Firmware Measurement Descriptors

1. Scope

This document is a proposal for a file descriptor which can be used by roots of trust to measure, verify, and update device firmware. Additionally, architectures are proposed for three use-cases of this descriptor format; (1) measuring first mutable code, (2) verifying first mutable code with secure boot, and (3) secure update of device firmware.

1.1 Root of Trust Mechanisms

This proposal is not opinionated about the mechanism using the Firmware Measurement Descriptor (FMD). Whether this is discrete hardware, an integrated root of trust, or some other mechanism, the Root of Trust should be able to perform measurement and verification using the FMD.

1.2 Boot Media

This proposal should not be opinionated about the hardware medium from which the CPU boots the first mutable code. This proposal typically uses flash as an example boot medium, however this should not be a requirement.

This proposal does assume that the regions which are covered by the FMD are byte-addressable when measured by the RTM.

2. Terms and Definitions

2.1 Root of Trust for Measurement (RTM)

An entity (hardware or software) which bootstraps the process of building a measurement chain on a booting system.

2.2 Root of Trust for Update (RTU)

An entity (hardware or software) which verifies firmware updates and ensures only properly verified firmware updates can be applied.

2.3 Root of Trust for Verification (RTV)

An entity (hardware or software) which verifies first mutable code on a system and ensures only properly verified code is allowed to boot.

2.4 Type-Length-Value (TLV)

A binary encoding scheme which is used for optional information in a protocol. A TLV structure contains, in order, (1) an integer type field, (2) an integer size field, and (3) a value with the specified size.

2.5 TLV Section

Any FMD structure which begins with a TLV header is referred to as a “TLV Section”.

2.6 Verifier

An entity which verifies the measurement chain bootstrapped by an RTM.

2.7 S-RTM

An RTM where the initial integrity measurement occurs at platform reset. The S-RTM is static because the PCRs associated with it cannot be re-initialized without a platform reset.

2.8 S-HRTM

An S-RTM implemented using a Hardware RTM.

2.9 Validate

Validation refers to the act of hashing code or data and then making a determination as to the validity of that data.

2.10 Measure

Measurement refers to the act of hashing code or data and storing that value so that it may be reported later to some external party.

2.11 Enforcement

Enforcement refers to an RoT making a judgment about the validity of an entity and gating some action on whether or not this entity is valid.

2.12 Natural Alignment

Natural alignment refers to the fields in a structure being aligned as follows:

- 1 byte data may be at any address
- 2 byte data must contained in an aligned 4-byte word
- 4 byte data must have a start address that is a multiple of 4
- 8 byte data must have a start address that is a multiple of 8
- etc.

This ensures that there is no implicit padding added between structure members.

2.13 FMD Section

An FMD section is any structure which has a TLV header. Each Section must have a fixed length in bytes.

3. Goals and Requirements

3.1 Describe firmware to Roots of Trust

A root of trust should be able to use an FMD to determine how to validate or measure regions of a firmware image.

3.2 Enable RTM, RTV, and RTU to operate on firmware images

In order to bootstrap trust in a measurement chain at boot, this proposal aims to define a platform-agnostic format to provide necessary data required by an RTM, RTV, or RTU.

3.3 Enable RTU to verify firmware updates

Verifying a payload at boot-time is very similar to verifying it for updates. An FMD should be able to define regions which should be verified. An RTU should be able to use this definition to verify a firmware update without additional information about the contents of the firmware image.

3.4 Enable RoT to enforce measurements

In some use-cases, it is necessary for an RoT to enforce a known good hash of a payload image; in particular, for secure update and secure boot. The FMD should be able to include sufficient information for an RoT to make a judgement of the validity of the image.

4. Background and Motivation

Roots of Trust are important for establishing trust in a system before using the system to perform important tasks. This proposal is particularly interested in two ways of establishing trust:

1. Validating and reporting the first mutable code which runs on a system before it runs.
2. Validating updates to first mutable code.

