Delphi-Windows Digital Signature Library Interface

Joseph Snyder
SW Process Engineer
snyderj@osehra.org

Purpose

The purpose of these three files is to provide the interface between Delphi executables and the built-in Windows security functions. The push to supplement the traditional sign-on with the usage of a Personal Identity Verification (PIV) card is rapidly gaining momentum. One major focus of using these PIV cards is to further secure the prescribing of controlled substances.

There are already existing versions of these interface files which are available to the public. One version was released in the version 29 FOIA copy of the Computerized Patient Record System (CPRS) for the VistA EHR. However, it was determined that the license for those files, which was the Mozilla Public License, was incompatible with the license that the OSEHRA VistA uses for its repository. These files are released with the Apache 2.0 license to match the license for the OSEHRA VistA repository. The files contained here are written by blanking the three necessary files and filling in the function calls based upon what was missing during compilation time and testing.

Code Walkthrough

The interface consists of three separate files:

- XlfMime.pas
- Wcrypt2.pas
- WinSCard.pas

Each file contains different functions corresponding to a different focus around acquiring and verifying the security certificate of the card holder.

WinSCard.pas

WinSCard.pas contains the constant values and function signatures necessary to connect to a Smart Card reader. Once a connection has been established to a local Smart Card reader, it is able to access the information about the certificate contained on a supplied card. These functions are all found in the Winscard.dll library.

Wcrypt2.pas

Once the user's certificate has been accessed from the PIV or Smart Card, the next step is to verify that the certificate is valid and corresponds to a trusted user. The functions found in Wcrypt2.pas are to be used to open a certificate store and perform various checks to verify that the user has the

correct identity and permissions. The functions in this file are called from one of two Windows DLLs: crypt32.dll or Advapi32.dll.

XlfMime.pas

The functions found in the XlfMime.pas file are used to encode information for transmission between two sites. This library encodes and decodes the information using a Base64/MIME encryption. The Delphi environment has a built-in library to accomplish this, named EncdDecd, to encode both strings and arbitrary streams of data

External Interfaces/Dependencies

The files in this submission were originally released with the CPRS code that corresponded to version 29 of the software. CPRSv29 corresponds to the OR*30*306 patch for the Order Entry Results Reporting package. If these files are placed into the source tree of an earlier version of CPRS, they should not have an effect on the compilation. **To fully utilize the files, it is recommended that they be used to compile CPRSv29 or later.**

The files were rewritten by building the CPRS v29 code in the Delphi 2007 compiler. It was also tested with a different program using the Delphi XE3 compiler. Both of these compilation environments were run on Windows 7 SP 1 machines. These two compilers both showed no issue in compiling the submitted files. **It is recommended that a Delphi compiler which is Delphi 2007 or more recent is used to compile the files**.

The testing code was verified using the NIST Test Personal Identity Verification (PIV) card set. More information about the NIST testing cards can be found here:

http://csrc.nist.gov/groups/SNS/piv/testcards.html

As mentioned above, the development and testing was done on two Windows 7 Service Pack 1 computers. Of the three Windows DLLs that are used, none are specific to Windows 7. Therefore, there should be no dependence on the version of Windows environment necessary to build with these files.

The versions used in development and testing are:

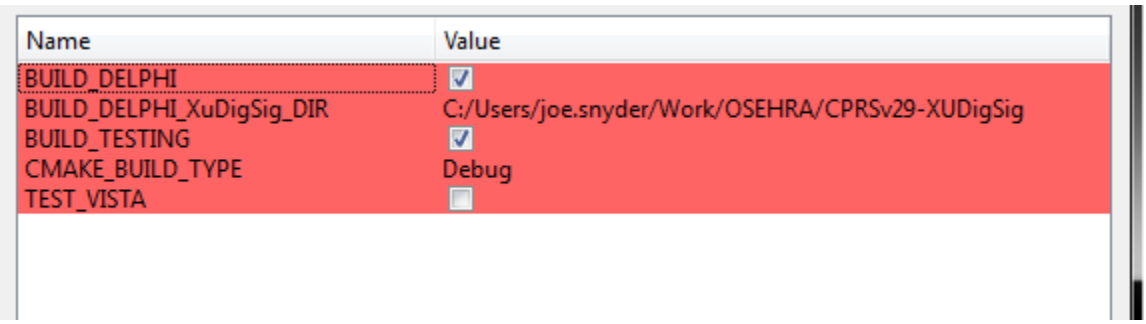
- crypt32.dll
  - 6.1.7601.18526
- Advapi32.dll
  - 6.1.7600.16385
- Winscard.dll
  - 6.1.7600.16385

The values were found with by utilizing the sigcheck tool and entering the path to the `C:\Windows\System32` version of each file.

