Cerberus 项目：证明规范

作者：

- Christopher Weimer，微软首席固件工程师
- Akram Hamdy，微软高级固件工程师

修订记录

- v1.00 (2022-07-01)
    - 初始版本。

概括

Cerberus Challenge Protocol 的目标是确保系统中的所有组件上只运行经过身份验证和兼容的固件. 这是通过直接测量受保护的固件和远程设备的测量挑战来实现的。Cerberus Challenge Protocol提供了确保平台的完整性和真实性所必需的原语，但没有提供有关如何实现这一点的许多细节。本规范提供了这些细节。

Device Measurements - 设备测量

任何证明流程的基础是维护和报告设备加载的固件和配置的测量值的能力。在 Cerberus 设备中，测量通过平台测量寄存器 (PMR) 报告，其具有与 TPM PCR 相似的行为和属性。

每个 PMR 代表一个单独的测量，由多个单独的组件扩展。扩展操作遵循与 TPM PCR 相同的公式，因此 PMR 值表示为“PMRnew = HASH (PMRold || Data)”。每个 Cerberus 设备之间用于生成 PMR 和初始 PMR 值（即执行任何扩展操作之前的 PMR 值）的散列算法可能不同。

Platform Measurement Register(PMR) - 平台测量寄存器

Cerberus Challenge Protocol 允许 5 个 PMR 值（PMR0 - 4），每个值都可以根据具体实现来衡量不同类型和数量的数据。不要求设备支持所有 5 个 PMR，但每个设备都必须至少支持 PMR0。除此之外，它们的使用方式有很大的灵活性。此处记录的内容被认为是不同类别 Cerberus 设备的“最佳实践”。

PMR0

作为唯一需要的 PMR，每个设备都必须生成此测量以供证明，并且期望包含此测量中的元素的最小集合。根据特定实现的要求，可以有其他条目，但 PMR0 必须至少包含以下测量列表。

- 已加载的所有可变固件层。遵循 TCG DICE 引导模型，需要测量所有层 0 到 N，其中层 N 是正在运行的应用程序固件。
- 安全启动密钥。这包括硬件支持的根密钥和用于加载后续引导阶段的密钥清单。
- 当前的反回滚计数器。

仅证明其自身固件状态的简单 RoT 设备可能没有 PMR0 以外的任何 PMR。

外部 RoT

外部信任根是一种将插入处理器的非易失性固件存储（例如 SPI 闪存）并代表处理器验证和证明固件状态的设备。由于外部 RoT 证明的不仅仅是它自己的状态，它需要使用更多可用的 PMR。

对于这些设备，PMR0 应该包含 RoT 设备本身的固件测量值。它不应包括任何清单或应用于设备的其他配置的测量。这为证明 RoT 状态提供了一致的测量。

PMR1 和 PMR2 应包含任何可配置状态的测量值。这将包括设备用于验证处理器固件的任何 PFM 以及验证结果。在这两个 PMR 中实际如何组织测量并不重要。例如，在 RoT 保护两个处理器的情况下，设备同样可以将所有测量值放在单个 PMR 中，或者将一个处理器放在 PMR1 中，另一个放在 PMR2 中。

提供 PMR3 和 PMR4 以提供与 TPM 类似的功能。处理器可以将自己的数据扩展到 Cerberus 管理的 PMR 中。然后可以将这些 PMR 的状态用作证明声明的一部分。

平台 RoT

PA-RoT（平台主动信任根）负责证明系统内所有组件的健康状况。它可能也可能不是外部 RoT，但它对 PMR 的使用有类似的期望。

PMR0 将是 PA-RoT 设备的固件测量值，PMR1 和 PMR2 将用于报告系统组件的配置和认证状态。这包括用于证明组件和证明结果的任何 PCD 或 CFM。

PMR3 和 PMR4 在此配置中可能是必需的，也可能不是必需的。根据实现方式，可能仍然希望系统中的处理器能够进行测量以进行扩展。

内在根

固有信任根是 SoC 封装内和芯片上的设备，但与主应用程序处理器分开。内在 RoT 强制执行安全启动和对加载到应用处理器中的所有固件进行测量，并且可以选择性地提供对来自应用处理器运行时的额外测量的管理。由于内在 RoT 嵌入在设备中，因此它可能不会像外部 RoT 那样使用 PFM。

PMR0 将仅包含内部 RoT 固件的测量值。其他 PMR (1-4) 可以任意组合使用，以报告应用处理器的固件和安全状态的测量结果。为 PMR0 中的 RoT 测量的相同类型的信息应该为应用处理器测量。如果 RoT 公开可由应用程序处理器扩展的测量，它将为此目的使用 PMR3 或 PMR4。

如果固有 RoT 也被用作 PA-RoT 或证明系统内的其他组件，则需要分配至少一个 PMR 来测量 PCD、CFM 和证明结果。

证明测量

Cerberus Challenge Protocol提供有状态（`Challenge`/`Get PMR`）和无状态（`Message Unseal`）机制来证明设备测量。在证明命令中，每个设备只需要挑战，因为这可能是证明简单 AC-RoT 所需的唯一命令，该简单 AC-RoT 仅包含 PMR0 中的测量值。强烈建议支持额外 PMR 的设备实施“获取 PMR”。鼓励 PA-RoT 设备实施“消息解封”，为分布式证明工作流程提供灵活性。

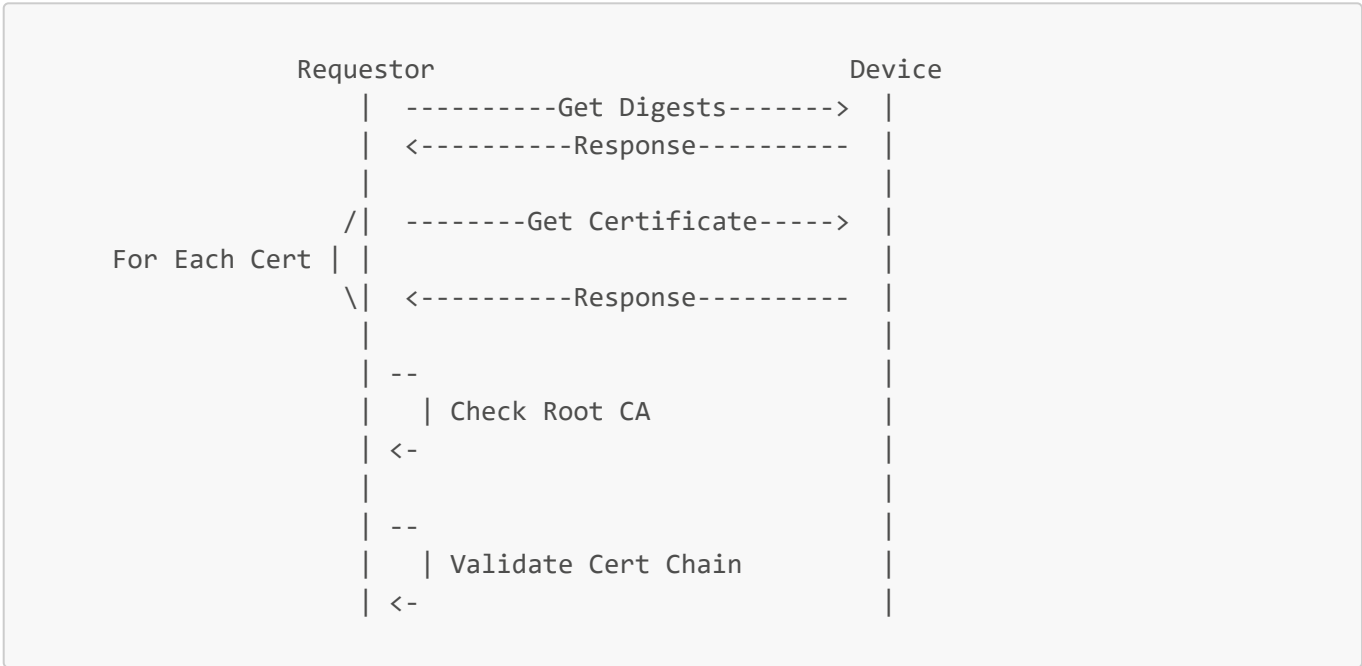
本节仅描述安全证明测量的机制。它没有说明如何知道报告的测量值是否正确。在系统中，这由 CFM 处理，稍后将详细描述。

设备证书

所有证明工作流程都使用由设备的 DICE 身份证书认可的设备唯一密钥。使用哪个设备密钥取决于正在执行的特定证明流程和设备实现细节，但检索和验证此密钥的过程是相同的。Cerberus 最多支持 8 个不同的证书链（插槽），但只需要设备支持单个证书链即可支持证明。

获取和验证设备的证书链涉及执行“获取摘要”请求，然后按以下顺序执行多个“获取证书”请求。

1. 为目标证书链发出“Get Digests”。这提供了链中每个证书的哈希值，可用于两个目的。 首先，它告诉请求者这个链中到底有多少证书。其次，它为请求者提供了一种方法来缓存来自先前会话的信息并检测证书是否已更改。缓存证书信息时，请求者可以缓存完整证书、每个证书的摘要或整个链的摘要，并能够验证报告的证书链是否与先前验证的相匹配。
2. 为链中的每个证书颁发“获取证书”。最后一个证书将是用于证明请求的密钥的最终实体证书。如果存在与当前摘要匹配的有效缓存，则可以跳过任意数量的这些请求。
3. 验证根 CA 是否可信。这种信任如何建立取决于请求者。另一个 Cerberus 设备将使用 CFM 中的信息来确定根 CA 是否可信。
4. 一旦知道根 CA 是好的，X.509 路径验证将在整个证书链上执行，证明认证证书已被根 CA 认可。
5. 如果证书链验证失败，则设备不可信。此设备不需要进一步证明。



一旦获得受信任的设备证书，它就可以用作安全交换的一部分来证明设备状态。

挑战/获得PMR

使用“挑战”或“获取 PMR”命令证明 Cerberus 设备是检查 PMR 值的最简单机制。这两个命令的响应仅返回 PMR 的原始值。在“挑战”的情况下，返回的 PMR 是 PMR0。使用 `GetPMR`，可以查询设备支持的任何 PMR。请求中的新鲜度随机数和使用先前获取的证明密钥的签名提供与特定设备的绑定和抗重放。