4.1 Describing Firmware to Roots of Trust

Many secure boot and measured boot schemes involve each piece of software in the boot chain measuring/validating the next stage. In order to verify the first piece of software in this boot chain, typically a firmware image, the root of trust needs to know how to parse and validate this firmware image.

Ideally, a firmware vendor should be able to use a single format to describe firmware to multiple roots of trust from different hardware vendors. Additionally, a system owner should expect each RoT to handle the image in the same way. For example, two different RTMs should produce the same boot measurement given the same firmware image and FMD. This allows system owners to have the same attestation model across heterogeneous machines.

In order to meet these needs, this proposal defines a descriptor format which all firmware vendors can produce for firmware images and which RoTs can read in order to validate these firmware images.

4.2 Separating “what to measure” from hardware

Without a firmware image that describes how it should be measured, the decision of what to measure is left to individual vendors. This may mean that system owners will see different measurements for the same software booted on different machines. Additionally, it may be challenging for system owners to change what is measured during boot to fit their attestation model.

The introduction of FMD allows system owners to produce an FMD which hardware RoTs can consume, but which don't need to be produced by the RoT vendors. This allows RoT vendors to produce hardware for broad use without restricting disjoint usages of this hardware.

4.3 Region-based Validation

A simple approach to describing a firmware image is to tell the RoT the start address and size of the firmware image. This would allow the RoT to hash the entire image and either validate or measure it. There are, however, a few issues with this approach in practice:

1. Firmware images often contain regions which are not static. Some regions are used as scratch space by boot code.
2. Not all code/data in a firmware image are controlled by the entity which signs and distributes the image. For example, some regions contain firmware blobs controlled (and separately signed/validated) by the CPU vendor.
3. In many cases, it is desirable during boot for the RoT to only validate first mutable code. This code can then contain more complex logic to validate subsequent boot stages. It is unnecessary for the root of trust to explicitly validate all the code in the boot medium.

In order to resolve these problems, FMD defines region groups. A region group is the set of regions which the RoT should measure/validate. A region group allows the RoT to operate on the relevant portion of the payload image, ignoring portions irrelevant to validation and measurement.

4.4 Supporting Multiple Use-Cases

In order to describe a payload image to an RoT, there is significant overlap in the information needed by an RTM, RTV, or RTU. For this reason, this specification proposes a single flexible format for describing firmware to each of these RoT types.

In particular, each of these require the following information about an image:

- A description of which parts of the image to measure or validate
- Metadata about the size and location of the descriptor
- (Optional) Integrity protection
- (Optional) A way to enforce the proper contents of the payload image.

Additionally, a single piece of software or hardware may be used to implement each of these RoT types. The use of a single format means that the same implementation can be used for parsing and operating on the descriptor.

5. Firmware Measurement Descriptor (FMD) Format

5.1 TLV Format

Each distinct structure in the Firmware Measurement Descriptor must begin with a Type-Length-Value (TLV) header.

BYTES	FIELD NAME	COMMENTS
0:1	Type	Type of this section.
2:3	Length	Length of this section (including this header).

4:5	Version	Version of this section.
6:7	uint16_t	reserved_0

This allows optional fields to be added to the descriptor which may be ignored by a RoT implementation. If the RoT's FMD parser does not understand a structure with a given type, it can skip that structure by reading the length field and moving to the next structure.

5.1.1 TLV Versioning

Every TLV section contains a version field. Any time a structural change is made to the TLV section, the corresponding version must be incremented. Examples of structural change include:

- A field is added or removed
- An existing field changes in size
- An existing field changes in semantic meaning

See section 5.9.2 for the current versions of all sections defined in this proposal.

5.2 FMD Container Format

An FMD is a collection of the above TLV structures. The FMD must contain exactly one fmd_header as the first structure. All other TLV sections must follow the header, and the total size of all TLV sections must not exceed the "Descriptor Size" field in the header.

