CHAPTER 11

NV Indexes

The TPM requires the use of nonvolatile memory for two general classes of data:

- Data structures defined by the TPM architecture.
- Unstructured data defined by a user or a platform-specific specification

One use of TPM nonvolatile memory is for architecturally defined data, or fields defined in the TPM library specification. This includes hierarchy authorization values, seeds and proofs, and private data that the TPM won't reveal outside its secure boundary. It also includes counters, a clock, and more: nonvolatile data that the caller can read. Nonvolatile memory can also hold structured data made persistent, such as a key.

This section describes a second use of NV memory: unstructured platform or user-defined space. This is sometimes called a *user-defined index*, because the user assigns an index (a handle) to each area and accesses data using the index value.

TPM 1.2 includes user-defined indexes that can hold unstructured data. The user defines the size and attributes of the index. The user can write data without any restriction on the data value. The TPM provides authorization, controlling access to the index via a shared secret keyed-hash message authentication code (HMAC) key, Platform Configuration Register (PCR) values, locality, and physical presence, and provides various read and write locks.

TPM 2.0 expands the 1.2 features in several ways:

- An index can have the state "uninitialized, not yet written." Reads will fail until the index is first written. Further, the index can't be used in a policy. A party relying on a value can be assured that a party with write authority initialized the index and that the data doesn't simply have a default or uninitialized value.
- As with any other protected entity, TPM 2.0 indexes may have either an authorization value or a policy.
- Another entity's policy can include an NV index value. The policy specifies an operation to be performed on all or part of the index value: a comparison to policy data. The operations include equal, not equal, signed, and unsigned comparisons, and a check for bits set or clear.

Another new NV index feature is the data type. It augments the 1.2 unstructured data type (now called *ordinary*) with three others, giving four NV index types:

- Ordinary
- Counter
- Bit field
- Extend

NV Ordinary Index

An ordinary index is like a TPM 1.2 index. It holds unstructured data of arbitrary length. In contrast with counter, bit-field, and extend indexes, there is no restriction on the type of data that can be written.

USE CASE: STORING A SECRET

A platform contains a 20-byte secret that must be available early in a boot cycle. It stores the secret in an NV index. The index attribute `TPMA_NV_PPREAD` specifies that reads require platform authorization. The platform software, running early in the boot cycle, knows this authorization and so can read the secret. It's trusted not to reveal the secret once it completes its task. Because other software later in the boot cycle or beyond doesn't know the platform authorization, it can't read the secret.

The TPM commands are as follows:

- `TPM2_NV_DefineSpace`: Create an ordinary index, size = 20 bytes, with platform authorization to read and write
- `TPM2_NV_Read`: Uses platform authorization

USE CASE: STORING A CERTIFICATE

A platform OEM creates a certificate stating that an endorsement key is fixed to the platform and that the platform was manufactured with certain security guarantees. The OEM stores the certificate in NV during manufacturing. Read access is unrestricted. Write access is restricted by policy to the OEM and is used to update the certificate.

The TPM commands are as follows:

- `TPM2_NV_DefineSpace`: Create an ordinary index, size of certificate, platform authorization to write, read with authorization value, and a null (zero-length) password
- `TPM2_NV_Write`: Run with platform authorization
- `TPM2_NV_Read`: Run with a null password

USE CASE: STORING A COMMON PASSWORD

A user creates a set of keys with an identical policy, authorizing use if a password in the NV authorization field is known. The user permits access to all keys by supplying the correct secret value. The user writes one NV location to change the common password for all keys.

The TPM commands are as follows:

1. `TPM2_NV_DefineSpace`: ordinary index, size = 0 bytes (the NV data is not used in this use case), common password, policy password to change authorization.
2. Create a common policy: `TPM2_PolicySecret` with the name of the NV index.
3. `TPM2_Create`: Creates multiple keys with the common policy. `userWithAuth` is clear so that a policy is mandatory.
4. `TPM2_NV_ChangeAuth`: Changes the password for all keys in one operation, using the current password.

