CHAPTER 19

Startup, Shutdown, and Provisioning

Startup here is defined as software operations that occur each time a platform boots. The boot can be a cold boot, or it can be what in PC terms is called a *resume from suspend* or a *boot from hibernate*. The TPM holds several classes of volatile state, including PCR values, loaded sessions and keys, enables, authorization and policy values, hybrid NV indexes, and clock state. Based on the type of power cycle, this volatile state must either persist or be initialized. The TPM provides two commands that, in various combinations, permit external software to manage the power-cycle requirements.

Provisioning, on the other hand, is a rare occurrence. It might happen only once over the lifetime of the platform. A TPM vendor, platform manufacturer, IT department, or end user generates keys and other secrets, inserts certificates, and enables or disables certain TPM features. The other side of provisioning is deprovisioning: what the parties do before they repurpose, surplus, or discard a platform to ensure that secrets are erased.

This chapter discusses startup first, followed by the TPM provisioning tasks that various parties may perform. Those parties may include the TPM manufacturer, the platform manufacturer (also called the OEM), and the end user (either an individual or an IT department).

Startup and Shutdown

Startup (and shutdown as well) is handled by low-level software. On a PC platform, this is the BIOS and operating system. The intent is that state is reset or restored as required so that resuming applications are unaware of these events. For example, an application doesn't expect loaded keys or sessions to suddenly disappear. It may not be able to reload keys, and it may not want to rerun a policy evaluation because a session vanished. The TPM, with support from the operating system and boot code, makes power cycling transparent to applications by saving volatile state to its nonvolatile memory on power down and restoring state on power up.

The TPM specification defines three startup events: *TPM Reset*, *TPM Resume*, and *TPM Restart.* They follow a signal called *TPM Init* during a platform reset. In a typical hardware TPM, Italicize is assertion of the TPM reset pin, possibly preceded by a power cycle. At this time, it's assumed the TPM's volatile state is lost, and only the saved (if any) nonvolatile state remains.

TPM Reset normally occurs when the platform is booting after a power on or rebooting without a power cycle. The TPM receives a startup command to reset the TPM volatile state. *Reset* in this case can mean either setting state to a specified initial value or generating new random values for nonces. *TPM Reset* establishes a new trusted platform state. All required software components are measured into the set of reset PCRs. All the TPM's resources are reset to their default provisioned state.

TPM Resume typically occurs when the platform resumes from suspend, sometimes also called a *sleep state* or *low-power state*. Because the platform is continuing rather than rebooting, all state, including PCR values, is restored. *TPM Resume* restores the TPM's state to that before the power loss or reset, because the platform trust state has not changed since the reset or power off.

TPM Restart typically occurs when the platform comes out of hibernation. Before the power cycle, the TPM receives a command to save state, and most of the state is restored at startup. The exception is PCR values, which are initialized, not restored. This permits a booting platform to extend new measurements to the TPM, while the TPM state used by the operating system and applications are restored. *TPM Restart* is a special case where the platform reestablishes its trusted state (by creating new measurements), but the user's state (operating system and applications) is restored.

The TPM provides two commands to support these startup events: `TPM2_Shutdown` and `TPM2_Startup`. Shutdown is typically performed by the operating system just before transitioning to a platform reset or power down. `TPM2_Shutdown` has two options: `CLEAR` and `STATE`. Startup is executed by the root of trust for measurement (RTM) in the initialization firmware (for example, BIOS on the PC). `TPM2_Startup` also has two options: `CLEAR` and `STATE`.

Here are the commands in combination:

- TPM Reset (reboot) is `TPM2_Shutdown` with the `CLEAR` option (or no shutdown command) followed by `TPM2_Startup` with the `CLEAR` option.
- TPM Restart (hibernate) is `TPM2_Shutdown` with the `STATE` option followed by `TPM2_Startup` with the `CLEAR` option.
- TPM Resume (suspend, sleep) is `TPM2_Shutdown` with the `STATE` option followed by `TPM2_Startup` with the `STATE` option.

