Attestation of System Components v1.0 Requirements and Recommendations

EDITOR: Elaine Palmer, IBM Corporation

CONTRIBUTORS:
Yigal Edery, Kameleon
Joe Foster, Microchip
Ahmed Abbas Hassan, CyShield
Brett Henning, Broadcom, Inc.
Bryan Kelly, Microsoft
Nate Klein, Google
Jubin Mehta, Facebook, Inc.
Alberto Munoz, Intel Corporation
Rajeev Sharma, Open Compute Project
Wojtek Powiertowski, Facebook, Inc.
Eric Spada, Broadcom, Inc.
Ben Stoltz, Google

Revision History

Revision	Date	Guiding Contributor(s)	Description
1.0	2020-11-04	Elaine Palmer, IBM Corporation	Initial Release

Executive Summary

The environment. In cloud data centers, servers are filled with a plethora of subsystems, peripherals, accelerators, hardware, and firmware from multiple global suppliers. To add to the complexity, those servers are typically configured on demand.

The problem to be solved. Until this document, servers had no standardized, open, and automated mechanism to dynamically establish and verify trust in those products. For example, does a network adapter still contain the initial firmware that was installed by its manufacturer? Has the latest security patch been applied to the firmware in a memory controller? Which country's certified cryptographic algorithms are implemented in a storage unit? Cloud service providers cannot rely on procurement agreements alone to assure that the products they buy are secure. These products must be protected during design, development, manufacture, testing, shipping, provisioning, installation, and operation.

The Open Compute Project solution. This document presents a design for dynamically establishing and verifying trust in the components in a server. In this design, a platform (e.g., server, baseboard management controller, or trusted external service acting in this role) communicates with attester devices (e.g., roots of trust for subsystems and adapters) to determine whether or not the device is trustworthy.

The platform must:

1. Determine which devices are present
2. Collect measurements (e.g., firmware version and cryptographic functions) from each device
3. Verify the device's certificate(s) and the certificate chain back to a trusted root
4. Verify the device's digital signature over the measurements
5. Either accept the device or decide on a remedial action

The attester device must:

1. Contain a tamper-protected, immutable hardware root of trust
2. Be provisioned with a unique identity, firmware, and cryptographic keys in a secure facility
3. Implement secure boot, executing only digitally signed and verified firmware
4. Respond to platform requests for digitally signed measurements (evidence) of the device's configuration.

This document identifies required and optional functionality for platforms and attester devices. Feedback on version 1.0 of this document is invited, especially from vendors implementing it.

Table of Contents

Executive Summary **3**
Purpose **5**
Audience **5**
Syntax and conventions **5**
Requirements, Recommendations, and Choices **5**
REQUIREMENTS - Conformance Statement 5
Introduction **6**
Platforms, Attesters, and Verifiers 6
Interactions Between Verifiers and Attesters 7
Supply Chain Assurance 8
Keys, seeds, and device identifiers **9**
REQUIREMENTS - Keys, Entropy, and Random Bits 11
Certificate chains and credentials **12**
Protocols **13**
Participants 13
Provisioning Facility 13
REQUIREMENTS - Initial Provisioning Environment, Operations, and Equipment 18
Device Ownership Provisioning 19
REQUIREMENTS - Device Ownership Provisioning 19
Discovery and Interrogation protocol 21
REQUIREMENTS - Discovery and Interrogation 22
Authentication, Attestation, and Enrollment protocol 23
REQUIREMENTS - Authentication, Attestation, and Enrollment 25
REQUIREMENTS - SPDM Standards Support 25
Measurement collection and storage **27**
REQUIREMENTS - What to measure and what not to measure 27
REQUIREMENTS - Security-relevant configuration data 27
REQUIREMENTS - When to Measure 28
Policies **29**
Glossary and Abbreviations **30**
Relevant standards, guidelines, and documents **30**
License **31**
About Open Compute Foundation **32**
Appendix A - Summary of Requirements and Recommendations **33**

1. Purpose

This document is intended to create a specification for the functionality and interoperability of attestation operations. These operations produce information about the ownership and configuration of systems (servers) and system components (devices). This document is part of a larger specification created by the OCP Security Project.

2. Audience

The audience for this document includes, but is not limited to, system and system component designers, security information and event management (SIEM) system developers, and cloud service providers.

3. Syntax and conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

The roles "attester", "verifier", and "reference integrity measurements" are defined in the draft Reference Terminology for Attestation Procedures.

4. Requirements, Recommendations, and Choices

Critical requirements, recommendations, and choices described in this document are highlighted *in this style.*

4.1. REQUIREMENTS - Conformance Statement

*The manufacturer / Provisioner **MUST** provide a statement of conformance describing how the attester device satisfies the critical requirements, follows the recommendations, and selects from the choices allowed by this document.*

5. Introduction

5.1. Platforms, Attesters, and Verifiers

A platform verifier (a system such as a server, a storage controller, or a trusted service acting in this role) must assess the trustworthiness of the devices within a platform (physically or logically). It must also determine whether to admit the devices into the platform in their full capacity, admit them in a reduced capacity, exclude them entirely, or disable them. In order to make this determination, it uses attestation from the devices to reliably ascertain their trustworthiness. In this specification, certain devices within a platform are the attesters. Reliability of attestation (over and above simple logs) is established by using proven industry standard cryptographic methods to mitigate unscrupulous behaviors such as (but not limited to) the "lying endpoint" and the "man-in-the-middle".

An attester is a collection of hardware, software, firmware, and a root of trust (RoT) with the ability to provide reliable evidence of trustworthiness (i.e. measurements) to the verifier. For example, an attester may be a network interface controller (NIC), redundant array of independent disks (RAID) controller, or non-volatile memory express (NVMe) solid state drive (SSD). An attester's RoT may be a discrete component with its own firmware and policy, separate from the device it attests.

The relationship between a platform, verifier, and attester is shown in the platform attestation UML model below. A platform must have its own RoT and a verifier that can verify attestations from attester devices. The platform's RoT and verifier may reside in the main processing unit, in a trusted baseboard management controller (BMC), in a dedicated device, or trusted service.

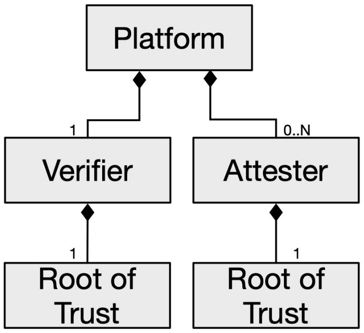


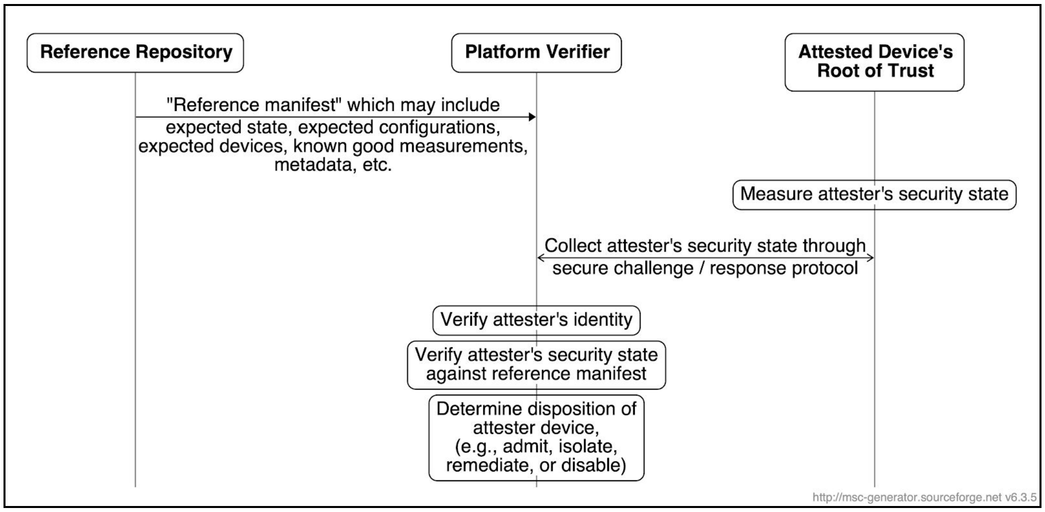
Figure 1. Platform Attestation UML Model

Notice that in this model, attestation is not hierarchical. That is, an attester does not include other attesters, and it is not responsible for verifying another attester attached to it. However, attested devices may act as a bridge, with the responsibility of relaying communication between an attester and a verifier. For example a PCIe bridge may relay communication from a NVMe SSD attached to it, but it is not responsible for verifying the trustworthiness of it. Other examples of bridges include host bus adapters (HBAs), RAID controllers, and NICs.

The detailed content (hardware, software, and firmware) of an attester is out of scope of this specification. However within a platform, there are natural attester boundaries - such as PCIe Card Electromechanical (CEM) form factors, Enterprise & Data Center SSD Form Factor (EDSFF), a SPI bus connecting the attester RoT to its attested device, etc. (generally called field replaceable units).

5.2. Interactions Between Verifiers and Attesters

The figure below is an overview of interactions that may take place between platform verifiers and attesters. Details of these interactions appear later in this specification.


Reference Repository
Platform Verifier
Attested Device's Root of Trust
"Reference manifest" which may include expected state, expected configurations, expected devices, known good measurements, metadata, etc.
Measure attester's security state
Collect attester's security state through secure challenge / response protocol
Verify attester's identity
Verify attester's security state against reference manifest
Determine disposition of attester device, (e.g., admit, isolate, remediate, or disable)
http://msc-generator.sourceforge.net v6.3.5

Each attester device must have a root of trust (RoT).[1] Its RoT is trusted to calculate measurements of the security state of the attester device (e.g., firmware digests, boot parameters, etc.). At a minimum, the RoT is responsible for measuring and reporting the security state of the lowest layer of firmware and the initial security-relevant data in the attester device. Additional layers of firmware measure and record subsequent layers, prior to executing them.

The attester device reports its identity and its measurements to the verifier, which collects and verifies them. The platform verifier may "pull" the measurements from the attester, or the attester may "push" the measurements to the platform verifier. Verification typically involves verifying a digital signature applied by the attester device's RoT, and comparing the reported measurements against a reference manifest. The reference manifest may include, for example, a list of allowed device identities, or a list of hashes of known good firmware for attester devices. Based on the results of the verification, the platform decides what to do with the attested device, e.g., admit it, repair it, isolate or "fence" it, or disable it.

[1] Establishment of the root of trust is outside the scope of this document.

The communication between the attester device’s RoT and the verifier, and between the reference repository and the verifier must be secure against man-in-the-middle attacks, replay attacks, spoofing, and undetected modification.