USE CASE: STORING A ROOT PUBLIC KEY

The IT administrator places the hash of a public key in NVRAM, which is locked so the user can't write to it. It's used to verify a public key, which is used in turn to verify that signatures are from IT. Or it's the hash of the root public key of the certificate chain.

The TPM commands are as follows:

1. IT creates the signing key and digests the public key.
2. Create a read policy: `TPM2_PolicyCommandCode` with the command `TPM2_NV_Read`. This policy allows anyone to read the index essentially without authorization.
3. `TPM2_NV_DefineSpace` - ordinary index, size = digest size, IT administrator password, password to write, policy to read with the above read policy.
4. `TPM2_NV_Write` - with the IT admin password, storing the public key digest.

And here's how to verify a signature:

1. `TPM2_NV_Read` read the public key digest.
2. Validate the public key against the digest.
3. Validate the signature against the public key.

USE CASE: STORING AN HMAC KEY

In the Linux Integrity Measurement Architecture (IMA) Extended Verification Module (EVM), store an HMAC key that is released to the kernel early in the boot and then used by the kernel to verify the extended attributes of files to see that they have been approved for loading or use by the kernel.

The TPM commands are as follows:

- `TPM2_NV_DefineSpace` - ordinary index, size = HMAC key size, IT administrator password, password to write, policy to read with the above (anyone can read) read policy
- `TPM2_NV_Write`: With the IT admin password, stores the HMAC key
- `TPM2_NV_Read`: Reads the HMAC key

NV Counter Index

An *NV counter* is a 64-bit value that can only increment. At the beginning of the first increment command, it's initialized to the largest value that any counter has ever had on the TPM. This includes counter indexes currently defined and counters that were defined in the past but are no longer on the TPM. Thus a counter can never roll back, even by deleting and re-creating the index.

TPM 1.2 users might be familiar with monotonic counters. These NV counters are the equivalent, but the user is free to define none or as many as are needed up to TPM resource limits.

USE CASE: REVOKING ACCESS TO A KEY

A key holder wants to revoke access to a key. The key is created with a policy that says the key can be used (the policy can be satisfied) as long as the counter value is equal to its current value. Incrementing the counter revokes access.

The TPM commands are as follows:

`TPM2_NV_DefineSpace`: counter index, password of key holder, password to write, and a policy to read. The policy is `TPM2_PolicyCommandCode` with the command `TPM2_PolicyNV`. This policy allows anyone to use the index in a policy essentially without authorization.

`TPM2_Create`: Create a key with `userWithAuth` clear, requiring a policy to authorize the key. The policy is `TPM2_PolicyNV` with the NV value equal to all zero.

`TPM2_NV_Increment`: Revokes authorization to use the key.

NV Bit Field Index

A bit field contains 64 bits, initialized to all bits clear at the beginning of the first write, which can also optionally set bits. A bit (or bits) can then be set but never cleared.

USE CASE: MULTIPLE-USER KEY REVOCATION

A key holder wants to grant and later revoke access to a key for up to 64 users. The key is created with a policy with up to 64 OR terms. Each term combines some authorization specific to each user (a biometric or smart card digital signature, for example) and an assigned bit being clear. The key is revoked for that user by setting the bit in the NV bit field.

The TPM commands are as follows:

`TPM2_NV_DefineSpace`: bit field index, password of key holder, password to write, and a policy to read. The policy is `TPM2_PolicyCommandCode` with the command `TPM2_PolicyNV`. This policy allows anyone to use the index in a policy essentially without authorization.

`TPM2_Create`: Create a key with `userWithAuth` clear, requiring a policy to authorize the key. The policy is `TPM2_PolicyNV` with the operand `TPM_EO_BITCLEAR` (the bit assigned to the user is clear).