消息解封

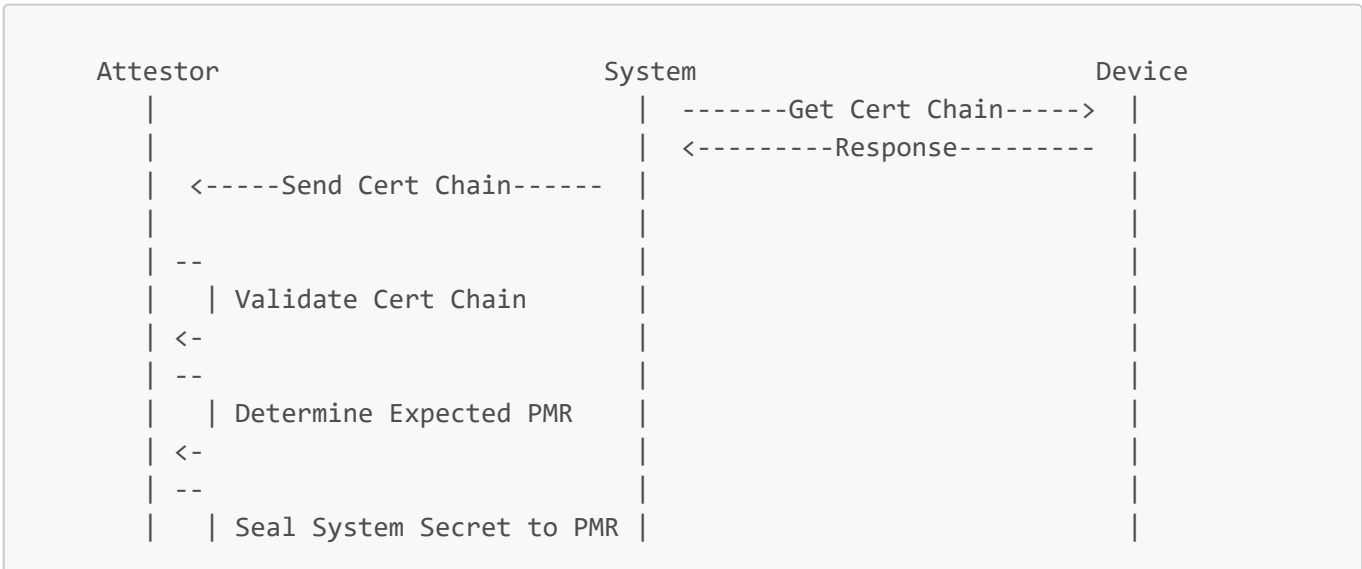
`Message Unseal` 也可以证明设备 PMR 的状态，但不一定直接从设备获取 PMR 值。如果证明者拥有系统正常运行所需的一些秘密或其他工件，则此流程有效。外部证明者可以使用设备证书的知识和预期的 PMR 值来将此系统秘密密封到预期的 PMR 值。如果具有匹配身份证书链的设备具有预期的 PMR 值，目标系统将只能解封此秘密。使用以下顺序。

1. 接收并验证设备证书链。

2. 确定设备的一组预期 PMR 值。
3. 生成随机种子。
    - 如果 Seal/Unseal 将使用 ECDH，则此种子是使用随机 ECC 密钥对和 Cerberus ECC 公钥的 ECDH 共享秘密。
    - 如果 Seal/Unseal 将使用 RSA，则此种子是一个随机数，然后使用 Cerberus RSA 公钥对其进行加密。
4. 从种子中，使用 NIST800-108 计数器模式派生一个 256 位 HMAC 签名密钥。
    - Ki = 种子 -标签="签名密钥"
    - 没有上下文
    - r = 32 -`key = HMAC (种子, 0x00000001 || 签名密钥 || 0x00 || 0x00000100)`
5. 从种子中，使用 NIST800-108 计数器模式导出 256 位对称加密密钥。
    - Ki = 种子 -标签="加密密钥"
    - 没有上下文
    - r = 32 -`key = HMAC (种子, 0x00000001 || 加密密钥 || 0x00 || 0x00000100)`
6. 使用加密密钥对系统机密进行加密。加密方式和算法可由平台自行定义。如果需要不同的密钥长度，可以通过额外的 KDF 进一步变形 256 位加密密钥以派生最终加密密钥。
7. 生成密封策略。这是一个包含五个 64 字节值的数组，每个值对应一个 PMR。由于测量未使用完整的 64 字节分配而导致的未使用字节应设置为 0。未使用的 PMR，无论是因为设备不支持它还是因为 PMR 的值无关紧要，应设置为 0。此数组映射直接到"Message Unseal"请求结构中的"PMRx Sealing"值。
8. 计算密封负载的 HMAC 签名，使得 `sig = HMAC (signing key, ciphertext || sealing policy)`
9. 将加密的系统秘密、密封策略、HMAC 签名和种子发送到目标系统。
    - 如果使用 ECDH 密封，则种子是证明者生成的随机 ECC 公钥。
    - 如果使用 RSA 密封，则种子是 RSA 加密的随机数。
10. 一旦目标系统收到完整的数据包，它必须执行"Message Unseal"请求，将这四个数据提供给 Cerberus 设备。
11. Cerberus 设备将使用提供的种子重新生成相同的签名和加密密钥，并在有效负载上验证 HMAC 签名。如果 HMAC 签名有效且 PMR 值与密封策略中指定的值匹配，则将加密密钥返回给请求者。
12. 然后目标系统可以使用加密密钥来解密系统秘密或执行任何其他需要该密钥的必要操作。

注意：此流程生成的加密密钥可以从设备以明文形式发送。该设备还将基于相同的种子重新生成相同的加密密钥，只要 PMR 值保持不变，而无需了解谁在发送请求或此有效负载已被解封多少次。

由于这些原因，加密密钥必须被认为是一次性使用的密钥，并且用密钥密封的系统秘密也需要是一次性使用的。如果其中任何一个被违反，系统就会通过重放受到损害。



```
        | <-                                |                                   |
        |                                   |                                   |
        |  ----Send Sealed Secret---->      |                                   |
        |                                   |  -------Message Unseal----->      |
        |                                   |  <---------Response---------      |
        |                                   |                                   |
        |                                   |  ---Message Unseal Result-->      |
        |                                   |  <---------Response---------      |
        |                                   |                                   |
        |                                   | --                                |
        |                                   |   | Decrypt System Secret         |
        |                                   | <-                                |
```

“消息解封”流程的一个重要方面是证明者必须知道设备的 PMR 值。此外，它必须确切地知道这些值是什么，而不仅仅是可接受值的范围。Cerberus Challenge Protocol提供了多种机制来确定 PMR 状态。或者，可以向证明者提供足够的离线知识以了解这些值应该是什么。

- **获取 PMR**
  获取设备 PMR 值的第一个也是最明显的方法是发出“获取 PMR”请求。

- **证明日志**
  Cerberus 设备可以选择提供为每个 PMR 记录的所有测量的证明日志。使用“获取日志”请求，可以为设备检索证明日志并将其与证书链一起发送给证明者。证明日志不仅包含每个 PMR 的最终值，还包含促成该值的每个测量值。这为证明者提供了更多粒度来验证 PMR 状态。

  如果证明日志本身不足以验证 PMR 状态，则可以检索证明日志中每次测量的实际数据。如果设备支持“获取证明数据”请求，则可以针对证明日志中的每个条目发出此请求，并发送给证明者。

- **TCG 日志**
  如果 Cerberus 设备支持，“获取日志”请求可用于检索符合 TCG 标准的测量日志，该日志可发送给证明者。此日志将包含与 Cerberus 证明日志和“获取证明数据”的组合相同的信息。

清单

Cerberus 清单是应用于设备以驱动固件的身份验证和证明的配置文件。不同的清单类型适用于不同的场景，但基于相同的基本结构。

清单的设计预期它们将存储在设备外部的闪存中，并且设备中没有足够的内存来在任何时候将整个清单存储在 RAM 中。将整个清单放在 RAM 中显然是可行的，但可能有更有效的方法来构建此用例所需的数据。

一般清单结构

每个清单至少包含三个主要部分：清单标题、目录和签名。每个清单都以固定长度的标头开头，以签名结尾，签名的长度在标头中指定。

仅使用标题信息，就可以验证完整的清单。目录紧跟在清单标题之后，描述了清单中包含的所有信息。

清单本身由数量可变的元素组成，其类型和顺序在目录中指定。目录是一个可变长度的结构，包含一个标题、清单中每个元素的一个条目、元素的可选散列列表以及可用于独立验证表数据的整体表散列。

清单中的保留字段应设置为 0，但解析器不会验证此值。清单的未来版本可能会对某些数据使用保留字段，并且为了保持向后兼容性，解析器不得拒绝这些字段中具有非零值的清单。这同样适用于指定字段内的保留位。

所有大于一个字节的字段都存储在小端。这还包括不落在字节边界上的字段。为了简化解析，不是偶数倍字节的字段被分解以与内存字节边界对齐。例如：

- 一个 12 位字段 [11:0]，从内存字节边界开始存储为
  `byte0[7:0] = 数据[7:0]`
  `byte1[7:4] = 数据[11:8]`
- 一个 12 位字段 [11:0]，从一个字节的中间开始存储为
  `byte0[3:0] = [3:0]`
  `byte1[7:0] = [11:4]`

一个例外是字节数组的字段，尤其是摘要。除非另有说明，否则所有摘要和其他字节数组都存储在大端。

所有长度都以字节表示。

清单头

位域Manifest.Header

类型	名称	说明
16	总长度	清单中所有数据的总长度，包括签名。
16	清单类型	指示清单类型的标识符。
32	version_id	清单的版本标识符。版本 ID 是一个单调递增的值，防止回滚到早期的清单版本。
16	signature_length	附加到清单数据末尾的签名长度。
公钥类型	public_key_type	用于生成清单签名的公钥类型。
KeyStrength	key_strength	用于生成清单签名的密钥强度。
哈希类型	哈希类型	用于生成清单签名的哈希算法。
0x00	_	预订的。

枚举 Manifest.PublicKeyType

价值	名称	说明
00b	rsa	公钥是 RSA。
01b	ecc	公钥是ECC。

枚举 Manifest.KeyStrength

价值	名称	说明
000b	rsa_2k_ecc_256	密钥是 2048 位 RSA 或 256 位 ECC。
001b	rsa_3k_ecc_384	密钥是 3072 位 RSA 或 384 位 ECC。

价值	名称	说明
010b	rsa_4k_ecc_521	密钥是 4096 位 RSA 或 521 位 ECC。

枚举清单.HashType

价值	名称	说明
000b	sha2_256	哈希是使用 SHA2-256 计算的。
001b	sha2_384	哈希是使用 SHA2-384 计算的。
010b	sha2_512	哈希是使用 SHA2-512 计算的。

清单目录

位域Manifest.TOC.Header

类型	名称	说明
8	entry_count	表中的条目数。
8	哈希计数	表中具有关联散列的条目数。不需要所有条目都具有散列。
00000b	_	填充未使用的 HashType 位。这必须是 0。
哈希类型	哈希类型	指定为表验证添加的哈希类型。这也是用于生成单个元素哈希的哈希类型。
0x00	_	预订的。

清单中的每个元素都必须在目录中有一个条目。

条目必须根据清单中的位置排序。

位域Manifest.TOC.Entry

类型	名称	说明
8	type_id	指示元素类型的标识符。
8	父母	表示此元素的父级的类型 ID。此元素之前具有匹配类型 ID 的第一个元素是父元素。如果元素没有父元素，这将被设置为“0xff”。
8	格式	元素中包含的数据的格式版本。新格式版本必须向后兼容旧版本。字段不能移动或重新排序。
8	hash_id	标识符指定此元素在哈希表中的索引。元素散列是可选的。哈希 ID 等于或大于表中哈希的总数表示该元素没有哈希。为此，建议使用散列 ID“0xff”。
16	偏移量	元素数据所在的清单开头的偏移量。
16	长度	元素数据的长度。

