Trusted Platform Module Evolution

Justin D. Osborn and David C. Challener

For more than a decade, commercial PC platforms have been shipping with a standards-based embedded security subsystem on the motherboard known as the Trusted Platform Module, or TPM. TPMs have been used in a wide variety of applications, but some issues have hampered large-scale adoption. During the last 8 years, the Trusted Computing Group has been working on revising the specification to increase its flexibility, manageability, and utility. This article presents TPM use cases and explains the motivation for the major changes made to improve the TPM specification.

INTRODUCTION

The Trusted Platform Module (TPM) is a cryptographic coprocessor chip that has been included on most enterprise-class PC and laptop motherboards produced in the past decade. The specification for TPMs is produced by the Trusted Computing Group (TCG), an industry consortium. TPM chips are produced by a variety of vendors including Infineon, Broadcom, Atmel, STMicroelectronics, and Nuvoton. PC manufacturers shipping TPM-enabled PCs include Dell, Lenovo, HP, Toshiba, and Fujitsu. The current specification version is TPM 1.2,[1] but the TPM 2.0 specification has been released to the public for comment.[2] Microsoft has announced that all systems submitting to the Windows Certification Program after 1 January 2015 will be required to have TPM 2.0.[3] TPMs have been used in a wide variety of applications from secure military platforms,[4] to secure industrial control systems,[5] to secure electronic voting systems.[6]

The TCG's architecture for hardware-based security was motivated by increasingly sophisticated malware attacks in the late 1990s. Then—and still today—the most popular way to defend against network attack on the PC client is through antivirus software. The fundamental flaw with a software-only defense approach is that software cannot effectively verify itself. Malware that gains execution on a computer at the same privilege level as the antivirus software can simply disable the program to conceal its existence. Malware had also become more sophisticated by targeting firmware and bootloader code, which is rarely verified by antivirus software. To securely verify software configuration, a computing platform needs a place to record and verify the state of software that is outside of the system memory space. In the computer security world, the concept of a hardware security monitor that verifies software in main memory has been around for decades. For the TCG, however, it was

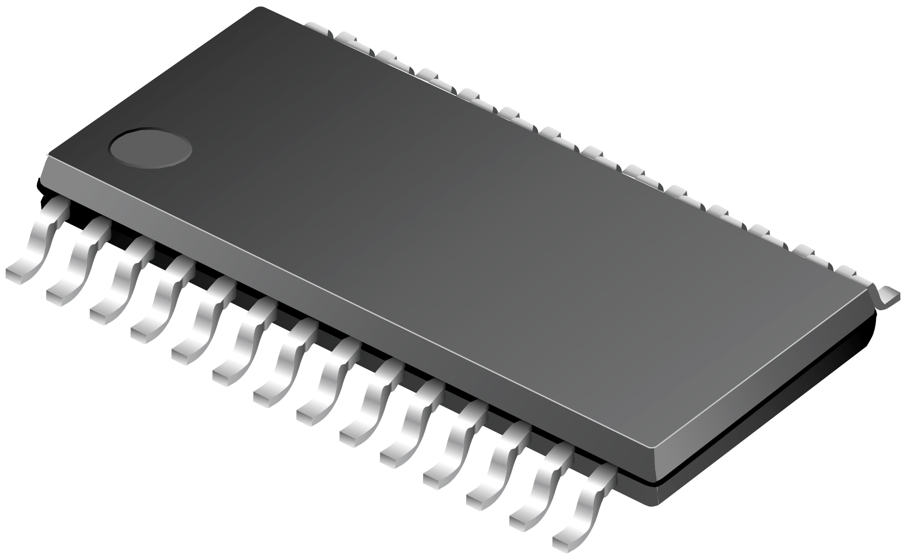

Figure 1. A TPM 1.2 chip.

important to design a hardware module that increased the cost of a PC by $1 or less to allow for widespread adoption. Thus, the TPM was designed as a passive chip (it responds only to commands from platform software) that helps platform software to verify itself, as well as a general-purpose cryptographic module.