`TPM2_NV_SetBits`: Setting the bit assigned to the user, thus revoking authorization to use the key.

NV Extend Index

An extend index is defined with a specified hash algorithm, and it's fixed for the lifetime of the index. The data size of the index is based on its hash algorithm. It's initialized to all zero before the first write. The write is an extend operation, similar to that performed on a PCR.

The most likely use case combines an extend and a hybrid index to create flexible PCRs, as discussed in the "Hybrid Index" section. Another general use case is a secure audit log, because any extend operation creates a cryptographic history that can't be reversed.

USE CASE: SECURE AUDIT LOG OF CA KEY USE

A certificate authority wants to log each time its key is used to sign a certificate. It wants to be able to detect whether the log, kept on disk, has been altered. It creates an NV extend index for which it has exclusive write authority through a policy.

Each time the CA signs a new certificate, it logs the certificate and extends a digest of the certificate into the NV index. To validate the log, it walks the log, re-creating the extend value, and compares it to the NV index value. For additional security, it can even get a signature over the NV index value.

The TPM commands are as follows:

- `TPM2_NV_DefineSpace`: extend index, policy to write by CA, and a policy to read by anyone. The CA signs a certificate, logs the certificate in an audit log, and digests the certificate.
- `TPM2_NV_Extend`: Adds the digest to the NV audit log. If the certificate is small enough, it can be extended directly. However, it's probably faster to digest outside the TPM and extend just a digest.

Hybrid Index

Yet another new TPM 2.0 feature is the hybrid index. As with a nonhybrid, the NV index metadata (its index handle, size, attributes, policy, and password) are nonvolatile; its data is created in volatile memory. Except for hybrid counters (described later) the index data is only written to NV memory on an orderly shutdown. Any of the four index types (ordinary, counter, bit-field, or extend) can be a hybrid index.

Hybrid indexes may be appropriate when the application expects frequent writes. Because NV technology is often subject to wear out, a TPM may protect itself by refusing to write at a high rate. Volatile memory doesn't have wear-out issues, so a hybrid index can be written as often as required.

Hybrid index data may only be present in volatile memory if the index is deleted before an orderly shutdown. An application could define the index, write data, use the values in a policy, and then delete the index.

USE CASE: ADDITIONAL PCRS

The simplest use case is adding PCRs beyond the number the TPM vendor provides, typically 24 for a PC Client TPM. As with the permanent PCRs, the index persists through power cycles, but the value is reset back to zero. This means PCRs are no longer a scarce resource ("beach front property," in TCG work group slang).

The TPM commands are as follows:

- `TPM2_NV_DefineSpace`: hybrid extend index, well known null password, so that anyone can read and extend
- `TPM2_NV_Extend`: Now equivalent to `TPM2_PCR_Extend`

USE CASE: PCRS WITH DIFFERENT ATTRIBUTES

An application requires PCRs, but the standard TPM PCRs have fixed attributes specified by the platform. For example, the operating system may restrict access, the application may want PCR authorization restrictions, the application may need a hash algorithm different from those in effect for the TPM PCRs, or the application might want extends restricted to an extended locality.

The application creates a hybrid extend index with the desired attributes, uses them as PCRs, and then deletes the index when the application terminates. Note that, because the index is a hybrid, the extend doesn't write NV memory, avoiding performance and wear-out issues.

The TPM commands are as follows:

- `TPM2_NV_DefineSpace`: hybrid extend index, application-specific digest algorithm, application-specific extend policy more restrictive than "anyone can extend."
- `TPM2_NV_Extend`: as needed by the application.
- `TPM2_NV_UndefineSpace`: when the application terminates.

These PCRs can have read authorizations: policy authorizations for either read or write different from those of the platform. They can be used in virtual TPMs to record the state of a helper VM inside the vTPM of a VM (see the Virtualization Specification).