在元素条目之后，有一个用于元素验证的哈希表。该表是一个大小相等的散列数组，其长度由目录标题中指定的散列算法确定。必须使用相同的算法计算所有哈希值。具有散列的元素必须在使用数据之前验证散列，即使内存中只需要一部分元素数据。哈希是在目录条目中指定的整个长度上计算的。

目录可以通过几种不同的方式使用，具体取决于性能要求和内存容量。

1. 为了优化内存消耗，在清单验证期间将缓存目录标头和哈希。目录的其余部分和清单元素保留在外部闪存中。读取的每个元素都需要一系列操作。

    - 从闪存中读取每个元素条目以找到所需的条目。
    - 读取所需条目的散列条目（如果可用）。
    - 确保闪存上的所有目录数据作为这些读取的一部分进行哈希处理。
    - 将计算的目录哈希与缓存的哈希进行比较以确保有效性。
    - 读取并散列元素数据。与目录中的散列条目进行比较以确保有效性。

    此流程的性能较低，但只需要少量数据存储即可处理任何清单。

2. 如果设备内存充足，清单验证时可以完整读取目录并缓存，以便以后的元素访问更快。元素访问不需要重新验证目录，只需要验证元素数据。

    这种方法可能不适用于内存受限的设备，因为使用 SHA2-512 的具有 255 个条目和 255 个哈希值的目录需要大约 18k 内存的缓存。

清单元素

虽然在大多数情况下元素的顺序和数量是可变的，但元素可以限制在特定的上下文中。某些元素可能只是顶级结构的一部分（即，它不会有任何父元素），而其他元素则需要特定类型的父元素。错误上下文中的元素将被解析器忽略。此外，元素可以定义为单例。

单例元素的重复条目将被忽略。所有标识符字符串都限制在 255 个字节以内，并且没有任何终止符存储在清单中。

定义了一组基本元素，这些元素在所有清单类型中都是通用的。特定清单可以定义其他类型。

类型编号	元素类型	格式	顶级	单例	说明
0x00	平台ID	1	×	×	使用此清单的平台的标识符。
0x01 - 0x0f	保留				保留以供将来使用。
0xe0 - 0xfe	供应商扩展				为供应商特定的元素类型保留。
0xff	不可用				不能用作元素类型，因为它用作父/子关系的一部分。

平台 ID 元素

Platform ID 元素用于将特定清单关联到特定类型的平台。它还可以作为识别特定清单的辅助方法。由于清单 ID 只是整数，并且可能在不同类型和不同系统的清单之间重叠，因此清单 ID 和平台 ID 的组合提供了一种唯一标识特定清单的方法。此外，在清单验证期间使用平台 ID 以确保仅使用同一平台的清单来替换现有清单。

虽然清单结构不需要任何特定元素的存在，但如果清单不包含平台 ID 元素，清单解析器可能会拒绝该清单作为无效清单。

位域清单.PlatformID

类型	名称	说明
8	plat_id_length	平台 ID 字符串的长度。
0x000000	_	预订的。
plat_id_length	plat_id_string	平台 ID 字符串。这是一个非空终止的 ASCII 字符串。
对齐 (32)	_	零填充。

平台固件清单

外部 RoT 设备使用平台固件清单 (PFM) 来验证处理器固件存储的内容。PFM 结构是为 SPI 闪存设备量身定制的，尽管相同的结构可能适用于其他存储类型。

PFM 包含可由处理器加载的允许固件版本列表，并提供验证图像所需的信息。该结构很灵活，可以考虑 Flash 的许多不同内容和布局。在简单的情况下，一个连续的闪存块中包含一个固件映像，但在某些情况下，一个闪存设备可能有多个固件组件，每个组件都由处理器独立加载。所有这些场景都可以在 PFM 中描述。

PFM 将在清单标头中报告 0x706d 的 manifest_type。PFM 定义了以下额外的清单元素类型。

类型编号	元素类型	格式	顶级	单例	说明
0x10	闪存设备	0	×	×	定义与整个闪存设备相关的全局信息。
0x11	固件	1	×		定义闪存上存在的单个固件。这仅在清单中有 Flash Device 元素时才有效。
0x12	固件版本	1			单一版本固件的说明。这将仅用作固件元素的子元素。

闪存设备元素

每个 PFM 旨在描述单个闪存设备的内容。Flash Device 元素向 RoT 提供关于物理闪存的整体信息，这可能是内容验证所必需的。

注意：在为单个处理器使用两个冗余闪存的配置中，两个闪存设备将有一个 PFM。预计 RoT 会在任何时间点管理对一个闪存或另一个闪存的访问。任一设备的布局和验证应该相同。

位域 PFM.FlashDevice

类型	名称	说明

类型	名称	说明
8	空白字节	表示闪存未使用字节的值。在验证期间，将根据此值检查所有未使用的闪存区域，以确保没有存储意外数据。
8	fw_count	闪存中存储的固件组件数。
0x0000	_	预订的。

固件元素

Firmware 元素描述闪存设备的逻辑分区，代表单个可更新的部分。大多数处理器只需要一个固件元素，因为只有一个映像从闪存加载。然而，一些更复杂的 SoC 可能有多个独立的内核，每个内核都加载固件。在这种情况下，可以对闪存进行分区，以便单独验证每个组件。

位域 PFM.Firmware

类型	名称	说明
8	version_count	此固件组件允许的版本数。
8	fw_id_length	固件组件标识符的长度。
0000000b	_	预订的。
更新应用	run_time_update	指示是否可以在不重新启动的情况下应用更新的标志。
0x00	_	预订的。
fw_id_length	fw_id_string	固件组件的 ASCII 字符串标识符。这不是空终止的。
对齐 (32)	_	零填充。

枚举 PFM.Firmware.UpdateApplied

价值	名称	说明
0b	重启	更新仅在处理器重启时应用。
1b	运行时间	可以在不通知 RoT 重启的情况下应用更新。

固件版本元素

固件版本元素描述了验证单个固件版本所需的所有信息。每个固件组件都可以在一个 PFM 中支持多个版本。如果闪存上的固件与固件版本元素中包含的信息不匹配，则认为它是不受信任的。

位域 PFM.FirmwareVersion

类型	名称	说明
8	img_count	此版本中包含的签名图像数。
8	rw_count	此版本使用的 R/W 数据区域数。
8	version_length	版本标识符字符串的长度。

类型	名称	说明
0x00	_	预订的。
32	版本地址	闪存上存储版本字符串的地址。这必须在每次都经过验证的签名图像中。
version_length	version_string	版本的 ASCII 字符串标识符。这不是空终止的。
对齐 (32)	_	零填充。
RWRegion(rw_count)	rw_regions	包含 R/W 数据因此无法静态验证的闪存区域列表。
SignedImage(img_count)	图像	必须用于验证闪存内容的散列列表。

位域 PFM.FirmwareVersion.RWRegion

类型	名称	说明
000000b	_	预订的。
RWO操作	auth_fail_op	当更新验证失败时，RoT 应该对该 R/W 区域执行的操作。
24	_	预订的。
地区	地区	包含 R/W 数据的闪存区域。

位域 PFM.FirmwareVersion.SignedImage

类型	名称	说明
00000b	_	填充未使用的 HashType 位。这必须是 0。
Manifest.HashType	哈希类型	指定用于图像身份验证的哈希类型。
8	region_count	要为此图像散列的闪存区域数。
0000000b	_	预订的。
必须验证	验证	指示是否应在每次启动时验证映像的标志。
0x00	_	预订的。
哈希类型	img_hash	图像数据的预期哈希值。
地区 (地区计数)	_	应散列的闪存区域列表。

位域 PFM.FirmwareVersion.Region

类型	名称	说明
32	开始地址	定义区域内的第一个有效地址。
32	end_addr	定义区域内的最后一个有效地址。

枚举 PFM.FirmwareVersion.MustValidate

价值	名称	说明
0b	更新	固件映像是从启动时验证的映像链式加载的，因此只有在有更新时才验证它。
1b	each_boot	在每次系统启动时验证固件映像。

枚举 PFM.FirmwareVersion.RWOperation

价值	名称	说明
00b	没什么	不对 R/W 区域执行任何操作。
01b	恢复	如果可能，从备份中恢复区域。
10b	擦除	擦除整个 R/W 区域。
11b	_	保留（什么也不做）。

PFM 生成的 XML 表示

每个固件组件的每个版本都需要创建一个可用于生成 PFM 的元数据文件。这是一个使用 XML 存储元数据的示例，生成脚本可以使用这些元数据来创建二进制格式。

```
<固件类型=“标识符”版本=“标识符”平台=”标识符”>
    <版本地址>0x00000000</版本地址>
    <UnusedByte>0xff</UnusedByte>
    <RuntimeUpdate>假</RuntimeUpdate>
    <读写>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
            <OperationOnFailure>无</OperationOnFailure>
        </区域>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
            <OperationOnFailure>恢复</OperationOnFailure>
        </区域>
    </读写>
    <签名图片>
        <哈希>
            0x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f
        </哈希>
        <哈希类型>SHA256</哈希类型>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
        </区域>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
        </区域>
        <ValidateOnBoot>真</ValidateOnBoot>
```

```
    </签名图片>
    <签名图片>
        <哈希>

0x000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f2021222324252627
28292a2b2c2d2e2f
        </哈希>
        <哈希类型>SHA384</哈希类型>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
        </区域>
        <地区>
            <开始地址>0x00000000</开始地址>
            <结束地址>0x00000000</结束地址>
        </区域>
        <ValidateOnBoot>假</ValidateOnBoot>
    </签名图片>
</固件>
```

领域	说明
类型	固件组件字符串标识符。
版本	固件版本字符串标识符。
平台	平台字符串标识符。
版本地址	版本字符串标识符的闪存地址。
未使用的字节	在未使用的闪存区域中检查的字节。
运行时更新	指示固件是否可以在运行时更新的标志。
读写	定义单个 R/W 数据区域。
地区	定义连续闪光区域。
开始地址	区域内的第一个有效地址。
结束地址	该区域内的最后一个有效地址。
OperationOnFailure	验证失败后对 R/W 数据区域执行的操作。
SignedImage	定义需要身份验证的单个图像。
哈希	图像的预期哈希值。
哈希类型	使用的哈希类型。

XML 枚举的允许值

具有默认值的枚举表示该标记是可选的。如果它不存在于 XML 中，将使用默认值。

操作失败

- 无（默认）
- 恢复
- 擦除

哈希类型

- SHA256（默认）
- SHA384
- SHA512

平台配置数据

平台配置数据 (PCD) 清单用于自定义 Cerberus 设备并指定策略信息。此外，PCD 用于通知 RoT 它所在的平台以及如何联系其他组件以在平台上进行证明。此清单对于每个平台都是唯一的，并且独立于 PFM 和 CFM 应用。

