CHAPTER 4

Existing Applications That Use TPMs

Even though more than 1 billion TPMs are deployed in the market, and they exist on almost all commercial PCs and servers, very few people know about them. And many people who do know about TPMs are surprised to discover that many applications are written for them. There are also a large number of ways to easily write applications that take advantage of TPM 1.2 devices. Because TPM 2.0 devices are just beginning to appear on the market, it's perhaps not surprising that not as many applications can use TPM 2.0 directly. The purpose of this book is to enable you to write programs that take advantage of all the features of TPM 2.0, both basic and advanced.

This chapter starts by looking at the various application interfaces that are used by programs to interface with the TPM hardware. Then you examine a number of applications that already use TPMs. Perhaps the most interesting part of the chapter—and one we hope you will help make out of date—is a short list of types of programs that should use TPMs but don't.

We follow up with some considerations that any programmer using a TPM must take into account, and a description of how some existing programs have handled them.

Application Interfaces Used to Talk to TPMs

A number of different types of applications have been written already for use with TPM 1.2 and 2.0. These can be classified by the programming interface they use:

- Proprietary applications written directly to the TPM (available for both 1.2 and 2.0).
- Legacy applications that use a middleware interface to talk with the TPM, specifically Public-Key Cryptography Standard (PKCS) #11 and Microsoft Cryptographic Application Programming Interface (CAPI). When PKCS #11 stacks are available for TPM 2.0, they work with it as well. They are available for TPM 1.2 in all operating systems. Beginning with Windows 8, Microsoft has made its cryptographic interfaces able to use both TPM 1.2 and TPM 2.0.

- Applications that use the TCG Software Stack (TSS) interface to talk with the TPM (multiple proprietary TSSs are available from IBM, Infineon, and NCP; an open source TSS code named TrouSerS is also available for multiple OSs). These are 1.2 implementations. TSS 2.0 is in development.
- Applications that use Java interfaces to talk with the TPM. So far, only 1.2 implementations that interface between Java code and the TPM exist, but 2.0 versions should soon appear. Mobile devices, especially those running the Android OS, use Java interfaces.
- Applications that use the Microsoft TPM Base Services (TBS) library: These can be used with either TPM 1.2 or TPM 2.0. Some functions work with either. Those that use new capabilities of the TPM 2.0 only work with it.
- Microsoft TSS.net works with TPM 2.0 and comes with a TPM 2.0 emulator! TSS.net is not compatible with the TCG standards, and only currently works on Microsoft products.

The first applications to use the TPM were proprietary applications that were shipped with the machines that had the first versions of TPMs. These included IBM's password manager and file and folder encryption, which used the TPM to store encryption keys. Dell, HP, and Infineon have their own varieties of these applications. Generally speaking, they work well, but are intended to focus on very specific usage models.

The next type of applications that use TPMs use it through cryptographic service providers (CSPs). There are two main kinds: those that use CAPI and those that use the RSA Corporation's PKCS #11. Any application written to use either of these APIs for cryptographic services can use a TPM via a standard means of pointing those cryptographic services to the TPM. Fortunately, most software that uses cryptography uses one of these two services, for good reason. Cryptographic services are notoriously difficult to program correctly, particularly if the programmer is worried about weak implementations that may be vulnerable to attacks such as side-channel attacks.[1] The best practice is to rely on experts to write those cryptographic services. Additionally, those cryptographic services may be certified by NIST as providing validated services that behave as expected, and hence can be used in government institutions.

Both of these APIs contain hooks that allow other cryptographic services to be substituted for those done in software by the service. This lets software take advantage of a hardware interface that provides protection against software attacks by implementing cryptographic services in a separate memory space. Such CSPs are available for Windows for both CAPI and PKCS. These implementations are available from Security Innovation, Wave Systems, Infineon, and Lenovo for a fee. They're often bundled with computers from major manufacturers. Infineon's CSP is noteworthy in that it can find applications on the machine that can use its services and give the user the opportunity to use the

[1]Side-channel attacks occur when the time or power it takes to perform a calculation can give hints to an attacker about what key is being used.