Attestation may be performed any time during the lifecycle of the platform: during the provisioning process, at initial deployment, periodically while the platform is deployed, at redeployment, or during decommissioning.

Because the degree of certainty needed to establish trustworthiness is a balance between risk tolerance, cost, and ease of use, this specification permits multiple methods of:

1. Establishing the Root of Trust (RoT)
2. Measuring attester state
3. Reporting measured attester state for consumption by the verifier
4. Communicating expected attester state to the verifier
5. Verifying the identity of the attester
6. Verifying the measured attester state
7. Mitigating an untrustworthy attester

5.3. Supply Chain Assurance

Security-critical components in platforms and attester devices should be designed assuming a security-hostile manufacturing environment and should be protected as early as possible in the supply chain. Key material should be protected end-to-end, assuming manufacturing networks are fully compromised. Safeguards include, but are not limited to securing provisioning facilities, limiting physical access, disabling hardware debug interfaces (e.g., JTAG), maintaining chain of custody, auditing quantities of production and scrapped components, protecting firmware development systems, digitally signing firmware, protecting and limiting access to key material, and enabling secure boot. An extensive list of recommendations for securing the supply chain appears in Secure Firmware Development Best Practices and Secure Device Manufacturing: Supply Chain Security Resilience.

At points in the supply chain, provisioning operations establish an attester device’s unique identity and its Device Owner. The Device Owner puts the device into service and determines the authority to update the device. Details of these operations appear later in this specification.

Device ownership may transition from the initial owner, to interim owners, and eventually to the final Device Owner. For example, during final testing, a manufacturing facility may temporarily establish ownership of a device to enable frequent replacement of the firmware. Eventually, it may transfer device ownership to a final customer. The following are common models for changing ownership:

- Send the device back to provisioning to be changed
- Generate new ownership credentials in the field (back to factory state)
- Transfer ownership as directed by the current owner

The following white paper and presentation discuss those and additional approaches to changing ownership.

6. Keys, seeds, and device identifiers

The table below lists the keys, random seeds, and device identifiers used in this document. Official sources of information on cryptographic algorithms, key types, key strength, and cryptoperiods are listed in the section REQUIREMENTS - Keys, Entropy, and Random Bits below. Helpful, but unofficial information is available in D. Giry's interactive article Cryptographic Key Length Recommendation.

Short name	Long name, symbol used in protocols	Key Type / Key Usage	Purpose	Stored where?	Protections	Where generated / created	How initialized	Revocable or updated in field? Under what authority?	Value registered or recorded? Usage auditable?
UDS	Unique Device secret	Seed value generated by random bit generator (RBG) e.g., primary seed	Input to key generation function according to NIST SP800-133	Device persistent protected store or regenerated deterministically on power up	read forbidden except by key generation function write forbidden after provisioning	In device or in secure provisioning facility	Self-generated by device or injected during provisioning	Not recommended Allowed only by device Provisioner to assure device provenance	no
DevIK$_{pr}$	Device identity private key	Private authentication key	Unique for each device Used to sign certificate for DevAK$_{pub}$ and tie DevAK$_{pr}$ to device identity	Device persistent store or regenerated on demand	read forbidden after mfg except by the signing operation write forbidden after provisioning	During device provisioning	Generated on board by key generation function using UDS or injected during provisioning (optionally DICE compliant)	Not recommended Allowed only by device Provisioner to assure device provenance	no
DevIK$_{pub}$	Device identity public key	Public authentication key / digitalSignature	Used to verify device identity	X.509 Cert for public key in device persistent store or external to device	write forbidden after mfg	During device provisioning	Same as DevIK$_{pr}$	Not recommended Allowed only by device Provisioner to assure device provenance	Recorded by platform at discovery
DevAK$_{pr}$	Device attestation private key	Private signature key	Used when attesting device state	Device persistent store or regenerated on demand	read forbidden after mfg except by the signing operation write forbidden after provisioning except when triggered by Device Owner	During device provisioning	Generated on board by key generation function (optionally DICE compliant)	yes by Device Owner	no
DevAK$_{pub}$	Device attestation public key	Public signature - verification key digitalSignature, contentCommit-ment	Used when verifying device state	X.509 Cert for public key in device persistent store or external to device	write forbidden after provisioning except when triggered by Device Owner	During device provisioning	Same as DevAK$_{pr}$	yes by Device Owner	Recorded by platform at discovery
pCA$_{pr}$	Provisioner's Certificate Authority private key	Private authentication key	Used to sign certificate for DevIK$_{pub}$	pCA's HSM for private key	HSM	pCA's HSM	pCA's HSM	Revocable by Provisioner	Audit number of certificates signed

pCA$_{pub}$	Provisioner's Certificate Authority public key	Public authentication key / keyCertSign	Used to verify certificate for DevIK$_{pub}$	X.509 Cert for public key		pCA's HSM	pCA's HSM	Certificate revocation list made available to platform	Recorded in platform in advance of or during device discovery

Short name	Long name, symbol used in protocols	Key Type / Key Usage	Purpose	Stored where?	Protections	Where generated / created	How initialized	Revocable or updated in field? Under what authority?	Value registered or recorded? Usage auditable?
DevOwnCA$_{pr}$	Device Owner's Certificate Authority private key	Private authentication key	Used to sign alternate certificate for DevIK$_{pub}$	DevOwn CA's HSM for private key	HSM	DevOwn CA's HSM	DevOwn CA's HSM	Revocable by Device Owner	Audit number of certificates signed
DevOwnCA$_{pub}$	Device Owner's Certificate Authority public key	Public authentication key / keyCertSign	Used to verify alternate certificate for DevIK$_{pub}$	X.509 Cert for public key stored on or off device		DevOwn CA's HSM	DevOwn CA's HSM	Revoked by Device Owner's CA or removed from platform if Device Owner changes	Recorded in platform in advance of or during device discovery
DevUpdtK$_{pr}$	Device Update private key	Private authorization key	Used to authorize updates to device's critical configuration	Device Updater's HSM for private key	HSM	Device Updater's HSM	Device Updater's HSM	Revocable by device updater	Audit all device updates that were signed / authorized
DevUpdtK$_{pub}$	Device Update public key	Public authorization key digitalSignature, contentCommitment	Used to verify updates to device's critical configuration	Device persistent store	Authenticated update	Device Updater's HSM	When provisioned or when deployed by Device Owner	Removed at change in ownership or by device-specific mechanism	May be attested
FWKeys Manifest	Key Manifest (for secure boot)	List of Public authentication keys	Used to contain list of FWSignK$_{pub}$	Device persistent store	Write authorized only by DevUpdtK$_{pr}$	Device Updater's deployment system	When provisioned or when deployed by Device Owner	Revocable by device updater	Audit all key manifests that were signed / authorized
FWSignK$_{pr}$	Firmware Signer's private key(s)	Private authentication key	Used to sign firmware or critical data	Firmware signer's HSM	HSM	Firmware Signer's HSM	Firmware Signer's HSM	Revocable by Firmware Signer	Audit all firmware that was signed
FWSignK$_{pub}$	Firmware Signer's public key(s)	Public authentication key digitalSignature, contentCommitment, codeSigning	Used to verify signature on firmware or critical data	Device persistent store or Key manifest	Write authorized only by DevUpdtK$_{pr}$	Firmware Signer's HSM	Delivered in Key Manifest	Removal from device authorized by DevUpdtK$_{pr}$	May be attested

6.2. REQUIREMENTS - Keys, Entropy, and Random Bits

*Symmetric keys, asymmetric keys, entropy, and random bits in the key table above **MUST***

- *Follow recommendations in NIST Special Publication 800-57 Recommendation for Key Management*

- *Follow recommendations in NIST Special Publication 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

- *Follow recommendations in NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*

- *Follow recommendations in NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*

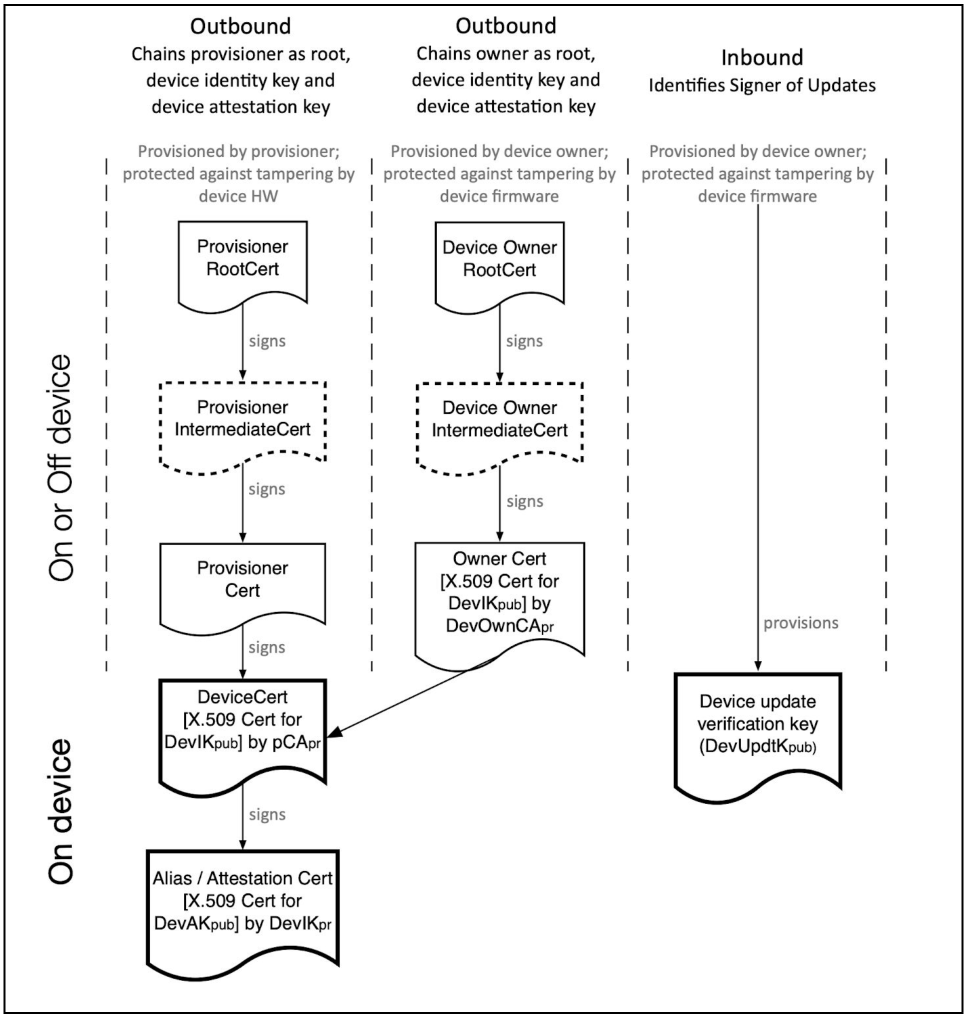
- *Follow the guidance in the Commercial National Security Algorithm (CNSA) Suite regarding quantum resistant algorithms and key sizes.*

- *Provide a statement of minimum key strength and cryptoperiods of the values in the key table above.*

7. Certificate chains and credentials

The figure below depicts certificate chains that link the keys associated with an attester device. Three credentials (drawn in bold lines) are either stored or dynamically generated onboard the device: the Device Identity Key certificate, chained to the Provisioner, the Device Attestation Key certificate, chained to the Device Identity key, and the Device Update public key. The most common configuration is shown in the two outer columns. The middle column shows an alternative configuration in which the

Device Owner adds to or replaces the Provisioner's certificate chain prior to putting the device in service. Note that if the Device Owner replaces the original certificate chain (perhaps because there is not enough storage in which to keep it), then the device cannot be reset securely in the field to its initial, post-provisioning state. Instead, it must be returned to the Provisioner to restore the Provisioner's root and certificate chain.


