TPM 2.0 TSS System API and Test Application

Will Arthur
Intel Corporation
7/28/15

About

This code repository contains the TSS stack as implemented so far: the System API (SAPI), two TPM Command Transmission Interface (TCTI) layers, the TPM Access Broker/Resource Manager (TAB/RM), and test applications for exercising the stack.

The TPM 2.0 TSS SAPI library code implements the system layer API level of the TSS 2.0 specification. These functions can be used to access all TPM 2.0 functions as described in Part 3 of the TPM 2.0 specification. The usefulness of this code extends to all users of the TPM, even those not planning to use the upper layers of the TSS. The SAPI code is highly modularized and built as a library of functions, tpm.lib (Windows) or libtpm.a (Linux). Applications should be linked against this library, and this means that only the code needed for a given application will actually be included in the final binary image. This makes this code highly suitable to everything from highly constrained embedded applications to large multiprocessor servers.

A TCTI layer is used to communicate between the SAPI and the TAB/RM.

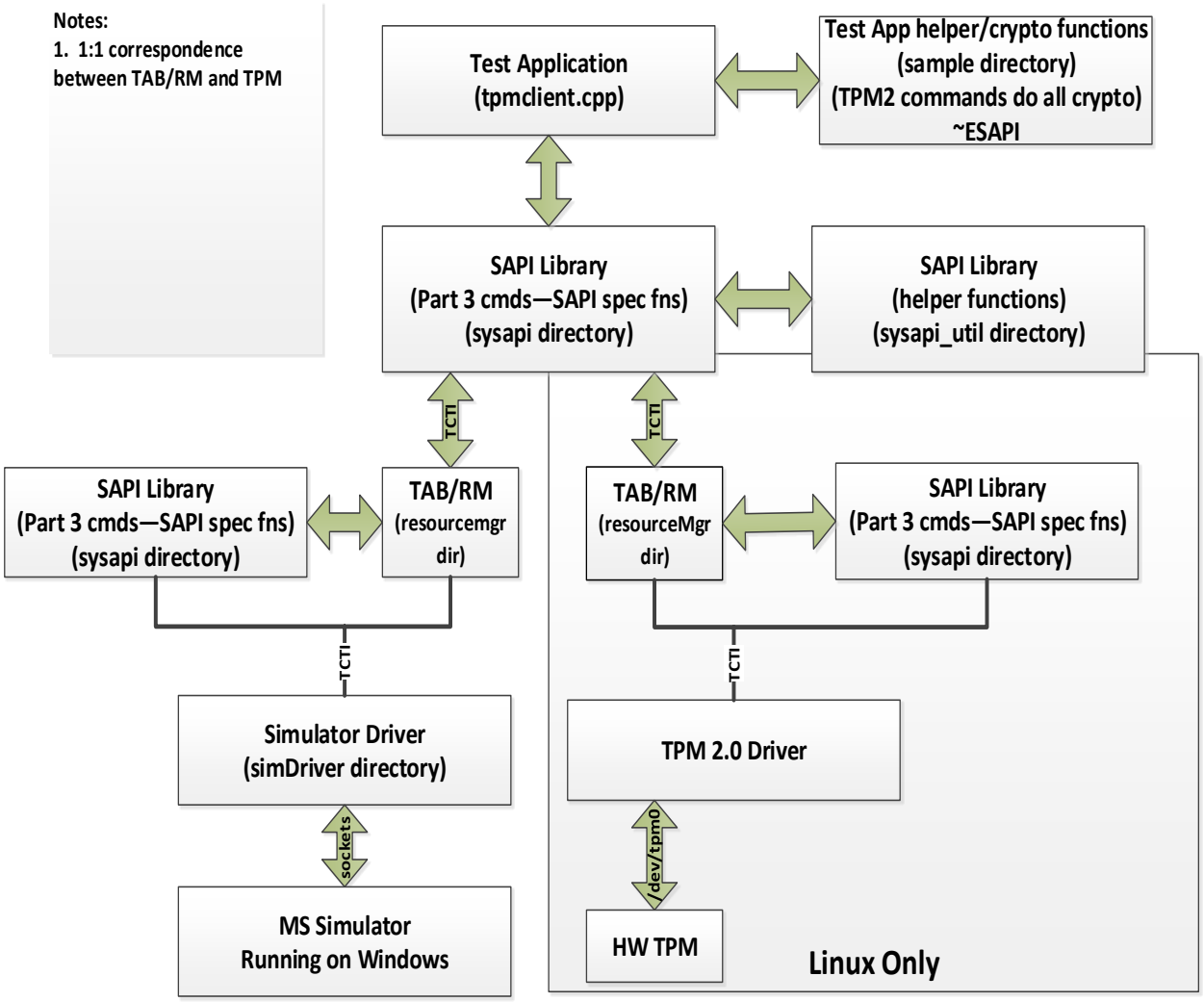
Between the system API library code and the TPM, there is a TAB/RM (TPM Access Broker/Resource Manager) daemon that handles all multi-process coordination and manages the TPM's internal resources transparently to applications.