TPM with them. In other OSs, such as Linux, BSD, MAC OS, and Solaris, PKCS #11 implementations allow the substitution of TPM functions for public-key generation and random-number creation; these are available for free. Additionally, some companies, such as Charismathics, have made middleware suites that can use the TPM to provide cryptographic services.

The problem with using legacy interfaces (PKCS #11 and MS CAPI) is that they only utilize basic services available with a TPM, such as key generation and signing. Advanced applications that use the TPM's ability to attest to the health of the machine or allow controlled migration of keys without exposing them in the clear aren't available using these middleware solutions. As a result, TSS was created. An open source implementation called TrouSerS was implemented by IBM and ported to Windows by the University Politecnico di Torino in Italy.[2] Proprietary implementations are also shipped by a number of companies. TSS is currently available for TPM 1.2; an updated specification and implementation are being developed for TPM 2.0.

The TSS library is much more suitable to C programming than Java programming. Therefore, some people at MIT created a Java interface to the TPM. It is available from MIT.[3]

Microsoft, starting with Windows Vista, provides almost direct access to the TPM through a programming interface called TPM Base Services (TBS). The TBS interface accepts TPM-formatted byte streams and returns TPM-formatted responses or errors. Because this is a low-level interface, you're expected to use one of the many libraries that convert high-level-language callable functions to the underlying TPM byte-stream representation.

TBS performs several additional functions. First, it provides multiprocess, multithread access to the TPM by "maintaining" an internal queue of commands submitted. Second, the TPM performs under-the-covers TPM context management by using the TPM context `save` and `load` commands. This allows TBS to present each application with a virtual TPM that appears to have essentially unlimited resources like key slots, and ensures that one application cannot interfere with the keys or slots created by another. Third, TPM commands are submitted via a TBS context, and TBS automatically cleans up resources when the context is closed or the process dies.

Windows also layers additional security mechanisms on top of the TPM's administrative controls. The problem addressed is that the use of certain TPM commands can impact the stability or correct operation of the operating system or other applications, but the TPM commands are not properly protected by the TPM's protection mechanisms. For example, most Platform Configuration Registers (PCRs) should be updated only by the trusted computing base, but the TPM does not require special authorization to extend a PCR. In Windows Vista and 7, Windows limited TBS access to administrative applications only. In Windows 8, commands are grouped into three sets:

- *No Access*: Including `TPM2_ContextSave` and `TPM2_ContextLoad`
- *Administrative-token processes only*: Including `TPM2_PCR_Extend` and privacy-sensitive operations
- *Standard-use access*: Creation and use of keys, and so on

[2] http://security.polito.it/trusted-computing/trousers-for-windows/.
[3] http://projects.csail.mit.edu/tc/tpmj/.

The set of standard-use and administrative commands can be edited by the operating system administrator. The OS keeps copies of the TPM's authorization values in access-protected entries in the registry. This behavior is described in much more detail in the document *Using the Windows 8 Platform Crypto Provider and Associated TPM Functionality*.[4]

In addition to the low-level TPM access provided by TBS, Windows also exposes a subset of TPM behavior through five much higher-level interfaces.

TPM Administration and WMI

Windows exposes many common TPM administrative tasks through GUI tools and through a scriptable and remote programming interface called Windows Management Instrumentation (WMI). This interface lets an administrator switch on TPMs, clear them, disable them, and so on. It transparently supports both TPM 1.2 and TPM 2.0.

The Platform Crypto Provider

Most Windows programs use cryptography through a set of interfaces called Cryptography Next Generation (CNG). CNG provides a uniform library for performing both software-based and hardware (such as High Security Module) based cryptography. Windows 8 lets you specify the TPM as a key protector for a subset of TPM-supported cryptography by specifying use of the Platform Crypto Provider. The Platform Crypto Provider has been extended to include a few specific TPM-like behaviors, such as quoting and key certification.

Virtual Smart Card

Windows 8 further extracts the TPM to behave like a smart card in any and all cases where a smart card can be used. This includes both enterprise and web logon.

Applications That Use TPMs

Table 4-1 lists applications that are currently available that use the TPM, along with the interface they use and the OS on which they run. All these work with TPM 1.2. Some of them, as noted, also work with TPM 2.0.