USE CASE: VIRTUALIZATION

A VMM creates an NV extend hybrid index for each VM. When the VMM creates a VM, it creates a corresponding PCR using a hybrid extend index. As the VMM starts the VM, it uses introspection to read and measure the VM's boot code, extending the measurements into the VM's PCR.

This requires a single command:

`TPM2_NV_DefineSpace`: Create a hybrid extend index

Hybrid ordinary, extend, and bit-field indexes are only written to NV memory on an orderly shutdown. Hybrid counters are more complicated because of the restriction that they never roll back or miss an increment operation. This must be ensured even if the shutdown isn't orderly, when the volatile value would be written to NV memory.

To achieve this, the value is written (flushed) to NV memory every so many increments. Thus a hybrid counter may not be solely in volatile memory, even if it's deleted before an orderly shutdown. If an application wants to avoid the flush, or at least determine when it will occur, a `get capability` command can report the flush period.

The second hybrid counter complication occurs on startup. If the most recent value wasn't flushed to NV (through an orderly shutdown), the count is set to the highest value it could have had without causing a flush. That is, it might skip some counts, but it will never roll back or miss an increment.

NV Access Controls

We previously discussed the major NV attributes: whether it's an ordinary, counter, bit-field, or extend index, and whether it's a hybrid index. An NV index also has unique controls that are different from objects like keys. Perhaps the most interesting is that it can

have separate controls for read and write. In particular, each index can be defined to use its policy, its password authorization, or the owner password or authorization, and the attributes can be set independently for read and write.

The TPM supports a set of NV index read and write locks. An index may be write-locked permanently. It can be write- or read-locked until the next TPM reset or restart. An index may be part of one set of indexes that can be locked in one operation (a global lock), again until the next reset or restart.

Finally, many TPM entities are protected by the dictionary-attack protection mechanism. After some number of failed authorization attempts, the TPM rejects authorization until a certain amount of time has passed. An NV index may be protected as well, but an attribute can be set to remove the protection. Removing the protection might be applicable if the authorization password is known to be a strong secret.

NV Written

Each NV index, when first created, has an implied value: not written. In TPM 1.2, an index was always created with all-zero data. A read could not distinguish between all-zero data and a not-yet-written index. In TPM 2.0, not written is a separate state. A policy can specify that the index must or must not be written.

USE CASE: WRITE-ONCE NV INDEX

The creator wants an index that can be written exactly once, perhaps during provisioning. Once written, it can be read by anyone with the correct password.

To implement this, create an OR policy with two terms. The first term permits the NV `Write` command code only if the index has not been written. The second term permits a read if the index has been written and the password is supplied.

Here are the steps:

1. Create a policy with two terms:
   - `TPM2_PolicyCommandCode` (`TPM2_NV_Write`) *AND* `TPM2_PolicyNvWritten` (`writtenSet` clear)
   - `TPM2_PolicyCommandCode` (`TPM2_NV_Read`) *AND* `TPM2_PolicyPassword`
2. `TPM2_NV_DefineSpace` - create an ordinary index, policy to write and read.

NV Index Handle Values

When the user creates an NV index, the user assigns an index value.[1] In TPM 1.2, certain bits had special properties, such as the D bit used for locking. In the TPM 2.0 library specification, there is no index assignment other than an overall handle range, and no bits of the index value have any special meaning. The TPM doesn't enforce any index properties based on the index value. However, platform-specific specifications or a global TCG registry can assign index values.

For example, the TCG registry assigns handle ranges to the TPM manufacturer (specifically, 0 to 0x3fffff), to the platform manufacturer, and for endorsement and platform certificates. It further reserves ranges for platform-specific specifications, such as the PC Client, server, mobile, and embedded platforms. All these assignments are by convention and aren't enforced in any way by (current) TPMs.

USE CASE: STANDARD CERTIFICATES

We expect that the TCG Infrastructure work group will define standard NV indexes for endorsement key certificates. Whereas TPM 1.2 has two such certificates, for the TPM vendor and for the platform OEM, TPM 2.0 can have certificates for multiple key algorithms and even different creation templates.

