Trusted Platform Modules: Building a Trusted Software Stack and Remote Attestation

Hardeep Uppal
University of Washington
hardeepu@cs.washington.edu

Dane Brandon
University of Washington
dbrandon@cs.washington.edu

ABSTRACT
In a networked environment where computers are required to collectively work together, it is frequently the case that a single compromised machine can compromise the functionality of the entire system. In such an environment, standard protocols such as password authentication may not provide the security level we desire in that a computer attempting to gain access to the network cannot prove that it has not been compromised.

The Trusted Computing Group (TCG) provides a specification for using a Trusted Platform Module (TPM) to build a trusted software stack which can be verified by a remote machine. As a proof-of-concept, we build a rudimentary trusted software stack, verify that its validation mechanisms work and how long they take, and measure how much time is added to the boot process by the TPM's added code.

1. INTRODUCTION

The End to the Middle project here at the University of Washington (UW) attempts to do away with the concept of a static gateway server or node in a local area network by distributing the gateway's duties across the participating nodes in the network. This will ideally be accomplished by running dedicated software such as an implementation of Paxos, for shared key/value consensus within the group, a NAT filter for internal packet routing, and use of OpenFlow switches to permit or deny group members in the network control over packet flow. Such software would run in parallel or on top of an OS which would let the software handle the distributed duties of the gateway server. [1]

In a distributed system where machines are connected to the internet and are executing client software, there is a strong possibility that machines will be attacked to steal information from the system by downloading and executing malicious programs. Such attacks will cause a change in the software stack running on a machine. The changes caused by the malicious program need to be detected in order to preserve the security of the system.

The question arises: When a new node attempts to join a group of trusted computers within a LAN (or a known node reboots), how can we be sure that they are who they say they are and that they are running the correct software to ensure correct behavior while interacting with the group? One answer is to use TPMs to measure the integrity of the system. TPMs are designed to securely measure the software stack loaded at boot time on a machine which can be attested and signed to be in a particular state at the time of joining the group. Malicious programs, such as malware or Trojans will be detected by the changes made to measurement of the system.

2. TRUSTED COMPUTING

Trusted computing as specified by the TCG provides a set of attestation functionality that can be used to keep track of all software measurements inside the TPM. Thus a machine can use the TPM to measure its software components at boot, sign the measurements, and send it them to a second party which can verify that the software configuration is valid.

2.1 Trusted Platform Module

The TPM is a hardware chip soldered onto the motherboard of a computer and is shipped with almost all business model (Lenovo ThinkPad, Dell Latitude, etc…) laptops today. Figure 1 shows the hardware components of the TPM chip. The TPM consists of a cryptographic processor, persistent memory and versatile memory. The TPM is capable of integrity measurement, secure storage of measurements, authentication, key creation and storage, key migration, encryption/decryption of data and keys, attestation and signing. Much of the functionality of the TPM is omitted from this paper as it is out of the scope of this project. This paper focuses on the functionality needed to build a trusted software stack.

2.1.1 TPM Memory

The versatile memory inside the TPM, which is volatile, is used to store measurement data and private keys. The measurement of a software stack is stored in registers called Platform Configuration Registers (PCRs). The current version of TPMs (version 1.2) consists of 24 PCRs. The values inside the PCRs can only be changed by an extend operation. To store new values in a PCR, the extend operation is used which appends a new value to the current value in a PCR, computes a hash, using Secure Hash Algorithm 1 (SHA-1), of the concatenated values and replaces the current value in the PCR with the output of the hash operation [2].

$$PCR[i] = SHA\text{-}1(PCR[i] \mid \text{new value})$$

So each measurement of a software stack component is a hash of an array of bytes, which can be either code about to be loaded or a hash of it, appended to the previous PCR value and stored back the PCR. The output of a SHA-1 hash is a 20 byte value. Its small size helps to store all measurements of the software stack with a very small memory footprint. A slight change in the measurement bits will result in a completely different hash value. A change in even one bit in the input for SHA-1 will result in approximately half of the output bits changing. This helps detect even the smallest change in the code being loaded at boot.