The TPM specification was written to be platform independent, with the idea that additional specifications would be written to govern implementation on specific platforms. Although the PC is the most popular platform on which TPMs are widely deployed, the TCG has working groups for mobile,[7] embedded,[8] and virtualized platforms.[9] On the PC, the TPM is usually implemented as a discrete integrated circuit in a 28-pin thin-shrink small-outline package (illustrated in Fig. 1). It is attached to the Low Pin Count bus. It can be integrated into other chips on the motherboard (e.g., the Broadcom TPM is part of a security chip that includes a fingerprint scanner and smartcard reader[10]). Firmware and operating system software interact with the TPM by accessing memory-mapped registers, which the main PC chipset converts into bus commands. Because TPM commands are somewhat complicated, the TCG Software Stack (TSS) was created to make interfacing easier for programmers.[11] Applications wishing to use TSS services link in a library that provides a simplified application programming interface. The TSS translates these high-level application programming interface calls into the equivalent TPM commands and sends them to the device driver, which transmits them to the chip.

The TPM contains two important functional components—a cryptographic engine that can perform encryption, digital signatures, and hashing, and a special register set called Platform Configuration Registers (PCRs). These registers store a representation of the state of software on the platform through a process we will describe later. The TPM 1.2 specification is limited to RSA encryption and SHA-1 hashing, but the TPM 2.0 specification has been written to allow for flexibility in encryption algorithms. The cryptographic engine can be combined with PCRs for some very interesting use cases that we will also describe later. In addition, the TPM contains a small amount of nonvolatile random access memory (NVRAM), which can be used to store keys or data. Software can also use the TPM to create monotonically increasing counters (counters that only increase and cannot be reset until power is reset). All of these items (keys, encrypted data, NVRAM, and counters) are known as "objects" in TCG parlance.

Because of limited storage space, the TPM does not normally store keys permanently. Rather, it contains a Storage Root Key that is stored persistently. When software requests that a key be created, the TPM generates a new RSA key, concatenates it with a value known only to the TPM called the "proof," adds any authorization information, and then encrypts these data with the public portion of the Storage Root Key, returning an encrypted "blob" to the requesting program. The program is responsible for storing the blob. When software wants to use the key, it sends a key load command along with the key blob. The TPM decrypts the key with the Storage Root Key, checks that the proof value is its own, checks that the supplied password matches, and loads the key into TPM memory. Software can then specify this loaded key in an encryption or digital signature command. Subsequent key creation requests may specify this loaded key as the "parent" key, allowing for key hierarchies.

TPM BACKGROUND AND USE CASES

In this section, we describe some terms and use cases that are commonly used in TCG architecture.

Platform Configuration Registers

PCRs are special registers within the TPM. They are the size of a hash value (20 bytes for SHA-1, used in TPM 1.2). They cannot be set directly. Instead, software sends an extension command to the TPM specifying a PCR number and a hash value. The TPM sets the new PCR value to the hash of the old PCR value concatenated with the input hash value. Because the hash function is assumed to be a one-to-one function, PCRs represent a unique sequence of extension operations.

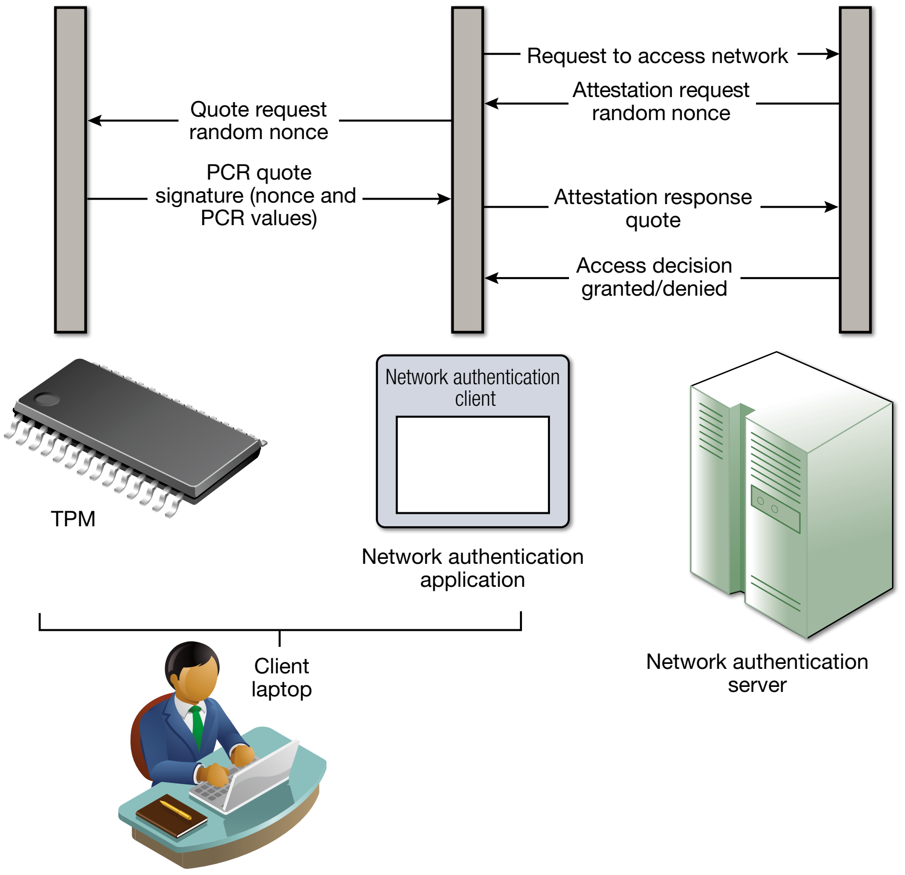
Request to access network
Attestation request random nonce
Quote request random nonce
PCR quote signature (nonce and PCR values)
Attestation response quote
Access decision granted/denied
TPM
Network authentication client
Network authentication application
Client laptop
Network authentication server

