REFERENCE

TCG

TCG Guidance for Securing Network Equipment

Version 1.0
Revision 26b
July 3, 2017
Committee Draft

Contact: admin@trustedcomputinggroup.org

- ***Work in Progress:***
  This document is an intermediate draft for comment only and is subject to change without notice. Readers should not design products based on this document

TCG Public Review

Disclaimers, Notices, and License Terms

Acknowledgements

Laffey, Mr. Tom (Co-chair)	Hewlett Packard Enterprise
Eckel, Mr. Michael (Co-chair)	Huawei Technologies Co., Ltd.
Fedorkow, Mr. Guy (Editor)	Juniper Networks, Inc.
Kuntze, Dr. Nicolai	Huawei Technologies Co., Ltd.
Latze, Ms. Carolin	Carolin Latze
Bell, Mr. Bob	Cisco Systems
Pritikin, Mr. Max	Cisco Systems
Sulzen, Mr. Bill	Cisco Systems
Birkholz, Mr. Henk	Fraunhofer Institute for Secure Information Technology (SIT)
Fuchs, Mr. Andreas	Fraunhofer Institute for Secure Information Technology (SIT)
Kotani, Dr. Seigo	Fujitsu Limited
McDonald, Mr. Ira	High North Inc
Rudolph, Dr. Carsten	Huawei Technologies Co., Ltd.
Zhang, Ms. Yi	Huawei Technologies Co., Ltd.
Hanna, Mr. Steve	Infineon Technologies North America Corp.
Rudy, Mr. Gregory	INTEGRITY Security Services, Inc.
Watsen, Mr. Kent	Juniper Networks, Inc.
Shah, Mr. Atul	Microsoft
Thungathurthi, Mr. Bhavani	Microsoft
Proudler, Mr. Graeme	Graeme Proudler
Lorenzin, Ms. Lisa	Pulse Secure, LLC
Schmidt, Mr. Charles	The MITRE Corporation
Fitzgerald-McKay, Ms. Jessica	United States Government

Table of Contents

1. Scope, Audience and Purpose 7
1.1 Scope 7
1.2 Audience and Purpose 7
2. Preface 8
2.1 Network Equipment Reference Model 8
2.2 Internal Structure of Network Equipment 9
2.3 Stakeholders 10
2.4 Key Differences between Network Equipment and PC Applications 11
2.4.1 Long Life Cycle 12
2.4.2 Privacy and Networking Equipment 12
3. Use Cases 14
3.1 Device Identity 14
3.1.1 OEM Device Identity and Counterfeit Protection 14
3.1.2 Identity for Network Access 15
3.2 Secure Zero Touch Provisioning 15
3.3 Securing Secrets 16
3.4 Protection of Configuration Data 17
3.5 Remote Device Management 17
3.6 Software Inventory 18
3.7 Attestation of Boot Integrity for Network Devices (“Health Check”) 18
3.8 Inventory of Composite Devices 18
3.9 Integrity-Protected Logs 19
3.10 Entropy Generation 19
3.11 Deprovisioning 20
4. Building Blocks 21
4.1 Chain of Trust 21
4.2 Secure Boot vs Measured Boot 22
4.3 802.1AR Device Identification 23
5. Implementation Guidance 25
5.1 Device Identity 25
5.1.1 OEM Device Identity and Counterfeit Protection 26
5.1.1.1 Configuring Initial Identity Credentials 27
5.1.1.2 Signing and Encryption Key Types 27
5.1.1.3 Identity for Attestation 28
5.1.1.4 Initial Identity with TPM1.2 29