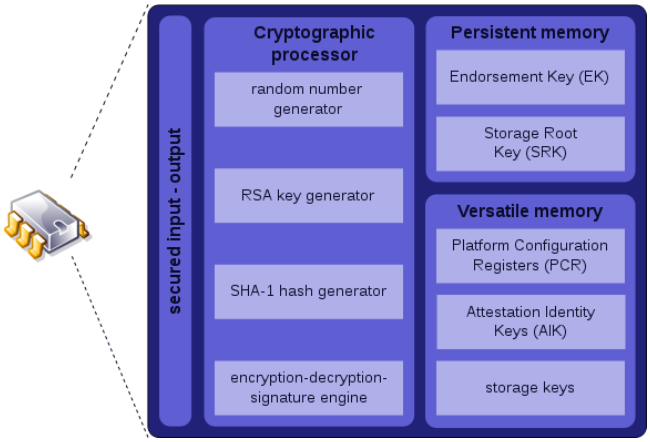

Figure 1: Trusted Platform Module chip. [3]

2.1.2 TPM Cryptographic Processor

The cryptographic processor in the TPM has an RSA key generator used to create RSA key pairs. The encryption-decryption engine provides the necessary functions for encryption/decryption of data on disk or signing PCR values. Finally it contains a SHA-1 hash generator used for hashing values that are stored in the PCR.

2.1.3 TPM Persistent Memory

The persistent memory in the TPM holds two keys: the Endorsement Key (EK) and the Storage Root Key (SRK). The EK is a 2048-bit RSA private key generated by the manufacturer of the TPM and hard encoded onto the chip or generated the first time ownership is taken of the TPM. The EK is never revealed to the user and stays in the TPM during its lifetime. [2][4] To preserve the privacy of a user, an EK is not allowed to encrypt any data and can only decrypt data. Similarly, the SRK is also a 2048-bit private RSA key but it is generated at time of taking ownership of the TPM by the user and used thereafter or until the TPM is cleared. The SRK like EK never leaves the TPM and is never revealed to the user.

TPM have limited memory and can only store a few number of keys in memory. When a key is created inside the TPM it is encrypted with the SRK's public key and stored on disk. This enables the ability to create an arbitrary number of keys and not be restricted to create keys because of the limitation in TPM's storage space. Since the key stored on disk is encrypted with the SRK's public key, the only way to use a key is to decrypt it with the SRK inside the TPM.

2.2 Trusted Software Stack

2.2.1 Trusted Boot

Trusted boot is a security property where software components involved in a bootstrap process are measured before being loaded and executed, i.e. when a computer is turned on, each layer of software involved in the boot process measures the subsequent software component being loaded. [5] Validation of the system is done at the application level after the hashes of the components are appended inside the TPM's PCRs. The boot process continues regardless if one of the software components being loaded does not hash to the correct value. In fact, the values are not checked at all during boot. They are simply stored. Therefore the user of the system must check the hashes within software to determine if unwanted modifications have been made.

It is important to differentiate trusted boot from secure boot. Secure boot is mechanically the same but PCR values are checked at each stage and if ever the expected value does not match the actual value, the boot process is interrupted.

2.2.2 Chain of Trust

Building a chain of trust requires a foundation that can be built upon. We need a bottom level immutable block on which we can build the additional layers which make up the trusted software stack. The boot block code in the BIOS is considered trustworthy and is the Core Root of Trust for Measurement (CRTM). Over the lifetime of the system, the boot block code in the BIOS remains unchanged (and cannot be changed) and therefore can reliably measure other components.

The CRTM will be the first instance of the boot process and it measures the BIOS and appends the value into a PCR before passing control. The BIOS then measures parts of hardware including ROMs and the boot loader (located in the master boot record) and appends the values inside specific PCRs. The control is then handed to the boot loader which measures the OS kernel and passes control to the operating system. After the OS is loaded, a third party can check the PCR values inside the TPM to verify that the state of the system has not been compromised and the system is configured as expected. Any changes made to the system will result in different PCR values and the user can then decide to trust the system or not.

It should be noted that the values in the PCRs are only representative of the state of the machine at boot time. If malicious software is loaded or changes are made to the system thereafter, the changes will not be reflected in the PCRs until the machine is rebooted.