Installation

These files were created to replace files from the FOIA release of the CPRS executable, so there is already a designed path to include these files. Please be aware that these variables and options will only be shown on a system with the proper Delphi environment.

To use these files with the OSEHRA VistA repository to build the CPRS executable, one would only need to set the "BUILD_DELPHI_XuDigSig_DIR" variable in the CMake GUI to the path to a directory with these files. An example is found in the image below:



This information would then be propagated to the Delphi project file upon running the CMake configure and generate steps.

These files are obviously not limited to just the CPRS application. To install the files into any other Delphi project, one would follow the standard method of including the files via the 'uses' directive. The files should be placed into a directory and added to a 'uses' subheading in the Delphi project file (*.dpr). Again, a small code snippet example follows:

```
Uses
  ShareMem,
  Forms,
  WinHelpViewer,
  ORSystem,
  Wcrypt2 in 'C:/Users/joe.snyder/Work/OSEHRA/CPRSv29-
XUDigSig\Wcrypt2.pas',
  WinSCard in 'C:/Users/joe.snyder/Work/OSEHRA/CPRSv29-
XUDigSig\WinSCard.pas',
  XlfMime in 'C:/Users/joe.snyder/Work/OSEHRA/CPRSv29-
XUDigSig\XlfMime.pas',
<snip>
```

How to Test

There are DUnit tests written for the three files which included in this download:

- UTXlfMime.pas
- UTWcrypt2.pas
- UTWinSCard.pas

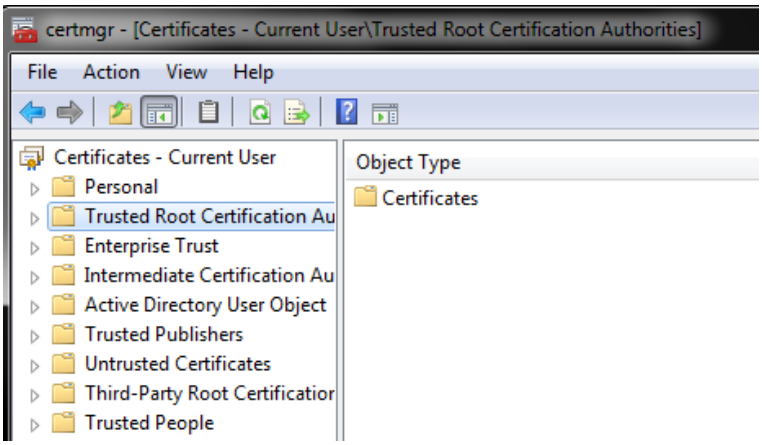
Note about testing:

One of the Wcrypt2 tests has multiple pass conditions based upon the return of the function being tested. The TestCertVerifyRevocation function has two passing conditions:

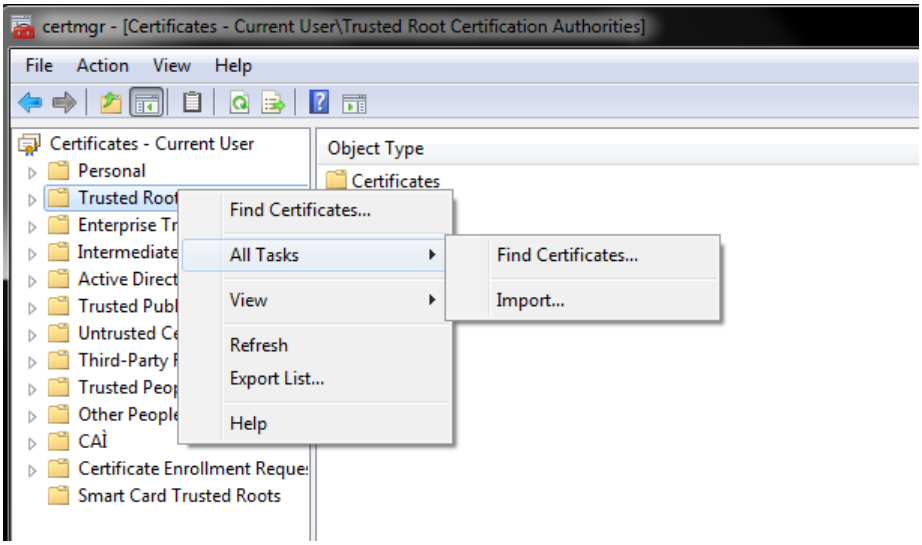
- the CertVerifyRevocation function returns **true** and has **no error** expressed
- The CertVerifyRevocation function returns **false** and the dwError value is set to '2148081683'.
    - This corresponds to a CRYPT_E_REVOCATION_OFFLINE error as expected.

If the dwError value is any other value, the test will fail. Due to the absence of a revocation server in the testing environment, there is no dashboard result with the test returning true with no errors.

It was found that when the self-signed certificate from the NIST was imported into the "Trusted Root Certificate Authority" category, the VerifyRevocation error was no longer being expressed. The self-signed certificate can be found at the NIST information website and can be imported into the "Trusted Root Certificate Authority" via the Certificate Manager utility. This is found by searching and running for the "certmgr.msc" program from the Windows' start menu.



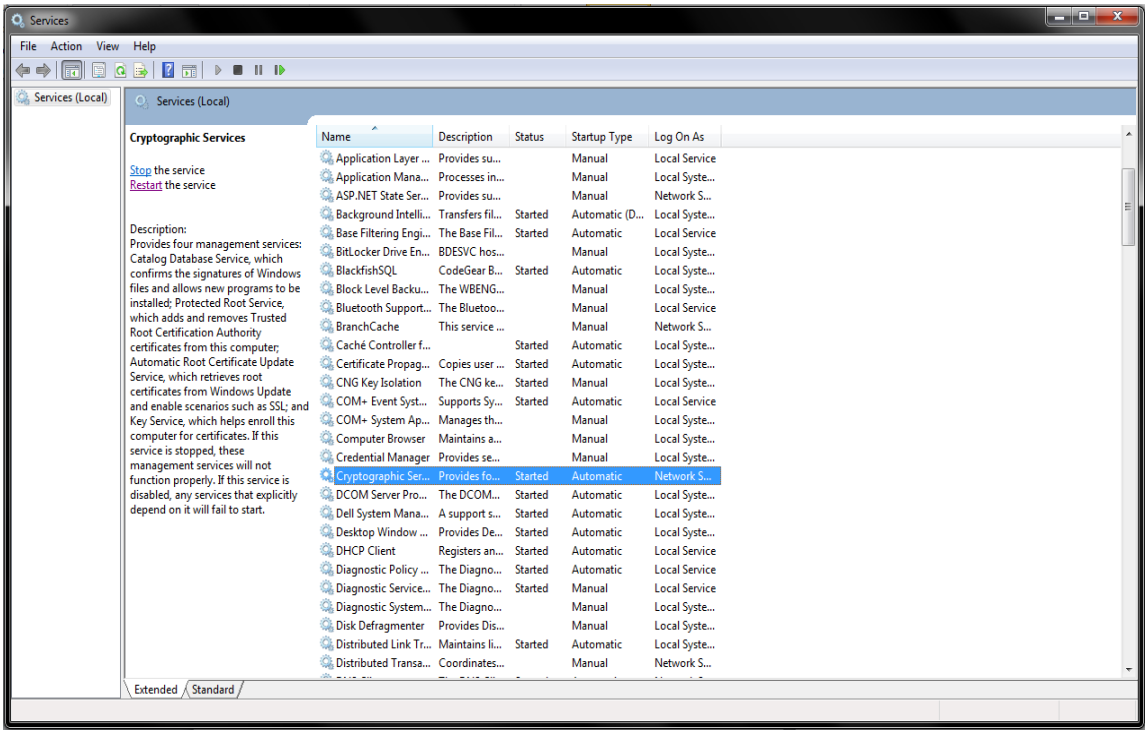
Once there, right click on the "Trusted Root Certification Authority" and select the "import" option under the "All Tasks" menu.