[4] http://research.microsoft.com/en-us/downloads/74c45746-24ad-4cb7-ba4b-0c6df2f92d5d/.

Table 4-1. *Applications and SDKs That Use TPMs, by Interface and OS*

Application Type	Application Name	Interface	OS
VPN	StrongSwan clients (used in Linux, BSD, Solaris, and so on)	TrouSerS (1.2)	Linux
	Cisco client VPNs.	Wave Systems (MS CAPI) Charismathics (1.2)	Windows
	Microsoft embedded VPN or DirectAccess can directly use either TPM 1.2 or TPM 2.0 in Windows 8.	Microsoft TBS TPM Base Services (1.2 or 2.0)	Windows
	Checkpoint Firewall VPN can use the TPM.	(1.2)	
	TypeSafe (TPM-backed TLS).	jTSS (1.2)	Linux
Attestation	Wave Systems Embassy client/ERAS server package.	TrouSerS (1.2)	Windows
	Wave Systems Endpoint Monitor	TrouSerS (1.2)	Windows
	Strong Swan TNC solution hooked to the TPM with PTS.	(1.2)	Linux
	NCP's Secure VPN GovNet Box (a separate box interposed between a computer and the network that establishes a secure VPN). The software is tested using TPM attestation.	(1.2)	Unknown
	AnyConnect	(1.2)	
	JW Secure has written an application that is Kerberos-like for Windows.	Microsoft TBS TPM Base Services (2.0)	Windows
	Integrity Measurement Architecture.	TrouSerS (1.2)	Linux, Unix-like OSs

(*continued*)

Table 4-1. (*continued*)

Application Type	Application Name	Interface	OS
	TPM Quote tools (SourceForge)	TrouSerS (1.2)	Linux, Windows
	TrustedGRUB	Direct (1.2)	Linux
	TVE	Trousers(1.2)	Linux
	Tboot	Direct(1.2)	Windows, Linux
	Flicker	Direct / Trousers (1.2)	Windows
Full disk encryption	Microsoft BitLocker	Microsoft TBS TPM Base Services (1.2, 2.0)	Windows
	dm-crypt	Direct (1.2)	Linux, Android
	SecureDoc		
File and folder encryption	Pretty Good Privacy (PGP)	PKCS #11 (1.2)	Windows
	OpenPGP	PKCS #11(1.2)	Linux
E-mail	Thunderbird for encrypted e-mail and signed e-mail	PKCS #11(1.2)	Windows, Linux
	Outlook	MS CAPI(1.2, 2.0)	Windows
Web browsers	Internet Explorer	MS CAPI(1.2, 2.0)	Windows
	Firefox	PKCS #11(1.2)	Windows Linux
	Chrome	PKCS #11(1.2)	Windows Linux
TPM Manager	TPM Manager (SourceForge)	microTSS (1.2)	Linux

As the table demonstrates, many applications use TPMs. There are even some large companies that use them.[5] BitLocker is one of the most widely used of these programs that use extended capabilities of the TPM. Wave Systems Embassy Suite is another. Often, conflicting management software requires multiple TPM programs to be used on the same system.

[5]See Ellen Messmer, *Network World* (2010), "PwC Lauds Trusted Platform Module for Strong Authentication," www.networkworld.com/news/2010/091510-trusted-platform-authentication.html.

With a 1.2 TPM, there was a single storage root key (SRK), which had to have an authorization that was shared by all applications using the TPM. Unfortunately, there was not unanimity in how to create the SRK—it could be created without needing any authentication, needing only a well-known secret of 20 bytes of 0, or needing the hash of a well-known secret for its password. Additionally, there was an owner authorization that was somewhat sensitive, because it was used to reset the dictionary attack mechanism as well as reset the TPM or create an attestation key (thought by some to be privacy sensitive).

Unfortunately, the owner authorization was also used to authorize allocation of non-volatile RAM space, which meant applications that needed to allocate nonvolatile RAM space had to know it. But if a different application took ownership of the TPM and set the owner authorization to a random number, protected by a back-end management function, it was unknown even to the end user. Some applications did this. If applications did not know how to coordinate with that back-end management application, they could not function.