Outbound
Chains provisioner as root, device identity key and device attestation key
Outbound
Chains owner as root, device identity key and device attestation key
Inbound
Identifies Signer of Updates
Provisioned by provisioner; protected against tampering by device HW
Provisioned by device owner; protected against tampering by device firmware
Provisioned by device owner; protected against tampering by device firmware
On or Off device
On device
Provisioner RootCert
signs
Provisioner IntermediateCert
signs
Provisioner Cert
signs
DeviceCert [X.509 Cert for DevIKpub] by pCApr
signs
Alias / Attestation Cert [X.509 Cert for DevAKpub] by DevIKpr
Device Owner RootCert
signs
Device Owner IntermediateCert
signs
Owner Cert [X.509 Cert for DevIKpub] by DevOwnCApr
provisions
Device update verification key (DevUpdtKpub)

8. Protocols

8.1. Participants

In this specification, the following organizations and equipment participate in the protocols:

- Attester Device
- Attester Device Provisioner
- Attester Device Provisioner's Certificate Authority
- Certificate Registry
- Device Registry
- Device Owner
- Platform
- Verifier

A platform verifier uses the protocols described in this specification to communicate with attester devices to determine whether or not to trust those devices and allow them into the system. These protocols, however, do not describe how the platform verifier actually makes that determination. Some common ways to establish trust include one or more of the following acceptance criteria:

- device certificates chain back to a trusted root certificate authority
- device is certified by a vendor with whom there is a business relationship
- device measurements match a predetermined list (manifest) of measurements
- device accepted on first use, and subsequent measurements match the first

8.2. Provisioning Facility

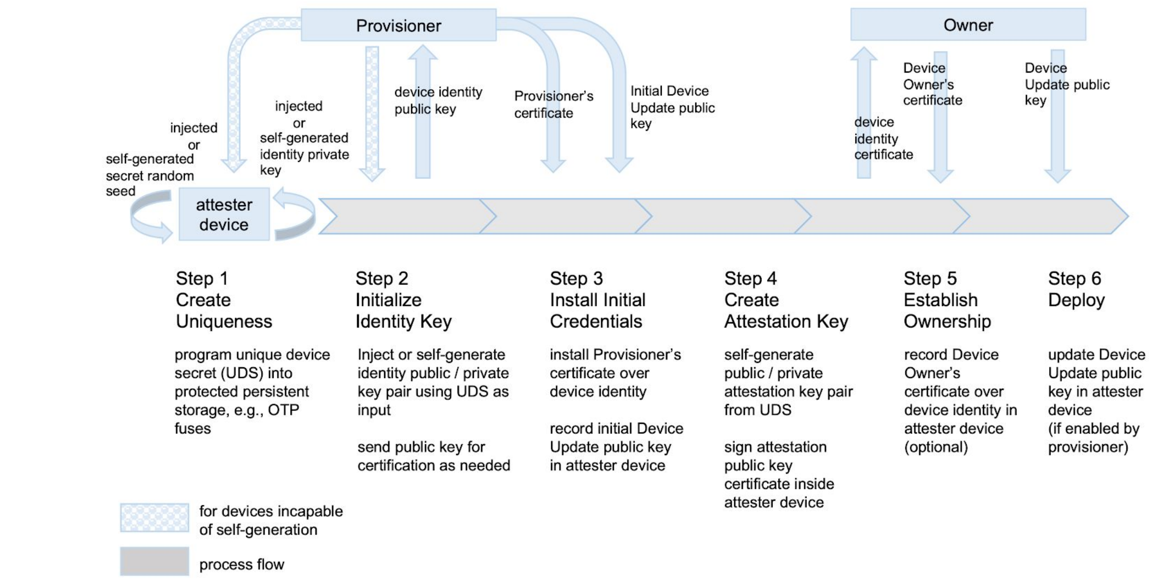
An attester device Provisioner provisions the attester device with a unique device secret, a unique device id key pair, and a corresponding certificate.

Provisioning establishes the **hardware root of trust and the unique, unclonable, and immutable identity of an attester device**. It also creates the device's initial credentials. Use cases for these and subsequent credentials are described in TCG TPM 2.0 Provisioning Guidance, Section 5.1, "Identity," and Section 5.3, "Attestation of Firmware Integrity Measurements."

The participants in this operation are the Provisioner's certification authority (pCA) and the device. An additional participant in this protocol is an optional secure value generator.

Often, however, Provisioners prefer to generate secure values offline and inject (aka "squirt") them into the device. Provisioners prefer to use injection instead of self-generation, because they can use external processes that are much faster at generating keys and deterministic random bits.

Additionally, external processes can perform more extensive entropy tests that a single device might pass, but multiple devices would fail collectively. [2] Thus, there are trade-offs among the throughput of provisioning processes, protection of secure values, and assurance that secure values are properly generated.


Provisioner
Owner
injected or self-generated secret random seed
injected or self-generated identity private key
device identity public key
Provisioner's certificate
Initial Device Update public key
device identity certificate
Device Owner's certificate
Device Update public key
attester device
Step 1 Create Uniqueness
program unique device secret (UDS) into protected persistent storage, e.g., OTP fuses
Step 2 Initialize Identity Key
Inject or self-generate identity public / private key pair using UDS as input
send public key for certification as needed
Step 3 Install Initial Credentials
install Provisioner's certificate over device identity
record initial Device Update public key in attester device
Step 4 Create Attestation Key
self-generate public / private attestation key pair from UDS
sign attestation public key certificate inside attester device
Step 5 Establish Ownership
record Device Owner's certificate over device identity in attester device (optional)
Step 6 Deploy
update Device Update public key in attester device (if enabled by provisioner)
for devices incapable of self-generation
process flow

The figure above illustrates the provisioning of secrets, device identity, device identity and attestation key pairs, the Provisioner's identity, the Device Owner's certificate, and the device update public key. The left to right arrows in the middle row show the progression of an attester device through the provisioning process.

Provisioning step 1 shows two methods of generating the device's unique device secret (UDS): 1) a device generates its own secret, or 2) a hardware security module (HSM) generates it, for later injection into the device. Injecting the UDS exposes it to potential attacks that are not present (or minimized) when the device generates the UDS on its own. Therefore, it is recommended that each attester device generates its own UDS.

In **provisioning step 2**, the UDS is provided as input to a key generation function, which generates a unique device identity key pair (DevIK$_{pub,pr}$). The device identity key can be thought of as a "trustworthy serial number." In some devices, the device identity key never changes throughout the lifetime of the device. In others, the device identity key will change if the UDS or the cryptographic identity of the first mutable firmware changes.

Provisioning step 2 shows two methods of generating the device's identity key pair:

1. A device generates its own keypair, then sends the public key to the Provisioner, or
2. the Provisioner's HSM generates it, for later injection into the device.

[2] Bernstein D.J. et al. (2013) Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild. In: Sako K., Sarkar P. (eds) Advances in Cryptology - ASIACRYPT 2013. ASIACRYPT 2013. Lecture Notes in Computer Science, vol 8270. Springer, Berlin, Heidelberg.

In either method, the Provisioner's HSM, which is a component of the Provisioner's Certificate Authority (pCA), signs a certificate for the device identity public key ($DevIK_{pub}$).

Some devices may simplify the generation of the device identity key pair by following the Device Identifier Composition Engine (DICE) architecture (see TCG Implicit Identity Based Device Attestation).   In DICE, additional values, such as the Compound Device Identifier are incorporated into the key generation function.

Note that injecting the device identity private key exposes it to potential attacks that are not present (or minimized) when the device generates the key pair on its own. Therefore, it is recommended that each attester device generates its own identity key pair. There is a performance benefit to generating the key pair externally: The key pair and certificate for the public key can be created at the same time, without having to wait for the attester device to send the public key to the Provisioner.