5.1.1.5 Key Types in TPM2.0 ... 29
5.1.1.6 Initial Identity with TPM2.0 ... 31
5.1.1.7 Loss of IDevID Credentials ... 33
5.1.1.8 Compromise of Keys ... 34
5.1.2 Local Device Identity ... 34
5.1.2.1 How to Install a Local Device ID ... 34
5.1.2.1.1 Conveying TPM Certify Information ... 35
5.1.2.2 Attestation with Local Identity ... 36
5.1.3 Identity for Network Authentication ... 36
5.1.3.1 Proving a Link to the TPM ... 36
5.1.3.2 Why Not Just Use EK and Platform Cert for Identity? ... 37
5.2 Secure Zero Touch Provisioning ... 37
5.3 Securing Secrets ... 38
5.4 Protection of Configuration Data ... 39
5.5 Remote Device Management ... 40
5.6 Software Inventory ... 40
5.7 Attestation of Integrity for Network Devices (“Health Check”) ... 42
5.7.1 Linux Integrity Measurement Architecture (IMA) ... 43
5.7.1.1 IMA Appraisal ... 46
5.7.2 PCR Definitions ... 47
5.8 Composite Networking Devices ... 47
5.9 Integrity-Protected Logs ... 49
5.9.1 Functional Requirements ... 49
5.9.2 Integrity-Protected Log Implementation ... 50
5.9.2.1 Timekeeping in TPM 1.2 ... 50
5.9.2.2 Timekeeping in TPM 2.0 ... 51
5.9.2.3 Assembling the Log ... 51
5.10 Entropy Generation ... 52
5.11 Deprovisioning ... 53
6. References ... 54
7. Glossary ... 56
8. Appendices ... 57
8.1 Use-Cases For Further Study ... 57
8.1.1 Secure Identity in Virtual Machines ... 57
8.1.2 Distributed Composite Devices ... 57
8.2 Standardization Work For Further Study ... 57
8.2.1 Known Good Hashes ... 57

8.2.2 Composite Systems 57
8.2.3 Reference Manifests 58
9. Revision History 59

1. Scope, Audience and Purpose

This document, the *TCG Guidance for Securing Network Equipment*, is part of the Trusted Computing Group's collection of Reference Documents, written by the Network Equipment Sub Group (NE-SG) to assist architects and designers determine where and how TCG technology, including the Trusted Platform Module, can best be used to secure network equipment such as routers, switches, and firewalls.

1.1 Scope

The *TCG Guidance for Securing Network Equipment* provides recommendations and detailed advice on how TCG standards should be used to secure network equipment such as routers, switches, and firewalls. Physical network functions are considered in this document; virtualized network functions are not considered in this version.

While the TPM's resistance to physical attack can help protect device identity, and can effectively prevent the leakage of credentials and other secrets, defense against physical attack is generally beyond the scope of this document.

TCG technology users are in the midst of a transition between TPM1.2 and TPM2.0; this document is constructed to cover both TPM1.2 and TPM2.0 applications, and to highlight differences when they're important.

There is a short glossary of terms in Section 7.

1.2 Audience and Purpose

Preserving the integrity and security of network equipment is essential to maintaining customer privacy and network reliability. In the face of increasingly sophisticated attacks on network equipment, Trusted Computing solutions are desperately needed in this area. Yet little information is available on how Trusted Computing should be used to secure network equipment and thus the networks that depend on this equipment. This reference document provides guidance for using TCG technologies to improve the security of network equipment and networks..

2. Preface

2.1 Network Equipment Reference Model

Figure 1 shows a simplified reference model often used for Network Equipment. Customer Premise Equipment (CPE) or Residential Gateways are often positioned between administrative domains, and may require special attention for management of access and identity.