Figure 5.2.1: Rough outline of format of an FMD

The TLV sections required to be present in an FMD depend on the use case. Each of the use cases—measurement, verification, and update—require specific structures to be present. An FMD which is intended to be used for multiple use-cases must include all the required structures for that case. More details for each use case can be found in the later "Use Case" sections.

5.3 Region Group

A Region Group consists of one fmd_region_info struct, followed by N fmd_region structs. FMD supports 3 types of region groups, MEASURE, VERIFY, and UPDATE. These types denote in which circumstances the regions in this group should be measured or verified.

All the regions in a region group must immediately follow the fmd_region_info struct.

An FMD shall not include multiple region groups of the same type. This means at most, an FMD can contain the following region groups:

1. One group of type FMD_REGION_GROUP_MEASURE
2. One group of type FMD_REGION_GROUP_UPDATE
3. One group of type FMD_REGION_GROUP_VERIFY

5.4 Region Group Hashing

Given an FMD region group, the hash produced by the RTM shall be

```
REGION_GROUP_HASH = HASH(REGION_0 || REGION_1 || … || REGION_N)
```

Where the hash function to be used is defined in the corresponding fmd_region_info struct for the Region Group.

It is important to not only measure the contents of a region, but also the offset and size of that region. This prevents attacks where the verifier approves of the contents of a region, but that region does not reside at the expected offset. For example, the descriptor may cover code at one location, but the CPU boots from a different location. To achieve this, the data to be hashed for a given region A at offset X with size Y shall be

```
REGION_N_HASH = X || Y || FLASH_CONTENTS[X … X+Y])
```

Only regions of type FMD_REGION_TYPE_STATIC should be included as part of the region group hash. Other region types are to support RoT handling of dynamic data.

Take as an example the following regions:

```
REGION_A = {
  start: 0x1000,
  size: 0x0100,
  type: FMD_REGION_TYPE_STATIC
}
REGION_B = {
  Start: 0x2000,
  size: 0x0010,
  type: FMD_REGION_TYPE_STATIC
}
REGION_C = {
  Start: 0x2000,
  size: 0x0010,
  type: FMD_REGION_TYPE_MIGRATE
}
```

The measurement for this region group shall be

```
REGION_GROUP_HASH = HASH(0x1000 || 0x0100 || FLASH[0x1000 … 0x1100] ||
                                        0x2000 || 0x0010 || FLASH[0x2000 … 0x2010])
```

5.5 Compound FMD

While this proposal lists 3 separate use cases for an FMD, a single FMD may be used to describe a payload image in all of these cases. To accomplish this, an FMD should include all the required sections for each use-case. An RoT which does not understand a section meant for a different use case can safely skip that section.

Figure 5.5.1: A compound FMD used for measurement, update, and verification

5.6 Descriptor Signature

Zero or more fmd_signature TLV sections may be included in an FMD. The purpose of allowing multiple signatures is to allow an FMD to be signed by multiple different keys or with multiple different algorithms.

5.6.1 Signature Verification

Verifying an FMD signature amounts to a standard signature verification over all TLV sections except for any fmd_signature sections.

Take the following FMD as an example:

In order to verify this FMD, the verifier should compute a hash over the signed TLV sections:

```
HASH(fmd_header || fmd_region_info || fmd_region).
```

Then using this hash and the fields in the fmd_rsa_signature struct, the verifier can do a standard RSA signature verification.

Note, the actual image contents pointed to by each fmd_region are not hashed during this step, this signature is only over the descriptor itself. After the validity of a signed FMD is determined, the it can be used by an RoT for one of the use cases described later in this document.

5.6.2 Verification Key

Each fmd_signature section contains the public key which should be used to verify the signature. This is to provide the most flexibility for how an RoT chooses to validate that a key is trusted.