PCD 将在清单标头中报告 `0x1029` 的 `manifest_type`。PCD 定义了以下额外的清单元素类型。

类型编号	元素类型	格式	顶级	单例	说明
`0x40`	`RoT`	2	×	×	提供设备的一般配置。
`0x41`	`SPI 闪存端口`	1			SPI 闪存上的单个受保护固件存储。这是 RoT 元素的子元素。
`0x42`	`I2C 电源管理控制器`	1	×		定义 RoT 使用的电源管理设备。
`0x43`	`具有直接 I2C 连接的组件`	1	×		定义通过 I2C 直接连接到设备的 AC-RoT。
`0x44`	`带有 MCTP 桥接连接的组件`	1	×		定义通过 MCTP 网桥连接的 AC-RoT。

根元素

RoT 元素包含 Cerberus 设备的配置选项及其直接保护的固件。RoT 通过 I2C 与 BMC 和其他设备通信。

`位域 PCD.RoT`

类型	名称	说明
`0000000b`	`_`	预订的。
`RoTType`	`rot_type`	指示证明层次结构中 RoT 的类型。
`8`	`port_count`	直接受外部 RoT 保护的外部处理器的数量。
`8`	`components_count`	RoT 必须证明的远程组件的数量。
`8`	`rot_address`	RoT 设备的物理总线地址。

类型	名称	说明
8	rot_default_eid	RoT 应使用的默认 MCTP EID。外部 MCTP 网桥可以在运行时分配不同的 EID。
8	bridge_address	连接的 MCTP 桥的物理总线地址。
8	bridge_eid	MCTP 网桥的 EID。
0x00	_	预订的。
32	attestation_success_retry	设备成功证明后等待重新证明的持续时间（以毫秒为单位）。
32	attestation_fail_retry	设备认证失败后等待重新认证的持续时间（以毫秒为单位）。
32	discovery_fail_retry	设备发现步骤失败后等待重试的持续时间（以毫秒为单位）。
32	mctp_ctrl_timeout	MCTP 控制协议响应超时时间（以毫秒为单位）。
32	mctp_bridge_get_table_wait	如果 MCTP 网桥不支持动态 EID 分配，RoT 启动后在向 MCTP 网桥发送 MCTP 获取路由表请求之前等待的持续时间（以毫秒为单位）。如果设置为 0，RoT 将只等待 EID 分配。
32	mctp_bridge_additional_timeout	除了 MCTP 桥接导致的设备超时时间外，还要等待的时间（以毫秒为单位）。
32	attestation_rsp_not_ready_max_duration	当 SPDM 设备响应 ResponseNotReady 错误时，这是 RoT 在重试请求之前允许等待的最长持续时间（以毫秒为单位）。
8	attestation_rsp_not_ready_max_retry	每个请求允许的 SPDM ResponseNotReady 重试的最大次数。如果超出此范围，RoT 将无法通过设备认证。
0x000000	_	预订的。

枚举 PCD.RoT.RoTType

价值	名称	说明
0b	PA-RoT	平台主动信任根。
1b	AC-RoT	主动组件信任根。

SPI 闪存端口元素

SPI 闪存端口元素表示直接受 RoT 保护的单个外部处理器。该处理器的固件存储在 SPI 闪存上。

位域 PCD.SPIFlashPort

类型	名称	说明
8	port_id	端口标识符。这映射到协议消息中的端口 ID 参数。
0b	_	预订的。
HostResetAction	host_reset_action	RoT 在主机重置时采取的操作。
看门狗监控	watchdog_monitoring	指示端口是否支持看门狗监控以触发恢复流。
RuntimeVerification	runtime_verification	指示端口是否支持固件更新的运行时验证。
FlashMode	flash_mode	包含固件的闪存设备的配置。
重置控制	重置控制	应与外部处理器一起使用的复位控制行为。
0000000b	_	预订的。
政策	政策	身份验证失败时应采取的操作。
8	pulse_interval	如果端口使用脉冲复位控制，复位信号的脉冲宽度。该值是 10 毫秒的倍数。
32	spi_frequency_hz	用于闪存通信的 SPI 时钟频率。该值以赫兹表示。

枚举 PCD.SPIFlashPort.HostResetAction

价值	名称	说明
0b	无	在主机重置时，RoT 不会采取任何其他操作。仍将执行正常的复位处理。
1b	reset_flash	在主机重置时，始终重置主机闪存。这是对任何正常重置处理的补充，意味着 RoT 将参与每个主机重置。

枚举 PCD.SPIFlashPort.WatchdogMonitoring

价值	名称	说明
0b	禁用	不支持看门狗监控。
1b	启用	支持看门狗监控。

枚举 PCD.SPIFlashPort.RuntimeVerification

价值	名称	说明
0b	禁用	不支持运行时验证。
1b	启用	支持运行时验证。

枚举 PCD.SPIFlashPort.FlashMode

价值	名称	说明
00b	双重	SPI 总线连接到两个用于存储固件的闪存设备。
01b	单身	处理器固件只有一个闪存设备。
10b	dual_filtered_bypass	用于固件的双闪存芯片。在这种模式下，RoT 保护永远无法完全禁用。即使在旁路模式下，插入的 RoT 也会阻止一些不需要的命令。
11b	single_filtered_bypass	与“dual_filtered_bypass”相同，但固件只有一个闪存设备。

枚举 PCD.SPIFlashPort.ResetControl

价值	名称	说明
00b	通知	复位控制信号不直接引起处理器的复位。相反，它通知处理器将在下一次重置时进行的待处理固件身份验证。在主机复位期间，释放复位控制信号以指示身份验证何时完成。
01b	重置	复位控制信号直接引起处理器的复位。处理器在验证期间保持复位状态。
10b	脉冲	复位控制信号直接引起处理器的复位。在验证期间，处理器在没有闪存访问的情况下保持运行，并且在验证后以脉冲方式进行复位。
11b	_	预订的。

枚举 PCD.Policy

价值	名称	说明
0b	被动	指示 RoT 测量中的证明失败，但允许失败的设备继续启动。
1b	活跃	防止认证失败的设备启动。

I2C 电源管理控制器元素

电源管理控制器元素包含有关系统电源管理设备的信息，RoT 使用该设备为未通过认证的设备提供电源。如果不存在电源管理控制器元素，则组件电源控制不可用。

元素中的 I2C 多路复用器信息以直接连接到 RoT 的多路复用器开始，以直接连接到电源管理设备的多路复用器结束。如果与电源管理控制器的通信不使用 MCTP，则 EID 字段可以设置为“0x00”。

位域 PCD.I2CPowerManagementController

类型	名称	说明
I2CDevice	设备	电源管理控制器的通信信息。

位域PCD.I2CDevice

类型	名称	说明

类型	名称	说明
4	mux_count	RoT 和设备之间的 I2C 路径中的多路复用器数量。
000b	_	预订的。
I2CMode	i2c_mode	设备的 I2C 操作模式。多路复用器始终是主从。
8	巴士	连接到设备的 I2C 总线的标识符。
8	地址	目标设备的 I2C 地址。
8	开斋节	设备 MCTP EID。如果设备不使用 MCTP，请将其设置为“0x00”。
I2CMux(mux_count)	多路复用器	I2C 多路复用器列表。

位域 PCD.I2CMux

类型	名称	说明
8	mux_address	多路复用器的 I2C 地址。
8	mux_channel	设备连接到的 Mux 通道。
0x0000	_	预订的。

枚举 PCD.I2CMode

价值	名称	说明
0b	multi_master	多主机 I2C 通信。
1b	master_slave	主从 I2C 通信。

组件元素

PCD 中的每个组件元素代表系统中必须证明的远程 RoT。不同的 AC-RoT 设备将具有不同的连接属性，因此有不同的元素类型来说明证明者将使用不同的方法与 AC-RoT 进行通信。组件类型标识符将匹配 CFM 中指定的类型。

位域 PCD.Component

类型	名称	说明
政策	政策	身份验证失败时应采取的操作。仅当存在电源控制器时才可能使用主动策略。
8	power_ctrl_reg	在管理该组件的电源控制器中注册地址。
8	power_ctrl_mask	一个位掩码，指示用于控制此组件电源的位。
0x00	_	预订的。
32	component_id	组件类型的标识符。此 ID 将用于匹配 CFM 中的证明策略。如果 PCD 中有多个具有相同组件 ID 的组件，它们将在 CFM 中使用相同的策略。

具有直接 I2C 连接元件的组件

该元素包含有关通过 I2C 直接连接到 RoT 的 AC-RoT 的信息。即使 I2C 路径中有 I2C 多路复用器，AC-RoT 也被视为直接连接。

元素中的多路复用器信息以直接连接到证明者的多路复用器开始，以直接连接到 AC-RoT 的多路复用器结束。

位域 `PCD.ComponentWithDirectI2CConnection`

类型	名称	说明
组件	组件	AC-RoT 证明配置。
I2CDevice	设备	AC-RoT 的 I2C 通信细节。

具有 MCTP 桥接元件的组件

此元素包含有关连接到设备的 AC-RoT 的信息，该设备将根据目标 EID 路由 MCTP 请求。在服务器中，这通常是 BMC。RoT 应该直接连接到这座桥，并了解如何与之通信。如果系统中有多个相同的设备并且可以从桥接设备动态发现 EID，则可以使用单个元素条目来标识多个设备以进行证明。

位域 `PCD.ComponentWithMCTPBridgeConnection`

类型	名称	说明
组件	组件	AC-RoT 证明配置。
16	device_id	用于在证明的发现阶段识别此设备的设备标识符。
16	vendor_id	供应商标识符用于在证明的发现阶段识别此设备。
16	subsystem_device_id	用于在证明的发现阶段识别此设备的子系统设备标识符。
16	subsystem_vendor_id	用于在证明的发现阶段识别此设备的子系统供应商标识符。
8	components_count	此元素描述的相同组件的数量。
8	开斋节	如果无法从 MCTP 网桥动态发现 EID，则使用默认 EID。
0x0000	_	预订的。