Figure 2. TPM-enabled network attestation concept showing notional requests and responses.

Measured Boot

Measured Boot is a process by which preboot software (firmware and operating system bootloaders) uses the TPM to verify that nothing has changed in the preboot environment. Ideally, on a trusted platform, all software is "measured" (that is, hashed) before it is executed. Measured Boot is a critical part of the TCG architecture and foundational to the other use cases. The core idea is that starting from power-on, each piece of code that is executed measures the next piece of code and extends a PCR before executing it. The specification describes the PCR number that should be used for each piece of software in the boot process. The first piece of code that executes (either the BIOS boot block or the entire BIOS chip) should be completely write-protected, guaranteeing that nobody can spoof this sequence of measurements with the known "good" values. If implemented correctly, this means that the PCRs contain a cryptographically unique representation of the state of software on the system. Measured Boot is different than "Secure Boot"[12] introduced by Microsoft in Windows 8 where the firmware simply verifies a signature of the bootloader before executing it. Measured Boot is a much more comprehensive security measure than Microsoft's Secure Boot.

Sealed Storage

Once the PCR values are properly configured, software can then "seal" keys or data to specific states of the platform. This means creating keys that can be used only when the software is in a "good" state, or encrypting data that can be decrypted only in a good state. For instance, Microsoft's BitLocker disk encryption product seals its encryption key to the TPM[13] and links it to PCRs, so the disk cannot be decrypted if malware changes the bootloader or firmware. Another application is single sign-on, where a TPM key is used by software to access all enterprise services but can be used only when the platform is in a specific state.

Network Attestation

Attestation is the process by which a platform can cryptographically prove to another platform that it is in a particular state. It does this by creating a special kind of key called an identity key that can be used only for this process. The network server creates a cryptographic nonce—a random value used to prevent replay attacks—and sends the nonce to the client. Software on the client creates a TPM "quote" request, sending the nonce to the TPM and specifying an identity key. The TPM hashes the PCR values along with the nonce and signs the hash. The client software sends this quote to the server, which can verify the platform configuration with the public portion of the identity key. See Fig. 2 for an illustration of this process. TPM-enabled network attestation can provide hardware-based assurance that only platforms owned by the enterprise are allowed access to the network and that the software on these platforms has not been modified.

TPM 2.0—TOWARD UBIQUITOUS TRUSTED COMPUTING

Although the TPM 1.1b and 1.2 specifications created the foundation for the use cases described, a number of roadblocks occurred that hampered adoption by industry. These included ambiguities in the specification, rigidness in the policy regarding how keys are used, tight coupling to specific cryptographic algorithms, and issues related to control of the TPM by platform software.

Algorithm Agility

Ideally, a flexible cryptographic module would handle both symmetric and asymmetric encryption algorithms. Symmetric encryption refers to algorithms where the same key is used for encryption and decryption. Asymmetric encryption (or public key cryptography) refers to algorithms that use a public and private key pair. To send an encrypted message, the sender encrypts the message with the receiver's public key—this ensures that only the receiver (who knows the private key) can decrypt it. Public key systems can also be used for digital signatures. The message sender can sign the message with his or her private key, and the receiver can use the public key to verify that the message originated with that person.

