SQLite Encryption Layer Implementation

Objective

The objective of this project is to implement a encryption layer on SQLIte3 in order to protect the DB File.

Technical Solution

The Technical solution is to wrap the functions that do the low-level read/write of bytes into the DB File in another function that will override the common behaviour of the read/write with encrypted buffers. The Encryption / Decryption mode and encription keys will be configured using SQLite3 pragmas.

Usage

From a C# perspective it is very easy to use, just create a normal SQLite3 connection:

```
 _fact = DbProviderFactories.GetFactory("System.Data.SQLi
te");
_cnn = _fact.CreateConnection();
_cnn.ConnectionString =
String.Format("Data Source={0};Pooling=true;FailIfMissing
```

```
=false",
"test.db");
_cnn.Open();
```

After that, you just set the encryption details:

```
var cmd = _cnn.CreateCommand();
cmd.CommandText = "PRAGMA encryption_method=AES";
cmd.ExecuteNonQuery();

cmd = _cnn.CreateCommand();
cmd.CommandText =
"PRAGMA encryption_keys=\"000102030405060708090a0b0c0d0e0
f\"";
cmd.ExecuteNonQuery();
```

And that is all.

###Technical Implementation

Encryption

It was created 2 functions: winWriteEncrypted and winReadEncrypted that replaced the original winWrite() and winRead() on the function map that points to the OS specific implementation for I/O operations:

Originally it was though to use Crypto++ library to implement AES

encryption, but it was too bulky for our use case, I chose then tiny-AES128, which is a very simple and embeddable solution for our use case.

One challenge is that AES needs to be used on multiple of 16 bytes, and it is more efficient to use it on 4096-byte blocks, so we had to make sure that all read/write was made on 4096 blocks.

Encryption Control

It was created 2 PRAGMA functions on SQLite3:

- **ENCRYPTION_METHOD**: this controls the Encryption method, currently it accepts AES or NONE. Example:

PRAGMA encryption_method=AES

- **ENCRYPTION_KEYS**: this controls the 128bit encryption key, it is passed as string:

PRAGMA encryption_keys="000102030405060708090a0b0c0d0e0f"

###Build

The implementation was made with Visual Studio Professional 2012, but targeting .NET Framework 4.6, in order to build the solution, just open the SqliteProjects.sln file and then build the C++ Project and then the .NET Project.

During the build a bin folder will be created with a subfolder for each Platform (Win32/Win64). The main files on the folder are the following:

- **SQLiteEncryptionTestApp.exe** - Our Unit Test application, which source code is included on the solution mentioned above
- **System.Data.SQLite.dll** - The main C++ dll which implements SQLite3, this is the binary that was changed to implement the encryption.
- **EntityFramework.dll** - This is a DLL for entity framework 6, installed via NuGet on the solution mentioned above
- **EntityFramework.SqlServer.dll** - This is a DLL for entity framework 6, installed via NuGet on the solution mentioned above
- **System.Data.SQLite.EF6.dll** - This is a DLL that provides support for EF with SQLite, it is available on the Solution Build Tree (No Change was made)
- **System.Data.SQLite.Linq.dll** - This is a DLL that provides support for EF with SQLite, it is available on the Solution Build Tree (No Change was made)

All the code and binaries needed were downloaded from https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki] (https://system.data.sqlite.org/index.html/doc/trunk/www/downloads.wiki) it is the sqlite-netFx-full-source-1.0.102.0.zip package, this package contains several projects, from which only the C++ DLL: System.Data.SQLite is of importance.

Besides System.Data.SQLite Source code downloaded from the website

above, we also use NUnit and XUnit++ for unit tests, and they are available on the 3rdParty folder on the solution.

Unit Tests

It was developed a small console application to do the unit testing of the implemented solution. This application contains 2 main classes:

- **SQLite3UnitTests**: This is a collection of TestCases that were migrated from System.Data.SQlite original source code.
- **SQLite3EF6UnitTests**: This is a simple program that implements Entity Framework 6 to create a database of school students, it was inspired by this example: http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/creating-an-entity-framework-data-model-for-an-asp-net-mvc-application

Both classes above have a execute method that takes a boolean parameter in order to determine if encryption should be used or not for the tests. The SQLite3UnitTests can take up to 20 minutes, so there is also a argument on that class to determine if we should only run the minimum set of tests.

Issues

The main issue faced on testing with Entity Framework is that System.Data.SQLite.EF6.dll Provider doesn’t implement automatic migrations from the Code to the Database, so **WE NEED TO CREATE**

THE SQLITE3 DB AND CREATE THE SCHEMA TO BE USED, PRIOR TO LOADING ENTITY FRAMEWORK. Please see this SO Article: http://stackoverflow.com/questions/35304899/entity-framework-6-not-creating-tables-in-sqlite-database

In the School example that I downloaded from Microsoft, I made a work around creating the method: SchoolConnectionFactory.CreateSqLiteDatabase() on the connection factory, so just before Entity Framework gets the first connection, it creates the DB, the schema and encrypts it.