The result was that the user was restricted to using a single suite of applications with the TPM, in order to allow all applications to have access to the authorizations they needed. In practice, this meant software that directly used the TPM had to be from the same developer as the management software used to set up the TPM.

This issue was somewhat mitigated when using only PKCS #11 or MS CAPI enabled applications, because they only required that there be a single application for managing the TPM; but they also couldn't use the higher functions of the TPM, such as attestation. This problem seems to be gradually disappearing. For example, Wave Systems software can manage TPMs for attestation and also for BitLocker.

TPM 2.0 still requires some coordination for authorization; but it lets you use multiple SRKs with the TPM, allowing completely separate applications to use the TPM with less coordination.

In researching applications that use the TPM, most of the use cases that come quickly to mind are supported by commercial software. However, some obvious use cases for software that uses a TPM, don't seem to exist in the marketplace.

Applications That Should Use the TPM but Don't

In the past few years, the number of web-based applications has increased. Among them are web-based backup and storage. A large number of companies now offer such services, but as far as we are aware, none of the clients for these services let the user lock the key for the backup service to a TPM. If this were done, it would certainly be nice if the TPM key itself were backed up by duplicating it on multiple machines. This appears to be an opportunity for developers.

Another application that has become more useful recently is remote management. Many companies now offer ways of allowing one computer to "take over" management of another computer. For instance, you can use this functionality to monitor your network remotely or to give troubleshooting advice to remote members of your family. But again, the security models we are familiar with, use passwords to gate the remote access. Although long, hard-to-remember passwords can provide some security, they aren't fun to use. This seems to be an ideal place for TPMs to be used—restricting remote access to machines that have been linked together with public/private keys. There do not appear to be any commercial applications that use the TPM for this—most commercial applications

don't even support use of other cryptographic devices, including smart cards, for increased security. This is not due to lack of software development kits for writing such software, because several of these kits exist.

Building Applications for TPM 1.2

When you're building an application that will use a TPM, it is important to first decide if you are going to use the advanced facilities of the TPM beyond those that are exposed by PKCS or MS CAPI. If not, then it makes the most sense to write your application to these interfaces. This way, your application can be used on those machines with and without TPMs. But to use unique TPM features such as attestation, extended authorization, localities, an NVRAM locations, you have no choice but to use one of the custom TPM interfaces.

A number of API libraries are available for writing applications using custom interfaces. TSS 1.2 had a reputation for being hard to learn, so other suites were developed. TPM/J was developed at MIT to provide an object-oriented means of programming to the TPM.[6] Institute for Applied Information Processing and Communication (IAIK), of Graz University also delivered a version of Java integration with the TPM through trustedJava.[7] Sirrix provided a microTSS, an attempt to simplify the TSS specification.[8]

Additionally, command-line tools for the TPM were released by IBM together with a TPM emulator on SourceForge. As a result, it was possible to exercise TPM base commands in batch file.

Microsoft's TBS interface started out as a basic interface with the TPM, but its API is growing, and it may turn into a very nice means of programming TPMs. The biggest news in TBS programming came in Windows 8, where the TBS interface abstracted the difference between TPM 1.2 and TPM 2.0 so that all the APIs work with either chip. This is particularly useful for applications that use only those APIs, but it doesn't (yet) expose the new functions in the TPM 2.0 specification. TSS.net, which Microsoft also released, lets all commands be sent directly to the TPM, although it doesn't, as yet, have a high-level interface for the new TPM 2.0 commands.

TSS.Net and TSS.C++

Windows 8 and TPM 2.0 were released before there were standards for TPM programming. To fill this gap, Microsoft developed and open sourced two libraries that let application programmers develop more complicated TPM-based applications than CNG or virtual smart cards allowed.

TSS.Net and TSS.C++ provide a thin veneer over TPM 2.0 for both managed code (such as C#) and native code (C++) applications. Both libraries allow applications to be built for a real TPM device (on TBS) or a TPM simulator (over a TCP/IP network connection.)

---

[6] http://projects.csail.mit.edu/tc/tpmj/.
[7] http://trustedjava.sourceforge.net/.
[8] http://www.filewatcher.com/p/tpmmanager-0.8.tar.gz.3959086/tpmmanager-0.8/src/microtss/TSS.cpp.html.