For example, a resource constrained RoT could store hashes of trusted keys rather than the whole key.

Alternatively, an RTM could measure the key used for verification. This then allows a verifier to make the determination as to the trustworthiness of this key.

5.7 Data Alignment

All TLV section structures must be naturally aligned. Where natural alignment is not possible, explicit “reserved” members should be added to ensure natural alignment.

5.8 Endianness

All multi-byte fields are stored in big endian format.

5.9 TLV Sections

- fmd_header: The overall header of the Firmware Measurement Descriptor.
- fmd_region_group: Metadata about a Region Group.
- fmd_region: A region in a Region Group.
- fmd_payload_info: Information about the payload image described by this FMD.
- fmd_signature: A signature over all the sections in a given FMD other than this and other fmd_signature sections.

6. Use Case #1: Boot Measurement

While FMDs are intended to be a generic proposal consumable by any Root of Trust, this section provides sample architectures to depict how FMD may be used during boot by an RTM.

6.1 Measured Boot Background

In distributed systems, a common goal is to ensure a machine is in a verifiably good state before allowing it to handle potentially sensitive data and/or workloads. One way to address this problem is through measured boot.

During boot, each piece of code, from system firmware to OS components, will measure (take a hash of) the next component before transferring control. After boot is complete, these measurements can be used to prove to a Verifier what code was booted on the system.

The above sample boot stack roots trust in the system firmware. It is the responsibility of the first boot stage to measure the next boot stage into the Root of Trust; in this case a TPM.

1. CPU begins booting firmware from flash
2. Firmware measures a hash of the boot loader into the TPM
3. Firmware boots the boot loader
4. Boot loader measures the hash of the OS kernel into the TPM
5. Boot loader boots the OS kernel.

This sequence allows a later entity to query the TPM and to determine what software was booted on the machine. If the Verifier assumes the firmware is trusted and bug-free, then this will bootstrap trust and the Verifier can trust the subsequent measurements made into the TPM.

6.2 Motivation & Rooting Trust

The primary issue with the above measured boot architecture is that it assumes the system firmware is trusted. Because it is the first code that boots on the system, no previous software boot stage is measuring this code into the TPM.

A Verifier cannot attest to the system firmware image used during boot. If the version of running system firmware has any known security flaws, this cannot be detected by the verifier.

6.3 Required FMD Sections

In order for an RTM to consume an FMD, the following fields must be included:

FMD SECTION	COMMENTS
fmd_header	
fmd_region_info	This must be a region group of type FMD_REGION_GROUP_MEASURE
fmd_region[fmd_region_info.count]	

6.4 Performing TPM S-HRTM measurement using an FMD

Given a boot medium with a Firmware Measurement Descriptor, an RTM can make an S-RTM measurement into the TPM.

Note: This is a logical diagram and is not intended to depict any particular hardware topology.

Before the system boots and before the TPM2_Startup command is issued to the TPM, the RTM can perform the following steps to measure the first mutable code of the system and anchor a measurement chain.

1. RTM comes out of power-on reset.
2. RTM holds CPU in reset.
3. RTM locates FMD in flash and parses it.
4. RTM issues `_TPM_Hash_Start` command to TPM.
5. For each region in FMD, RTM issues `_TPM_Hash_Data` command to TPM with the region to be measured as described in section 5.4.
6. RTM issues `_TPM_Hash_End` command to TPM to initialize the value of PCR0.
7. RTM brings CPU out of reset.
8. CPU begins booting from flash, unimpeded by the RTM.

Following this initial measurement, PCR0 contains the measurement of first mutable code described by the FMD.

As an example, if the FMD covers the initial boot block of firmware, which in turn measures other mutable code, then the measurement chain is rooted to the RTM. A Verifier can verify compare this measurement to the expected hash of the initial boot block to ensure expected code was booted.