The TAB/RM daemon communicates through another TCTI layer to either the TPM 2.0 simulator or a real TPM's device driver (Linux only).

Additionally, the test application, tpmclient, tests many of the commands against the TPM 2.0 simulator. The tpmclient application can be altered and used as a sandbox to test and develop any TPM 2.0 command sequences, and provides an excellent development and learning vehicle.

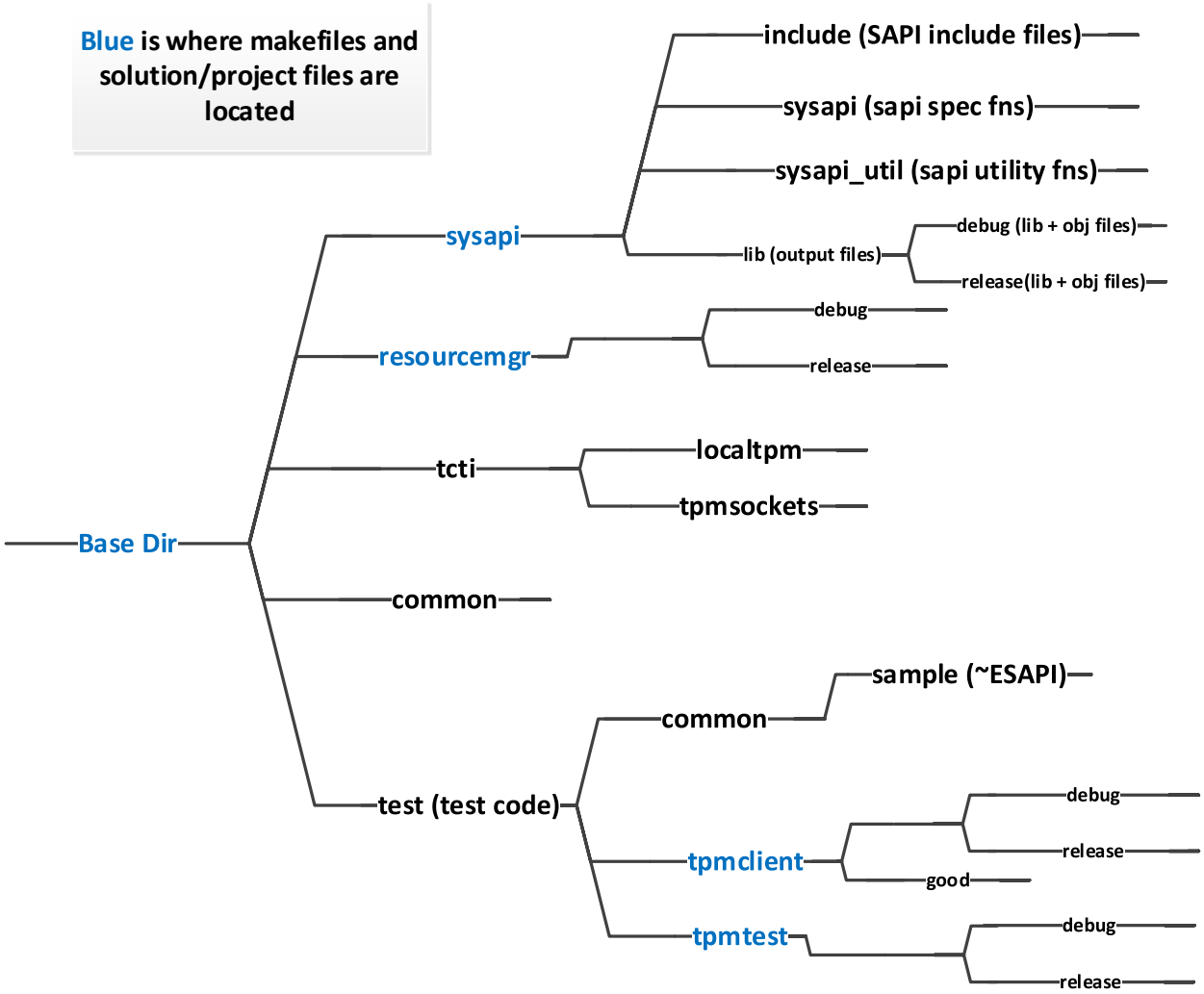
Architectural Block Diagram

This shows the blocks and interfaces in the SAPI library, TAB/RM, and test code as described above.



Code Organization

Below is a diagram of the directory structure for the code:



Some explanations:

- Sysapi: this is where the SAPI library code lives. The output library binaries are in lib\debug or lib\release.
- Resourcemgr: this is where the resource manager code is located. The resource manager is an executable program that is located in the debug or release directory. Since the resource manager sends TPM 2.0 commands itself, it is linked to the SAPI library.
- TCTI: there are two TCTI variants:
    - local TPM: the local TPM interface is used by the resource manager when it is communicating with a real TPM. This option is only available when running under Linux.
    - sockets. The sockets interface is used in two places:
        - Between the test application and the TAB/RM

- If the TPM 2.0 simulator is being used, the sockets interface is used for the lower TCTI connection to the “tpm”, in this case the simulator.