As noted earlier, TPM 1.2 and earlier versions were restricted to using the RSA algorithm for encryption and the SHA-1 algorithm for cryptographic hashing. Using symmetric encryption (such as DES or AES algorithms) would have greatly simplified the TPM 1.2 specification. However, at the time it was being developed, exporters of symmetric cryptographic chips were subject to record-keeping requirements under U.S. law. Because it would be prohibitively difficult for computer manufacturers to meet these requirements, the decision was made to use only RSA. RSA is an asymmetric algorithm that is primarily designed to encrypt very small messages. RSA is normally used to exchange symmetric encryption keys, which are then used to encrypt larger streams of data. Constraining the TPM to encrypt data and keys with RSA keys required some serious design shortcuts.

The algorithms themselves have also weakened over time as more cryptanalytic attacks have been published. In 2009, a 768-bit RSA key was factored,[14] and there is disagreement in the cryptographic community over the strength of 2048-bit RSA used in TPM 1.2. Recent attacks have surfaced on the SHA-1 algorithm, which was designed to have 80-bit collision resistance (that is, requiring at least 2^{80} evaluations of the hash function to discover two messages with the same hash value). In 2005, an attack was discovered that weakened it to 69 bits.[15] Current estimates are around 61.5 bits.[16] In 2005, the National Institute of Standards and Technology started work to phase out use of SHA-1 in favor of the SHA-2 family and began working on a new algorithm to be called SHA-3. Simultaneously, the TPM specification committee began working on the next version of the TPM specification, which eventually became TPM 2.0.

The key structure of TPM 1.2 did not allow simply replacing the SHA-1 algorithm with SHA-256. The additional 12 bytes in SHA-256 made the resultant data structure too large to encrypt in one step with an RSA key. Truncating the SHA-256 hash to 160 bits would make the data fit the structure but would weaken the strength of the hash. Ultimately it was decided to change the key structure entirely.

In the TPM 1.2 specification, a 2048-bit RSA key encrypted another key by using a trick. An RSA private key consists of two large prime numbers. The public key is the product of those numbers. Only one of the prime numbers is required to recover the other prime number if the product of the two primes is also known. Therefore, a parent key is only used to encrypt one of the prime numbers of a child key, and when the child key is loaded into the TPM, both the child's first prime number (encrypted with the parent key) and the public key are loaded into the chip. The TPM then uses the parent key to decrypt the child's first prime number and then divides the result into the public key to obtain the other. This has the effect of reducing the amount of data that needs to be encrypted and obviates the need for the TPM to contain a symmetric algorithm. However, it is also slow.

In TPM 2.0, keys are generally stored symmetrically encrypted, with the key used to encrypt them itself encrypted with a parent key called a key encryption key, or KEK. See Fig. 3 for an illustration. This two-step process adds very little execution time overhead to the TPM 1.2 method, because symmetric encryption is much faster than asymmetric encryption. It also allows for the data structures encrypted to be much larger than before when only an asymmetric method was used. This change also allows any combination of algorithms to be used in the specification regardless of the key sizes they individually could encrypt. As a result, the specification abstracted the algorithms supported to an algorithm ID table, which is referenced by the specification but is not part of it.

Currently, the algorithm ID table lists a variety of public/private key algorithms, symmetric algorithms, hash algorithms, key derivation functions, cryptographic modes, and signing schemes, including National Institute of Standards and Technology–certified algorithms. The algorithm ID table is independent of the specification and can be added to without changing the specification at all by simply allocating a number to represent a new algorithm addition.

Independent of the specification, platform-specific specifications delivered by the TCG will provide mandated sets of algorithms that will be implemented in