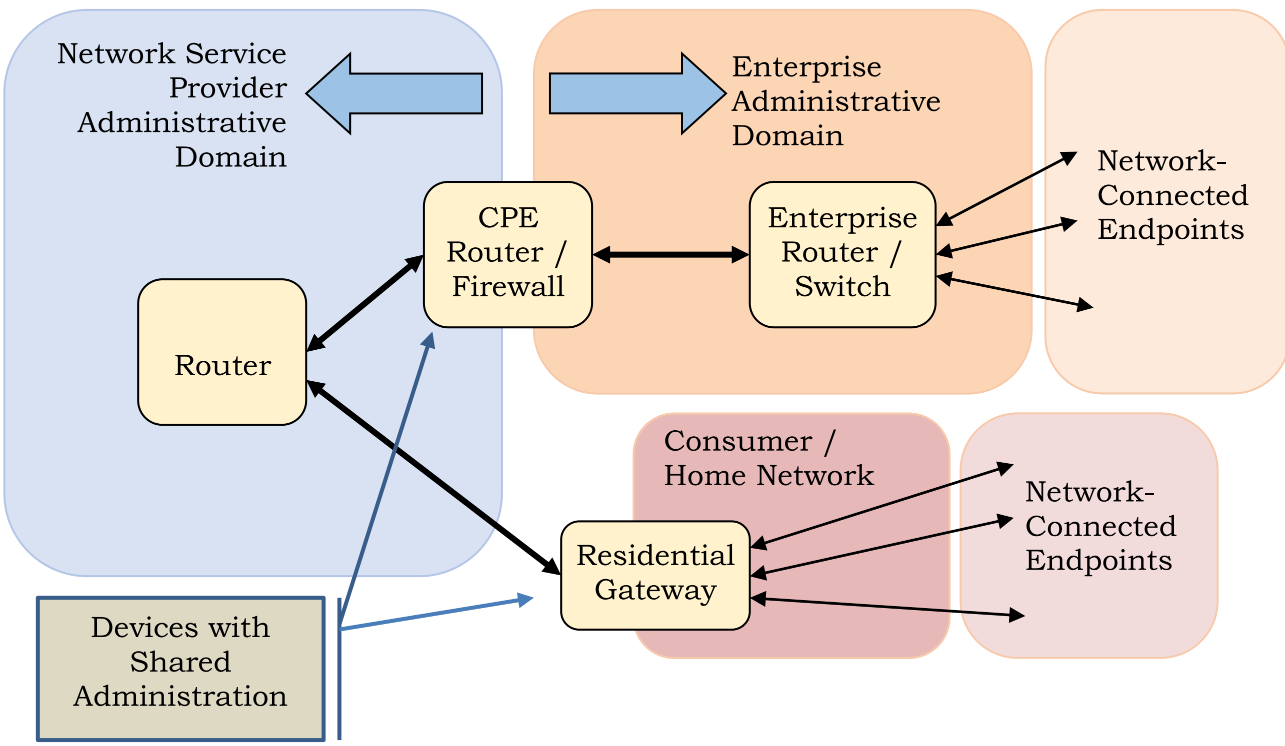


Figure 1: Simplified Network Reference Model

It should be noted that there are often many more administrative sub-domains within each side of the domains depicted in Figure 1. Large enterprises in particular often have IT departments that act as internal service providers, offering connectivity to departments and groups, each of which may have considerable autonomy.

In this context, there may be several points in an overall network where different administrative domains must share control of a single device; for example, an Enterprise Managed Gateway CPE device might have interfaces configured by a Service Provider to ensure contracted class of service, while the Enterprise would control interfaces that connect to its own equipment.

Further, it should be noted that the administrator of a given device normally will not have direct physical connectivity to the device, and will require authenticated remote access to carry out management functions.

2.2 Internal Structure of Network Equipment

Networking equipment like routers and switches often comprises several major units:

- The Forwarding Plane inspects each packet and performs the networking function required (e.g., L3 routing, L2 switching, Firewall security inspection, etc). The forwarding plane may be implemented in hardware, microcode, or software on a conventional processor, but in any case, the forwarding plane is typically subservient to a Control Plane, and would rely on the Control Plane for security.
- The Control Plane is the part of the device that initializes, monitors, configures and diagnoses the Forwarding Plane. The Control Plane is typically implemented as software running on an operating system, hosted on a conventional processor. The Control Plane typically manages every aspect of Forwarding Plane operation, including assurance that whatever code runs in the forwarding plane is authenticated.
- Some devices implement a separate Management Plane that provides an external interface used to configure and manage the device. A management plane would also typically execute in a conventional OS and processor environment.

