
    c. **Redistributable** license is for software manufacturers who resell solution they want to protect, so you can embed their technology in the software deliverables.

When you take into consideration the pros and cons of writing your own solution, you will see very quickly that a reasonably priced solution with source code is the way to go. Fortunately XP_CRYPT is a great solution and is available for a good price*!*