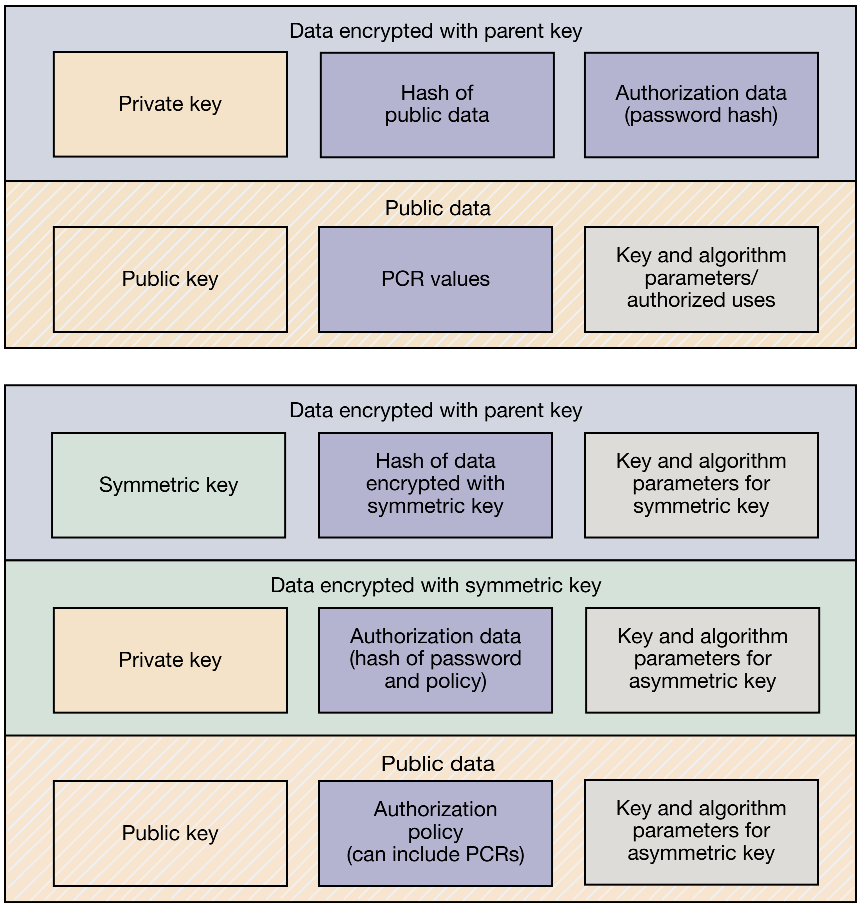
Data encrypted with parent key
Private key
Hash of public data
Authorization data (password hash)
Public data
Public key
PCR values
Key and algorithm parameters/ authorized uses
Data encrypted with parent key
Symmetric key
Hash of data encrypted with symmetric key
Key and algorithm parameters for symmetric key
Data encrypted with symmetric key
Private key
Authorization data (hash of password and policy)
Key and algorithm parameters for asymmetric key
Public data
Public key
Authorization policy (can include PCRs)
Key and algorithm parameters for asymmetric key

Figure 3. Asymmetric key structure for TPM 1.2 (top) and TPM 2.0 (bottom).

various physical TPMs. This will ensure interoperability among the various implementations. These are likely to include 1024-bit and 2048-bit RSA, 256-bit elliptic curve cryptography, SHA-1, SHA-256, and 128-bit AES in the first implementations. RSA and SHA-1 will be used for backward compatibility.

TPM 2.0 will also probably include a suite of Chinese commercial cryptographic algorithms (SM2 elliptic curve cryptography, SM3 hash, and SMS4 symmetric cipher). Western encryption algorithms are illegal in China, and Chinese-developed algorithms are mandated for commercial products. China developed its own version of the TPM 1.2 specification (called the Trusted Cryptography Module) based on these algorithms but only published a partial specification for the software application programming interface.[17] PC manufacturers such as Dell and Lenovo use the Chinese module on their TPM-enabled systems when selling in China. To have these algorithms included in the TPM 2.0 specification, the Chinese had to publish their algorithms. The algorithms were included in the first publicly released TPM 2.0 draft specification, but debate is still ongoing within TCG as to their inclusion in the final version. A similar situation exists with Russia. TPMs are banned in Russia, but there is no Russian equivalent. Russia's commercial crypto algorithms have been published; however, their standards do not specify certain critical parameters (presumably for the purpose of keeping the knowledge of the most secure parameters limited to government). Multiple implementations of the same algorithm have prevented TCG from standardizing, so Russian algorithms will probably be left out of TPM 2.0 for the foreseeable future.