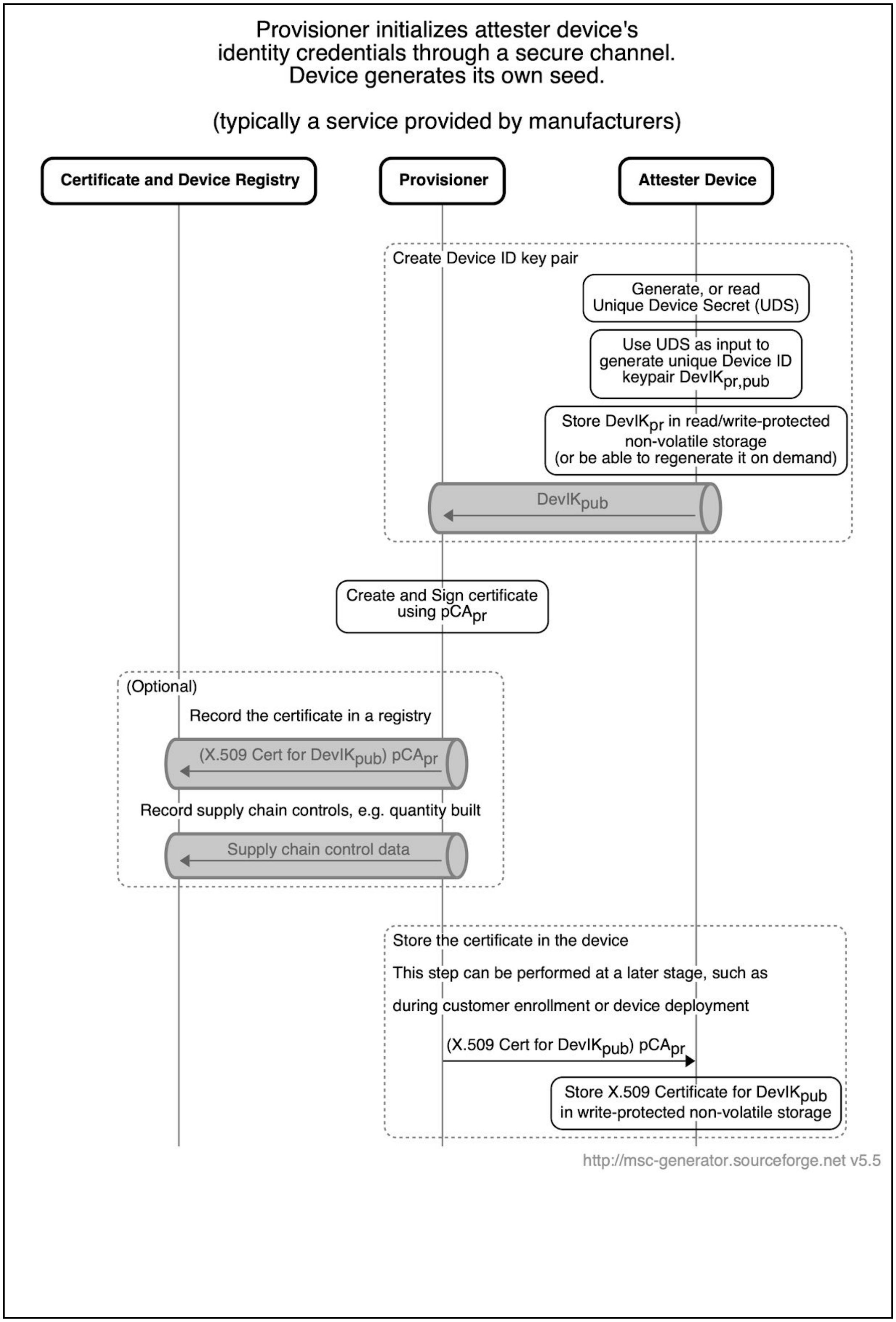
For those familiar with TPMs, provisioning step 2 is somewhat similar to creating the endorsement key, EK, except that there is no concern for anonymity when using the DevIK.

In **provisioning step 3**, the device records the Provisioner's authorization of the device's identity key pair (an X.509 certificate signed by the Provisioner). The Provisioner may optionally record an initial device updater's public key. This key may be temporary, such as one used to verify the signature of manufacturing test key manifests, or an interim one until the Device Owner replaces it later. Alternatively, the Provisioner may record a device updater's key and lock it permanently in the device.

In **provisioning step 4**, the device uses an initial random secret to generate a device attestation key pair ($DevAK_{pu}$, $DevAK_{pr}$). As in provisioning step 2, the key generation function may optionally follow the DICE architecture. The device fills in a certificate template with the attestation public key, then signs the certificate using $DevIK_{pr}$.

In **provisioning step 5**, the Device Owner establishes ownership of the device (see "Device Ownership Provisioning" below).

In **provisioning step 6**, the Device Owner provides the device update public key, and the device records it to use later to verify signatures on updates to its critical configuration.

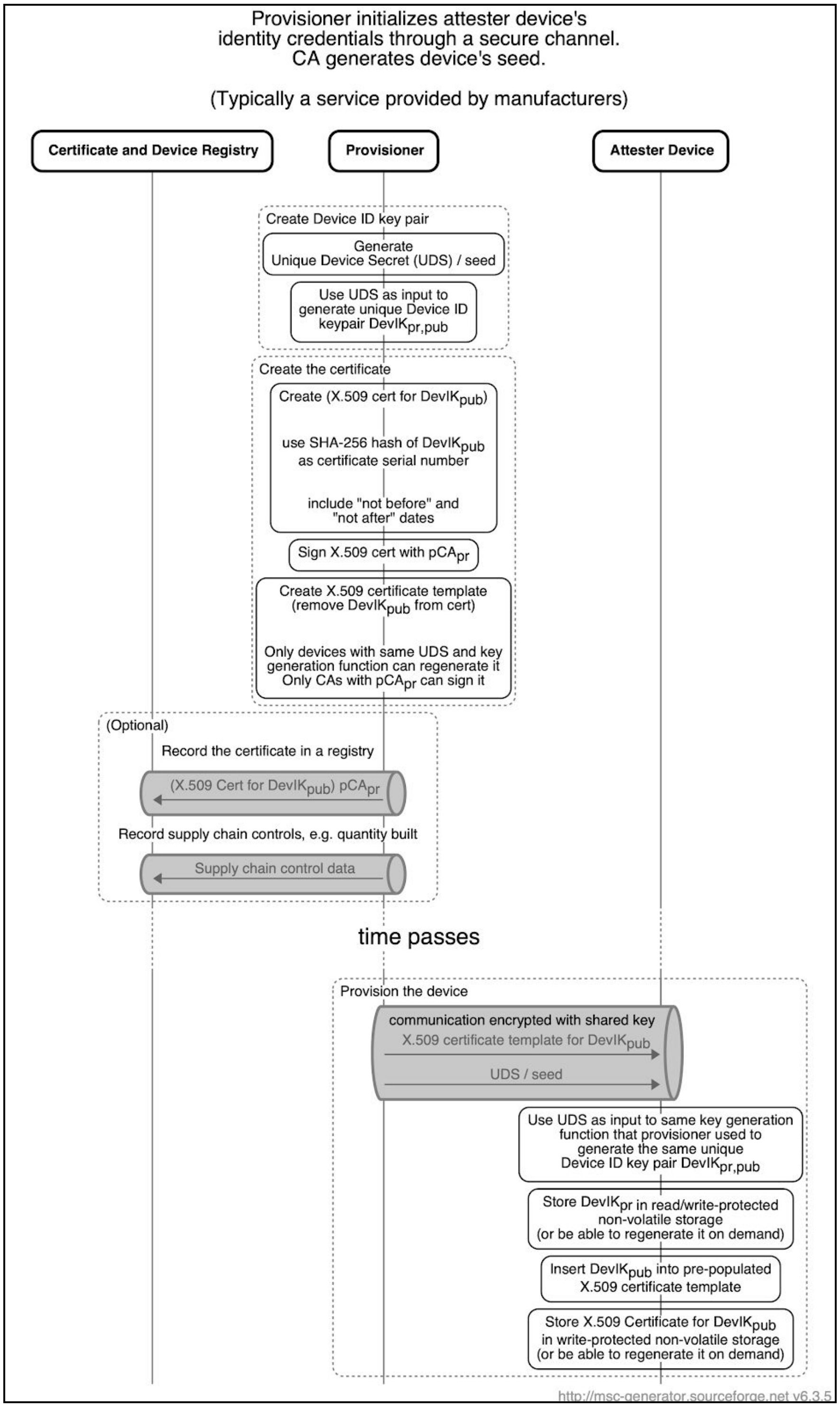


Provisioner initializes attester device's identity credentials through a secure channel. Device generates its own seed.

(typically a service provided by manufacturers)

Certificate and Device Registry
Provisioner
Attester Device
Create Device ID key pair
Generate, or read Unique Device Secret (UDS)
Use UDS as input to generate unique Device ID keypair $\mathrm{DevIK}_{pr,pub}$
Store DevIK_{pr} in read/write-protected non-volatile storage (or be able to regenerate it on demand)
DevIK_{pub}
Create and Sign certificate using pCA_{pr}
(Optional)
Record the certificate in a registry
(X.509 Cert for DevIK_{pub}) pCA_{pr}
Record supply chain controls, e.g. quantity built
Supply chain control data
Store the certificate in the device
This step can be performed at a later stage, such as
during customer enrollment or device deployment
(X.509 Cert for DevIK_{pub}) pCA_{pr}
Store X.509 Certificate for DevIK_{pub} in write-protected non-volatile storage
http://msc-generator.sourceforge.net v5.5

The protocol diagram above shows the interaction between a newly manufactured attester device and a Provisioner, as they interact to provision the identity credentials for a device. In this interaction, the device creates its own unique device secret (UDS). There are no challenge-responses, because there is not yet a root of trust in the device on which the pCA can rely. The benefit of this approach is that the UDS is never exposed outside of the device.


Provisioner initializes attester device's
identity credentials through a secure channel.
CA generates device's seed.
(Typically a service provided by manufacturers)
Certificate and Device Registry
Provisioner
Attester Device
Create Device ID key pair
Generate Unique Device Secret (UDS) / seed
Use UDS as input to generate unique Device ID keypair DevIK$_{pr,pub}$
Create the certificate
Create (X.509 cert for DevIK$_{pub}$)
use SHA-256 hash of DevIK$_{pub}$ as certificate serial number
include "not before" and "not after" dates
Sign X.509 cert with pCA$_{pr}$
Create X.509 certificate template (remove DevIK$_{pub}$ from cert)
Only devices with same UDS and key generation function can regenerate it
Only CAs with pCA$_{pr}$ can sign it
(Optional)
Record the certificate in a registry
(X.509 Cert for DevIK$_{pub}$) pCA$_{pr}$
Record supply chain controls, e.g. quantity built
Supply chain control data
time passes
Provision the device
communication encrypted with shared key
X.509 certificate template for DevIK$_{pub}$
UDS / seed
Use UDS as input to same key generation function that provisioner used to generate the same unique Device ID key pair DevIK$_{pr,pub}$
Store DevIK$_{pr}$ in read/write-protected non-volatile storage (or be able to regenerate it on demand)
Insert DevIK$_{pub}$ into pre-populated X.509 certificate template
Store X.509 Certificate for DevIK$_{pub}$ in write-protected non-volatile storage (or be able to regenerate it on demand)
http://msc-generator.sourceforge.net v6.3.5

The protocol diagram above shows a variation of the interaction between a newly manufactured attester device and a Provisioner. In this interaction, the Provisioner creates the unique device secret (UDS), and both the Provisioner and the device use it as input to a key generation function to derive the device’s unique identity keypair. The benefit of this approach is that a certificate authority is not required on the manufacturing line. The drawback is that the UDS is vulnerable to disclosure at the Provisioner’s site.

8.2.1. REQUIREMENTS - Initial Provisioning Environment, Operations, and Equipment

- *Initial provisioning operations **MUST** be carried out in a trusted facility, in which a secure channel between the Provisioner and the device is guaranteed.*

- *The Provisioner **MUST** report which of the following provisioning methods is used:*
  {attester device self-generates both UDS and DevIK$_{pr}$,
  Provisioner injects UDS and device self-generates DevIK$_{pr}$, or
  Provisioner injects both UDS and DevIK$_{pr}$}.