Although essentially all networking devices have one or more control-plane processors that configure and run the device, the term *Network Device* can cover very small to very large devices:

- Many small networking devices are packaged as non-modular units with no field-replaceable units (FRUs). A small switch, wireless access point or customer premise device might all be non-modular. In this case, the identity (e.g. serial number) of the chassis encompasses all components of the device.
- Larger networking devices may have pluggable I/O or feature cards. The base chassis may have a unique identity, but the pluggable modules usually have their own serial numbers or identity. I/O or feature cards may also have distributed control plane processors that should be secured.
- Many large networking devices have field-replaceable control plane processors, often with several independent control plane processors for redundancy. In this case, each control processor may have its own serial number and identity. In some cases, to meet redundancy goals, the chassis itself may be completely passive, with no active components to prove identity.

  Some architectures use a distributed control plane with many independent, cooperating processors; in that case, the identity and integrity of each processor may need to be determined separately.

Figure 2 shows an abstract view of *composite* networking devices, that is, those comprising several independent units. While there are many ways to construct a composite system, ranging from single-chassis devices with plug-in cards through to self-organizing networks of elements, a key factor in this context is that there are usually a small number of elements that serve as the external management interface for the system, combined with a larger number of internal units, which mostly are not directly reachable by the external management system. Each of the elements in Figure 2 may have its own Trusted Computing Base (TCB).

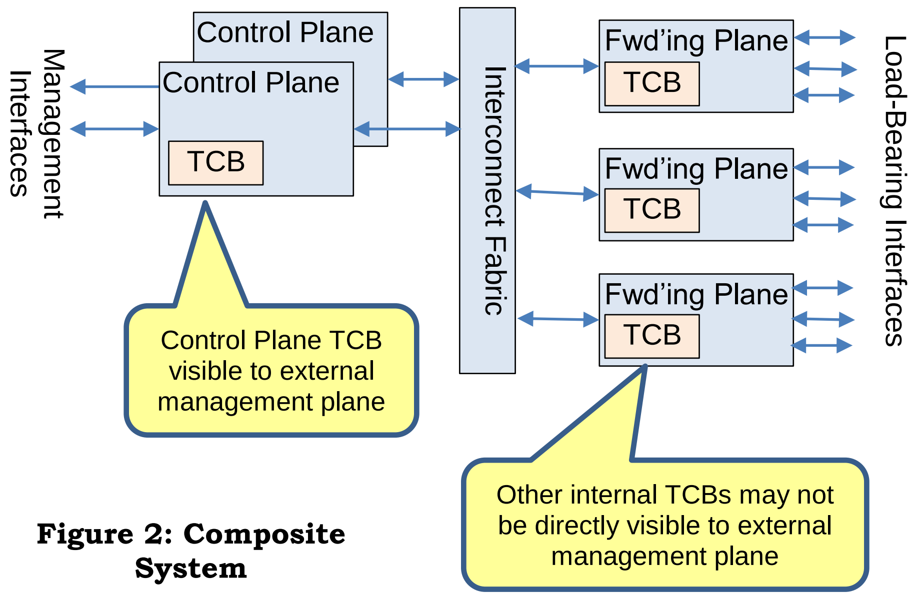


Figure 2: Composite System

Recommendations in this document relate to security, identity and integrity of the Control Plane; if there is Networking Equipment where the forwarding plane can act independently of the Control Plane, the forwarding plane may need to be protected as if it were a control plane element.

2.3 Stakeholders

There are many stakeholders involved in the use and secure deployment of Network Equipment:

- TPM Manufacturer
  - The TPM manufacturer creates and initializes the TPM silicon. TPM manufacturers usually equip their devices with a unique Endorsement Key (EK) and EK Certificate.
- Device Manufacturer
  - Device Manufacturers develop and sell Networking Equipment, often referred to also as OEMs (Original Equipment Manufacturers).
  - The Device Manufacturer is usually responsible for supplying the equipment, a software operating environment.
  - Device Manufacturers are often called upon to diagnose and remediate complex networking problems encountered by Service Provider and Enterprise customers.
- Network Service Provider
  - Service Providers generally provide public network service through defined gateways.

