CHAPTER 12

# Platform Configuration Registers

Platform Configuration Registers (PCRs) are one of the essential features of a TPM. Their prime use case is to provide a method to cryptographically record (measure) software state: both the software running on a platform and configuration data used by that software. The PCR update calculation, called an *extend*, is a one-way hash so that measurements can't be removed. These PCRs can then be read to report their state. They can also be signed to return a more secure report, called an *attestation* (or *quote*). PCRs can also be used in an extended authorization policy to restrict the use of other objects.

The TPM never passes judgment on the measurements. Internally, it doesn't know which measurements are good or bad, or more or less secure or trusted. At the time of measurement, TPM PCRs just record values. Security or trust comes later, when an application uses PCR values in an authorization policy, or a remote party asks for a signed attestation (quote) of the values and judges their trustworthiness.

New for TPM 2.0, TPMs no longer hard-code the SHA-1 algorithm for PCRs. The algorithm can be changed. Some implementations include banks of PCRs, with each bank implementing a different algorithm.

A TPM implements a number of PCRs: for example, 24 for a PC TPM. The PCRs are allocated by convention to the various software layers, from early boot code to the operating system and applications. They're also allocated for both the software to be run (often the even-numbered PCRs) and the configuration files that customize the boot process (typically the odd-numbered PCRs.)

## PCR Value

The primary use case for a PCR is to represent the platform software state, the history of the critical software (and configurations) that have run on the platform until the present. The TPM initializes all PCRs at power on, typically to either all zeroes or all ones, as specified by the TPM platform specification. The caller can't directly write a PCR value. Rather, a PCR value is changed through what the TPM calls an *extend* operation, as described in Chapter 2. Cryptographically, it is as follows:

```
PCR new value = Digest of (PCR old value || data to extend)
```

In words, it takes the old PCR value and concatenates some data to be extended. The data to be extended is almost always a digest, although the TPM can't enforce this. The TPM digests the result of the concatenation and stores the resulting digest as the new PCR value.

After reboot, a platform begins with trusted software called the *core root of trust measurement (CRTM)*. The CRTM measures (calculate a digest of) the next software to be run and extends that digest into an even PCR. It then extends that software's configuration data into an odd PCR. This software, perhaps a BIOS, in turn measures and extends the next software, perhaps a master boot record. The measurement chain continues through the early OS kernel code and perhaps further. Security-critical configuration files are also measured.

The net result is that the PCR value represents the history of all measurements extended into it. Because of the one-way nature of a secure digest, there is no way to undo a measurement (to extend the PCR back to a desired value).

As a typical example, the PC Client specification allocates the PCRs as shown in Table 12-1.

***Table 12-1.*** *Example PCR Allocation*

| PCR Number | Allocation |
|---|---|
| 0 | BIOS |
| 1 | BIOS configuration |
| 2 | Option ROMs |
| 3 | Option ROM configuration |
| 4 | MBR (master boot record) |
| 5 | MBR configuration |
| 6 | State transitions and wake events |
| 7 | Platform manufacturer specific measurements |
| 8–15 | Static operating system |
| 16 | Debug |
| 23 | Application support |

The security of this process depends on the security of the CRTM. The CRTM, being the first software to run, can't be measured or validated. It's a *root of trust*. The platform manufacturer can protect the CRTM from attack by making it immutable, putting it in ROM, or otherwise preventing software updates. Because this precludes bug fixes, an alternate method is to use signed code and have the current CRTM validate the signature before updating itself.

The Linux open source Integrity Measurement Architecture (IMA) integrates boot-time measurements into the kernel. An IMA policy determines which software elements are measured. These typically include libraries and executables run under root privilege during boot, as well as Linux configuration files that determine the boot path. It doesn't typically measure user-level applications.

## Number of PCRs

In practice, a TPM contains multiple PCRs. The PC Client platform requires 24 PCRs, and this minimum is expected to be the actual number in PCs. Automotive TPMs may have many more. The platform TPM specification specifies the PCR attributes, and a platform software specification standardizes what measurements go into which PCRs.