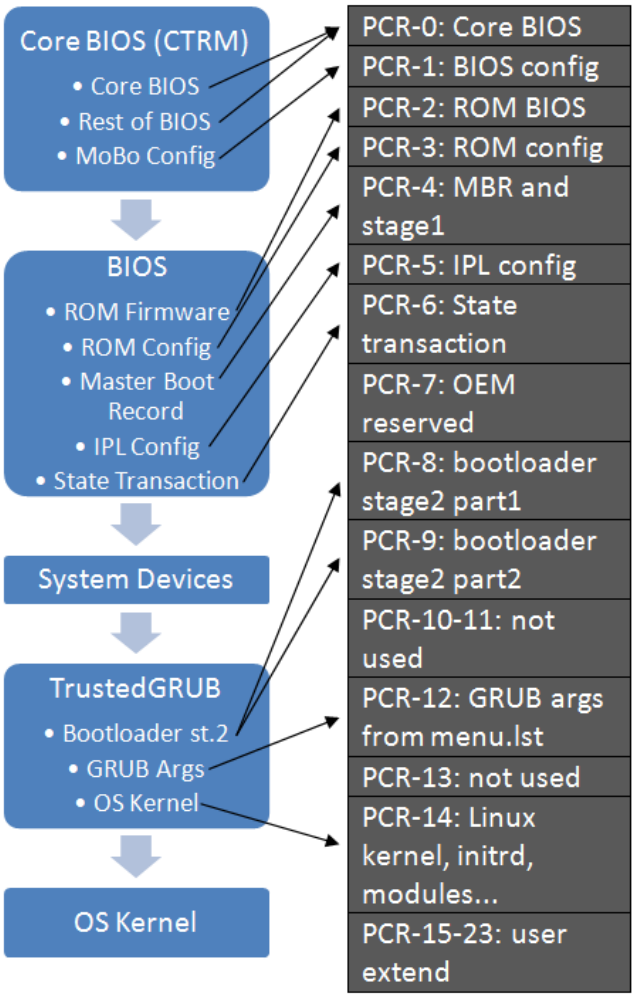

Figure 2: Execution and Measurement diagram for Chain of Trust

Figure 2. shows the boot sequence and measurements taken. The blue column shows the major software components as they are loaded. The bullets in each component show what the component measures once it has been loaded and starts to run. The arrows point to the specific locations in the PCRs where the measurements are stored. With the exception of the CTRM, which measures itself in addition to the rest of the BIOS, each software component is measuring the next before it is loaded. Once the OS Kernel is loaded all PCR values have been appended and the system integrity can be verified or attested to.

2.2.3 TrustedGRUB

With no additional software other than a typical Linux installation (we used Ubuntu 9.04) the chain of trust measurement stops after the BIOS measures the boot loader stage 1. In order to measure stage 2 of the boot loader and the operating system, which prepares the system to load the operating system kernel, we need a modified version of GRUB (Linux boot loader) which has additional code to measure stage 2 and the OS kernel.

TrustedGRUB extends GRUB-0.97 and uses functions provided by the Trusted Computing Group at IBM to continue the chain of trust. TrustedGRUB continues the integrity measurements by extending the chain of trust to measure stage 2 of the boot loader and the operating system kernel. It also measures the menu.lst file loaded by stage 2 which contains configuration options (loading different operating systems and location/version of the kernel) for GRUB. To completely measure the boot process, TrustedGRUB measures all inputs used by stage 2 of the boot loader which includes the kernel, multi-boot modules and configuration information. Hence TrustedGRUB completes the building of the chain of trust upward into a booted state for attestation. TrustedGRUB does not determine the integrity of a system but completes the measurement chain after which another process can determine the integrity of the system.

2.2.4 TrouSerS

The TCG Software Stack (TSS) provides an API for applications and the operating system to use the functionality provided by the TPM to create keys, sign PCR values, seal, unseal etc. TrouSerS is an open-source implementation of TSS which provides a set of functions written in C code to manage and configure the TPM. Some of the basic functionality to administer the TPM is implemented in a package called tpm-tools. Tpm-tools provides commands to take ownership of the TPM, view the public EK, view the version of the TPM, seal/unseal data in TPM, etc… We used Tpm-tools to generate an EK while taking ownership of the TPM for the first time and view PCRs after boot. Tpm-tools also provides commands to create a Secure Root Key (SRK) which is used for secure storage.

2.3 Attestation

Once the machine is booted with PCR values in the TPM, the system can be verified by a second party. In general the process of validating state on a remote machine is subject to typical security threats such as a replay-attack and a standard approach to securely attesting to a machine's state is taken by use of a privacy Certificate Authority (CA). In this section we outline a protocol for attestation to a second party by using a third party privacy CA.

2.3.1 Delivering the PCR Values for Verification

To prove our state to a second party, we need to send them some sort of representation of our PCR values which they can check against some expected value. Unfortunately this is very easy to spoof. All an attacker needs to do is intercept the valid values once, regardless whether they are encrypted, and send them whenever they wish to fool the second party into believing they are a computer running the trusted stack. This can be prevented by use of a random number called a *nonce* and the TPM's quote function.

When a computer wants to prove itself to a second party, the second party challenges it by sending a nonce. The Tspi_TPM_Function of the TPM then appends the PCR

values together along with the nonce, performs a SHA-1 hash, and signs the resulting hash with a private key. The second party can now decrypt the results with a public key from the TPM, and compare the value to its own hash of the known PCR values with the nonce it created for the challenge. By adding the nonce to the hash, the results are known to be "fresh", i.e. there cannot be a replay attack.