  - Service Providers may also offer private networking services to enterprises, often delivered over infrastructure also used for public interconnect.
  - Security and reliability may be offered as a value added service.
- Enterprise
  - An Enterprise usually has network equipment for its own use, i.e., not to sell as a service.
  - An Enterprise may use specialized monitoring and configuration services provided by a remote Maintenance Provider as a way to outsource some of its own IT support.
- Consumer
  - Network Equipment may be purchased by individual consumers or small businesses (e.g., WiFi gateway routers).
  - Consumer devices may need to take care of themselves, with little expectation for active administration.

In this document, the “Administrator” is considered to be the Service Provider, Enterprise or Consumer that has primary authority over the device, and can authorize or delegate access for other stakeholders, including Secondary Administrators. The Administrator must be able to erase private information without making the device unusable for a subsequent owner. (See Sections 3.11 and 5.11.)

2.4 Key Differences between Network Equipment and PC Applications

Networking Equipment almost always contains a general-purpose computing environment to configure and manage the device. But there are distinct differences between Networking Equipment and the common PC client and server applications:

- While Network Equipment may be highly modular, it is usually shipped as a closed embedded system, integrating hardware and software.[1],[2]
- The chain of security typically does not stop when the OS boots; what matters is security of the networking function that’s provided by the unit as a whole.
- Network Equipment typically must boot and operate without manual intervention (e.g., no Owner Password can be expected at boot time).
- While Network Equipment has an important role in protecting user privacy, the equipment itself typically should not have an ability to hide or mask its own identity. See Section 2.4.2 for more on Privacy.

[1] Even though many Device Manufacturers support an SDK environment, and some are moving more aggressively to an open-source model, the manufacturer is still usually responsible for infrastructure security of the device.

[2] For a counter-example, see Open Compute Project Networking, http://www.opencompute.org/projects/networking/

2.4.1 Long Life Cycle

Network Equipment often has a long life cycle, and must stay operational in the network for many years. Certificates with 10-year expiration are often too short for many Network Equipment applications, as equipment may stay in operation for upwards of 20 years. Device Manufacturers should ensure that any immutable certificates have lifetimes long enough to extend through the expected life of the platform.

IEEE802.1AR Secure Device Identity [1] requires the following related to the "notAfter" field in a DevID certificate in Section 7.2.7.2

> *Devices possessing an IDevID are expected to operate indefinitely into the future and should use the value 99991231235959Z. Solutions verifying a DevID are expected to accept this value indefinitely. Any other value in a DevID notAfter field are expected be treated as specified in RFC 5280.*

Network Equipment manufacturers should consider using long or indefinite lifetimes for any immutable certificates stored in devices (including the EK Certificate).

Along with certificates that don't expire prematurely, manufacturers may want to consider mechanisms to accommodate cryptographic agility. Key lengths and algorithms do evolve over time, and mechanisms to update algorithms and replace dated certificates in fielded gear may extend useful equipment life.[3],[4]

Conversely, Administrators may want to configure relatively short lifetimes for locally-generated certificates such as a Local Device ID (LDevID, see Section 5.1.2), or other vendor-specific applications, as a mechanism to ensure that devices don't remain forgotten in their networks past the time when they should be replaced, or past the time when cryptographic parameters such as key length may have become outdated.

2.4.2 Privacy and Networking Equipment

Networking Equipment such as routers, switches and firewalls has a key role to play in guarding the privacy of individuals using the network:

- Packets passing through the device must not be sent to unauthorized destinations.

  Some examples include:
  - Routers often act as Policy Enforcement Points, where individual subscribers may be checked for authorization to access a network. Subscriber login information must not be released to unauthorized parties.

---

[3] See National Institute of Standards and Technology, *Report on Post-Quantum Cryptography, NISTIR 8105* [15]

[4] While Algorithm Agility is not part of TPM1.2, TPM2.0 can accommodate multiple cryptographic algorithms.

  - Networking Equipment is often called upon to block access to protected resources, or from unauthorized users.
- Routing information, such as the identity of a router’s peers, must not be leaked to unauthorized neighbors.
- If configured, encryption and decryption of traffic must be carried out reliably, while protecting keys and credentials.

