How to compile SQLCipher for Windows Desktop.

This is my tutorial on how to compile SQLCipher for Microsoft Windows use in Visual Studio 2012 or 2013.

SQLCipher is a cryptographically enhanced version of the popular public domain database project SQLite. The developers at Zetetic LLC have been talented enough to produce a cryptographically strong cross platform database and have been generous enough to provide access to a community edition, an awesome BSD style licensing, and a public Github repository. These guys rock and my thanks go out to them.

This tutorial is being written in March of 2014 and I am using the most recent version of all of the files and utilities listed below. I give no garuantee that this procedure will work for you. It has successfully worked for me on several separate runs now, and I do feel that I have worked enough of the kinks out of the procedure to be able to credibly share a tutorial.

I will have to assume that since you are trying to hand compile a building block for secure encrypted applications that you are not a complete novice with computers and programming. This tutorial is for use with Windows Desktop only and at this time does not cover compiling for Windows Phone, RT or Windows Store apps.

Prerequisites.

- Install Visual Studio Express 2012 or 2013 for Desktop and update them (should work with older versions as well but I have not verified that)
- Install Activestate Perl (a reboot is sometimes required or applying some manual path environment settings)
- Get the most recent stable source for openssl (1.0.1f at this time) and extract to C:\opensslsrc32
- Get the source zip for SQLCipher from github and extract to C:\sqlcipher
  Watch out for nested directories; winrar in particular likes to extract the files to c:\sqlcipher\sqlcipher-master\(base dir here). If that happens just move the inner files and directories up so the base directory is C:\sqlcipher. The same principle also goes for openssl; make its base directory C:\opensslsrc32.
- Install Mingw making sure to install all of the developer and autotools binaries.
  - Installing Mingw is a multi step process. First you download and install the package manager/installer and then use that to install the compiler and msys system.
  - After installing the installer run it and under the basic pane install the msys base, developer-tools, mingw32-base, and the gcc-g++.
  - Apply changes (go make a sandwich, this takes a bit.)
  - Set the fstab as it does not ship with a custom one. Directions to do this are in the getting started document on the mingw site. It is most likely subject to change upstream so I wont explain it here.

At this point you should have your pre reqs out of the way and we are ready to start the build process.

OpenSSL

We start by compiling openssl for 32 bits with both the libraries and dll.

- Open the 32 bit visual studio command prompt for your version. (Not the regular cmd.exe)
- In the command window change directories to your openssl 32 bit source directory C:\opensslsrc32
- Once you are in the correct directory run the following commands in order:
  - `perl Configure VC-WIN32 --prefix=C:\opensslbuild32`
  - `ms\do_ms`
  - `nmake -f ms\nt.mak`
  - `nmake -f ms\nt.mak install`
  - `nmake -f ms\ntdll.mak`
  - `nmake -f ms\ntdll.mak install`
- Now you can close your visual studio command prompt window.

In your openssl build directory (c:\opensslbuild32) you will (should) have: lib\libeay32.lib and bin\libeay32.dll among some other files.

Building SQLCipher

In the last step should have left you with: lib\libeay32.lib and bin\libeay32.dll among some other files in your C:\opensslbuild32 directory.

- Copy, don't move, those two files to the base source directory for sqlcipher. If you are using the same directory names a I am it will be C:\sqlcipher.
- In explorer (local files, not the internet one) navigate to where you installed Mingw. The default (in March 2014) is C:\mingw\msys\1.0\ and run msys.bat.
- That should open a msys shell. Due to the cross breed nature of this beast it is a bit different from either windows or *nix and takes some getting used to.
- Navigate to your sqlcipher directory. If you are using my example directory names you should be able to enter
  - `cd /c/sqlcipher`
- Notice the leading forward slash before the drive letter and the fact that we have switched to the forward slash not the backslash like windows normally expects.
- Now you have enter the most cryptic and error prone portion of this whole adventure. Manually enter into the msys command line all on one continuous line (unfortunately you cannot paste):
  - `./configure --with-crypto-lib=none --disable-tcl CFLAGS="-DSQLITE_HAS_CODEC -DSQLCIPHER_CRYPTO_OPENSSL -I/c/opensslbuild32/include /c/sqlcipher/libeay32.dll -L/c/sqlcipher/ -static-libgcc" LDFLAGS="-leay32"`
- Now that SQLCipher is configured to build you can start the build process, enter the following commands, in order, in the same msys command window:
  - `make clean`
  - `make sqlite3.c`
  - `make`
  - `make dll`
- You should now have a fully functional sqlcipher executable dll file and an amalgamated sqlite3.c file.

Test Drive

If things have gone well, you should now have a fully functional sqlcipher executable and dll file. Open a normal command (cmd.exe) prompt, navigate to the sqlcipher build directory and take SQLCipher for a test drive:

- sqlcipher test.db
- sqlcipher>PRAGMA key = 'password';
- sqlcipher>create table testtable (id int, name varchar(20));
- sqlcipher>insert into testtable (id,name) values (0,'alice'), (1,'bob'), (2,'charlie');
- sqlcipher>select * from testtable
    - 0:alice
    - 1:bob
    - 2:charlie
- sqlcipher>.exit

Now that we have a database with some values in it let's exit and try to access it without a password.