- *Cryptographic algorithms and deterministic random bit generators **MUST** be validated under the NIST Cryptographic Algorithm Validation Program (CAVP)*

- *Cryptographic modules, if used, **SHOULD** be validated at overall level 2 or higher under FIPS 140-2 SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES or Security Requirements for Cryptographic Modules, FIPS 140-3*

- *Entropy, random bits, symmetric keys, and private asymmetric keys **MUST** be generated within the attester device itself, in a hardware security module, or locally, in a device with the following properties:*

  - *Follows recommendations in NIST Special Publication 800-90A Rev 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

  - *Follows recommendations in NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*

  - *Follows recommendations in NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*

  - *Complies with Annex C: Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules*

  - *Follows the guidance in the Commercial National Security Algorithm Suite regarding quantum resistant algorithms and key sizes.*

- *Each attester device has the following properties:*
    - ○ *Each attester device **MUST** have a unique, and immutable device ID key pair.*
    - ○ *Each attester device **MAY** be provisioned with a hash of the first mutable firmware.*
    - ○ *Each attester device **MUST** prevent exfiltration of device secrets through defined interfaces.*
    - ○ *The Provisioner **MUST** generate a certificate signed by its pCA private key, which links the unique device identity with its Provisioner.*
    - ○ *Each attester device **MUST** generate a certificate signed by the device ID private key, which links the attestation public key with the device identity.*

8.3. Device Ownership Provisioning

The first Device Owner provisions the attester device with a Device Update public key (DevUpdtK$_{pub}$). The attester device uses this key to verify the authenticity and integrity of updates to the device's critical configuration.

The device ownership provisioning operation should take place as early as possible in the lifecycle of the attester device, ideally as soon as the identity of the device (DevIK$_{pub,pr}$) is established. Until the legitimate Device Update public key is provisioned, the attester device is vulnerable to an attack in which an attacker's own Device Update public key is provisioned, then later authorizes malicious updates to the attester device.

Optionally, the Device Owner's Certificate Authority generates a certificate over the device identity public key (DevIK$_{pub}$). If present, this certificate can be presented during attestation operations to attest that an owner has taken ownership of the device.

The parties to this protocol are the Device Owner and the attester device. (Sometimes, the Provisioner and the Device Owner are one and the same.)

8.3.1. REQUIREMENTS - Device Ownership Provisioning

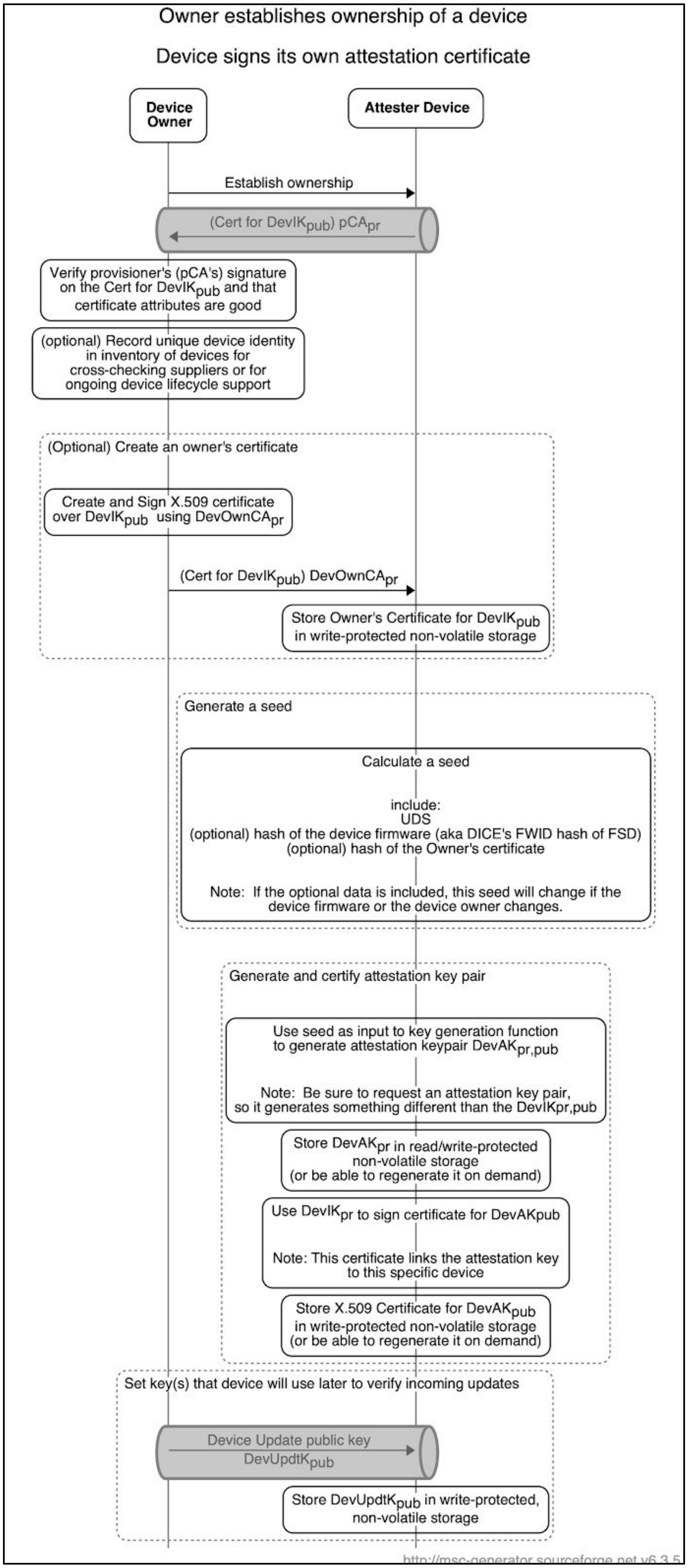
- *Each attester device **MUST** be provisioned with a Device Update public key, which is used to verify updates to the device's critical configuration.*

- *The Device Update public key, once provisioned on the attester device, **MUST** only be modified through an authenticated ownership transfer.*

- *Each attester device **MAY** be provisioned with a Device Owner's certificate over the device identity public key.*

- *Each attester device **MAY** be provisioned with a hash of the device firmware.*

- *Each attester device **MAY** be provisioned with a hash of the Device Owner's certificate over the device identity public key.*


Owner establishes ownership of a device
Device signs its own attestation certificate
Device Owner
Attester Device
Establish ownership
(Cert for $DevIK_{pub}$) pCA_{pr}
Verify provisioner's (pCA's) signature on the Cert for $DevIK_{pub}$ and that certificate attributes are good
(optional) Record unique device identity in inventory of devices for cross-checking suppliers or for ongoing device lifecycle support
(Optional) Create an owner's certificate
Create and Sign X.509 certificate over $DevIK_{pub}$ using $DevOwnCA_{pr}$
(Cert for $DevIK_{pub}$) $DevOwnCA_{pr}$
Store Owner's Certificate for $DevIK_{pub}$ in write-protected non-volatile storage
Generate a seed
Calculate a seed
include:
UDS
(optional) hash of the device firmware (aka DICE's FWID hash of FSD)
(optional) hash of the Owner's certificate
Note: If the optional data is included, this seed will change if the device firmware or the device owner changes.
Generate and certify attestation key pair
Use seed as input to key generation function to generate attestation keypair $DevAK_{pr,pub}$
Note: Be sure to request an attestation key pair, so it generates something different than the DevIKpr,pub
Store $DevAK_{pr}$ in read/write-protected non-volatile storage (or be able to regenerate it on demand)
Use $DevIK_{pr}$ to sign certificate for DevAKpub
Note: This certificate links the attestation key to this specific device
Store X.509 Certificate for $DevAK_{pub}$ in write-protected non-volatile storage (or be able to regenerate it on demand)
Set key(s) that device will use later to verify incoming updates
Device Update public key $DevUpdtK_{pub}$
Store $DevUpdtK_{pub}$ in write-protected, non-volatile storage
http://msc-generator.sourceforge.net v6.3.5

The protocol diagram above shows the interaction between a newly provisioned attester device and a Device Owner, as they interact to provision ownership information, attestation credentials, and update verification keys for a device.

First, the Device Owner verifies that the device was provisioned by a trusted provisioner and optionally issues its own certificate over the device's unique identity public key.

Next, the device calculates a seed which will be used as input to the key generation function for its attestation key.   It can use the same UDS and key generation function as was used when generating the device identity key (DevIK), but the key generation function must be informed that the key is an authentication key, otherwise it would create a key pair identical to DevIK.

In the DICE architecture, the attestation key is derived from the device firmware. If the device firmware changes, then a new attestation key is regenerated. In this specification, including a hash of the device firmware is optional. Another optional parameter to the key derivation function is a hash of the Device Owner's public key. If it is included and the owner changes, then a new attestation key is regenerated.

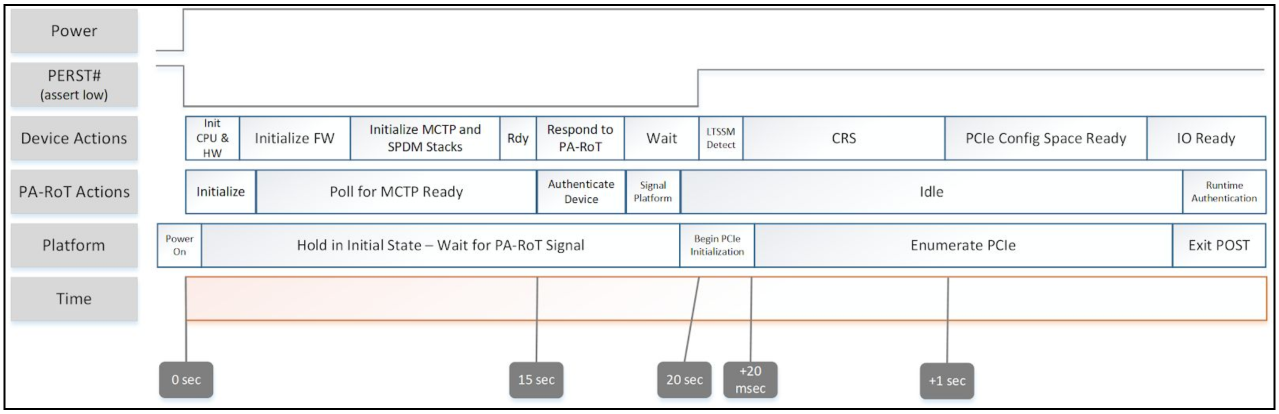
Some attester devices may not be able to generate and sign an X.509 certificate. In an alternative method, the device builds a lightweight TPM-style certificate containing the attestation public key, then signs it using its device ID private key. This TPM-style certificate is then sent to the Device Owner to be parsed, converted to an X.509 certificate, signed by the Device Owner's CA, and sent back to the device. Just as in the provisioning protocol above, these operations must be carried out in a secure facility or in a trusted platform.