The following is a brief overview of the command behaviors. There are many details surrounding the clock and time counters, session context, hybrid NV indexes, and more. These are discussed in other parts of the book as the concepts are introduced:

- `TPM2_Shutdown` with the `CLEAR` option is an orderly shutdown before the platform powers down or reboots. The TPM saves certain volatile values to nonvolatile memory: the clock and NV indexes with the orderly attribute that are normally shadowed in volatile memory.
- `TPM2_Shutdown` with the `STATE` option is a shutdown typically due to hibernation or suspend. The TPM stores the previously listed items plus tracking for session contexts, PCRs that the platform specification mandates should be saved, certain NV index flags, and state associated with audit.

- `TPM2_Startup` with the `CLEAR` option initializes TPM volatile state, including PCR and NV volatile state; enables the three hierarchies; and clears the platform authorization and policy.
- `TPM2_Startup` with the `STATE` option is only permitted after `TPM2_Shutdown` with the `STATE` option. PCRs are restored or initialized based on the platform specific specification.[1]

For example, the detailed behavior and rationale for PCRs though the three power-cycle types are as follows:[2]

- On a reboot, *TPM Reset*, all PCRs must be initialized. The `TPM2_Startup` with the `CLEAR` option always initializes PCRs, regardless of the type of shutdown.
- On a resume from hibernation, *TPM Restart*, the platform is rerunning BIOS code and doing its measurements, so the PCRs must be initialized. `TPM2_Startup` with the `CLEAR` option again initializes PCRs, even though they were saved during the power-down sequence.
- On a resume from suspend, *TPM Resume*, the PCR values may be lost on power down. However, the platform resumes without rerunning BIOS, boot, or OS initialization code. PCR values must therefore be restored. `TPM2_Shutdown` with the `STATE` option saves volatile PCRs as the platform suspends. `TPM2_Startup` with the `STATE` option restores those values.[3]

Startup Initialization

The TPM has several parameters that must be initialized at each startup. In TPM 1.2, there was one hierarchy with one owner authorization, and that authorization was persistent. It had one disabled and one deactivated flag. As described in Chapter 9, TPM 2.0 has three hierarchies, each with an authorization secret, a policy, and an enable flag.

TPM 2.0 has a platform hierarchy with a volatile authorization value and policy, which are reset on *TPM Reset* or *TPM Restart*. Software early in the boot cycle is expected to set these values. It also has a hierarchy enable flag, enabled at startup. We expect that platform OEMs will not provide a means for the operating system or applications to disable the platform hierarchy, because OEMs may use the platform hierarchy for runtime functions.

[1]The PC Client specification mandates restoring PCRs 0-15 and initializing PCRs 16-23.
[2]Chapter 18 discusses the management of objects and sessions in detail.
[3]The platform-specific specification indicates which PCR indexes must be restored and which must be initialized.

Platform policy is straightforward. If it isn't set, it's *empty*: a policy that can never be satisfied. If the platform OEM has a policy for platform-hierarchy control, its boot software sets the policy value using `TPM2_SetPrimaryPolicy`. The policy contains no secrets, but its value must be protected from tampering while in the boot software. If the OEM has no such policy, it leaves the platform policy empty, so the policy can't be satisfied.

Platform authorization (HMAC shared secret–style authorization) works differently. The TPM architecture expects the platform to set the platform authorization value to a strong secret using `TPM2_HierarchyChangeAuth` and make this value inaccessible to the operating system or applications. Alternatively, if the OEM has no need for the platform authorization, it can set it to a random value and then "forget" the value by erasing it from system memory. The platform hierarchy is still enabled, but it can't be authorized using a password or HMAC.

The specification designers first considered a persistent value, similar to the TPM 1.2 owner authorization. This raised two questions: how would the platform software remember the shared secret through boot cycles, and how would the platform ensure that a strong secret is used? The solution was to use a volatile value: a large, random number set at startup. The random number can even be obtained from the TPM random number generator. Now the platform software doesn't have to remember the value through power cycles. It's stored in platform volatile memory that's accessible only to the platform software.

We expect that the platform authorization value may not be set immediately. Because the platform software trusts itself, it may leave the value empty, a very easy value to remember, and set it later, before exiting to option ROMs or other untrusted software, to protect the platform hierarchy from other software.

Other hierarchies are persistent and need not be initialized at startup. These include the (storage hierarchy) owner authorization and policy, endorsement authorization and policy, and the lockout authorization and policy.

The storage and endorsement hierarchy enables are set at *TPM Reset* and *TPM Restart*. The platform is expected to remember the owner's and privacy administrator's requested state and disable them if required.

