CHAPTER 6

Execution Environment

Future chapters in this book present code examples to illustrate concepts. In order for you to be able to build and run these code examples, this chapter describes how to set up an execution environment and build TPM 2.0 sample applications. An execution environment consists of two things: a TPM and a software stack to communicate with the TPM. You can use a hardware or software TPM to run the code examples. In this chapter you learn how to set up the Microsoft TPM 2.0 simulator, a software implementation of TPM 2.0. For software stacks, currently there are two software API environments for TPM 2.0 programming: Microsoft's TSS.net and TSS 2.0. This chapter demonstrates how to set up both of these environments.

Setting Up the TPM

All TPM 2.0 programming environments require a TPM to run code against. For developers, the TPM that is easiest to use is the Microsoft TPM 2.0 simulator. Of course, you can also use other TPM 2.0 devices, hardware, and firmware, as they become available, to run the code examples. Because communication with a hardware or firmware TPM is platform specific, you must use the correct driver; setting up this driver isn't described here.

Microsoft Simulator

Provided by Microsoft, the Microsoft simulator is a full TPM 2.0 device implemented completely in software. Application code can communicate with the simulator via a sockets interface. This means the simulator can be run on the same system as the application or on a remote system connected via a network.

Two versions of the simulator are available. A binary-only version can be downloaded from: http://research.microsoft.com/en-US/downloads/35116857-e544-4003-8e7b-584182dc6833/default.aspx. For TCG members, the second, and better, option is to obtain the TPM 2.0 simulator source code and build it. The advantage of doing this is that it allows an application developer to step through the simulator itself, which is often quite useful when debugging errors. In either case, the simulator can only run under Windows.

You will first learn how to build the simulator from source code and set it up. Then, for non-TCG members, you will learn how to get the TSS.net or simulator binary and use the simulator executable. Finally, the chapter presents a simple Python program that you can use to test that the simulator is working.

Building the Simulator from Source Code

This option is available only to TCG members, because it requires downloading source code from TCG's web site. Go to the `www.trustedcomputinggroup.org` web site, click Member Login at top right, click the Groups pull-down at left, select TPMWG under My Groups, and then click Documents. At this point you should be at this web site: `https://members.trustedcomputinggroup.org/apps/org/workgroup/tpmwg/documents.php`. Find the latest version of the simulator, and download it; it will be called something like TPM 2.0 vX.XX VS Solution.

Building the simulator requires that Visual Studio 2012 or later be installed. Follow the directions in the TPM 2.0 Simulator release notes file to build the simulator.

Setting Up a Binary Version of the Simulator

Download the simulator from `http://research.microsoft.com/en-US/downloads/35116857-e544-4003-8e7b-584182dc6833/default.aspx`. Unzip the file into the directory of your choice.

Running the Simulator

Search for the simulator binary, `simulator.exe`, in the `install` directory, and start it. In some settings, you may need to configure the port numbers that the simulator listens to for commands. You can do this on the simulator command line.

The simulator uses two ports:

- *TPM command port*: Used for sending TPM commands and receiving TPM responses. The default port is 2321; if you need to change this, you can set it on the command line as follows:

  ```
  > simulator <portNum>
  ```

- *Platform command port*: Used for platform commands such as power on/off. The platform command port is always one greater than the TPM command port. For example, the default platform port number is 2322; and if you use the command-line option to set the TPM command port, the platform port is 1 greater than the command-line value.

There are two reasons to use a port other than the default port:

- If the network you're running on is using the default port for some other use
- If you want to run two instances of the simulator on the same machine, in which case you need to run one on a different port

Testing the Simulator

Let's look at three ways to test that the simulator is working: a simple Python script, TSS.net, and the system API test code.

Python Script

To test that the simulator is running correctly, you can use this Python script:

```
#!/usr/bin/python

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

platformSock = socket(AF_INET, SOCK_STREAM)
platformSock.connect(('localhost', 2322))
# Power on the TPM
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')
# 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 the size of the response, the response, and 4 bytes of 0's
reply=tpmSock.recv(18)
for c in reply:
  print "%#x " % ord(c)
```

The script sends the TPM `startup` command to the TPM. If the `startup` command works correctly, you should see the following output from the `for` loop `print` statement:

```
>>>for c in reply:
...   print "%#x " % ord(c)
...
0x0
0x0
0x0
0xa
0x80
0x1
0x0
0x0
0x0
0xa
0x0
0x0
0x1
0x0
0x0
0x0
0x0
0x0
```

If you're getting this result, the simulator is running correctly.

TSS.net

TSS.net is a C# library of code for communicating with the TPM. Download it from https://tpm2lib.codeplex.com, install it, and run a code example as described shortly.

System API Test Code

Follow the directions in the section "TSS 2.0" for the System API library and test code. If any TPM 2.0 command is successfully sent to the TPM, the simulator is working.

Setting Up the Software Stack

The two software stacks you can use to communicate with the TPM are TSS 2.0 and TSS.net.

TSS 2.0

TSS is a TCG standard for the TCG software stack. TSS 2.0 can be built on (and link to applications for) Windows and Linux. It consists of five or six layers and is implemented in C code except for a couple of Java layers. The layers at which TPM 2.0 code can be developed are as follows:

- *System API (SAPI)*: The lowest layer in TSS 2.0, which provides software functions for performing all variants of all TPM 2.0 functions. This layer also has tests that you can run against it. It requires detailed knowledge about TPM 2.0.
- *Enhanced System API (ESAPI)*: The next layer in TSS 2.0. It sits directly on top of the SAPI. This layer provides a lot of the glue code for doing encryption and decryption, HMAC sessions, policy sessions, and auditing. It also requires detailed knowledge about TPM 2.0, but it makes session handling much easier.
- *Feature API*: The layer to which most applications should be written. It provides APIs that isolate you from the messiness of the TPM 2.0 specification.
- *Feature API Java*: Layer that sits on top of the C code and performs the translation between C and Java so that Java applications can use TSS.

As of this writing, TSS 2.0 is implemented only at the System API level and includes a linked-in device driver for talking to the simulator. Currently, this code is only available to TCG members at `https://github.com/`. To access to the code, you must contact the TCG TSS workgroup chair to get permission. Follow the directions in the `readme.docx` file to install it and run the test code against the simulator.

TSS.net

As noted previously, you can download TSS.net from `https://tpm2lib.codeplex.com`, and then install it. To understand it, review the file: `Using the TSS.Net Library.docx`. Unfortunately, this doesn't tell you how to build and run the code examples. The `samples\Windows8` directory contains separate directories for sample projects; you can follow these directions for the `GetRandom` example and then apply those steps to other examples:

1. In Windows Explorer, open the solution file: `tss.net\tss.sln`.
2. Respond with OK to the prompts for loading the various projects.

3. Select Build > Build Solution.
4. Start the simulator. (See the earlier directions.)
5. Run the `GetRandom` executable: `tss.net\samples\Windows8\GetRandom\bin\Debug\GetRandom.exe -tcp 10` (10 is the number of random bytes).

You can now run other sample programs in a similar manner. Try them out!

Summary

Now that you have an execution environment (or maybe both of them) set up, you're ready to run the code samples from the following chapters of the book.

The next chapter describes the TCG Software Stack, TSS. This software stack is currently being defined and implemented and will be freely available under an open source license to application programmers. It's used for some of the subsequent code examples in this book.