The platform specifications may set aside several PCRs for user-level applications. And one PCR (16), the debug PCR, is reserved for testing software. As such, it's resettable without a power cycle.

As described in Chapter 11, TPM 2.0 provides for user-defined NV extend indexes, which are essentially PCRs. They have additional flexibility in that the hash algorithm, password, and policy can be individually set for each index. The metadata (mainly algorithm and authorization) is nonvolatile, whereas the actual data values are likely to be volatile through the use of a hybrid index.

The remainder of this chapter is limited to architecturally defined PCRs.

## PCR Commands

PCR commands include the following:

- `TPM2_PCR_Extend`: Likely to be the most-used PCR command. Extends a digest into a PCR.
- `TPM2_PCR_Event`: Permits the TPM to do the digest and then extend the digest in one operation. The message is limited to 1,024 bytes.
- `TPM_PCR_Read`: Reads a PCR, which is useful when validating an event log as described later.
- `TPM2_PCR_Reset`: Resets a PCR, which is useful for some application-defined PCRs that permit this. Most PCRs can't be reset.
- `TPM_PCR_Allocate`: Assigns digest algorithms to PCRs. This is likely to be done once at most, if the default algorithm is to be changed.
- `TPM2_PCR_SetAuthPolicy`: Assigns an authorization policy to a PCR group. It isn't required in the PC Client.
- `TPM2_PCR_SetAuthValue`: Assigns an authorization value to a PCR group. It isn't required in the PC Client.

## PCRs for Authorization

Authorization is a common use for PCRs. An entity can have a policy that prevents it from being used unless specific PCRs have specific values. Chapter 14 explains this in detail. The policy can specify a subset of PCRs and a value for each. Unless the PCRs are in this state, the policy is not satisfied and the entity can't be accessed.

### USE CASE: SEALING A HARD DISK ENCRYPTION KEY TO PLATFORM STATE

Full-disk encryption applications are far more secure if a TPM protects the encryption key than if it's stored on the same disk, protected only by a password. First, the TPM hardware has anti-hammering protection (see Chapter 8 for a detailed description of TPM dictionary attack protection), making a brute-force attack on the password impractical. A key protected only by software is far more vulnerable to a weak password. Second, a software key stored on disk is far easier to steal. Take the disk (or a backup of the disk), and you get the key. When a TPM holds the key, the entire platform, or at least the disk and the motherboard, must be stolen.

Sealing permits the key to be protected not only by a password but by a policy. A typical policy locks the key to PCR values (the software state) current at the time of sealing. This assumes that the state at first boot isn't compromised. Any preinstalled malware present at first boot would be measured into the PCRs, and thus the key would be sealed to a compromised software state. A less trusting enterprise might have a standard disk image and seal to PCRs representing that image. These PCR values would be precalculated on a presumably more trusted platform. An even more sophisticated enterprise would use TPM2_PolicyAuthorize, and provide several tickets authorizing a set of trusted PCR values. See Chapter 14 for a detailed description of policy authorize and its application to solve the PCR brittleness problem.

Although a password could also protect the key, there is a security gain even without a TPM key password. An attacker could boot the platform without supplying a TPM key password but could not log in without the OS username and password. The OS security protects the data. The attacker could boot an alternative OS, say from a live DVD or USB stick rather that from the hard drive, to bypass the OS login security. However, this different boot configuration and software would change the PCR values. Because these new PCRs would not match the sealed values, the TPM would not release the decryption key, and the hard drive could not be decrypted.