7. Use Case #2: Secure Firmware Update

7.1 Motivation

A standard example of a firmware update would have the existing firmware on a device validate an update to the next version. In this case, the existing firmware is acting the Root of Trust for update. For this reason, a runtime compromise of firmware can be persisted across firmware updates. A malicious or buggy firmware can

- Modify new firmware prior to update
- Pretend to perform an update, while in reality keeping old firmware
- Allow external access to the boot medium

Alternatively, a separate Root of Trust may mediate updates. A hardware RTU is less likely to be compromised than device firmware and can be trusted to update the device, even when the existing firmware is compromised. While a hardware RTU can provide a greater security promise, it makes it more complicated to validate firmware. A hardware RTU is expected to work for various types of firmware which have different formats and different contents.

To resolve this, an FMD can be constructed for any arbitrary firmware image. The FMD can be populated by any entity (firmware vendor, machine owner, etc.) to cover the proper regions to be validated during update.

7.2 Anti-Rollback Protection

7.2.1 Secure Version Number

In order to support anti-rollback protections, the fmd_payload_info section has an unsigned 32-bit Secure Version Number. This may be separate from any other versioning scheme used to describe the firmware payload.

This SVN shall be incremented by the FMD author whenever a security relevant change is made. The determination of what is a security relevant change is left up to the FMD author.

7.2.2 Minimum Acceptable Update Version

The Minimum Acceptable Update Version (MAUV) is the minimum payload SVN to which the RTU will update. It should be tracked by the RTU and enforced on updates. If an incoming update has an SVN in its FMD which is lower than the current MAUV, the update will be denied.

The MAUV can be updated by using the minimum_svn field in the payload_info section in the FMD of an incoming payload update. If the payload update is successful, and if the minimum_svn field is greater than the RTU's current minimum SVN, the RTU will update the internal MAUV to this new value. The updated MAUV will be enforced on all future payload updates.

7.3 Region Verification

Once the FMD has been properly verified, an RTU can use the FMD to validate regions of the payload. The RTU should compute the hash of the regions in the FMD_REGION_GROUP_UPDATE region group, as described in section 5.4.

The computed hash can then by compared to the fmd_region_group.expected_hash field to ensure the payload contents are correct. If the computed hash and expected_hash match, then the payload update can be safely applied.

7.4 Region Migration

In the fmd_region sections, a region may be marked as FMD_REGION_TYPE_MIGRATE. A region of this type should not be included when computing the hash of the FMD_REGION_GROUP_UPDATE Region Group. This feature is included to allow the RTU to preserve mutable data across updates. During update, the RTU should preserve these regions as they are in the destination instead of copying the contents of the update image.

7.5 Required FMD Sections

FMD SECTION	COMMENTS
fmd_header	
fmd_region_info	
fmd_region[fmd_region_info.count]	This must be a region group of type FMD_REGION_GROUP_UPDATE.
fmd_payload_info	
fmd_signature	

8 Use Case #3: Secure Boot

8.1 Motivation

As in measured boot, most secure boot schemes have each boot stage verify the next. If verification fails at any stage the system will refuse to boot. Also similar to measured boot, without a separate root of trust, the first mutable code to boot on the system is implicitly trusted.

An RTV which verifies first mutable code can use a signed FMD to verify first mutable code prior to boot in a similar way to how an RTU verifies updates.

8.2 Region Verification

In order to verify first mutable code during boot, the RTV should do the following:

1. Verify the FMD signature.
2. Hash the regions in the FMD_REGION_GROUP_VERIFY Region Group.
3. Compare the computed hash to the hash in the fmd_hash section.
4. If the hashes do not match, the RTV should not allow the system to boot. Otherwise, the system can continue normal boot.

8.3 Required FMD Sections

FMD SECTION	COMMENTS
fmd_header	
fmd_region_info	This must be a region group of type FMD_REGION_GROUP_VERIFY.
fmd_region[fmd_region_info.count]	
fmd_signature	