Although the previous assignments are solely by convention, a TCG work group can also assign NV index values with implicit hardware properties. For example, the TPM may contain special hardware-package pins for general-purpose IO, called GPIO pins in the library and platform specifications. The platform specification determines the properties of the GPIO pins, including the following:

- The number of pins
- The assignment of a pin to an NV index value
- Whether a pin is mandatory or optional
- Whether the pin is fixed as an input or output, or is programmable
- Whether an output is volatile or persistent
- Whether the assignment is fixed by the TPM vendor firmware during manufacturing, or the index must be defined programmatically by the end user using the `TPM_NV_DefineSpace` command

The NV data is a hardware pin, but the NV metadata is identical to that of other indexes. Thus the GPIO comes with the full range of NV index controls, including an authorization value or policy, read and write controls, and locking features.

[1]This is different from starting a session or loading an object, where the TPM assigns the handle.

NV Names

The Name of a TPM entity uniquely (and cryptographically) defines the entity and is used for authorization. For an NV index, it's a hash of the public area, which includes the index (the handle), the attributes (including whether it has been written), the policy, and the size.

TPM2_PolicyNV permits an NV index value to be used in a policy. The policy can be based on a range of logical and arithmetic operations on the index. If the policy were based merely on the NV index value, it would offer little security: an attacker could delete the index and replace it with one with different access controls. For that reason, TPM2_PolicyNV uses the Name.

An example might help. Suppose you create an NV bit-field index that you intend to use for key revocation. The policy for the key includes a TPM2_PolicyNV term that can only be satisfied if the NV bit 0 is clear. The policy for the NV index says only the owner of a private key can write the index (TPM2_PolicySigned). To revoke the key, the owner signs a nonce to satisfy the NV policy and then sets bit 0 (TPM2_NV_SetBits).

Now suppose an attacker tries to remove the key revocation. They can't clear bit 0, because a bit-field index bit can only be set, never cleared. So, the attacker tries something more promising: they delete the index and re-create it with exactly the same Name. This fails because TPM2_PolicySigned fails on an index that has not yet been written. The attacker can't write the index because it can't satisfy the NV index policy TPM2_PolicySigned term.

The attacker makes one final attempt. They delete the index and re-create it with a policy that they can satisfy. They then write the index so that bit 0 is clear and use that index to authorize the key's policy using TPM2_PolicyNV. Because bit 0 is clear, it appears that the policy should succeed. However, the attacker had to change the policy, which causes the Name to change. When the new Name is used in TPM2_PolicyNV, the key's policy evaluation fails.

In summary, the "delete and re-create an index" attack fails because of two TPM features:

- An NV index can't be used in a policy until it has been written.
- The NV index in a policy uses the entire Name (public area), not just the index handle.

USE CASE: WRITE ONCE, READ ALWAYS NV INDEX

The user desires to create an index that they can write exactly once and that can then be read by anyone. An example is provisioning the TPM with a certificate.

The index has two OR terms. The first policy term is satisfied when the index has not been written and the owner supplies the correct password; it permits only the NV `write` command. The second term is satisfied once the index has been written; it permits only the NV `read` command.

Another subtle point is that the Name changes when the index is written, because the index public area includes the `written` attribute.

USE CASE: SECURING A POLICY SECRET

A policy secret permits authorization for a set of objects to be linked to a single secret. For example, a set of keys can have identical policies that authorize the key if an NV index-authorization password is known. The policy would use the NV index Name after it has been written.

■ **Note** `TPM2_PolicySecret` ties authorization to the NV password. `TPM2_PolicyNV` ties authorization to the NV data.

If the Name didn't change when the index was written, an attacker could delete the index and create a new one with the same Name but their own secret and thus gain access to keys tied to the secret. The attack doesn't work, because the attacker's index has not been written and thus has a different Name than the one required in the key's policy.