Although the TSS.Net and TSS.C++ libraries are low level, the authors have made every effort to make programming the TPM easy. For instance, here is a complete program for obtaining random numbers from the TPM:

```
void GetRandomTbs()
{
    // Create a TpmDevice object and attach it to the TPM. Here you
    // use the Windows TPM Base Services OS interface.
    TpmTbsDevice device;

    if (!device.Connect()) {
        cerr << "Could not connect to the TPM device";
        return;
    }

    // Create a Tpm2 object "on top" of the device.
    Tpm2 tpm(device);

    // Get 20 bytes of random data from
    std::vector<BYTE> rand = tpm.GetRandom(20);

    // Print it out.
    cout << "Random bytes: " << rand << endl;

    return;
}
```

All of these interfaces work, but of course some, such as TBS, are specific to the Windows OS. If you want to write programs that are portable to other OSs, you are better off with one of the others. For TPM 1.2, TSS was the interface with the broadest OS adoption. The next section considers an application that was written using TSS to take advantage of advanced TPM functions.

Wave Systems Embassy Suite

Wave Systems has written software to a TPM-specific interface, rather than to a higher-level interface such as PKCS #11. It needed to be done that way, to take advantage of the TPM's attestation capabilities. Because these capabilities aren't addressed in any other crypto-coprocessor, they aren't available in standard interfaces such as PKCS #11. Wave Systems uses the TCG TSS interface implemented in TrouSerS to talk to the TPM, manage the TPM owner password, create attestation identity keys (AIKs), and attest to those values via a standard called Trusted Network Connect, which communicates back to an administrative server. This server notices when PCR values have changed, and it can send alerts to IT staff when that happens. Some PCRs (like 0, which represents the BIOS firmware) should not change, unless the BIOS of a device has been upgraded, an event that IT should be aware of. TSS 1.2 was available for Windows, Linux, Solaris, BSD, and even the MAC OS. TSS 2.0 will be a good selection for the same reasons, if you want to be able to port your code to other OSs.

TSS 2.0 has been designed specifically with the aim of making programming TPM 2.0 as easy as possible. It is designed in layers so that at the lowest level, direct access to the TPM is still possible. Common design patterns that use a cryptographic coprocessor are made particularly easy to use at the highest application level programming interface. However, there are still some ground rules that every application developer should remember when developing applications that use a TPM.

Rocks to Avoid When Developing TPM Applications

When using the TPM in an application, there are two major pitfalls to avoid. First, the TPM (or another component on the motherboard) may die, or users may upgrade their equipment. If the motherboard is replaced, any keys that are locked to the TPM go away. Second, if data is locked to PCRs (a process called *sealing*), and the things measured into the PCRs are updated, that data is no longer unsealable.

Both of these problems amount to the same thing: management of the keys and data locked to a TPM needs to be carefully considered. An example of how do this well is found in Microsoft's BitLocker application, which first came out with Windows Vista Enterprise.

Microsoft BitLocker

Microsoft gave careful consideration to both of the previously described problems when it created the BitLocker application, originally embedded in the Enterprise edition of Vista. This program was used to do full-disk encryption of the hard disk on which Windows resided. To do this, early in the boot sequence BitLocker obtained a key from the TPM. This key was sealed to PCRs that represented the boot sequence of the computer up to the point where the kernel was loaded into memory. BitLocker could also require the user to enter a password. To enable management of the encryption key used for full-disk encryption, the sealed key was used as a key encrypting key (KEK) and used to encrypt the full-disk encryption key. The actual key used for the full-disk encryption key could be then backed up by also encrypting it using a very long random password. This password could be kept secure elsewhere (for example, on a USB key locked in a safe). This way, if the motherboard was replaced, the TPM died, or the hard disk was moved into a new system, the data stored on it was still accessible.