Provisioning

Here, *provisioning* includes all TPM setup that occurs less frequently than once per boot cycle. In a typical TPM lifetime, these actions may be performed only once.

This book divides the provisioning operations among three parties: the TPM manufacturer, the platform manufacturer (often referred to as the OEM), and the end user. Although this is a typical partition, the TPM doesn't enforce it, and enterprises may deviate from this pattern based on their trust model. For example, a platform in a large enterprise may further partition end-user provisioning between an actual user and an IT department administrator. A high security use case may replace a TPM vendor-supplied endorsement primary key or certificate with its own.

In TPM 1.2, certain provisioning steps could only be performed once. For example, although it had the concept of a *revocable* endorsement key that could be deleted and regenerated with a different value, this was optional and not implemented in commercial hardware TPMs.

In the TPM 2.0 architecture, there are no once-per-lifetime values. However, a platform specification may, for example, make `TPM2_ChangeEPS` optional, and a vendor may not implement it. In that case, an endorsement key created with a known template (see Chapter 15 for details) could not be permanently destroyed, although it could be flushed from the TPM. TPM 2.0 can also provision additional endorsement keys.

TPM Manufacturer Provisioning

The TPM manufacturer is uniquely qualified to certify that its hardware is authentic. Once the TPM part enters the supply chain, most purchasers don't have the expertise to distinguish a counterfeit from a genuine part.

The TPM generates[4] a primary seed in the endorsement hierarchy when it's first powered on. The TPM manufacturer then uses `TPM2_CreatePrimary` one or more times to create endorsement primary keys—the ones at the root of the endorsement hierarchy.[5] This command returns the public key, which the manufacturer uses to create a certificate.

The certificate, typically in X.509 format, asserts that the public key belongs to a genuine vendor TPM. It typically includes manufacturer identification. There is no security-related reason for the vendor to store either the primary key or its certificate on the TPM. Practical concerns drive the decision.

The primary key is generated from the primary seed and a caller-supplied template. The template contains the key algorithm and size, and possibly other entropy. If the seed isn't changed, the same template will always generate the same key. Thus, the vendor need not ship the TPM with the key stored in persistent memory. The user can re-create it at any time. This avoids consuming valuable NV memory in cases where the TPM vendor generates many primary keys for different templates, or when the key is likely to be used infrequently.

Why does the TPM use seeds? TPM 1.2 generated the endorsement key directly, but there was one algorithm (RSA) and one key size (2,048 bits). TPM 2.0 can have many algorithms and key sizes. If the TPM 1.2 pattern was used, each key would have to be stored on the TPM, consuming valuable NV memory. The TPM 2.0 design requires only a single persistent seed. The derived keys can be generated (and then discarded) as needed.

The advantage of shipping the TPM with a primary endorsement key is performance. Why have the user create the primary key when the vendor has already created it?

In practice, the TCG platform working groups are expected to specify one or more standard templates based on anticipated application needs. The TPM vendor will generate multiple keys but only provision one for a commonly used algorithm and size on the part before shipment.

---

[4]The manufacturer is permitted to create the endorsement primary seed externally and "squirt" it into the TPM in a vendor-specific process. This potentially saves manufacturing time, because the primary keys can also be calculated external to the TPM.

[5]Chapter 10 explains the general process of creating primary keys, and Chapter 15 goes into even more detail.

A similar practical concern determines whether the TPM ships with certificates. Although it's true that a user can usually ask the TPM manufacturer for a certificate corresponding to their public key, it's certainly more convenient to read it from TPM NV storage. There may be use cases where the platform isn't connected to a public network, making certificate retrieval even more inconvenient. In practice, we expect the TPM vendor to provision a certificate corresponding to the one or more primary keys. The certificate likely resides in the TPM's NV storage.

There can be use cases where the user doesn't completely trust the TPM vendor processes. Other use cases require an end user such as a government agency to prevent any link in the supply chain from tracking which machines are used by that agency. They want to remove any unique key that may aid in that tracking. This user can use `TPM2_ChangeEPS` to change the endorsement primary seed;[6] generate new, different primary keys;[7] and sign their own certificates. The user can also change the template to include a random number, which is unknown to the vendor, thus generating an endorsement primary key unknown to the vendor without invalidating existing primary key certificates.