These are the steps to seal:

1. Construct the policy, a `TPM2_PolicyPCR`, specifying the PCR values that must be present at the time of the unseal operation.
2. Use either of the following (similar to TPM 1.2 seal)
    - `TPM2_GetRandom()` to create the symmetric key external to the TPM
    - `TPM2_Create()`, specifying the symmetric key and the policy to create the sealed object
    - or (new TPM 2.0 alternative)
    - `TPM2_Create()`, specifying just the policy, to let the TPM create the symmetric key used in the sealed data object

Use the following to unseal:

- `TPM2_Load()` to load the object
- `TPM2_PolicyPCR()` to satisfy the sealed object policy
- `TPM2_Unseal()` to return the symmetric key

## USE CASE: VPN KEYS

Similar to the previous use case, a VPN private key can be locked to PCRs. The TPM permits the use of the VPN to connect to the enterprise intranet only if the software is in an approved state.

## USE CASE: SECURELY PASSING A PASSWORD FROM THE OS PRESENT TO OS ABSENT ENVIRONMENT

A platform administrator (for example, the IT administrator) wishes to grant the end user permission to change a BIOS setting, perhaps changing the boot order. The BIOS needs the administrator password. The administrator must pass the privileged-access password to the BIOS but doesn't want to reveal the password to the end user.

The administrator seals the password to the PCR state present while the BIOS is running (after a reboot). The admin supplies this sealed password to the user at the OS level. The user can't unseal the password while the OS is running, but the BIOS can unseal and use it after a reboot.

These are the steps at the OS level:

1. Construct a policy, a `TPM2_PolicyPCR` specifying that `PCR[2]` is all zeroes. This PCR will only have this value very early in the boot cycle, when the CRTM passes control to the first part of the BIOS.
2. Use `TPM2_Create()`, specifying the password and the policy to create the sealed object. The password is supplied via an encrypted session (see Chapter 17), essentially a secure tunnel into the TPM.

These are the steps at the BIOS level:

3. Use `TPM2_Load()` to load the object.
4. Use `TPM2_PolicyPCR()` to satisfy the sealed object policy.
5. Use `TPM2_Unseal()` to return the secret.

A typical use of PCRs for authorization would be to tie the use of an entity to the platform software state, but other uses are possible. For example, a password can be extended into a PCR, thus unlocking access. When access is no longer desired, the PCR can be reset (if permitted) or just extended with some other value.

## PCRs for Attestation

Attestation is a more advanced use case for PCRs. In a non-TPM platform, remote software can't usually determine a platform's software state. If the state is reported through strictly software means, compromised software can simply lie to the remote party.

A TPM attestation offers cryptographic proof of software state. Recall that a measurement can't be undone. A PCR can't be rolled back to a previous value. The attestation is a TPM quote: a number of PCR are hashed, and that hash is signed by a TPM key. If the remote party can validate that the signing key came from an authentic TPM, it can be assured that the PCR digest report has not been altered.

We say this is a more advanced use because it's insufficient to simply validate the signature and the key's certificate. The party has to next validate that the digest of the PCR matches the reported PCR values. This is straightforward.

Next, the party has to read an event log—a log of all software and other states measured, with their hashes—and validate that the event log matches the PCR values. This is still not too hard; it just involves some math.

The TCG Infrastructure Work Group (IWG) and PC Client Work Group specify the details of the event log format. The Platform Trust Services (PTS) specification from the IWG specifies how to report measurements through Trusted Network Connect (TNC). Standardizing the logging and reporting formats permits standard software to parse and validate the log against the attestation (quote).

The Integrity Measurement Architecture (IMA) for Linux specifies an event-log file format. Typical entries looks like this and includes a PCR index, a template hash, a template name, the file hash, and a hint (untrusted) as to the file name:

```
10 88da93c09647269545a6471d86baea9e2fa9603f ima
a218e393729e8ae866f9d377da08ef16e97beab8 /usr/lib/systemd/systemd

10 e8e39d9cb0db6842028a1cab18b838d3e89d0209 ima
d9decd04bf4932026a4687b642f2fb871a9dc776 /usr/lib64/ld2.16.so

10 babcdc3f576c949591cc4a30e92a19317dc4b65a ima
028afcc7efdc253bb69cb82bc5dbbc2b1da2652c /etc/ld.so.cache

10 68549deba6003eab25d4befa2075b18a028bc9a1 ima
df2ad0965c21853874a23189f5cd76f015e348f4 /usr/lib64/libselinux.so.1
```

The hardest part comes next. Through the TPM signed attestation quote, the party knows the platform software state. It now has to decide whether that software state is secure. The party has to match the measurement hashes against a whitelist, potentially requiring cooperation from third-party software providers.