8.4. Discovery and Interrogation protocol

The platform determines what attester devices are present and their authentication and attestation capabilities.

The platform begins to build a platform inventory of the attester devices present. (It completes the inventory after the authentication and enrollment protocol.) Platforms should be able to perform such a discovery as soon as power is up. On some platforms, discovery would happen after the firmware completes device initialization. However, it is expected that some platforms would perform early discovery, and sequence the startup process by holding devices in reset until they are discovered and optionally checked for integrity, using a sideband for the discovery. It is also expected that some devices may not support discovery before they are taken out of reset, mainly for legacy and flashless devices, but for modern and smart devices, this should be made possible.

Platforms can optionally hold the main data bus (i.e. PCIe) in reset while the platform interrogates the attester devices. This process is referred to as a split reset sequence. In this use case, the attester device is required to respond to requests while its main data bus interface is held in reset. One example of this mechanism is to hold PCIe reset (PERST#) asserted while the interrogation process occurs. The following diagram shows an example of this sequence using PCIe as the main data bus. In the following diagram, the platform holds PERST# asserted while attester devices are allowed to initialize their CPUs and firmware stacks. When the attester devices are ready to respond, the platform interrogates the attester devices according to its policies. When the platform is complete with the interrogation and satisfied with its results, the platform allows the attester devices out of reset.



Another possible reset sequence is for the platform to be held in a state that cannot cause harm. This process is referred to as a unified reset sequence. In this case, the attester devices are brought out of reset as normal and interrogated by the platform. When the platform is complete with the interrogation and satisfied with its results, the platform is admitted into production servicing, or allowed to move to its fully operational state. When using a unified reset sequence, the previous diagram does not apply.

An implementation can also use an external root of trust chip that controls the attester device. In such an implementation, the RoT chip controls the sequence of the attestation operations and attester device reset. In this case, the platform initialization appears to be unaltered from the perspective of the attester device.

This protocol is highly dependent on the specific technology of the platform, bus, and devices, and thus is out of scope. The goal, however, is in scope. The goal is to build a platform inventory containing a list of all security-relevant devices, whether or not they support authentication and attestation, and, if they do, what commands they support.

8.4.1. REQUIREMENTS - Discovery and Interrogation

- *Attester devices **MUST** be capable of communicating their authentication and attestation capabilities to the platform.*

- *Attester devices **SHOULD** be capable of communicating their capabilities to the platform within 15 seconds of being provided with power, even if their data plane bus (e.g. PCIe) is held in reset by the platform.*

- *Platforms **MUST** be capable of interrogating potential attester devices and recording their authentication and attestation capabilities.*

- *Platforms **MUST** be capable of interrogating attester devices that do not communicate their capabilities before being taken out of reset, e.g., by interrogating them later in the boot cycle or by having them pre-configured as such, in the platform reference manifest.*

- *Platforms **MAY** use the message formats for GET_CAPABILITIES and NEGOTIATE_ALGORITHMS as described in* Security Protocol and Data Model (SPDM) Specification or *Device Capabilities as described in Project Cerberus Firmware Challenge Specification . Where necessary, bridge components may be responsible for translating from the native bus protocol into the GET_CAPABILITIES/ NEGOTIATE_ALGORITHMS message formats.*

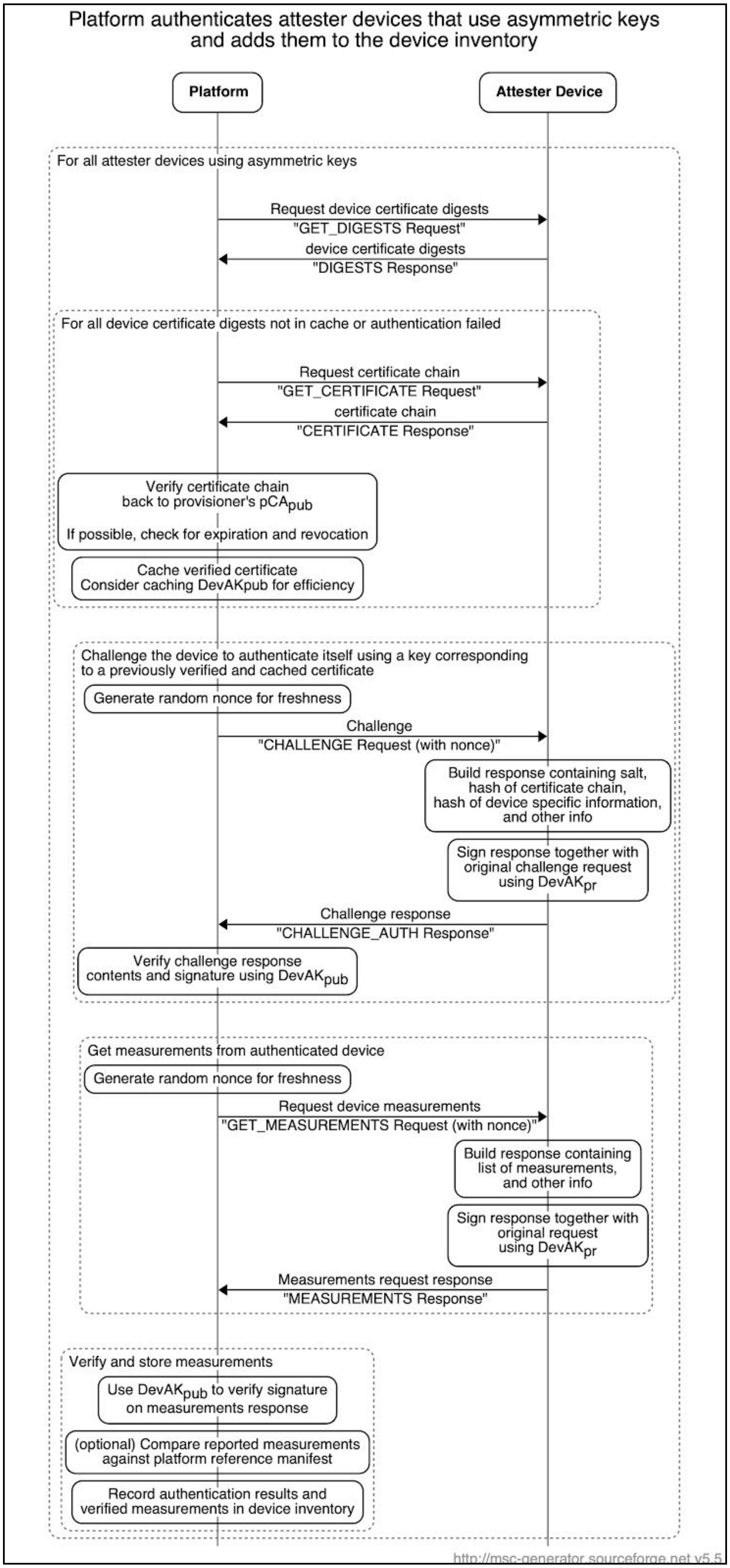
8.5. Authentication, Attestation, and Enrollment protocol

After the platform completes the interrogation phase, it authenticates attester device identities, and completes an inventory of authenticated devices and their measurements.

In this protocol,

1. The platform authenticates the identity of each attester device by collecting (from the device) and verifying the certificate chain of its attestation key, all the way back to the Provisioner's root public key pCA_{pub}. This operation assures that the attestation key is on a device that was provisioned by a trusted provisioner. For efficiency, the platform may cache the digests of the certificates to avoid having to verify them again, and to reduce the amount of storage required to store the entire chain.
2. The platform verifies that each attester device possesses and can use a private authentication key corresponding to the certificate chain that was verified in the previous step. Optimization allows a platform to skip re-verification of a certificate chain. An attacker device may attempt to take advantage of this optimization by presenting the certificate chain (and hash) of a good device. If so, then this step will fail, because the attacker device cannot sign the challenge response with a private key corresponding to the certificate hash in the response, which also matches the certificate hash previously verified and cached by the platform.
3. The platform verifies the authenticity and integrity of measurements of the firmware on authenticated attester devices.
4. The platform assembles an inventory of authenticated attester devices, their identities, and their associated measurements.
5. The platform optionally compares the assembled inventory to a platform reference manifest of expected devices and measurements.
6. Should any steps in the protocol fail, a platform-dependent action is taken, such as admit the attested device, repair it, isolate or "fence" it, or disable it.

Example message formats and content are as described in Security Protocol and Data Model (SPDM) Specification or Project Cerberus Firmware Challenge Specification.


Platform authenticates attester devices that use asymmetric keys and adds them to the device inventory
Platform
Attester Device
For all attester devices using asymmetric keys"
Request device certificate digests
"GET_DIGESTS Request"
device certificate digests
"DIGESTS Response"
For all device certificate digests not in cache or authentication failed
Request certificate chain
"GET_CERTIFICATE Request"
certificate chain
"CERTIFICATE Response"
Verify certificate chain back to provisioner's pCA_{pub}
If possible, check for expiration and revocation
Cache verified certificate
Consider caching DevAKpub for efficiency
Challenge the device to authenticate itself using a key corresponding to a previously verified and cached certificate
Generate random nonce for freshness
Challenge
"CHALLENGE Request (with nonce)"
Build response containing salt, hash of certificate chain, hash of device specific information, and other info
Sign response together with original challenge request using $DevAK_{pr}$
Challenge response
"CHALLENGE_AUTH Response"
Verify challenge response contents and signature using $DevAK_{pub}$
Get measurements from authenticated device
Generate random nonce for freshness
Request device measurements
"GET_MEASUREMENTS Request (with nonce)"
Build response containing list of measurements, and other info
Sign response together with original request using $DevAK_{pr}$
Measurements request response
"MEASUREMENTS Response"
Verify and store measurements
Use $DevAK_{pub}$ to verify signature on measurements response
(optional) Compare reported measurements against platform reference manifest
Record authentication results and verified measurements in device inventory
http://msc-generator.sourceforge.net v5.5

8.5.1. REQUIREMENTS - Authentication, Attestation, and Enrollment

1. *Attester devices **MUST** provide certificate digests and certificates when requested by the platform.*

2. *Attester devices **MUST** build and sign responses to challenges from the platform. Although this step is optional in the SPDM specification, it is required here.*

3. *Platform verifiers **MUST** request certificate digests and certificates from attester devices.*

4. *Platform verifiers **MUST** verify ASN.1 DER encoded X.509v3 certificates and certificate chains from attester devices back to each device's Provisioner root public key.*

5. *Platform verifiers **MUST** present challenges to attester devices and verify the content in the responses.*

6. *Platform verifiers **MUST** verify attester device signatures on the challenge responses.*

7. *Platform verifiers **MAY** build a platform inventory containing authentication status, firmware signing keys, firmware measurements, and Device Owners of attester devices.*

8. *Platform verifiers **MAY** accept a predefined manifest (an expected inventory of devices) or build it dynamically.*

9. *Platform verifiers **MAY** compare the platform inventory to a platform manifest containing expected devices and their configurations.*

8.5.2. REQUIREMENTS - SPDM Standards Support

10. *Attester devices that support the SPDM standard **MUST** conform to the set of capabilities as defined in the table "Required Capabilities for SPDM."*

11. *Attester devices that support the SPDM standard **SHOULD** support the set of algorithms as defined in the table "Recommended Algorithms for SPDM".*

12. *Attester devices that support the SPDM standard **MUST** support SPDM version 1.0 or higher.*

13. *Attester devices that support the SPDM standard **SHOULD** support the current version.*

8.5.2.1. Required Capabilities for SPDM

The following table lists the SPDM capabilities, as defined in the CAPABILITIES response, that are required for attester devices that are compliant with this specification. Note, this table is based on version 1.1.0 of the SPDM specification, and capabilities that are only defined in version 1.1.0 are not required if the attester device does not support version 1.1.0.