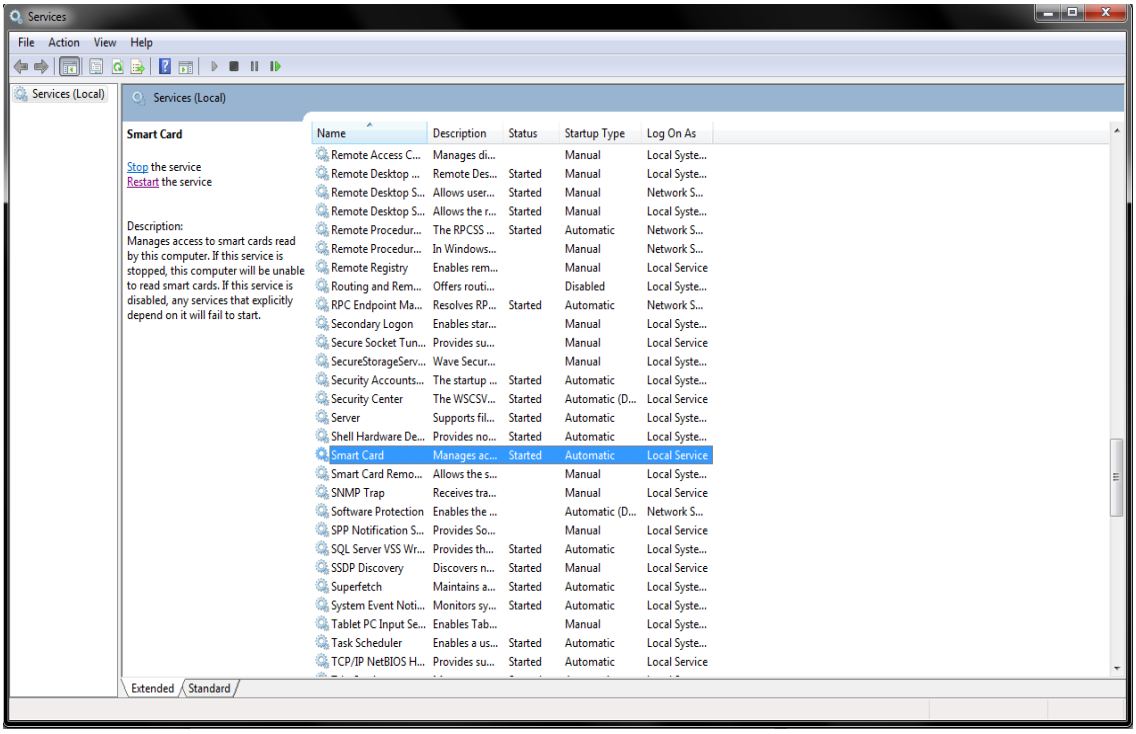
Setup Prior to Testing

The usage of the smart card requires some set up on the Windows environment. There are two services that are required in order to access the smart card and the certificates contained within. To ensure the services are started, Search for and run the "services.msc" functionality from the Windows start menu.

Once the Services window appears, ensure that the "Smart Card" and "Cryptographic Service" entries both have the "Started' status.



Cryptographic Service



Smart Card Service

Integration of Testing

OSEHRA VistA Repository

These files, much like the source files, have a place in the OSEHRA VistA repository. To test the files with the CPRS build process, place the files into the `Packages\Order Entry Results Reporting\CPRS\Testing\Tests` directory. Then, you would add an entry for each file under the 'uses' header to the CPRSTesting.dpr.in:

```
uses
  UTXlfMime in '@SOURCE@\Tests\UTXlfMime.pas' {UTXuDsigS},
  UTWcrypt2 in '@SOURCE@\Tests\UTWcrypt2.pas' {UTXuDsigS},
  UTWinSCard in '@SOURCE@\Tests\UTWinSCard.pas'{UTXuDsigS};
```

After running a Configure, Generate, and make, you can execute all of the tests at once by running the executable or by using the CTest program.