This is the essence of the Trusted Computing concept. PCRs provide a means to trust that a list of software modules indeed reflects the software state of a platform. It doesn't make any value judgments as to whether that software is secure.

## USE CASE: QUOTE

A networking device wants to decide whether to let a client platform connect to a network. It wants to know whether the platform is running fully patched software. The device quotes the TPM PCR and validates the result against a whitelist of patched software modules. If the platform is current, it's permitted on the network. If not, it's routed to a patch server but not otherwise permitted network access.

The StrongSwan open source VPN solution can use the TCG TNC standard, combining TPM quotes and a policy to gate access to a VPN.[1]

[1] http://wiki.strongswan.org/projects/strongswan/wiki/TrustedNetworkConnect.

The Kaspersky antivirus software end user license agreement (EULA) permits the software to report on the files processed, versions of the software, and more. The license permits use of the TPM, if present, to authenticate the report.[2]

## PCR Quote in Detail

It's interesting to examine the quote data in detail. Through this data, the reader can understand the security properties of the quote. A quote's structure—the structure that is hashed and signed—contains these fields:

- *Magic number* `TPM_GENERATED`: Prevents an attacker from signing arbitrary data with a restricted signing key and claiming later that it was a TPM quote. See Chapter 10 for the interaction between restricted signing keys and `TPM_GENERATED`.
- *Qualified name of the signing key*: A key could appear strong but be protected by an ancestor with a weaker algorithm. The qualified name represents the entire ancestry of the key.
- *Extra data provided by the caller*: This data is typically an anti-replay nonce, which is proof that the quote is current.
- *TPM firmware version*: Included so that the verifier can decide if it trusts a particular TPM code version.
- *TPM clock state*: The variable `resetCount` is of particular importance for the next use case. For privacy, the clock information is obfuscated when signing with a key outside the endorsement hierarchy.[3] This isn't an issue, because the attester only wants to detect if `resetCount` changes, not read its actual value.
- The type of attestation structure (a quote, in this case).
- The selection of PCRs included in the quote.
- A digest of those selected PCRs.

[2]http://support.kaspersky.com/8752.

[3]For a detailed explanation of this privacy issue, see the "Other Privacy Considerations" section of Chapter 9.

**USE CASE: DETECTING A REBOOT BETWEEN TRANSACTIONS**

A platform is performing financial transactions. A monitoring device performs a quote every 15 minutes to detect changes to the platform software state. However, an attacker sneaks in between quotes, reboots into compromised software, performs an unauthorized transaction, and then reboots the platform back to the trusted state. The next quote will show the same trusted PCR values. However, the `resetCount` change tells the monitoring software that two unexpected reboots occurred.

## PCR Attributes

Each PCR comes with several attributes. The attributes are defined in the TPM library specification, but which PCR indexes have which attributes is left to the platform-specific specification. Generally, most PCR indexes are assigned by convention to specific software, but a few are unassigned and open for use by applications.

The PCR `Reset` attribute indicates whether the PCR can be reset using the `TPM2_PCR_Reset` command. Typically, the reset value is all zeroes. Most PCRs are not resettable, because this would permit compromised software to set the PCR value to a known good state. Some PCRs are resettable only in a certain locality, corresponding to dynamic root of trust measurement (DRTM) sequences.

The PCR `Extend` attribute indicates whether the PCR can be extended using the `TPM2_PCR_Extend` or `TPM2_PCR_Event` command. Obviously, a PCR that couldn't be extended would be useless, but some can be extended only in some localities.