Capability	**Description**
CERT_CAP	Supports certificate exchanges
CHAL_CAP	Supports challenge
MEAS_CAP	Supports MEASUREMENTS and should support signed MEASUREMENTS (SPDM MEAS_CAP = 10b)

8.5.2.2. Recommended Algorithms for SPDM

Attester devices are allowed a large number of algorithm combinations under the SPDM Specification. To improve compatibility, attester devices should follow the guidelines in this section.

Minimum Support	
MeasurementHashAlgo	TPM_ALG_SHA_384
BaseAsymAlgo	TPM_ALG_RSASSA_3072
BaseHashAlgo	TPM_ALG_SHA_256

Recommended Support	
MeasurementHashAlgo	TPM_ALG_SHA_512
BaseAsymAlgo	TPM_ALG_ECDSA_ECC_NIST_P384
BaseHashAlgo	TPM_ALG_SHA_384

9. Measurement collection and storage

9.1. REQUIREMENTS - What to measure and what not to measure

- *The measurements **MUST** include everything that affects the security of the attester device, such as executable code, headers, security state and configuration data.*
- *The measurements **MUST** exclude information that will make the measurements brittle, such as run-time configuration data that does not impact the security of the device, and information which is expected to be updated frequently on the device.*

Some measurements may not be obvious, may affect the security of the device, and may or may not make the measurements brittle. For example, many flash devices contain a foundry-installed serial number which could be included in the measurements to detect flash replacement attacks (which can bypass flash read-only protections).

9.2. REQUIREMENTS - Security-relevant configuration data

*In the event that configuration data for the device may lead to compromise of the security of the device (such as fuses or straps that enable JTAG or other test interfaces), this class of configuration data **MUST** be discoverable from the device and/or cause the measurements of the device to be distinguishable from production measurements. The mechanism for detecting/providing this information to the attester device **MUST** be enforced through pure hardware means.*

Reset state

*On attester device reset, the measurement registers **MUST** be cleared (reset to 0s) and a measurement indicating an attester device reset event **MUST** be extended to the measurement register.*

*Resetting the attester measurements independently from the system that it measures **MUST NOT** be possible through a purely software mechanism (avoid separate reset and power signals).*

Security / integrity of the measurement storage

*The measurement storage **MUST** be integrity protected to prevent malicious or inadvertent modification, but it is not confidential.*

Measurement logs

*Measurement storage **MAY** include a structured log of measurements.*
This log is used by the platform to derive and verify extended components or measurements. The log may also contain unprotected metadata associated with the measurements. Desirable properties of measurement logs are

- Tamper evident
- Tamper resistant
- Contains a sufficient number of events to support analysis
- Indicates the relative time spanned in the log

The platform collects any logs it collects from attester devices. Additionally, the platform may maintain a log which contains an aggregation of the state of the attester devices. One such event log has been standardized by the Trusted Computing Group (TCG). Tooling already exists that supports parsing and verifying TCG event logs. (See section 5 of TCG EFI Protocol Specification) Although the TCG logs are not a perfect fit, platforms and attester devices may be able to map their events to TCG events and measurement storage to platform configuration registers, in order to produce standardized logs.

Algorithms for cumulative measurements

When multiple measurements are accumulated in one register, writes should update the measurement register as follows:

new value ≔ Hash (old value || input provided)

This calculation follows that of Trusted Computing Group's "Extend" operation as described in TCG EFI Protocol Specification. Note that the size of "input provided" is the same as the size of the digest of the hash algorithm. The size also must be constant to avoid length extension attacks as described in https://en.wikipedia.org/wiki/Length_extension_attack .

Security / integrity of object that was measured

Firmware (data and code) left exposed on external flash is vulnerable to time-of-check time-of-use problems (TOCTOU). The problem is that it may be modified between the time the verification is done, but before it is executed. Therefore, it must be validated on EVERY read.

Devices with limited memory and execute-in-place devices are particularly vulnerable to TOCTOU problems. If firmware must be staged or loaded in pieces, then it must also be verified in stages or pieces, before it is executed.

9.3. REQUIREMENTS - When to Measure

Before execution, following a layered approach

*Before executing or transferring control to mutable code, immutable code **MUST** record measurements of the mutable code and of relevant configuration settings*. For example, a read-only section of secure boot firmware on an attester device, before executing the remainder of the device's boot firmware stored in writable flash, must record a hash of that writable firmware. It may also record the public key it used to verify the digital signature on the writable firmware.

Next, the mutable code must record measurements and relevant configuration settings of the next layer of mutable code (also *before* executing or transferring control to it).

Cold vs. warm boot

Devices should reset any measurement state, and perform a full boot from a hardware root-of-trust whenever the device undergoes a full power-reset.

Resets and updates

*If an attester device's state changes (e.g., reset or chain of trust has become invalidated), the attester **MUST** generate a signal or counter to notify the platform of the change.*

*If the attester device's state changes, the platform and device **MUST** repeat the complete attestation protocol.*
The device may maintain an optional non-volatile counter that is incremented on each secure boot. This counter can be included in the set of values returned during attestation, and can be used by the verifier to detect any unexpected device resets (which could be indicative of suspicious activity). However, in certain power states, some devices will reboot, thus inflating the counter and making it an unreliable indicator of suspicious activity.

Dynamic modification of device

*If a device allows for the runtime modification of any state that can affect the security properties of the device, such modifications **SHOULD** be reflected in the measurements and measurement logs of the device.* Examples of such dynamic modification include loading new firmware without rebooting the device, or enabling debug functionality (that can affect the security state of the device).

Entry into Debug mode

If the activation of debug capabilities in the device can have security implications (including ability to read or modify registers or memory, bypass secure boot capabilities, load untrusted firmware, read performance state that can be used to extract side-channel information, etc.), such an activation needs to be reflected in the measurements and measurement logs.

*It **SHOULD NOT** be possible for any debug mode to reset the measurement values, or make arbitrary changes to them.* Only extensions must be permitted.

Continuous monitoring

In order to support continuous monitoring by the platform, *it is **RECOMMENDED** that attester devices be able to respond to an attestation request at any time during the device's normal runtime operation.*

11. Policies

When putting this specification into practice, there are many decisions that must be left to the implementation. These decisions cover topics such as

- What to measure
- What authority is authorized to request attestations

- What to do with devices with certificates that expired while sitting in a warehouse
- How to handle errors, e.g. ignore, log and keep going, or log and fail
- Whether to admit devices incapable of attestation
- Whether to admit immutable devices that cannot be updated (or smart devices that may be masquerading as immutable ones)
- Whether to admit devices that can reset themselves without the intervention or knowledge of the platform
- How to handle attested devices where attestation has not succeeded: e.g., admit it with a notice of unsuccessful attestation, repair it, isolate or “fence” it, or disable it.

12. Glossary and Abbreviations

See Glossary and Abbreviations

13. Relevant standards, guidelines, and documents

[1] Approved Random Number Generators for FIPS PUB 140-2 Annex C (DRAFT) Security Requirements for Cryptographic Modules

[2] Automated Proof for Authorization Protocols of TPM 2.0 in Computational Model (full version)

[3] Bernstein D.J. et al. (2013) Factoring RSA Keys from Certified Smart Cards: Coppersmith in the Wild. In: Sako K., Sarkar P. (eds) Advances in Cryptology - ASIACRYPT 2013. ASIACRYPT 2013. Lecture Notes in Computer Science, vol 8270. Springer, Berlin, Heidelberg.

[4] BCP 14 - Key words for use in RFCs to Indicate Requirement Levels

[5] Commercial National Security Algorithm Suite

[6] Cryptographic Key Length Recommendation

[7] Device Identifier Composition Engine (DICE) Architectures

[8] IEEE 802.1AR Secure Device Identity

[9] Implicit Identity Based Device Attestation v1 rev93

[10] Implementing DICE, Trusted Computing Group 3/20/2018

[11] Length Extension Attacks

[12] NIST Special Publication 800-57 Part 1 Rev. 5, Recommendation for Key Management, Part 1: General

[13] NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation

[14] NIST Special Publication 800-108 Recommendation for Key Derivation Using Pseudorandom Functions (Revised)

[15] NIST Special Publication 800-131a Rev. 2, Transitioning the Use of Cryptographic Algorithms and Key Lengths

[16] NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation

[17] NIST Special Publication 800-155 (DRAFT), BIOS Integrity Measurement Guidelines

[18] NIST Special Publication 800-193, Platform Firmware Resiliency Guidelines

[19] Open Compute Project, Project Cerberus Firmware Update Specification

[20] Project Cerberus Firmware Challenge Specification

[21] Reference Terminology for Attestation Procedures

[22] RFC 2119, Key words for use in RFCs to Indicate Requirement Levels

[23] RFC 5280, Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, Section 4.2.1.3

[24] RFC 8174 Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words

[25] Secure Device Manufacturing: Supply Chain Security Resilience

[26] Secure Firmware Development Best Practices, July 2019

[27] Security Protocol and Data Model (SPDM) Specification version 1.0.0, 2019-12-11

[28] Security Requirements for Cryptographic Modules, FIPS 140-2

[29] Security Requirements for Cryptographic Modules, FIPS 140-3

[30] Security Requirements for PMCI Standards and Protocols

[31] TCG EFI Platform Specification For TPM Family 1.1 or 1.2, Specification Version 1.22 Revision 15, January 27, 2014

[32] TCG EFI Protocol Specification, Family“2.0”Level 00 Revision 00.13 March 30, 2016

[33] TCG Infrastructure WG TPM Keys for Platform Identity for TPM 1.2, Specification Version 1.0 Revision 3, August 21 2015

[34] TCG TPM 2.0 Provisioning Guidance v1.0 rev 1.0, March 15, 2017

[35] TCG Trusted Platform Module Library Part 1: Architecture, Family “2.0” Level 00 Revision 01.59 November 8, 2019

[36] USB Authentication Specification Rev. 1.0 with ECN and Errata through January 7, 2019

[37] Universal Serial Bus Revision 3.2 Specification

14. License

OCP encourages participants to share their proposals, specifications and designs with the community. This is to promote openness and encourage continuous and open feedback. It is important to remember that by providing feedback for any such documents, whether in written or verbal form, that the contributor or the contributor's organization grants OCP and its members irrevocable right to use this feedback for any purpose without any further obligation.