The implementation of algorithm agility has inherent problems that had to be solved in the specification. If more than one algorithm is used in an implementation (as is expected), then it is clear that some algorithms will be stronger than others. It is unlikely that someone will want to encrypt a key with nominal strength of 120 bits with a key with nominal strength of 80 bits. Similarly, one would want to use hash algorithms with strength at least as strong as the key whose integrity they are protecting. However, there is not necessarily agreement as to the strength of algorithms. Therefore, when a public key is created, a suite of algorithms is selected by the creator—asymmetric, symmetric, and hash algorithms, etc. The hash algorithm selected will be used for calculations of the name of the key, for integrity calculations of things it encrypts or signs, and for authorization calculations. Mixing and matching of algorithms at this level is not allowed.

Because the creator of a key gets to select the parent of the key being created, it is possible for the creator to pick a key with less strength than the key being created. However, to make this a deliberate choice, the specification is designed to make it easy for a user to determine the strength of all of the keys in a key chain that terminates at the TPM. It is possible to get proof from the TPM of that list of key ancestors.

Additionally, although a primary key created under a TPM can be of any strength selected by the creator, any storage key underneath a key encryption key has to follow a rule: If the key cannot be independently duplicated or moved to a different TPM, it must have exactly the same algorithm set as its parent. Keys that cannot be used as key encryption keys are called leaf keys, and they can have any set of algorithms selected by the creator.

Enhanced Authorization

In TPM 1.2, authorization (the process by which software proves to the TPM that it is allowed to use a key, counter, or NVRAM object) is very limited in scope. The only way to restrict access is by passwords (represented as SHA-1 hashes) and PCR values. To use a key, software will have to prove knowledge of the password hash as part of the command. It is also possible to seal a key to specific PCR values so that the key cannot be used unless the password is known and the PCRs are in the chosen state.

This means that TPM 1.2 authorization is fairly inflexible. When multiple users share a platform, it makes it difficult to share TPM keys and data. Users often have their own sets of keys, because they individually know their passwords. This also makes system administration difficult because users must physically enter their passwords to authorize usage of their keys.

Enhanced Authorization (EA) in TPM 2.0 greatly expands the methods by which key and data use can be authorized, and the policy has become much more flexible. In TPM 1.2, software would prove it had knowledge of the password in an authorization session. A single command would be sent to the TPM before the command requiring authorization to start the session. One of the parameters to the command was a hashed message authentication code that included the password hash along with other values so that the TPM could verify knowledge of the password.

EA extends these authorization sessions into policy sessions, which allow multiple authorization methods to be combined by Boolean logic. For instance, let's say one wants to have a key that is accessible by both Alice and Bob and they have individual passwords. A policy can be created that says, "Authorize access if and only if Password(Alice) or Password(Bob)." Or say one wanted to create multifactor authentication for both users but allow either user to access the key. A policy can be created that says, "Authorize access if and only if ((Password(Alice) and SmartCard(Alice)) or (Password(Bob) and SmartCard(Bob))." The way this works is that software will create the policy and then specify the hash of that policy when creating the key or data. The TPM does not need to know the details of the policy—the hash is sufficient. Later, when software wants to use the key, it will start an authorization session and send one command to the TPM for each token in the policy equation. The TPM will verify that the policy specified in that sequence of commands is satisfied and also verify that the hash of the policy command sequence is the same as the policy hash specified at creation time.

Not only does EA create a flexible policy language, it also adds more possible authorization methods. Previously, one was limited to passwords and PCR values. Now the authorization methods include the following:

- Passwords—similar to TPM 1.2
- PCR values—same as TPM 1.2, but the addition of Boolean logic means multiple PCR states can be used
- TPM counter or NVRAM value—require that these items have a particular value
- Physical presence— require that a user be physically present at the PC
- Commands—require that the object can only be used with a given set of commands
- Digital signature from a public key—this allows smartcards such as Common Access Cards to be used for authorization

Management for some things is much easier to do with simple passwords than with complex policies, and so the design allows for each object to have a simple password associated with it, a policy, or both. If both are present, the creator of the object can split roles between the two, so that (for example) a simple password may be used for signing with a key, but administrative tasks such as creating a certificate for the key or creating a backup of the key may require an IT administrator's smartcard authorization.

Compilable Specification

In the TPM 1.2 specification, behavior of the TPM when responding to a given function is written in pseudocode. Although this is more specific than a natural language description, it still leaves a lot of wiggle room for interpretation. Many issues arose over the years when vendors implemented slightly different versions of the same command. Some vendors chose to fix these issues with firmware updates to the TPM, but other TPMs cannot be updated (their code is permanently burned in at the factory). To avoid similar problems, the TPM 2.0 specification is written in C. This should truly standardize TPM behavior across vendor implementations.