The tpm sockets code is somewhat modified depending on whether it is being used for the top or bottom TCTI layer.

If an application wanted to run without the TAB/RM, either the lower version of the sockets interface or the local TPM interface could be hooked directly to the application via TCTI.

- Common: this is code that is common between components (currently the resource manager and test applications).
- Test: this is where test applications are located. Currently there are two:
  - Tpmclient: this is designed to test the SAPI library and the TAB/RM and as such will only run against the simulator.
  - The common directory contains code that is shared between multiple test applications. Right now there is only one test application but there will be more in the future. In particular, a future app, tpmtest, will be capable of running against a real TPM. The sample subdirectory contains code that is very much like an ESAPI (Enhanced System API) layer.

Instructions

To use the TPM 2.0 library, simulator client, and test app:

1. Build and test TPM 2.0 simulator:
    2. Get the TPM 2.0 simulator, 1.22 version, from the TCG web site, trustedcomputinggroup.org, and install it.
    3. Unzip the files into a root directory.
    4. Build and test the simulator:
        a. Follow directions in the simulator release notes to build the simulator.
        b. In Visual Studio 2012, open TPMcmd\simulator.sln solution file and build it.
        c. Copy libeay32.dll into TPMcmd\debug directory.
        d. Run TPMcmd\debug\simulator.exe
        e. To test it the following python script can be used.

NOTE: you may have to cut and paste these commands into Python interpreter one by one. I'm not a python expert, and I couldn't get the script to just run:

```
import os
import sys
import socket
from socket import socket, AF_INET, SOCK_STREAM

platformSock = socket(AF_INET, SOCK_STREAM)
platformSock.connect(('localhost', 2322))
platformSock.send('\0\0\0\1')

tpmSock = socket(AF_INET, SOCK_STREAM)
tpmSock.connect(('localhost', 2321))
# Send TPM_SEND_COMMAND
tpmSock.send('\x00\x00\x00\x08')
# Send locality
tpmSock.send('\x03')
# Send # of bytes
tpmSock.send('\x00\x00\x00\x0c')
s# Send tag
tpmSock.send('\x80\x01')
# Send command size
tpmSock.send('\x00\x00\x00\x0c')
# Send command code:  TPMStartup
tpmSock.send('\x00\x00\x01\x44')
# Send TPM SU
tpmSock.send('\x00\x00')
# Receive 4  bytes of 0's
reply=tpmSock.recv(18)
```

5. Build system API and test code:
    a. Windows:
        i. Create an environment variable, TSSTOOLS_PATH that points to your Visual Studio C nmake.exe, cl.exe, link.exe, and lib.exe utilities.
        ii. In Visual Studio 2010, open the tss.sln solution file in the base directory.
        iii. Build it. This will build the System API library (tpm.lib), the TAB/RM daemon (resourcemgr.exe) and the test application (tpmclient.exe). tpm.lib is linked into both the TAB/RM daemon and the test application. The test\tpmclient\debug or test\tpmclient\release directories are where the resourcemgr and tpmclient executable files are located after building, depending on the type of build.
        iv. If you're going to run against the simulator, start the TPM 2.0 simulator (this assumes you have a working version of this installed).
        v. Start the TAB/RM daemon. If you're running against the simulator, select the "sim" command line option and use the command line parameters for selecting the TPM host/port. Also, either use the default port or select a different one (if there's a conflict) for the port that will be used by applications to communicate with the daemon. The easiest way is to run everything on the same machine (currently this is only possible if running against the simulator since there is no implementation of Linux simulator), in which case no command line parameters are needed. For help with the command line parameters, type "resourcemgr -?". For comparing to known good output, redirect the resourcemgr output to a file.
        vi. Run the tpmclient or tpmtest(not currently available) application.
            1. Tpmclient will test various TPM 2.0 library functions and the resource manager. Redirect the output to a file: tpmclient -host –dbg 3 out_file 2>&1. Type "tpmclient -?" to see help text that describes the debug message levels and other command line options. Compare the output file to the good sample output files, test\tpmclient.out.good and test\resourcemgr.out.good. The same number of commands should have run and pass/fail status should be the same for all the tests. There will be miscompares due to randomness in some forms of output data from the TPM, but these are not errors.
            2. Tpmtest, a future enhancement, will self-configure, depending on the capabilities of the installed TPM, and run some tests against that TPM.
    b. Linux:
        i. Go to the base directory and run make: > make
        ii. Follow steps iv through vi as detailed above for Windows with slight modifications for running under Linux.

Reporting Bugs

Intel requests that any bugs found in this code and bug fixes be reported to Intel so that all users of this code can benefit. Please report any issues to: Will Arthur, will.c.arthur@intel.com.