It is acknowledged that any such documentation and any ancillary materials that are provided to OCP in connection with this document, including without limitation any white papers, articles, photographs, studies, diagrams, contact information (together, “Materials”) are made available under the Creative Commons Attribution-ShareAlike 4.0 International License found here: https://creativecommons.org/licenses/by-sa/4.0/, or any later version, and without limiting the foregoing, OCP may make the Materials available under such terms.

As a contributor to this document, all members represent that they have the authority to grant the rights and licenses herein. They further represent and warrant that the Materials do not and will not violate the copyrights or misappropriate the trade secret rights of any third party, including without limitation rights in intellectual property. The contributor(s) also represent that, to the extent the Materials include materials protected by copyright or trade secret rights that are owned or created by any third-party, they have obtained permission for its use consistent with the foregoing. They will provide OCP evidence of such permission upon OCP’s request. This document and any "Materials" are published on the respective project's wiki page and are open to the public in accordance with OCP's Bylaws and IP Policy. This can be found at

http://www.opencompute.org/participate/legal-documents/.

If you have any questions please contact OCP.

15. About Open Compute Foundation

The Open Compute Project Foundation is a 501(c)(6) organization which was founded in 2011 by Facebook, Intel, and Rackspace. Our mission is to apply the benefits of open source to hardware and rapidly increase the pace of innovation in, near and around the data center and beyond. The Open Compute Project (OCP) is a collaborative community focused on redesigning hardware technology to efficiently support the growing demands on compute infrastructure. For more information about OCP, please visit us at http://www.opencompute.org

Appendix A - Summary of Requirements and Recommendations

A1. REQUIREMENTS - Conformance Statement

a. *The manufacturer / Provisioner **MUST** provide a statement of conformance describing how the attester device satisfies the critical requirements, follows the recommendations, and selects from the choices allowed by this document.*

A2. REQUIREMENTS - Keys, Entropy, and Random Bits

*Symmetric keys, asymmetric keys, entropy, and random bits in the key table **MUST***

a. *Follow recommendations in NIST Special Publication 800-57 Recommendation for Key Management*

b. *Follow recommendations in NIST Special Publication 800-90A, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

c. *Follow recommendations in NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*

d. *Follow recommendations in NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*

e. *Follow the guidance in the Commercial National Security Algorithm (CNSA) Suite regarding quantum resistant algorithms and key sizes.*

f. *Provide a statement of minimum key strength and cryptoperiods of the values in the key table.*

A3. REQUIREMENTS - Initial Provisioning Environment, Operations, and Equipment

a. *Initial provisioning operations **MUST** be carried out in a trusted facility, in which a secure channel between the Provisioner and the device is guaranteed.*

b. *The Provisioner **MUST** report which of the following provisioning methods is used:*
{attester device self-generates both UDS and DevIK$_{pr}$,
Provisioner injects UDS and device self-generates DevIK$_{pr}$, or
Provisioner injects both UDS and DevIK$_{pr}$}.

c. *Cryptographic algorithms and deterministic random bit generators **MUST** be validated under the NIST Cryptographic Algorithm Validation Program (CAVP)*

d. *Cryptographic modules, if used,* ***SHOULD*** *be validated at overall level 2 or higher under [FIPS 140-2 SECURITY REQUIREMENTS FOR CRYPTOGRAPHIC MODULES]* *or [Security Requirements for Cryptographic Modules, FIPS 140-3]*

e. *Entropy, random bits, symmetric keys, and private asymmetric keys* ***MUST*** *be generated within the attester device itself, in a hardware security module, or locally, in a device with the following properties:*

   i. *Follows recommendations in NIST Special Publication 800-90A Rev 1, Recommendation for Random Number Generation Using Deterministic Random Bit Generators*

   ii. *Follows recommendations in NIST Special Publication 800-90B, Recommendation for the Entropy Sources Used for Random Bit Generation*

   iii. *Follows recommendations in NIST Special Publication 800-133 Recommendation for Cryptographic Key Generation*

   iv. *Complies with Annex C: Approved Random Number Generators for FIPS PUB 140-2, Security Requirements for Cryptographic Modules*

   v. *Follows the guidance in the Commercial National Security Algorithm Suite regarding quantum resistant algorithms and key sizes.*

f. *Each attester device has the following properties:*

   i. *Each attester device* ***MUST*** *have a unique, and immutable device ID key pair.*

   ii. *Each attester device* ***MAY*** *be provisioned with a hash of the first mutable firmware.*

   iii. *Each attester device* ***MUST*** *prevent exfiltration of device secrets through defined interfaces.*

   iv. *The Provisioner* ***MUST*** *generate a certificate signed by its pCA private key, which links the unique device identity with its Provisioner.*

   v. *Each attester device* ***MUST*** *generate a certificate signed by the device ID private key, which links the attestation public key with the device identity.*

A4. REQUIREMENTS - Device Ownership Provisioning

a. *Each attester device **MUST** be provisioned with a Device Update public key, which is used to verify updates to the device's critical configuration.*

b. *The Device Update public key, once provisioned on the attester device, **MUST** only be modified through an authenticated ownership transfer.*

c. *Each attester device **MAY** be provisioned with a Device Owner's certificate over the device identity public key.*

d. *Each attester device **MAY** be provisioned with a hash of the device firmware.*

e. *Each attester device **MAY** be provisioned with a hash of the Device Owner's certificate over the device identity public key.*

A5. REQUIREMENTS - Discovery and Interrogation

a. *Attester devices **MUST** be capable of communicating their authentication and attestation capabilities to the platform.*

b. *Attester devices **SHOULD** be capable of communicating their capabilities to the platform within 15 seconds of being provided with power, even if their data plane bus (e.g. PCIe) is held in reset by the platform.*

c. *Platforms **MUST** be capable of interrogating potential attester devices and recording their authentication and attestation capabilities.*

d. *Platforms **MUST** be capable of interrogating attester devices that do not communicate their capabilities before being taken out of reset, e.g., by interrogating them later in the boot cycle or by having them pre-configured as such, in the platform reference manifest.*

e. *Platforms **MAY** use the message formats for GET_CAPABILITIES and NEGOTIATE_ALGORITHMS as described in* Security Protocol and Data Model (SPDM) Specification or *Device Capabilities as described in Project Cerberus Firmware Challenge Specification . Where necessary, bridge components may be responsible for translating from the native bus protocol into the GET_CAPABILITIES/ NEGOTIATE_ALGORITHMS message formats.*

A6. REQUIREMENTS - Authentication, Attestation, and Enrollment

a. *Attester devices **MUST** provide certificate digests and certificates when requested by the platform.*

b. *Attester devices **MUST** build and sign responses to challenges from the platform. Although this step is optional in the SPDM specification, it is required here.*

c. *Platform verifiers **MUST** request certificate digests and certificates from attester devices.*

d. *Platform verifiers **MUST** verify ASN.1 DER encoded X.509v3 certificates and certificate chains from attester devices back to each device's Provisioner root public key.*

e. *Platform verifiers **MUST** present challenges to attester devices and verify the content in the responses.*

f. *Platform verifiers **MUST** verify attester device signatures on the challenge responses.*

g. *Platform verifiers **MAY** build a platform inventory containing authentication status, firmware signing keys, firmware measurements, and Device Owners of attester devices.*

h. *Platform verifiers **MAY** accept a predefined manifest (an expected inventory of devices) or build it dynamically.*

i. *Platform verifiers **MAY** compare the platform inventory to a platform manifest containing expected devices and their configurations.*

A7. REQUIREMENTS - SPDM Standards Support

a. *Attester devices that support the SPDM standard **MUST** conform to the set of capabilities as defined in the table "Required Capabilities for SPDM."*

b. *Attester devices that support the SPDM standard **SHOULD** support the set of algorithms as defined in the table "Recommended Algorithms for SPDM".*

c. *Attester devices that support the SPDM standard **MUST** support SPDM version 1.0 or higher.*

d. *Attester devices that support the SPDM standard **SHOULD** support the current version.*

A8. REQUIREMENTS - What to measure and what not to measure

a. *The measurements **MUST** include everything that affects the security of the attester device, such as executable code, headers, security state and configuration data.*

b. *The measurements **MUST** exclude information that will make the measurements brittle, such as run-time configuration data that does not impact the security of the device, and information which is expected to be updated frequently on the device.*

A9. REQUIREMENTS - Security-relevant configuration data

a. *In the event that configuration data for the device may lead to compromise of the security of the device (such as fuses or straps that enable JTAG or other test interfaces), this class of configuration data **MUST** be discoverable from the device and/or cause the measurements of the device to be distinguishable from production measurements. The mechanism for detecting/providing this information to the attester device **MUST** be enforced through pure hardware means.*

b. *On attester device reset, the measurement registers **MUST** be cleared (reset to 0s) and a measurement indicating an attester device reset event **MUST** be extended to the measurement register.*

c. *Resetting the attester measurements independently from the system that it measures **MUST NOT** be possible through a purely software mechanism (avoid separate reset and power signals).*

d. *The measurement storage **MUST** be integrity protected to prevent malicious or inadvertent modification, but it is not confidential.*

e. *Measurement storage **MAY** include a structured log of measurements.*

A10. REQUIREMENTS - When to Measure

a. *Before executing or transferring control to mutable code, immutable code **MUST** record measurements of the mutable code and of relevant configuration settings.*

b. *If an attester device's state changes (e.g., reset or chain of trust has become invalidated), the attester **MUST** generate a signal or counter to notify the platform of the change.*

c. *If the attester device's state changes, the platform and device **MUST** repeat the complete attestation protocol.*

d. *If a device allows for the runtime modification of any state that can affect the security properties of the device, such modifications* ***SHOULD*** *be reflected in the measurements and measurement logs of the device.*

e. *It* ***SHOULD NOT*** *be possible for any debug mode to reset the measurement values, or make arbitrary changes to them.*

f. *It is* ***RECOMMENDED*** *that attester devices be able to respond to an attestation request at any time during the device's normal runtime operation.*