PCD 生成的 XML 表示

每个 RoT 都应该有 PCD 来定义与该特定配置相关的行为和组件。这对于在多个不同平台中使用的通用 RoT 设备尤其重要。这是一个使用 XML 存储平台配置的示例，生成脚本可以使用这些平台配置来创建二进制格式。

```
<PCD sku="标识符" version="十六进制整数">
    <RoT type="PA-RoT" mctp_ctrl_timeout="十进制整数"
        mctp_bridge_get_table_wait="十进制整数">
        <端口>
            <Port id="十进制整数">
                <SPIFreq>十进制整数</SPIFreq>
                <ResetCtrl>重置</ResetCtrl>
                <FlashMode>双</FlashMode>
```

```
                <政策>被动</政策>
                <脉冲间隔>0</脉冲间隔>
                <RuntimeVerification>启用</RuntimeVerification>
                <WatchdogMonitoring>启用</WatchdogMonitoring>
            </端口>
            <Port id="十进制整数">
                <SPIFreq>十进制整数</SPIFreq>
                <ResetCtrl>通知</ResetCtrl>
                <FlashMode>单一</FlashMode>
                <政策>有效</政策>
                <脉冲间隔>0</脉冲间隔>
                <RuntimeVerification>已禁用</RuntimeVerification>
                <WatchdogMonitoring>禁用</WatchdogMonitoring>
            </端口>
        </端口>
        <接口类型="I2C">
            <地址>十六进制整数</地址>
            <RoTEID>十六进制整数</RoTEID>
            <BridgeEID>十六进制整数</BridgeEID>
            <BridgeAddress>十六进制整数</BridgeAddress>
        </界面>
    </RoT>
    <电源控制器>
        <接口类型="I2C">
            <Bus>十进制整数</Bus>
            <EID>十六进制整数</EID>
            <地址>十六进制整数</地址>
            <I2CMode>多主控</I2CMode>
            <多路复用器>
                <Mux level="十进制整数">
                    <地址>十六进制整数</地址>
                    <Channel>十进制整数</Channel>
                </Mux>
            </多路复用器>
        </界面>
    </电源控制器>
    <组件>
        <组件类型="标识符" 连接="直接"
            attestation_success_retry="十进制整数"
            attestation_fail_retry="十进制整数"
            attestation_rsp_not_ready_max_retry="十进制整数"
            attestation_rsp_not_ready_max_duration="十进制整数">
            <政策>被动</政策>
            <接口类型="I2C">
                <Bus>十进制整数</Bus>
                <地址>十六进制整数</地址>
                <I2CMode>多主控</I2CMode>
                <EID>十六进制整数</EID>
                <多路复用器>
                    <Mux level="十进制整数">
                        <地址>十六进制整数</地址>
                        <Channel>十进制整数</Channel>
                    </Mux>
                </多路复用器>
```

```
            </界面>
            <电源控制>
                <Register>十六进制整数</Register>
                <掩码>十六进制整数</掩码>
            </PwrCtrl>
        </组件>
        <组件类型="标识符" 连接="MCTPBridge"
            count="十进制整数"
            attestation_success_retry="十进制整数"
            attestation_fail_retry="十进制整数"
            discovery_fail_retry="十进制整数"
            mctp_bridge_additional_timeout="十进制整数"
            attestation_rsp_not_ready_max_retry="十进制整数"
            attestation_rsp_not_ready_max_duration="十进制整数">
            <政策>被动</政策>
            <DeviceID>十六进制整数</DeviceID>
            <VendorID>十六进制整数</VendorID>
            <SubsystemDeviceID>十六进制整数</SubsystemDeviceID>
            <SubsystemVendorID>十六进制整数</SubsystemVendorID>
            <EID>十六进制整数</EID>
            <电源控制>
                <Register>十六进制整数</Register>
                <掩码>十六进制整数</掩码>
            </PwrCtrl>
        </组件>
    </组件>
</PCD>
```

领域	说明
sku	此平台配置的字符串标识符。这成为平台 ID。
版本	PCD 的版本。这成为清单 ID。
RoT	RoT 配置的容器。
类型	RoT 类型标识符。
mctp_ctrl_timeout	MCTP 控制协议响应超时时间（以毫秒为单位）。
mctp_bridge_get_table_wait	如果 MCTP 网桥不支持动态 EID 分配，RoT 启动后在向 MCTP 网桥发送 MCTP 获取路由表请求之前等待的持续时间（以毫秒为单位）。如果设置为 0，RoT 将只等待 EID 分配。
端口	外部 RoT 保护的端口集合。
端口	单个受保护端口。
id	端口的整数标识符。
SPIFreq	SPI 闪烁频率，以 Hz 为单位。
重置Ctrl	验证时重置控制设置。

领域	说明
FlashMode	受保护设备的闪存配置。
政策	鉴证政策。
脉冲间隔	如果端口使用脉冲复位，则复位脉冲宽度。
RuntimeVerification	端口运行时验证设置。
看门狗监控	端口看门狗监控设置。
接口	RoT 的通信接口。
类型	通信接口类型或组件类型标识符字符串。
地址	7 位 I2C 地址。
RoTEID	信任根的默认 MCTP EID。
BridgeEID	MCTP 网桥 EID。
BridgeAddress	MCTP 桥 7 位 I2C 地址。
巴士	设备所在的 I2C 总线的标识符。
开斋节	设备 MCTP EID。
I2CMode	I2C 通信模式。
多路复用器	连接到设备的一系列 I2C 多路复用器。
多路复用器	单个 I2C 多路复用器。
级别	I2C 路径中的多路复用器级别。0 如果第一个多路复用器，1 是第二个，依此类推。
地址	多路复用器的 7 位 I2C 地址。
频道	要在多路复用器上激活的通道。
组件	要证明的组件集合。
组件	单个组件的证明信息。
连接	组件连接类型。
attestation_success_retry	设备成功证明后等待重新证明的持续时间（以毫秒为单位）。
attestation_fail_retry	设备认证失败后等待重新认证的持续时间（以毫秒为单位）。
attestation_rsp_not_ready_max_retry	设备允许的 SPDM ResponseNotReady 重试的最大次数。
attestation_rsp_not_ready_max_duration	收到 SPDM ResponseNotReady 错误后重试之间的最长等待时间。

领域	说明
PwrCtrl	组件功率控制信息。
注册	电源控制寄存器地址。
掩码	电源控制位掩码。
计数	此元素描述的相同组件的数量。
discovery_fail_retry	设备发现步骤失败后等待重试的持续时间（以毫秒为单位）。
mctp_bridge_additional_timeout	除了 MCTP 桥接导致的设备超时时间外，还要等待的时间（以毫秒为单位）。
设备ID	设备ID。请参阅设备发现说明。
供应商ID	供应商 ID。请参阅设备发现说明。
SubsystemDeviceID	子系统设备 ID。请参阅设备发现说明。
SubsystemVendorID	子系统供应商 ID。请参阅设备发现说明。

XML 枚举的允许值

具有默认值的枚举表示该标记是可选的。如果它不存在于 XML 中，将使用默认值。

RoT/类型

- PA-RoT
- AC-RoT

端口/ResetCtrl

- 通知
- 重置
- 脉冲

端口/闪存模式

- 单身的
- 双重的
- 单过滤旁路
- 双过滤旁路

策略/运行时验证

- 启用
- 禁用

策略/看门狗监控

- 启用
- 禁用

接口/类型

- I2C

接口/I2C模式

- 多主控
- 主从

组件/连接

- 直接的
- MC TPBridge

政策/失败行动

- 被动的
- 积极的

组件固件清单

组件固件清单 (CFM) 描述了系统中每个组件的允许测量或策略列表。这类似于 PFM，但目的是证明远程 AC-RoT 设备。只有证明其他 AC-RoT 的 Cerberus 设备才会消耗 CFM。CFM 中的元素描述了如何使用 Cerberus Challenge Protocol或 DMTF SPDM 协议来证明设备。CFM 中的每个组件必须在 PCD 中至少有一个对应的条目。同一类型组件设备的多个 PCD 条目将映射到 CFM 中的单个组件。

CFM 将在清单标头中报告 `0xa592` 的 `manifest_type`。CFM 定义了以下额外的清单元素类型。

类型编号	元素类型	格式	顶级	单例	说明
`0x70`	组件设备	0	×		定义单一类型的 AC-RoT 来证明。
`0x71`	PMR	0			提供重新生成 PMR 时所需的信息。这是 Component Device 元素的子元素。
`0x72`	PMR 文摘	0			单个 PMR 的允许值列表。这是 Component Device 元素的子元素。
`0x73`	测量	0			测量摘要的允许值列表。这是 Component Device 元素的子元素。
`0x74`	测量数据	0			提供有关原始测量数据检查的信息。这是 Component Device 元素的子元素。
`0x75`	允许的数据	0			提供有关对原始测量数据进行单次检查的信息。这是测量数据元素的子元素。
`0x76`	允许的 PFM	0			单个端口允许的 PFM ID 列表。这是 Component Device 元素的子元素。
`0x77`	允许的 CFM	0			允许的 CFM ID 列表。这是 Component Device 元素的子元素。

类型编号	元素类型	格式	顶级	单例	说明
0x78	允许的 PCD	0			允许的 PCD ID 列表。这是 Component Device 元素的子元素。
0x79	允许的 ID	0			提供有关单次检查清单 ID 的信息。这是允许的 PFM、CFM 或 PCD 元素的子元素。
0x7A	根CA	0			可用于证书链身份验证的根 CA。这是 Component Device 元素的子元素。

组件设备元素

组件设备元素是用于证明单一类型 AC-RoT 所需的所有信息的顶级容器。组件设备的子元素指定必须采取哪些证明操作和检查。为了使 AC-RoT 的认证成功，所有认证检查都必须通过。PCD 中描述的每个 AC-RoT 都使用“类型”属性与组件设备元素匹配。每个 Component Device 元素需要至少有一个子元素。

位域 CFM.ComponentDevice

类型	名称	说明
8	cert_slot	用于证明挑战的证书链的槽号。
证明协议	证明_协议	用于向组件发出证明请求的协议。
Manifest.HashType	transcript_hash_type	用于 SPDM 脚本散列的散列类型。
Manifest.HashType	measurement_hash_type	用于生成测量、PMR 和根 CA 摘要的哈希类型。
00b	_	填充未使用的 HashType 位。这必须是 0。
0x00	_	预订的。
32	component_id	将被证明的组件类型的标识符。这必须是 CFM 中的唯一标识符。

枚举 CFM.AttestationProtocol

价值	名称	说明
0x00	cerberus_protocol	Cerberus Challenge Protocol。
0x01	dmtf_spdm	DMTF SPDM 协议。

位域 CFM.ComponentDevice.Check

类型	名称	说明
检查类型	检查	要执行的比较类型。
0000b	_	未使用的位。这必须是 0。

类型	名称	说明
1	字节顺序	多字节数据值的字节顺序。如果数据以 little endian 表示，则为 0；如果数据为 big endian，则为 1。如果数据只是单个字节，则此值无关紧要。

枚举 CFM.ComponentDevice.CheckType

价值	名称	说明
000b	等于	确保报告的数据等于指定值。
001b	不等于	确保报告的数据不等于指定值。
010b	小于	确保报告的数据小于指定值。
011b	小于或等于	确保报告的数据小于或等于指定值。
100b	大于	确保报告的数据大于指定值。
101b	大于或等于	确保报告的数据大于或等于指定值。

PMR元素

PMR 元素包含重新生成 PMR 摘要所需的信息。如果 PMR ID 不存在此元素，则零的初始值将用于计算 PMR 所需的任何流。

位域 CFM.ComponentDevice.PMR

类型	名称	说明
measurement_hash_type	初始值	生成 PMR 时使用的初始值。

PMR 摘要元素

PMR 摘要元素包含单个 PMR 的所有允许摘要的列表。这些摘要代表 PMR 报告的最终测量，如“挑战”或“获取 PMR”请求所报告的那样。

位域 CFM.ComponentDevice.PMRDigest

类型	名称	说明
8	pmr_id	PMR 要证明的标识符。Cerberus Challenge Protocol允许它介于 0 和 4 之间。
8	摘要计数	此 PMR 允许的摘要数。
0x0000	_	预订的。
measurement_hash_type(digest_count)	pmr_digest	PMR 允许的摘要列表。