Running '`ctest -N`' will show the available tests:

```
C:\Users\joe.snyder\Work\OSEHRA\VistA-delphi-dev>ctest -N
Test project C:/Users/joe.snyder/Work/OSEHRA/VistA-delphi-dev
  Test #1: DUnitUTSignonCnf
  Test #2: DUnitUTWcrypt2
  Test #3: DUnitUTWinSCard
  Test #4: DUnitUTXlfMime

Total Tests: 4
```

Then, the tests can be run together by executing 'ctest' or individually with a command like "ctest –R Wcrypt2". The tests run will be determined by matching the test names against the string after the "-R". Upon running the tests, all of the tests should pass. The output printed to the screen should look like this:

```
C:\Users\joe.snyder\Work\OSEHRA\VistA-delphi-dev>ctest
Test project C:/Users/joe.snyder/Work/OSEHRA/VistA-delphi-dev
    Start 1: DUnitUTSignonCnf
1/4 Test #1: DUnitUTSignonCnf ..................   Passed    0.05 sec
    Start 2: DUnitUTWcrypt2
2/4 Test #2: DUnitUTWcrypt2 ....................   Passed   25.21 sec
    Start 3: DUnitUTWinSCard
3/4 Test #3: DUnitUTWinSCard ...................   Passed    0.20 sec
    Start 4: DUnitUTXlfMime
4/4 Test #4: DUnitUTXlfMime ....................   Passed    0.05 sec

100% tests passed, 0 tests failed out of 4

Total Test time (real) =  25.69 sec
```

Be aware that running the tests will ask for a smart card to be inserted and accessed.

An earlier attempt of the integration of the testing code can be found in the OSEHRA Gerrit instance: http://review.code.osehra.org/#/c/633/ .

External project

For testing with DUnit without using the OSEHRA VistA repository, the files can be included into any existing DUnit testing setup. If one does not exist, the OSEHRA version is a good example to see how to generate a separate testing executable to run. The Delphi project directory which generates the OSEHRA testing can be found in the `Packages/Order Entry Results Reporting/CPRS/Testing` directory.

Coverage Calculations

The testing of the code reports that 95% of the executable code is covered by the three test files. The coverage was calculated using the Delphi Code Coverage tool with the OSEHRA VistA testing setup. The coverage is calculated over the testing executable that is generated instead of the CPRS executable.

The command used to generate the necessary HTML files for CTest to parse is:

```
VistA-delphi-dev$ ~/Downloads/CodeCoverage_win32_1.0_RC9/CodeCoverage.exe -e
~/Work/OSEHRA/VistA-delphi-dev/CPRS/Testing/Bin/Debug/CPRSTesting.exe -html -u XlfMime
WinSCard Wcrypt2 -sp ~/Work/OSEHRA/CPRSv29-XUDigSig/
```

An explanation of the flags for the function is below:

- -e
  - Path to the executable to run
- -html
  - Output the results of the coverage as HTML files
- -u

    - Names of the units to calculate coverage for
- -sp
    - Directory where the source for the targeted units can be found

The generated HTML will report 100% coverage due to the fact that it can calculate which lines where hit, but not the total of executable lines. To demonstrate the correct calculation, run the Coverage capability of a CMake whose version is later than **3.0.2** in the same directory that contains the output HTML of the previous step. An example run follows:

```
VistA-delphi-dev$ ~/Work/cmake-build/bin/Debug/ctest.exe -M Experimental -T Coverage
   Site: TUCHANKA
   Build name: Win32-
Performing coverage

   Accumulating results (each . represents one file):
    ...
        Covered LOC:         84
        Not covered LOC:     4
        Total LOC:           88
        Percentage Coverage: 95.45%
```

Future Work

The future of these three files does have a clear path for additional development. The files do not contain the signatures for every possible Windows function. As a new functionality is developed for these Delphi libraries, calls to other Windows functions will become necessary. These functions and any necessary data structures will be included as needed.