Additionally, Microsoft gave thought to the problem caused by people upgrading their BIOS. Such an upgrade prevented the TPM from being able to unseal the KEK. Although the random-number backup sufficed for recovery in this case, Microsoft decided it would make more sense for an administrator doing the BIOS upgrade, who already had access to the decrypted data, to have a means to temporarily leave the full-disk encryption key in the clear while the BIOS upgrade was performed and then reseal it to the TPM's new PCR values after the BIOS upgrade. It is important to realize that making things easy for the user at a small cost to security (leaving the drive open for the brief time while a BIOS upgrade was taking place) is usually a good tradeoff. Security that is hard to use is seldom used.

When IBM came out with its first TPM solutions, several years before BitLocker saw the light of day, it also had to keep manageability problems in mind.

IBM File and Folder Encryption

IBM had a similar problem when it allowed storage keys to be used for file and folder encryption to the TPM, and it solved the issue in a similar way. Instead of generating a random number, IBM wanted to let users type the answer to questions in order to recover the disk encryption key; this key was normally encrypted with the KEK, which in turn was protected by the TPM. This can be dangerous, because it may allow an attacker to simply try many answers to these questions in the hope of generating the correct answer and unlocking the drive. IBM's solution to this problem was clever. The company realized that although in normal use the key needed to be available almost immediately, in the case of recovery, it was fine if it took several minutes to recover the data. Therefore IBM performed a hash operation on the answers to the questions over and over again until a few minutes had passed, noted the number of operations, and then used the resulting value as a key to encrypt the file and folder encryption key. It then stored the number of operations and the encrypted blob on the hard disk. In order to decrypt this blob, someone had to spend several minutes for every attempt to answer the questions. This quickly becomes impractical for an attacker, but it costs a user only a few minutes in the case of recovery.

When TPM 2.0 was being designed, the architects had experience with the multitude of problems caused by managing TPMs, so new features were built into 2.0 to help solve these issues. One specific problem that is encountered repeatedly in security software is the need to manage authorizations (passwords). For example, someone changes a password while on a plane or late at night at a hotel, when they aren't connected to the network; then, the next day, they can't remember their password. Or someone working for a corporation quits or (worse yet) dies and leaves important corporate data encrypted on their hard disk without telling anyone their password. IT organizations are assumed to be able to fix problems like this—but it's hard to see how they can. TPM 2.0 enhanced authorization was designed to help fix the issue of managing passwords.

New Manageability Solutions in TPM 2.0

Programs to solve the manageability problem can use the same techniques used with TPM 1.2 devices; but with TPM 2.0, a number of new solutions are available. Loss of a password or authorization is unfortunately a big issue in the industry—in an enterprise, many people forget their passwords or lose their smart cards every day. There's no shame in admitting it: we've all done it.

Generally, setting up a certified key on a TPM takes some effort, but doing this during provisioning time in TPM 2.0 is much easier. If users need their TPMs reprovisioned in the field, this burdens IT staff. Because IT staff are major players in computer purchasing decisions, the architects of the TPM specification needed to solve this problem. The TPM 2.0 design allows management not just of keys (so they can be duplicated on other TPMs), but also of authorizations; this is demonstrated in detail in the chapter on enhanced authorization. For now, suffice it to say that major TPM 2.0 enhancements were designed to solve this problem.

Summary

In this chapter, you have seen that many different software interfaces can be used to take advantage of TPM capabilities, and many currently available applications use TPMs. Some of these only take advantage of standard capabilities such as those in any crypto coprocessor—creating, storing, and using keys. These basic interfaces, such as MS CAPI and PKCS, exist in a large number of applications. Taking advantage of higher-level capabilities, such as those used in attestation software, requires talking to TPM-specific interfaces instead of generic cryptographic interfaces. There are several of those for TPM 1.2 and currently at least two, Microsoft TBS and TCG's TSS, for the TPM 2.0 interface.

Finally, you saw that when creating applications that use a crypto coprocessor such as a TPM, there are rocks to avoid: the cryptographic processor may die, or a motherboard to which it's attached may have to be replaced. Even worse, the only user who knows a password may become unavailable. For the sake of manageability, you need a strategy to recover functionality after such an occurrence. Enhanced authorization, a new feature in TPM 2.0, meets this need; it is explained in chapter 14.

To continue your journey into the TPM 2.0 universe, in the next chapter we kick-start your ability to read and understand the TPM 2.0 specification.