测量元素

Measurement 元素包含单个测量的允许摘要列表。对于 Cerberus PMR，这将是在提供证明或 TCG 日志时由“获取日志”请求报告的特定 PMR 中的条目。

在 SPDM 中，这将是单个测量块的摘要，并且可以在由“SPDM Challenge”请求报告或由“SPDM Get Measurement”请求单独报告时与所有其他测量块聚合。

位域 `CFM.ComponentDevice.Measurement`

类型	名称	说明
`8`	`pmr_id`	包含要证明的条目的 PMR 的标识符。Cerberus Challenge Protocol 允许它介于 0 和 4 之间。对于使用 SPDM 的设备，这将为零，因为 SPDM 不提供与 PMR 等效的功能。
`8`	`measurement_id`	PMR 中特定条目的索引，以证明是否使用 Cerberus Challenge Protocol。如果使用 SPDM，这是测量块索引。
`8`	`allowable_digest_count`	此测量的允许摘要数。
`0x00`	`_`	预订的。
`AllowableDigest(allowable_digest_count)`	`digests_list`	所有预期固件版本的允许摘要列表。

位域 `CFM.ComponentDevice.Measurement.AllowableDigest`

类型	名称	说明
`16`	`version_set`	与设备上相同版本的固件关联的一组测量的标识符。如果同一组测量适用于所有版本的固件，则这将为 0。
`8`	`摘要计数`	此版本集允许的摘要数。
`0x00`	`_`	预订的。
`measurement_hash_type(digest_count)`	`摘要`	此版本集的预期测量摘要列表。

在证明单个测量时，有必要确保设备报告的测量代表所有测量的预期状态。单独证明单个测量并不能确保这一点，因为每个测量都将独立于其他检查进行检查。这将允许设备报告来自固件 A 的一个测量值和来自固件 B 的另一个测量值被视为健康，即使要求这两个测量值仅代表固件 A 或 B 的状态。`version_set` 标识符提供了一种方法来确保可以在更广泛的证明上下文中检查报告的测量，从而提供单独测量检查之间的耦合。

测量数据元素

测量数据元素包含一系列检查，以针对单个测量块测量的原始数据执行。对于 Cerberus PMR，这将是“获取证明数据”请求报告的特定 PMR 中的一个条目。在 SPDM 中，这将是单个测量块的原始形式，可以通过“SPDM 获取测量”请求单独报告。这允许比简单地比较摘要更复杂的策略检查，并且有可能更有效地使用 CFM 中的空间。对这些数据执行的检查与允许的子数据元素一样多，并且所有检查都需要成功才能成功证明。

在测量数据元素的单独检查中使用 `version_set` 与测量元素相同。此外，由于 version_set 而存在的 Measurement 元素之间的耦合也扩展到 Measurement Data 元素。这意味着所有 Measurement 和 Measurement Data 检查都必须在相同的“version_set”中进行证明才能通过证明。

位域 `CFM.ComponentDevice.MeasurementData`

类型	名称	说明
8	pmr_id	包含要证明的条目的 PMR 的标识符。Cerberus Challenge Protocol 允许它介于 0 和 4 之间。对于支持 SPDM 的设备，这将为零。
8	measurement_id	PMR 中特定条目的索引，以证明是否使用 Cerberus Challenge Protocol。如果使用 SPDM，这是测量块索引。
0x0000	_	预订的。

允许的数据元素

允许数据元素包含单个测量数据检查的允许值列表。对于“等于”和“不等于”检查，在单个允许数据元素中可以有尽可能多的具有相同“version_set”标识符的数据条目，以涵盖所有允许或不允许的值。对于任何其他类型的检查，每个 version_set 标识符的允许数据元素中应该只有一个数据条目。在所有情况下，允许具有不同“version_set”标识符的任意数量的数据条目。

要组合多个测量数据检查（例如“不等于”和“大于”），必须使用多个允许数据元素，每个检查一个。

位域 `CFM.ComponentDevice.MeasurementData.AllowableData`

类型	名称	说明
检查	检查	对数据执行的比较类型。
8	num_data	将检查的值的总数。
16	位掩码长度	在应用任何检查之前应用于测量数据的位掩码的长度。如果为 0，则不应用位掩码并检查原始数据。
位掩码长度	data_bitmask	应用于接收到的数据的位掩码。这允许比较在必要时忽略某些位或字节。必须创建此位掩码以说明数据的字节顺序。相同的位掩码适用于所有数据条目。如果位掩码比测量数据长，则忽略未使用的掩码位。
对齐（32）	_	零填充。
数据（num_data）	数据列表	支持的数据列表。

位域 `CFM.ComponentDevice.MeasurementData.AllowableData.Data`

类型	名称	说明

类型	名称	说明
16	version_set	与设备上相同版本的固件关联的一组测量的标识符。如果相同的测量数据检查适用于所有版本的固件，则这将为 0。
16	数据长度	用于比较的数据长度。
数据长度	数据	用于比较的数据。这必须以“Check.endianness”标志指示的格式存储。
对齐（32）	–	零填充。

为了将一组数据分组到单个允许数据元素中，每个数据检查必须使用相同的位掩码。每个唯一的位掩码必须是一个单独的允许数据元素。由于原始数据在不同版本的固件之间可能具有不同的长度，因此不要求数据长度相同。但是，将位掩码与不同长度的数据一起使用会带来一些挑战，因此必须满足以下属性才能有效地对检查进行分组。

1. bitmask_length 必须至少等于所有支持数据中最大的 data_length。
2. 如果对于任何给定的“data”，“bitmask_length”大于“data_length”，则比较将忽略任何超过“data_length”的“data_bitmask”字节。
3. data_bitmask 始终从数据的最低有效字节开始应用。如果数据和位掩码长度不匹配，位掩码的最高有效字节将被忽略。Check.endianness 标志表示数据和位掩码是如何存储的以及它们需要按什么顺序处理。

允许的 PFM 元素

允许的 PFM 元素提供了一种机制来证明基于活动 PFM 配置的 AC-RoT。每个允许的 PFM 元素将与单个端口的活动 PFM 进行比较。比较的数据将由“获取配置 ID”请求报告。Child Allowable ID 元素将提供针对 PFM ID 运行的检查。

位域 CFM.ComponentDevice.AllowablePFM

类型	名称	说明
8	port_id	应检查的返回 PFM ID 列表中的索引。这与端口 ID 相同。
允许清单	允许	为 PFM ID 执行的证明。

位域 CFM.ComponentDevice.AllowableManifest

类型	名称	说明
8	plat_id_length	预期平台 ID 的长度。
plat_id_length	plat_id_string	平台 ID 字符串。这是一个非空终止的 ASCII 字符串。
对齐 (32)	_	零填充。

允许的 ID 元素

Allowable ID 元素包含用于单个清单检查的允许 ID 列表。该元素可以是 Allowable PFM、CFM 或 PCD 元素的子元素。

对于“等于”和“不等于”检查，单个允许 ID 元素中可以有尽可能多的数据条目，以覆盖所有允许或不允许的值。对于任何其他类型的检查，允许 ID 元素中应该只有一个数据条目。

要组合多个清单 ID 检查（例如“不等于”和“大于”），必须使用多个允许 ID 元素，每个检查一个。

位域 CFM.ComponentDevice.AllowableID

类型	名称	说明
检查	检查	要执行的比较类型。
8	num_id	将检查的 ID 总数。
16	_	预订的。
32 (num_data)	ids	用于比较的清单标识符。

允许的 CFM 元素

允许的 CFM 元素提供了一种机制来证明基于活动 CFM 配置的 AC-RoT。每个允许的 CFM 元素将与单个 CFM 进行比较。比较的数据将由“获取配置 ID”请求报告。Child Allowable ID 元素将提供针对 PFM ID 运行的检查。

位域 CFM.ComponentDevice.AllowableCFM

类型	名称	说明
8	cfm_index	应检查的返回 CFM ID 列表中的索引。
允许清单	允许	为 CFM ID 执行的证明。

允许的 PCD 元素

允许的 PCD 元素提供了一种机制来证明基于活动 PCD 配置的 AC-RoT。比较的数据将由“获取配置 ID”请求报告。Child Allowable ID 元素将提供针对 PFM ID 运行的检查。

位域 CFM.ComponentDevice.AllowablePCD

类型	名称	说明
0x00	_	预订的。
允许清单	允许	为 PCD ID 执行的证明。

根 CA 元素

Root CAs 元素包含可用于验证 AC-RoT 的证明证书链的受信任根证书列表。如果组件不包含 Root CAs 元素，则 AC-RoT 的证书链必须与请求者共享一个根 CA。

位域 CFM.ComponentDevice.RootCAs

类型	名称	说明
8	ca_count	允许的根 CA 摘要数。

类型	名称	说明
0x000000	_	预订的。
measurement_hash_type(ca_count)	root_ca	受信任的根证书的哈希列表。这表示整个证书的散列，包括签名。

CFM 生成的 XML 表示

每个必须证明 AC-RoT 设备的 RoT 需要至少有一个 CFM，其中包含有关要证明的组件的信息。与仅处理一个闪存设备的 PFM 不同，CFM 旨在保存多个组件的认证信息。

带有 CFMComponent 元素的 XML 文件用于定义可证明的组件。单个 CFM XML 文件用于选择要包含在 CFM 二进制文件中的可证明组件。

每个组件的每个固件版本都有一个 XML，生成的 CFM 二进制文件将使用"version_set"字段区分属于不同固件版本的测量值。每个组件 XML 应至少有一个 Measurement 或 MeasurementData 元素，该元素对于每个固件版本都是唯一的。此唯一条目必须是 XML 中 CFMComponent 的第一个 Measurement 或 MeasurementData 子元素。

如果该条目是 MeasurementData 元素，则它必须有一个 AllowableData 子元素。生成的 CFM 二进制文件不能包含此条目的版本设置为 0 的任何子元素。如果此 MeasurementData 需要多个 AllowableData 元素，则必须将它们分成两个不同的 MeasurementData 元素。第一个仅包含唯一信息的单个 AllowableData，第二个包含剩余的检查。