The PCR `Reset` attribute via DRTM indicates whether a PCR can be extended through writes directly to the TPM interface, as opposed to the normal TPM command format. These are both platform specific and linked to the particular TPM hardware interface. This attribute typically varies by locality.

All PCRs are reset at reboot when `TPM2_Startup` is issued with the `CLEAR` parameter. Most are typically reset to all zeroes, but some can have other values, such as all ones or a value related to the locality at which the startup command was issued.

The `No Increment` attribute is tied to the `TPM2_PolicyPCR` command. A policy tied to a PCR is an immediate assertion. The PCR values at the time of the `TPM2_PolicyPCR` command are extended into the policy session hash. However, a PCR value could change after the immediate assertion, which should normally invalidate the policy session. This invalidation is implemented though a counter that is normally incremented whenever a PCR is changed. The policy session records the value during `TPM2_PolicyPCR` and then checks it when the session is used. If the count values aren't equal, the TPM knows that a PCR changed, and the policy session use fails.

Note the word *normally* in the previous paragraph. The TPM specification provides the `No Increment` attribute. PCRs with this attribute, when changed, don't increment the counter and thus don't invalidate policy sessions in use. Most PCRs don't have this attribute, but the PC Client specification assigns it to a debug PCR and a few reserved for applications.

**USE CASE: NO INCREMENT ATTRIBUTE PCRS FOR VMS**

An application-level PCR may be assigned to measure a virtual machine. This PCR is reset because the VM is instantiated and extended frequently over the lifetime of the VM. If each extend invalidated a policy session, the `TPM2_PolicyPCR` command would be useless.

**USE CASE: NO INCREMENT ATTRIBUTE PCRS FOR AUDIT**

An application-level PCR may be assigned to secure an audit log. See Chapter 16 for details on this use case. This PCR is reset when the audit log is initialized and is extended as the log is updated. If each extend invalidated a policy session, the `TPM2_PolicyPCR` command would be useless.

## PCR Authorization and Policy

As with other entities, a PCR may have an authorization value or policy. The library specification permits either to be set per PCR or per group of PCRs.

The PC Client TPM has neither. No authorization is required to access the PCR. The rationale is that authorization would increase the boot time, which is often an important parameter.

## PCR Algorithms

The first requirement that led to TPM 2.0 was the removal of TPM 1.2's hard-coding of the SHA-1 hash algorithm. Because PCRs are closely tied to hash algorithms, TPM 2.0 theoretically offers many PCR possibilities through the `TPM2_PCR_Allocate` command.

The key word is *theoretically*. PCRs can be allocated in banks, with each bank corresponding to a hash algorithm. The command permits PCRs to be allocated in any combination, and a PCR can be assigned to more than one bank and have more than one algorithm. The `TPM2_Extend` command must now specify not only a PCR index and a digest but also an algorithm. If no index exists with that algorithm, the extend operation is ignored.

So, in theory, software would perform multiple measurements, create multiple digests, and then extend each digest into the appropriate bank. What does the PC Client specification do in practice?

That specification requires only one bank with all PCRs in it. The bank defaults to SHA-1 but can be changed to SHA-256. Although a TPM vendor is free to implement more complicated combinations, we expect most TPMs to be operated as either purely SHA-1 or purely SHA-256. The supporting software knows the TPM's algorithm and measures, digests, and extends accordingly.

Further, we expect that TPMs won't change algorithms very often. If fact, the most likely scenario is that it's shipped with SHA-256 and remains SHA-256 forever, or that it's shipped with SHA-1 and then updates to SHA-256 once as the support software is simultaneously updated.

## Summary

PCRs have two basic uses. Their value may be reported in a signed attestation quote, permitting a relying party to determine the platform software's trust state. They may be used in a policy to authorize the use of other objects based on PCR values. Whereas TPM 1.2 PCRs were hard-coded to use the SHA-1 algorithm, TPM 2.0 PCRs can use other hash algorithms.