In summary, the four differences from TPM 1.2 that contribute to these scenarios are as follows:

- Primary endorsement keys can be re-created at any time as long as the primary seed doesn't change and the template is known.
- Because primary endorsement keys can be re-created, they need not be stored long term in the TPM.
- Because TPM 2.0 supports multiple algorithms and added template entropy, there can be more than one primary endorsement key.
- If rolling the EPS is supported, endorsement keys can be deleted, and thus the certificates invalidated.

Platform OEM Provisioning

Platform OEM provisioning has two concerns:

- Authenticity
- Control

As with the TPM manufacturer, a platform manufacturer certificate[8] (typically in X.509 format) asserts that the hardware is authentic. It asserts that the TPM is attached to the OEM's platform. It further asserts that the platform software meets certain TCG recommended standards. For a PC client, this includes a CRTM that performs measurements of software and extends those measurements to PCRs.

[6]Rolling the EPS also deletes any primary or descendent keys in the hierarchy.
[7]Keys generated using the new EPS are cryptographically unrelated to those generated using the old EPS.
[8]We know of no OEM that currently is provisioning platform certificates.

Although an attacker could physically remove a TPM from the OEM platform and put it in a counterfeit, the TCG technology doesn't defend against physical attacks. Further, this would compromise only one platform per attack. On the other hand, a successful attack that extracts a primary seed from a TPM would permit the attacker to manufacture an unlimited number of counterfeits.

Platform certification typically begins with an endorsement primary key generated by the TPM at the TPM manufacturer. The OEM wants to verify that the TPM is authentic before asserting that its platform is authentic. It reads the TPM certificate and validates it; it may go further, reading the public key and verifying that it matches, or even use the private key to prove that the key pair is present.

As with the TPM vendor endorsement key certificate, there is no security-related reason to store the platform certificates on the TPM before shipping the platform. Practical considerations will likely drive the OEM to include a certificate in the TPM's persistent memory, corresponding to the vendor TPM certificate.

TPM 2.0 specifies a platform hierarchy. The platform OEM may optionally provision that hierarchy with a platform policy. As explained earlier, the TPM initializes this policy to empty: a policy that can never be satisfied. The OEM must provision it at startup using `TPM2_SetPrimaryPolicy`.

Where does the platform policy value (a hash) come from? We expect that the value will be embedded in early platform boot software, protected by the same OEM mechanism that protects the CRTM. So, this value is provisioned during platform manufacturing not into the TPM, but into the platform CRTM. It's inserted into the TPM at startup.

End User Provisioning

The term *end user* is used loosely here. For a home computer, the end user is typically the literal user of the computer. For an enterprise, centrally managed platform, the end user may be the actual user or support personnel.

The end user must provision the endorsement and storage hierarchies. This is a case where an IT organization may decide to provision the endorsement hierarchy and/or the dictionary-attack reset authorizations, and leave the provisioning of the storage hierarchy to the person who is actually using the platform.

The first consideration is whether to disable the hierarchies (always enabled at startup) using `TPM2_HierarchyControl`. The method of disabling a hierarchy is platform specific. We expect something equivalent to a BIOS screen, with a means of disabling a hierarchy for the remainder of a boot cycle or having it persist through boot cycles. (We hope to present enough valuable use cases that you will never dream of disabling the TPM.)

Next, the end user must provision the endorsement and storage hierarchy, and the dictionary-attack protection policies and authorization values. Owner authorization is initially empty (no authorization required), as is owner policy (no policy is present). `TPM2_HierarchyChangeAuth` and `TPM2_SetPrimaryPolicy` change these values, which persist until cleared. The endorsement authorization and policy are set and changed using the same commands. The owner authorization and endorsement authorization should be set to high-entropy values, because they aren't guarded by the dictionary-attack protection.

The dictionary-attack logic has both a policy and an authorization value. The same party may provision all policies and authorization values, but they may choose different logic and values. It's particularly important to provision the dictionary-attack reset policy. If triggered, the dictionary-attack password can be used only once before a large wait is enforced. However, the policy can be used to reset the dictionary-attack counter even when the dictionary-attack password has been locked out.

The endorsement primary seed is generated during TPM vendor manufacturing. The end user doesn't typically change this value.