2.3.2 Attestation Identity Keys (AIK) and the Privacy CA

To prevent a replay-attack there must be a mechanism in place to prove that the first party is a genuine TPM to the second party. Ideally a TPM equipped computer would come with the EK hardcoded into the chip and a certificate from the manufacturer which can be used to prove that your signed values are from a genuine TPM. Currently not all TPM equipped machines come with a hardcoded EK and corresponding certificate. When we took ownership of our machine, we were prompted to create a permanent EK immediately. Functionally this EK is the same as a hardcoded EK from the manufacturer but without the accompanying certificate, we need a way to prove that our TPM is a TPM.

Matters are complicated further by the fact that by repeatedly using the same EK to sign the PCRs, other parties may determine that values are coming from the same computer. So to protect the user's privacy, the TCG has disabled the use of the EK for signing the PCRs. Instead AIKs can be created, used, and thrown out when they are considered too old to use.

AIKs can be created by the TPM and securely stored outside on disk, then reloaded into the volatile TPM memory by the function call Tspi_TPM_LoadKey(). To prove that an AIK comes from a valid TPM, the public half of the AIK is sent to a privacy CA with the public half of the EK certificate. The privacy CA should have a list of all valid TPM manufacturers' public keys and be able to verify that the request comes from a valid TPM by the public EK certificate. The privacy CA then puts the AIK in a certificate, signs it, and encrypts it again with the public EK. By encrypting the certificate with the public EK, the only party that can unwrap it is the computer with the TPM we are trying to associate with the AIK.

Now we have a signed certificate with which a third trusted party attests that the AIK is from a valid TPM to the second party. The certificate can now be sent over to the second party with the PCRs which will be signed with the private half of the AIK by the TPM. The second party has everything it needs to verify that the machine attempting to gain access to resources was in a trusted state after boot.

The problem of having to create our own EK on initialization of the TPM coupled with the fact that there is currently no privacy CA to verify TPMs makes following this protocol impossible for now. For the purposes of this project we created a fake certificate. Even without a known privacy CA, it would be possible for an organization to run their own privacy CA which they could use to register their own computers and EKs.

3. PERFORMANCE AND MEASUREMENTS

In this section we test the general performance of the TPM to show that it is reliable and that its performance is not prohibitive to use in a practical environment.

3.1 Verifying That Registers Change

Using TPM tools once the operating system has booted, we are able to do some basic functions with the TPM from the command line. One important thing to observe is that the hashed values in the PCRs actually change when the code being loaded changes. TrustedGRUB gives us options of which kernel and modules to load. By selecting two different kernels to load from the boot-menu, we have verified that the hashes change. Figure 3 illustrates the PCR digest from two different boots with different kernels being loaded. Changes are highlighted in bold.

As expected, PCRs 12 and 14 are different. PCR12 is the hash of the command line arguments selected from the boot menu and PCR 14 is the hash of the kernel code being loaded.

Similarly we observed that adding a three line script to a directory of startup programs changed PCR-14 thus validating that small changes result in different hashes as well as large changes like loading a different kernel.

Kernel: 2.6.28-16-generic
PCR-00: F6 44 C8 1F FE D3 62 65 BE EE 8D 72 BC 5A 45 E3 B1 B9 CF F2
PCR-01: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-02: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-03: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-04: 42 96 8A 3A 55 E1 67 1B D5 09 B3 9B C7 52 34 6D 70 9A FE CB
PCR-05: 1F 9A E5 C6 58 5A 57 C6 3B 93 FB FC A9 C0 7A 0E E4 F1 10 A6
PCR-06: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-07: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-08: 94 C3 C8 5F 09 EC 25 C2 C6 05 BA F7 2E CB 49 F9 7E E6 C0 FC
PCR-09: 93 CD A5 03 69 E2 41 BD BF 8B 22 D5 8B 07 21 6E F0 BF 7B 44
PCR-10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-11: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-12: 17 AE 0B D7 E1 B6 8C FE E0 4C 77 E1 B6 C1 ED 2C D3 1E A9 6B
PCR-13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-14: BC 09 40 C2 25 76 2C 23 5D 0F 58 26 6C 10 6D BD 9C 4B C4 D5
PCR-15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Kernel: 2.6.28-11-generic
PCR-00: F6 44 C8 1F FE D3 62 65 BE EE 8D 72 BC 5A 45 E3 B1 B9 CF F2
PCR-01: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-02: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-03: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-04: 42 96 8A 3A 55 E1 67 1B D5 09 B3 9B C7 52 34 6D 70 9A FE CB
PCR-05: 1F 9A E5 C6 58 5A 57 C6 3B 93 FB FC A9 C0 7A 0E E4 F1 10 A6
PCR-06: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-07: A8 9F B8 F8 8C AA 95 90 E6 12 9B 63 3B 14 4A 68 51 44 90 D5
PCR-08: 94 C3 C8 5F 09 EC 25 C2 C6 05 BA F7 2E CB 49 F9 7E E6 C0 FC
PCR-09: 93 CD A5 03 69 E2 41 BD BF 8B 22 D5 8B 07 21 6E F0 BF 7B 44
PCR-10: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-11: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-12: BF F2 05 B6 B0 41 4A B8 5C 76 85 F8 34 3D 0D D9 A0 4C 22 2C
PCR-13: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
PCR-14: 23 88 21 8F 0E A8 12 C7 0D EB A1 C6 1C A1 CC 4A 78 8B 25 69
PCR-15: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Figure 3. PCR digests of two different kernels