It's assumed here that the NV index policy (part of the Name) prevents the attacker from writing the index. For example, the NV write policy might require authentication with a public key (`TPM2_PolicySigned`).

■ **Note** Observe that the data value written to NV doesn't matter and serves only to prove that the index creator can write the index. The key policy is tied to the NV password, not the NV data.

The previous use case demonstrates an interesting property. You can create an NV index with a Name that no one else can reproduce. If the Name includes having `written` set, and the policy is such that only you can write the index, then only you can create that Name. This ensures that a policy points to your NV index, not one that an attacker created.

Further, the same NV index with the same Name can be created on multiple TPMs.

USE CASE DUPLICATING A SET OF KEYS

In the previous use case, the authorization for a set of keys was tied to an NV index password. A user can duplicate a set of keys to another TPM. Then the user can create an NV index with the same Name on that TPM so that the key's policy can be satisfied on the new TPM.

NV Password

A subtlety of the TPM is that a user can't really change an object's password. The `TPM2_ObjectChangeAuth` command can create an object with the new password, but the original object still exists. The user can delete all existing copies of the object, but the TPM can't enforce this.

This quirk isn't true of an NV index. The index exists only on the TPM. It can never be context-saved or in any way moved off the TPM. Thus, `TPM2_NV_ChangeAuth` really does change the password.

Separate Commands

The TPM API defines a set of commands dedicated to NV. `TPM2_NV_DefineSpace` creates an NV index. The caller specifies the NV metadata, including the size for an ordinary index, the policy, attributes, and the password. As explained earlier, a newly created index isn't initialized, or written, yet. It has no data.

The write commands are as follows:

- `TPM2_NV_Write` writes an ordinary index. Depending on the attributes, partial writes may or may not be permitted.
- `TPM2_NV_Increment` increments a counter index. Depending on the attributes and the `count` value, this may cause a write to nonvolatile memory.
- `TPM2_NV_Extend` extends arbitrary data (not necessarily a hash value) to an extend index.
- `TPM2_NV_SetBits` sets bits in a bit-field index. It `OR`s the current value and the input. An input of all zero is legally and useful. It makes the index `written` and initializes it to all zero.

`TPM2_NV_Read` reads any index data. A read can only occur after the index has been written at least once. `TPM2_NV_ReadPublic` reads the index public data. In combination with the session audit feature, a user can get a signature over the public area to prove its properties.

Several commands are dedicated to locking an index. The index attributes determine whether these locks can be set against a particular index:

- `TPM2_NV_WriteLock` can lock an index, forbidding further writes until the next boot cycle or forever.
- `TPM2_NV_GlobalWriteLock` can lock a set of indexes, again either forever or until the next boot cycle.
- `TPM2_NV_ReadLock` locks an index, preventing further reads until the next boot cycle.

`TPM2_NV_ChangeAuth` changes the index password. `TPM2_NV_Certify` can create a signature over index data. This command is optional in the PC Client specification. However, a similar result can be obtained by reading the index in an audit session and then getting a signed audit digest.

Summary

TPM 2.0 has four types of NV indexes: ordinary (unstructured data), bit-field, counter, and extend data indexes. An index can be read or written using the standard TPM password and policy controls. Hybrid indexes normally exist in volatile memory, but an orderly shutdown can save them to NV memory. They can avoid performance and wear-out issues. When an index is created, its state is "not written". Its data can't be read or used in a policy until it's written, and the "not written" state itself can be used on a policy.

Basic applications include provisioning with certificates or public keys. More advanced applications use an NV authorization in a policy, permitting it to be shared among entities. A policy referring to a bit-field or counter index value can be used for key revocation. An extend index offers PCR equivalents with different algorithms, authorizations, or lifetimes.

NV indexes have a separate set of commands and unique attributes to control authorization, read and write locking, and dictionary-attack protection.