这是可由生成脚本使用以创建二进制格式的两种 XML 格式的示例。

```
<CFM sku="标识符">
    <组件>
        "组件类型字符串"
    </组件>
    <组件>
        "组件类型字符串"
    </组件>
</CFM>

<CFMComponent type="identifier" attestation_protocol="Cerberus"
    slot_num="十进制整数" transcript_hash_type="SHA256"
    measurement_hash_type="SHA384">
    <RootCADigest>
        <!-- 摘要类型由 measurement_hash_type 决定。-->
        <摘要>
            根 CA 摘要
        </摘要>
        <摘要>
            根 CA 摘要
        </摘要>
    </RootCADigest>
    <PMR pmr_id="十进制整数">
        <SingleEntry>真</SingleEntry>
        <!-- InitialValue 类型由 measurement_hash_type 决定。-->
```

```
        <初始值>
            初始值摘要
        </初始值>
    </PMR>
    <PMRDigest pmr_id="十进制整数">
        <!-- 摘要类型由 measurement_hash_type 决定。-->
        <摘要>
            PMR文摘
        </摘要>
        <摘要>
            PMR文摘
        </摘要>
    </PMRDigest>
    <Measurement pmr_id = "十进制整数" measurement_id="十进制整数">
        <!-- 摘要类型由 measurement_hash_type 决定。-->
        <摘要>
            测量文摘
        </摘要>
        <摘要>
            测量文摘
        </摘要>
    </测量>
    <MeasurementData pmr_id="十进制整数" measurement_id="十进制整数">
        <允许数据>
            <字节序>BigEndian</字节序>
            <Check>等于</Check>
            <数据>
                允许数据 1
            </数据>
            <数据>
                允许数据 2
            </数据>
            <位掩码>
                测量数据位掩码
            </位掩码>
        </允许数据>
        <允许数据>
            <Endianness>LittleEndian</Endianness>
            <检查>大于或等于</检查>
            <数据>
                允许数据
            </数据>
        </允许数据>
    </测量数据>
    <MeasurementData pmr_id="十进制整数" measurement_id="十进制整数">
        <允许数据>
            <Endianness>LittleEndian</Endianness>
            <Check>等于</Check>
            <数据>
                “细绳”
            </数据>
            <位掩码>
                测量数据位掩码
            </位掩码>
```

```
        </允许数据>
    </测量数据>
    <AllowablePFM port="Decimal integer" platform="Identifier">
        <清单编号>
            <Check>等于</Check>
            <ID>十六进制整数</ID>
            <ID>十六进制整数</ID>
        </清单ID>
        <清单编号>
            <检查>大于</检查>
            <ID>十六进制整数</ID>
        </清单ID>
    </允许PFM>
    <AllowableCFM index="十进制整数" platform="标识符">
        <清单编号>
            <Check>等于</Check>
            <ID>十六进制整数</ID>
        </清单ID>
    </允许CFM>
    <AllowablePCD platform="标识符">
        <清单编号>
            <Check>等于</Check>
            <ID>十六进制整数</ID>
        </清单ID>
    </允许PCD>
</CFM组件>
```

领域	说明
sku	平台配置的字符串标识符。这成为平台 ID。
组件	定义要包含在 CFM 中的单个组件设备。这里使用组件的 ID。
CFMComponent	定义单个组件设备。
类型	组件类型字符串标识符。
证明_协议	AC-RoT 用于证明的协议。
slot_num	用于组件认证的证书链槽号。
transcript_hash_type	用于成绩单哈希的哈希算法。
measurement_hash_type	用于计算 PMR、测量和根 CA 摘要的哈希算法。
RootCADigest	定义用于证书链验证的可信根 CA。这是一个可选标签。
摘要	预期的摘要。
PMR	定义重新生成 PMR 所需的信息。这是一个可选标签。
SingleEntry	指示 PMR 是否具有单项测量值的布尔值。
初始值	生成 PMR 时要使用的初始值。
PMRDigest	为单个 PMR 定义允许的摘要。

领域	说明
pmr_id	PMR 的标识符。
测量	定义一组允许的测量值。
measurement_id	测量 ID。
测量数据	定义一组允许的测量数据。
允许数据	定义单个测量数据检查的允许值。
字节顺序	数据的多字节格式。
检查	对数据执行的比较类型。
数据	测量数据条目的预期数据。
位掩码	比较期间应用于数据的位掩码。
允许的PFM	定义要对 PFM ID 执行的证明。证明 SPDM 设备不支持此标记。
端口	端口 PFM 适用于。
平台	预期的清单平台 ID。
ManifestID	单个清单 ID 比较。
ID	要比较的清单 ID。
允许CFM	定义要对 CFM ID 执行的证明。证明 SPDM 设备不支持此标记。
索引	目标 CFM 的索引。
AllowablePCD	定义要对 PCD ID 执行的证明。证明 SPDM 设备不支持此标记。

XML 枚举的允许值

具有默认值的枚举表示该标记是可选的。如果它不存在于 XML 中，将使用默认值。

查看

- 平等的
- 不等于
- 少于
- 小于或等于
- 比...更棒
- 大于或等于

组件/证明_协议

- 地狱犬
- SPDM

字节顺序

- LittleEndian

- 大端

证明流程

根据设备配置，证明工作流程涉及 Flash 内容的身份验证、挑战远程设备或两者兼而有之。

Flash 内容的认证

闪存认证通常由连接到 SPI 闪存的外部 RoT 设备使用。PCD 用作此流程的一部分，以提供 SPI 时钟频率的配置，以便在访问闪存设备时使用以及与外部处理器交互相关的其他参数。其余的身份验证流程只需要一个 PFM。

固件更新认证

每当对 SPI 闪存设备执行固件更新时，RoT 将需要先验证内容，然后才能证明其有效。

验证采用以下流程：

1. 读取 PFM 以确定闪存上存在多少固件组件。 2.对于每个固件组件
    - 读取闪存设备以匹配 PFM 中的版本字符串。这将确定应该使用哪个版本条目进行身份验证。
    - 对于匹配版本中定义的每个签名图像，计算并比较闪存内容的哈希值。
2. 验证每个组件后，寻找未使用的闪存区域。 未使用的区域既不是 R/W 区域，也不是任何固件组件中的签名映像。这些未使用的区域必须用闪存设备 PFM 元素中定义的静态值填充。
3. 如果任何步骤产生意外结果，则图像验证失败。

在没有更新的情况下在系统启动时进行身份验证

如果闪存上的固件尚未更新，RoT 仍将验证闪存设备上的所有必要图像，但会对流程进行一些修改。

1. flash 未使用的区域不做空白检查。
2. 任何未声明“PFM.FirmwareVersion.MustValidate”标志的签名图像都将被跳过。

远程设备认证

AC-RoT 的证明遵循以下一般流程：

1. 确定被证明的设备类型。
2. 检索设备证书链，对其进行验证，并确定设备是可信的。
3. 从表示当前状态的设备获取测量报告。
4. 将测量值与 CFM 中的证明策略进行比较。
5. 将认证结果存储在请求RoT 的认证日志中。

Cerberus 设备可以使用 Cerberus 质询协议和/或 DMTF SPDM 证明协议支持证明。虽然协议之间的执行细节有所不同，但总体流程是相同的。

Cerberus Challenge Protocol规范中描述了证明流程中使用的所有 Cerberus 协议命令。MCTP 和 SPDM 命令在这些各自的规范中进行了描述。

Cerberus 到 AC-RoT 通信

作为 AC-RoT 组件认证的先决条件，需要启用与 AC-RoT 的通信。Cerberus 和 AC-RoT 组件设备都将充当 MCTP 端点，并且将直接连接在同一物理总线上或通过 MCTP 桥进行通信。需要证明其他 AC-RoT 的 Cerberus 实例需要了解寻址细节和整个平台配置。平台配置数据清单向 Cerberus RoT 提供这些详细信息。

使用 MCTP 桥发现 AC-RoT

要通过 MCTP 网桥与 AC-RoT 通信，Cerberus 首先需要通过向 MCTP 网桥发送“获取路由表条目”MCTP 控制请求来获取目标设备的 EID。然后 Cerberus 将向接收到的路由表中的每个 EID 发出“获取消息类型支持”MCTP 控制请求，以发现 AC-RoT 支持的证明协议。

使用 Cerberus Challenge Protocol的 AC-RoT 发现

当以下两个条件都为真时，确定对 Cerberus Challenge Protocol的支持：

1. AC-RoT 支持供应商定义的消息类型。
2. 支持 Microsoft PCI 供应商 ID (0x1414) 是为了响应“获取供应商定义的消息支持”MCTP 控制请求而公布的。

一旦确定 AC-RoT 支持 Cerberus Challenge Protocol，就会通过发送 Cerberus 协议“设备 ID”请求来识别设备类型。对该请求的响应用于在 PCD 中搜索组件条目以进行匹配。

使用 SPDM 的 AC-RoT 发现

由于 SPDM 在 MCTP 中定义了消息类型，因此通过简单地查找对该消息类型的支持来确定对 SPDM 的支持。一旦确定 AC-RoT 支持 SPDM，就需要将该设备与 PCD 中的组件条目进行匹配，就像使用 Cerberus Challenge Protocol 时一样。然而，从 SPDM 1.2 开始，没有协议定义的方法来实现这一点。在没有任何标准化 SPDM 工作流程的情况下，Cerberus 将使用以下内容，它必须由任何需要证明的基于 SPDM 的 AC-RoT 实施。如果 SPDM 规范的更新版本更新为包含确定此信息的方法，则该方法将用于支持它的 AC-RoT。

为了为设备提供一种符合 SPDM 的方式来识别自己以进行证明，Cerberus 协议“设备 ID”命令报告的相同 PCD ID 将在 SPDM 测量块中报告。对于支持 SPDM 1.1.x 的 AC-RoT，此信息将在设备支持的最后一个测量块索引中报告。对于其他 AC-RoT，此信息将位于“0xef”的固定测量索引处。

测量块的检索是通过以下 SPDM 命令序列实现的：

1.获取版本 2.获取能力 3.协商算法 4. 如果 AC-RoT 支持 SPDM 1.1.x，Cerberus 将使用“Get Measurements”获取设备支持的测量块总数，无需签名请求并将测量操作设置为“0”。Cerberus 紧随其后的是“获取测量”请求，没有为最后一个测量索引请求签名。 5. 如果 AC-RoT 支持 SPDM 1.2.x 或更高版本，Cerberus 将发送一个“Get Measurements”请求，不为测量索引“0xef”处的原始比特流请求签名。

在所有情况下，响应都应包含测量块的原始比特流。此测量块的格式基于 [DMTF DSP0267 1.1.0，用于固件更新的 PLDM](https://www.dmtf.org/sites/default/files/standards/documents/DSP0267_1. 1.0.pdf）。

位域 DeviceIds

类型	名称	说明
8	完成代码	PLDM_BASE_CODES 完成代码。
32	描述符长度	提供的所有描述符的总长度。

类型	名称	说明
8	描述符计数	描述符总数。
描述符 (descriptor_count)	描述符	设备描述符列表。

位域描述符

类型	名称	说明
16	类型	描述符的类型标识符。
16	长度	描述符数据的长度。
长度	数据	原始描述符数据。这代表什么取决于描述符的类型。

虽然响应中包含的描述符可能不仅包含 PCI ID，但至少必须包含“PCI 供应商 ID”、“PCI 设备 ID”、“PCI 子系统供应商 ID”和“PCI 子系统 ID”的描述符。此处包含一个快速参考值，用于为该信息分配给“描述符”字段的值。

类型	长度	数据
0x0000	2	PCI 供应商 ID。
0x0100	2	PCI 设备 ID。
0x0101	2	PCI 子系统供应商 ID。
0x0102	2	PCI 子系统 ID