3.2 TPM Function Times

For a trusted software stack to be used in the real world, the TPM must function fast enough to be practical. We chose four of the TPM's functions which are integral to the process of building the chain of trust and attestation. Figure 4 shows the mean time and standard deviation over thirty samples each with the exception of create_key which was sampled ten times.

	PCR Extend	Create Key	Sign PCRs	Load Key
Mean	0.02947	34.3019	3.07089	2.11058
Std. Dev.	0.00443	8.22022	0.26267	0.25812

Figure 4. Mean execution time and standard deviation of common TPM function. All times in seconds.

During the boot process, a series of twelve PCR extends occur. With an average extend time of .03s the total time to sequentially extend all value to the PCRs should be roughly .4s. In the context of a startup which takes over 50s this is a negligible cost.

The attestation process requires loading a key into the TPM and signing the PCRs. These functions alone total just over five seconds. While this is not an insignificant amount of time in terms of processor cycles, it is in no way prohibitive for the one time necessary to prove system state when joining a network.

Clearly creating keys with the TPM is an expensive operation with a mean creation time of 34.3s. We attribute this to the need to generate large random prime integers of similar bit-length. Since testing for primeness is a compute intensive algorithm and the TPM has limited processing power and memory, we expect that this would be a slow operation. Luckily keys do not need to be created for real-time operations and when needed AIKs could be created at boot in parallel with other system boot tasks which would entirely mask the required time.

4. FUTURE WORK

One of the important parts of creating AIKs is to have a Privacy CA certify that an AIK is created by a valid TPM. To accomplish this we would need to create our own privacy CA which will have a database of public EKs for various TPMs on different machines. The privacy CA will create certificates that will have a specific format containing the AIK. This certificate will be presented to a machine trying to validate other machines in the network.

After creating a privacy CA, we will be able to do complete benchmark of the attestation process where a machine will present the AIK's public key and PCR values signed by the AIK's private key to a trusted node, which will then validated the hash and allow a machine to join a group of nodes. To maintain freshness of PCR values, the challenger can present the machine with a nonce which will be appended and signed with the PCR values. The challenger can check the nonce to determine if the PCR values are fresh.

5. CONCLUSION

We have shown that it is within reach to build a trusted software stack with commodity hardware and open-source software which can be used to verify the state of the software stack at time of boot. The overall time-cost of using these tools while not negligible, is reasonable in the context they are intended for. We predict that the attestation process will not be prohibitively long to using TPMs and that with further development the use of TPMs shows great promise.

ACKNOWLEDGMENTS

We thank Colin Dixon, Prof. Tom Anderson, and Prof. Arvind Krishnamurthy for their ongoing support of our research.

REFERENCES

1. Dixon, C., Uppal, H., Brandon, D., Anderson, T., Krishnamurthy, A. An End to the Middle. University of Washington.
2. Challener, D., Yoder, K., Catherman, R., Safford, D., Van Doorn, L. *A Practical Guide to Trusted Computing*. IBM Press, Indianapolis, IN, USA, 2008.
3. Wikipedia: Trusted Platform Module. http://en.wikipedia.org/wiki/Trusted_Platform_Module
4. Trusted Computing Group. http://www.trustedcomputinggroup.org/developers/glossary
5. Selhorst, M., Stueble, C., Teerkorn, F. TSS Study Introduction and Analysis of the Open Source TCG Software Stack TrouSerS and Tools in its Environment. *Study on behalf of the German Federal Office for Information Security (BSI);* Bonn, May 2008.