The specification was designed in four parts:

- Part 1 is an introduction, but it also contains specifics of how sessions are set up with the TPM and used to authenticate the use of a TPM or TPM entity.
- Part 2 is a list of variables, structures, and constants used in the specification.
- Part 3 is a list of the commands that a TPM can execute. Those commands include a brief description in English of what they do, together with code that implements them.
- Part 4 consists of subroutines used by Part 3. Throughout the specification are special symbols, called decorations, that help special software extract code from the specification.

The symbols used in any part of the specification are listed at the beginning of each part. For example, when an ampersand is used in front of a variable, it means that the variable requires authorization via an authorization session when the command is executed. If there are multiple variables preceded with an ampersand, then there will be multiple authorization sessions in the command.

These types of special symbols, called decorations, are used by special software created by Microsoft that can extract the code from the specification and create a compilable reference TPM. Unfortunately, this special software is not freely available, although the resultant code has been provided by Microsoft to TCG members to use in verifying member's implementations. These decorations make the specification more precise but also require the reader to learn a special set of symbols to make perfect sense of the specification.

A compilable specification will be a huge leap forward for TPM. Clearly, this will reduce implementation differences between vendors. Research may be done to mathematically prove the correctness of parts of the specification now that it is written in a more formal language. Another major benefit is that a software emulator will always be available for TPM 2.0 without additional development. This will be useful for both virtualized platforms as well as resource-constrained platforms. For cost, space, or power-saving reasons it may not be feasible to field a discrete TPM chip. However, a compilable specification means that a platform designer can field a software TPM in a protected mode or virtualized mode of the main CPU.

Activation and Multiple Ownership

Without question, the biggest hindrance to widespread adoption of TPM 1.2 is that PC vendors ship machines with the TPM turned off by default. To activate the TPM, users must go into their firmware settings, find the appropriate menu, and choose to enable it. The off-by-default decision was made early on in response to outcries by the computer security community. At the time, opinions were expressed that TPM was simply a move by industry to enforce digital rights management and force people to use certain software packages. Some prominent researchers supported this position, which strengthened the argument against use of TPMs.[18] TCG therefore recommended that the TPM be off by default and that the user be required to manually activate it via firmware menus. Because most users never touch their firmware settings, this means that most TPMs have never been turned on.

Even if the TPM is activated, the platform firmware cannot do much with it. In TPM 1.2, once activated, the TPM starts out unowned, and the first person to establish an owner password via a specific command becomes the owner. The TCG decided early on that postboot applications would have to handle this process, cutting the firmware developers out of the picture because they must assume an unowned TPM. In the unowned state, a limited set of commands is available. Keys cannot be created or loaded, so the PCR state cannot be securely checked. With TPM 1.2, the firmware is unable to verify the boot state. Instead, the firmware is capable of hashing code and extending PCRs, and it is up to the operating system or applications to check them for Measured Boot.

TPM 2.0 fixes these problems by ensuring that the TPM is on by default and by creating a platform entity, ensuring that platform firmware has full access to TPM resources. Firmware will be able to create keys, encrypt data, and securely check PCR values. This means that simultaneously the platform user and platform firmware are "owners" of the TPM and can make use of its features. Firmware developers can use all the capabilities of the TPM to secure the preboot environment, in the same way that operating systems can use the TPM to protect their operation today.

CONCLUSION

The TPM 2.0 specification is a substantial improvement in sophistication over the TPM 1.2 design. It allows for algorithm agility, which should prevent early obsolescence. By additionally unifying and extending its authentication mechanisms via EA, it has simultaneously made the TPM more manageable, flexible, easier to program, and more useful. By making the specification compilable, it has become less prone to ambiguity and is itself a software reference implementation. By making TPMs activated by default, the vision of ubiquitous trusted computing may become reality.

REFERENCES

[1]Trusted Computing Group, *TPM Main Specification*, http://www.trustedcomputinggroup.org/resources/tpm_main_specification.

[2]Trusted Computing Group, *TPM Library Specification*, http://www.trustedcomputinggroup.org/resources/tpm_library_specification.