The storage primary seed is similarly generated during TPM vendor manufacturing. The end user can either use this value or generate a new one. Generating a new one invalidates all objects in the storage and endorsement hierarchies except the endorsement hierarchy primary keys. Thus, the endorsement primary key certificate is still useful.

Deprovisioning

Deprovisioning is primarily the process of removing secrets from the TPM, although a user may wish to remove public but unique data as well. A user typically deprovisions before surplusing a platform in an enterprise, selling used equipment, or scrapping the system.

What secrets aren't touched? Those in the platform hierarchy (if any) are controlled by the platform manufacturer. The endorsement hierarchy primary seed is typically fixed for the lifetime of the TPM. Changing this seed would invalidate endorsement primary keys and thus make their certificates useless.

A platform OEM may be tempted to use NV space to store end-user settings. For example, consider the BIOS configuration. Such an index could have a policy permitting the end user to write a value and anyone (specifically, the BIOS) to read it. Such use is discouraged if any value is even remotely secret or privacy sensitive, because end-user deprovisioning is easy to overlook.

Deprovisioning uses the `TPM2_Clear` command. Note that the authorization for this command is not, as you may expect, `TPM_RH_OWNER`, the role that controls the storage hierarchy. Rather, it's either `TPM_RH_LOCKOUT` (the dictionary-attack reset authorization) or `TPM_RH_PLATFORM` (platform authorization).

Lockout authorization is a reasonable choice. It's likely that the user knows this authorization. The TPM probably uses this value rather than owner authorization because, in some situations, the owner authorization may be more widely known. Because deprovisioning has wide-ranging effects, it's better to assign it to a more restrictive role.

Platform authorization is trickier, because it's available only early in the boot cycle, not at the OS level. This poses two problems. First, none but the most tech-savvy users can be expected to do *any* operation at the BIOS-screen level. Second, BIOS screens preclude remote deprovisioning, which is a requirement for cloud-type data centers. The solution to this dilemma is a platform policy, the alternative to a simple platform HMAC or password-based authorization. For example, suppose the platform owner wishes to allow a user to run the `TPM2_Clear` command with owner authorization. A platform policy includes an `OR` term that says, "command code = `TPM2_Clear` `AND` Policy Secret's handle == `TPM_RH_OWNER`". This policy permits owner authorization to be used, and the user can apply this authorization at the OS level.

What does `TPM2_Clear` do? Quite a lot. First, `shProof` and `ehProof` are changed. These proofs serve as HMAC keys for saved (nonresident) contexts. Once the proof changes, contexts in the storage and endorsement hierarchies can no longer be loaded. Next, any TPM-resident objects in either the storage or platform hierarchy are flushed. The storage primary seed is changed; this prevents primary storage keys from being regenerated, and thus, any object created under these keys can no longer be loaded. Finally, any NV index created by the owner is deleted. This is an improvement over TPM 1.2, where some indexes have to be individually deleted.

TPM2_Clear also prepares the TPM for the next owner. It resets authorization values. The lockout authorization is cleared (to a zero-length password), and the policy is cleared (to no policy in effect). The owner and endorsement authorization and policy are similarly reset. The storage and endorsement hierarchies are enabled; this is a departure from TPM 1.2, which required a reboot after an owner clear before a new owner could be installed.

The dictionary-attack mechanism count of failed authorization tries is reset. The clock is reset so that the new owner can accurately set it as desired. (The clock can't normally go back in time.) Reset count and restart count, which track boot cycles, are reset.

Summary

There are three startup cases, roughly corresponding to a PC cold boot, resume from suspend, and power up after hibernate. The TPM provides startup commands to reset or restore its state as appropriate for these cases.

So that startup authorization secrets need not be saved in the clear off the TPM or be accessible during boot, the TPM resets certain values and expects them to be provisioned during startup. Before it reaches the end user, the TPM may be provisioned with keys and certificates. These include TPM manufacturer-provisioned endorsement primary keys and corresponding certificates, and perhaps another set created by the platform manufacturer. The end-user provisioning steps include initialization of the storage hierarchy and installing endorsement and storage hierarchy authorization keys or policies.

Finally, deprovisioning through `TPM2_Clear` removes secrets and NV-defined indexes, resets authorizations and policies, and resets other values in preparation for the new owner.