- sqlcipher test.db
- sqlcipher>select * from testtable
- Error: file is encrypted or not a database
- sqlcipher>.exit

And again with the password.

- sqlcipher test.db
- sqlcipher>PRAGMA key = 'password';
- sqlcipher>select * from testtable
    - 0:alice
    - 1:bob
    - 2:charlie
- sqlcipher>.exit

If it looks like it is working properly, you can now open the file test.db in a hex editor and verify that it has no discernable structure. If you want some contrast verification, repeat the above experiment on a new database file without the PRAGMA key statement. It will make an unencrypted database that you will be able to easily see both clear structure and raw data.

Using it in Visual Studio

The next step is to get all of this working inside your Visual Studio projects.

First you need to open the 32 bit Visual Studio command prompt again and navigate to your sqlcipher directory. Verify that you have both files sqlite3.dll and sqlite3.def in the directory and enter:

- lib /def:sqlite3.def

This will create the library references to let visual studio link properly

Create a win32 console application(turn off precompiled headers) named something like SqlCipherTest. Pay close attention to the directory that it will be located in and save the project.

Outside of visual studio copy the five files : sqlite3.exp, sqlite3.lib, sqlite3.def, sqlite3.dll, and sqlite3.h to the source directory of your visual studio project.

Get back into Visual Studio and in the solution explorer add an existing header file and choos sqlite3.h

Open the project properties->configuration properties->linker->input->Additional dependencies and add sqlite3.lib

In your main source file enter something like the following c program:

```
// SQLCipherTest.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "sqlite3.h"
#include  //needed for the getch() to keep the console open


static int callback(void *NotUsed, int argc, char **argv, char **azColName) {
        //feed this callback function to handle theresultset returned by the select statement
        int i;
        for (i = 0; i < argc; i++) { //loop over results
                printf("%s = %s\n", azColName[i], argv[i] ? argv[i] : "NULL"); //gota love how human radable c is
        }
        printf("\n");
        return 0;
} //end callback

int _tmain(int argc, _TCHAR* argv[])
{
        sqlite3 *db;

        if (sqlite3_open("name_and_path_for_db", &db) == SQLITE_OK) {
                printf("DB file is open\n");
                if (sqlite3_exec(db, (const char*)"PRAGMA key ='password'", NULL, NULL, NULL) == SQLITE_OK){
                        printf("Accepted Key\n");
                };
                if (sqlite3_exec(db, (const char*)"CREATE TABLE testtable (id int, name varchar(20));", NULL, NULL, NULL) == SQLITE_OK) {
                        printf("Created Table\n");
                };
                if (sqlite3_exec(db, (const char*)"INSERT INTO testtable (id,name) values (0,'alice'), (1,'bob'), (2,'charlie');", NULL, NULL, NULL) =
                        printf("Gave it some data\n");
                };
                if (sqlite3_exec(db, (const char*)"SELECT * FROM testtable;", callback, NULL, NULL) == SQLITE_OK) {
                        printf("Sent Select\n");
                };
        }
        sqlite3_close(db);  //close it up properly

        getch(); //just a hack to keep the console open during debugging
```

```
        return 0;

}// end SQLCipherTest.cpp
```

Run it in the debugger and you should see the output of the select statemnt.

To test it more thoroughly comment out the two if statement blocks with the CREATE TABLE and the INSERT INTO and run it again and you should see the output of the select statement again.

To test further now change the key to something different and you should NOT see the output of the SELECT.

Notes

- This tutorial does not cover Windows RT or any variation of the new Metro UI found in Windows 8 and 8.1
- As of now this tutorial does not include info on how you use the amalgamated file to reduce dll dependency.
- There may be a few overkill items in the steps that I threw in for completeness and to have a better toolkit for future use in more projects.
- If you get the error telling you that your compiler cannot produce executables you most likely have a typo or have modified your CFLAGS environment variable in some way.
- There may be better ways to this but this method works for me, and I have repeated it several times from scratch on separate development boxes for verification. Also I am not a guru in recent versions of Visual Studio, I usually work in more *nixy build environments.
- The sample source code is just that, a bare essentials sample. It is not, by any means, secure production ready code. I would strongly advise against using it for any purpose other than testing the compilation capability.
- Directory names were chosen for ease of typing and are not mandatory. Modify them to fit your build environment.
- It is generally much easier to do this from the command line rather than fighting with the deep settings of an IDE.

Sources and Attribution

This tutorial would not have been possible without the many existing help threads on the net. This is really just an amalgamation of all of the solutions provided separately in the forums and released back to the community as a set of coherent steps.

References:

- http://mobileorchard.com/tutorial-iphone-sqlite-encryption-with-sqlcipher/
- https://groups.google.com/forum/#!msg/sqlcipher/VcYpa1a_RTM/Q3Q9YR7TzS8J
- https://groups.google.com/forum/#!topic/sqlcipher/BQAu43MQHqA
- https://groups.google.com/forum/#!topic/sqlcipher/ghFVyZPRFOw
- https://groups.google.com/forum/#!topic/sqlcipher/FHkvQvWQaWE

- Home
- About
- Contact
- Tutorials