[3]Microsoft Corporation, *Windows Certification Program: Hardware Certification Taxonomy & Requirements - Devices*, http://download.microsoft.com/download/F/1/C/F1C62227-2D6A-4915-9EF2-0BDAE-0F323BD/windows8-1-hardware-cert-requirements-device.pdf (accessed 27 Aug 2013).

[4]National Security Agency, *HAP Technology Overview*, http://www.nsa.gov/ia/programs/h_a_p/overview/index.shtml (accessed 27 Aug 2013).

[5]Trusted Computing Group, *Byres Security Demonstrates Industrial Control System (SCADA)*, http://www.trustedcomputinggroup.org/resources/byres_security_demonstrates_industrial_control_system_scada (accessed 27 Aug 2013).

[6]Fink, R. A., Sherman, A. T., and Carback, R., "TPM Meets DRE: Reducing the Trust Base for Electronic Voting using Trusted Platform Modules," *IEEE Trans. Inf. Forensic. Secur.* **4**(4), 628–637 (2009).

[7]Trusted Computing Group, *Mobile Working Group*, http://www.trustedcomputinggroup.org/developers/mobile (accessed 27 Aug 2013).

[8]Trusted Computing Group, *Embedded Systems Working Group*, http://www.trustedcomputinggroup.org/developers/embedded_systems (accessed 27 Aug 2013).

[9]Trusted Computing Group, *Virtualized Platform Working Group*, http://www.trustedcomputinggroup.org/developers/virtualized_platform (accessed 27 Aug 2013).

[10]Broadcom Corporation, *BCM5882: Secure Applications Processor*, http://www.broadcom.com/products/Security/PC-Client-Security/BCM5882 (accessed 27 Aug 2013).

[11]Trusted Computing Group, *TCG Software Stack Specification*, http://www.trustedcomputinggroup.org/resources/tcg_software_stack_tss_specification.

[12]Microsoft Corporation, *Secure Boot Overview*, http://technet.microsoft.com/en-us/library/hh824987.aspx (accessed 27 Aug 2013).

[13]Microsoft Corporation, *BitLocker Drive Encryption Overview*, http://technet.microsoft.com/en-us/library/cc732774.aspx (accessed 27 Aug 2013).

[14]Kleinjung, T., Aoki, K., Franke, J., Lenstra, A. K., Thomé, E., et al., "Factorization of a 768-Bit RSA Modulus," in *Proc. 30th Annual Conf. on Advances in Cryptology (CRYPTO 2010)*, Santa Barbara, CA, pp. 333–350 (2010).

[15]Wang, X., Yin, Y. L., and Yu, H., "Finding Collisions in the Full SHA-1," in *Proc. 25th Annual International Cryptology Conf. (CRYPTO 2005)*, Santa Barbara, CA, pp. 17–36 (2005).

[16]Stevens, M., "New Collision Attacks on SHA-1 Based on Optimal Joint Local-Collision Analysis," in *Proc. 32nd Annual International Conf. on the Theory and Applications of Cryptographic Techniques (EUROCRYPT 2013)*, Athens, Greece, pp. 245–261 (2013).

[17]Challener, D. C., and Osborn, J. D., *The Trusted Cryptography Module: An Analysis of China's Homegrown Trusted Computing*, AISD Report AI-09-124, JHU/APL, Laurel, MD (Aug 2009).

[18]Anderson, R., *'Trusted Computing' Frequently Asked Questions*, http://www.cl.cam.ac.uk/~rja14/tcpa-faq.html (accessed 27 Aug 2013).

The Authors

Justin D. "Ozzie" Osborn is a software reverse-engineer in APL's Asymmetric Operations Department and a member of the Senior Professional Staff. He has worked on several projects developing TPM software and performing vulnerability analysis of TPM solutions. **David C. Challener** is an applied mathematician and Senior Professional Staff member in the Asymmetric Operations Department. While at IBM, Dave worked on the design of the IBM PC embedded security subsystem and the first TPM chip. Dr. Challener coauthored the book *A Practical Guide to Trusted Computing* on programming with the TSS. He has been active in the TCG, serving on the Board of Directors, chairing the TSS Workgroup, and participating in the Technical Committee, Virtualization Workgroup, and Storage Workgroup. He currently cochairs the TPM Working Group. For further information on the work reported here, contact Justin Osborn. His e-mail address is justin.osborn@jhuapl.edu.