有关描述符类型和格式的更多详细信息，请参阅 DSP0267 1.1.0 中的表 8。

没有 MCTP 桥的 AC-RoT 发现

当系统不使用 MCTP 网桥时，Cerberus 设备将直接连接所需的 AC-RoT。在这种情况下，发现过程与 MCTP 网桥遵循的过程相同。唯一的区别是没有发送“获取路由表条目”命令。相反，AC-RoT 信息是根据 PCD 组件条目确定的。

AC-RoT 认证

为了执行 AC-RoT 的证明，远程 Cerberus 实例将闪存测量或策略信息发送回执行挑战的 Cerberus 设备。然后，执行质询的 Cerberus 设备会将接收到的信息与目标设备的组件固件清单中相应条目中包含的允许值进行比较。

Cerberus 设备可以支持 Cerberus Challenge 协议、DMTF SPDM 协议或这两种协议来执行 AC-RoT 的证明。预计 PA-RoT 设备将支持这两种协议，以允许它证明任何类型的组件。

使用 Cerberus 协议的证明程序

在成功接收和验证 Alias 证书链后，证明者将确保 CFM 条目中列出的所有摘要和测量数据与组件设备返回的值相匹配。

如果 CFM 仅包含寄存器 0 的平台测量寄存器 (PMR) 摘要，则只会发出“挑战”请求。来自响应的 PMR0 摘要值将与 CFM 元素内容进行比较。这个过程如下图所示：

```
                证明设备
                   | ------挑战请求----> |
                   | -- |
                   | 生成签名挑战 | |
                   | PMR 0 的响应 | |
                   | -> |
                   | <-----挑战响应----- |
                   | -- |
                   | | 验证挑战签名 |
                   | | 和测量 |
                   | <- |
```

如果 CFM 包含多个 PMR 摘要元素，则将为所需的每个 PMR 向设备发出“获取平台测量寄存器”请求。所有列出的摘要将与 CFM 元素内容进行比较，如果其中任何一个不匹配，则证明将失败。此流程如下所示：

```
                证明设备
                  /| --------获取PMR请求-----> |
对于每个 PMR | | -- |
                | | 生成签名响应 | |
                | | 对于请求的 PMR | |
                | | -> |
                | | <-------获取PMR响应------ |
                | | -- |
                | | | 验证响应签名 |
                | | | 和PMR摘要|
                 \| <- |
```

如果 CFM 包含测量元素，证明者将检查证明日志以将预期的测量条目与日志中的条目进行比较。

所有列出的摘要将与 CFM 元素内容进行比较，如果其中任何一个不匹配，则证明将失败。然后，证明者将为所有具有 CFM 测量条目的 PMR ID 发出“获取平台测量寄存器”请求。返回的 PMR 值将与证明日志中的值进行比较。此过程采用以下流程。

1. 认证者发出`Get Log`请求从设备中获取认证日志。
2. 证明日志中报告的摘要与 CFM 元素进行比较。
3. 证明者发出“Get Platform Measurement Register”请求以获取包含此条目的 PMR 值。
4. 预期的 PMR 值是从检索到的证明日志中计算出来的，并与设备报告的值进行比较。

如果任何比较失败，则该过程终止并且设备证明失败。如果证明者在单个 PMR 中有多个条目要验证，则可以优化流程以首先获取并检查所有测量值，然后继续向上验证层到 PMR 值。

这个过程如下图所示：

```
                证明设备
                   | ------获取日志请求------> |
                   | -- |
                   | 回应认证 | |
                   | 日志 | |
```

```
                    | -> |
                    | <--------获取日志响应----- |
                    | -- |
                    | | 将 CFM 测量值与 | 进行比较
                    | | 证明日志中的摘要 |
                    | <- |
                    | ------获取PMR请求------> |
                    | -- |
                    | 回应请求 | |
                    | PMR摘要 | |
                    | -> |
                    | <------获取PMR响应------ |
                    | -- |
                    | | 将 PMR 摘要与一个进行比较 |
                    | | 来自证明日志 |
                    | <- |
```

如果 CFM 包含测量数据元素，则要求证明者检查实际测量数据以确定设备是否在允许的配置下运行。此过程采用以下流程。

1. 证明者发出`Get Attestation Data`请求，从设备中获取测量数据。
2. 将返回的数据与 CFM 元素中的预期数据进行比较。
3. 证明者发出`Get Log`请求从设备中获取证明日志。
4. 较早收到的测量数据被散列并与证明日志中报告的摘要进行比较。
5. 证明者发出“Get Platform Measurement Register”请求以获取包含此条目的 PMR 值。
6. 预期的 PMR 值是根据检索到的证明日志计算的，并与设备报告的值进行比较。

如果任何比较失败，则该过程终止并且设备证明失败。如果证明者在单个 PMR 中有多个条目要验证，则可以优化流程以首先获取并检查所有数据，然后继续向上验证层到 PMR 值。

这个过程如下图所示：

```
                证明设备
                    | 获取认证数据请求-> |
                    | -- |
                    | 回应请求 | |
                    | 鉴证资料 | |
                    | -> |
                    | <-获取认证数据响应|
                    | -- |
                    | | 与 CFM 中的数据进行比较 |
                    | <- |
                    | ------获取日志请求------> |
                    | -- |
                    | 回应认证 | |
                    | 日志 | |
                    | -> |
                    | <--------获取日志响应----- |
                    | -- |
                    | | 比较摘要 |
                    | | 测量数据消化|
```

```
                    | | 在认证日志中 |
                    | <- |
                    | ------获取PMR请求------> |
                    | -- |
                    | 回应请求 | |
                    | PMR摘要 | |
                    | -> |
                    | <------获取PMR响应------ |
                    | -- |
                    | | 将 PMR 摘要与一个进行比较 |
                    | | 来自证明日志 |
                    | <- |
```

如果 CFM 包含 PFM、CFM 或 PCD 元素，则“获取配置 ID”命令用于从设备获取活动清单 ID。证明者会将收到的 ID 与 CFM 中包含的值进行比较。这个过程如下图所示：

```
                证明设备
                   | ---获取配置ID请求---> |
                   | -- |
                   | 响应所有 | 的 ID |
                   | 清单 | |
                   | -> |
                   | <---获取配置 ID 响应-- |
                   | -- |
                   | | 将所有 ID 与 CFM 进行比较 |
                   | | 元素 |
                   | <- |
```

使用 SPDM 认证协议的认证程序

版本和功能协商成功后，然后接收和验证 Alias 证书链，如果设备支持该命令，证明者将向设备发出“挑战”请求。无论 CFM 是否包含 PMR 摘要元素，都将始终执行此操作。在 SPDM 中，只有 PMR 0 被认为是有效的，如果 CFM 包含寄存器 0 的 PMR 摘要元素，则“挑战”响应返回的值将与 CFM 元素内容进行比较。如果不是，“挑战”仍然按照 SPDM 规范的建议执行，并确保先前交易的真实性。这个过程如下图所示：

```
                证明设备
                   | ------挑战请求----> |
                   | -- |
                   | 生成签名挑战 | |
                   | PMR 0 的响应 | |
                   | -> |
                   | <-----挑战响应----- |
                   | -- |
                   | | 验证挑战签名 |
                   | | 和测量 |
                   | <- |
```

如果设备支持 SPDM 1.2 或更高版本，并且不支持“挑战”命令，则使用“获取测量”命令（如果适用）。如果 CFM 包含 PMR 摘要元素但设备不支持“挑战”命令，则使用“获取测量”命令并请求所有测量块。然后，证明者将所有测量块组合成一个摘要，并将其与 PMR0 摘要 CFM 元素进行比较。此过程采用以下流程：

```
证明设备
   | ---获取测量请求--> |
   | -- |
   | 生成测量 | |
   | 用 | 的哈希响应 |
   | 所有测量块 | |
   | -> |
   | <--获取测量响应-- |
   | -- |
   | | 汇总所有测量 |
   | | 块散列和比较 |
   | | 结果到 CFM PMR 摘要 |
   | | 元素 |
   | <- |
```

如果 CFM 包含测量元素，并且设备支持 SPDM 1.2+，则证明者将使用将 RawBitStreamRequested 请求属性设置为 0 的“Get Measurement”命令从设备请求特定测量块，如测量 ID 所示作为摘要。流程采用以下流程：

```
证明设备
   | ---获取测量请求--> |
   | -- |
   | 生成测量 | |
   | 以 | 的散列响应 |
   | 要求测量 | |
   | 块| |
   | -> |
   | <--获取测量响应-- |
   | -- |
   | | 对比回复内容 |
   | | 到CFM测量元件|
   | <- |
```

如果 CFM 包含测量数据元素，则证明者将使用“获取测量”命令从设备请求特定测量块，如测量 ID 所示。

此过程采用以下流程：

```
证明设备
   | ---获取测量请求--> |
   | -- |
   | 生成测量 | |
   | 响应要求 | |
   | 测量块 | |
   | -> |
   | <--获取测量响应-- |
```

```
                    | -- |
                    | | 对比回复内容 |
                    | | 到CFM测量数据|
                    | | 元素 |
                    | <- |
```

SPDM 协议不明确支持配置 ID 的验证。

为了验证设备的配置ID，可以生成以配置ID为预期内容的测量块，并且可以利用上述用于测量或测量数据证明的流程。

处理测量版本集

无论是使用 Cerberus Challenge 协议还是 SPDM 来证明 AC-RoT，证明者都必须确保所有测量都代表相同版本的固件，正如 CFM 所要求的那样。在这些场景中，在证明 Measumement 和 Measurement Data 元素时会进行额外的检查。

CFM 中的第一个 Measurement 或 Measurement Data 元素必须对应于一个 measumerent，该 measumerent 具有针对所有支持的固件版本的条目并且在每个版本之间是唯一的。例如，可以在此处使用版本号或字符串的测量值。一旦第一次测量成功证明，证明者必须确定在 CFM 中为匹配条目指定的版本集标识符。所有后续的测量和测量数据元素都将使用这些附加条件进行证明，使用从第一次检查中选择的版本集标识符。

1. 如果元素包含版本集标识符等于 0 的条目，则此检查必须始终成功，而不管任何选定的版本集标识符。
2. 如果该元素包含与所选版本集标识符匹配的条目，则设备状态的证明必须仅与该元素条目匹配。
3. 如果该元素不包含所选版本集标识符的条目并且不包含版本集标识符为 0 的条目，则该元素作为当前证明的一部分被忽略。

参考

1. Cerberus挑战协议：
   https://github.com/opencomputeproject/Security/blob/master/RoT/Protocol/Challenge_Protocol.md
2. TCG 骰子：
   https://trustedcomputinggroup.org/work-groups/dice-architectures/
3. TCG TPM：
   https://trustedcomputinggroup.org/work-groups/trusted-platform-module/
4. 安全协议和数据模型规范：
   https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.2.0.pdf
5. 平台级数据模型 (PLDM) 基本规范：
   https://www.dmtf.org/sites/default/files/standards/documents/DSP0267_1.1.0.pdf
6. 管理组件传输协议 (MCTP) 基本规范：
   https://www.dmtf.org/sites/default/files/standards/documents/DSP0236_1.1.0.pdf