Functions that protect privacy are implemented as part of each layer of hardware and software that makes up the networking device.

In light of these requirements for protecting the privacy of users of the network, the Network Equipment must identify itself, and its boot configuration and measured device state (for example, PCR values), to the Equipment’s Administrator, so there’s no uncertainty as to what function each device and configuration is configured to carry out[5]. This allows the administrator to ensure that the network provides individual and peer privacy guarantees.

[5] For example, serious privacy violations could occur if embedded software was modified to snoop traffic, or if unauthorized configuration commands were added to encrypt traffic with an incorrect key or to replicate traffic to an unencrypted tunnel.

3. Use Cases

3.1 Device Identity

Providing reliable remotely-verifiable device identity for each piece of network equipment is a prerequisite for most use cases related to securing network equipment. This section contains a few use cases that demonstrate the value of device identity for network equipment and the variety of ways that device identity can be implemented and used. Within this section, the term “device” is used to mean a piece of network equipment.

Before getting into use cases, it’s helpful to distinguish two kinds of device identity: Manufacturer identity and Owner identity. The Manufacturer identity for a particular device is established, configured and managed by the Device Manufacturer, although it can also be used (e.g., verified) by the device owner. The Owner identity for a device is established by the device Administrator and is generally used only by the Administrator. Manufacturer identity is generally[6] unique across all products from that manufacturer (e.g., a model number and serial number) while Owner identity may be unique only within the Administrator’s domain (e.g., an asset number). These kinds of device identity are defined as Initial Device ID and Local Device ID, respectively, in IEEE 802.1AR (See Section 4.1).

In any case, device identity is generally implemented using a private key kept secret by the device and a certificate associated with that key, issued to the device by a Certification Authority (CA).

3.1.1 OEM Device Identity and Counterfeit Protection

Both network equipment owners and Device Manufacturers (OEM’s) need to verify the authenticity of network equipment, determining whether it is “counterfeit” (made by an unauthorized party or in an unauthorized manner) or “authentic” (made by authorized parties in an authorized manner).

Business Drivers: To achieve reliable network operation for end-user customers, Network Equipment Owners need to be assured that they are using authentic devices from known manufacturers. At the same time, Device Manufacturers want to reduce lost sales, protect their brand, prevent returns and warranty repairs of counterfeit equipment. Distributors want to detect counterfeit devices before they are resold. Auditors want to verify that network operators are meeting compliance regulations.

How It Works: During manufacturing, a signed manufacturer identity is instantiated in the device. Manufacturing processes and key types must be selected to ensure that the identity cannot be copied to, or instantiated in, another device. Later, the authenticity of the device can be verified by checking the validity of this identity.

This verification can be done by many parties, as detailed above: the network equipment owner, the manufacturer, the distributor, auditors, network operators, etc. The verification may be done locally (by connecting a cable directly to the device) or remotely (across a network).

[6] The Manufacturer and Administrator are responsible for managing uniqueness in their respective name spaces, but typically these identifiers will not be ‘cryptographically strong’; the public key associated with the device identity can also be used as a unique identifier where that’s a factor.

If a counterfeit device is detected, various actions can be taken: blocking the device from the network, alerting an operator, etc.

3.1.2 Identity for Network Access

Business Drivers: Network operators often wish to ensure that only authorized equipment is connected to their networks. This is true for telecommunications companies but also for cloud and data center operators, hospitals, enterprises, manufacturing facilities, and other environments where the network needs to be tightly controlled.

How It Works: Certificates proving device identity may be used to authorize network access. When a device connects to the network, its device identity is verified and checked against a list of authorized devices. If the device is on the list, it's permitted to connect to the network. Either identity configured by the Device Manufacturer, or identity configured by the Administrator may be used for this purpose.

In the case of CPE equipment, or other situations where administration is shared between two domains, it may be necessary for a Primary Administrator to grant rights to a Secondary Administrator to add their own identity.

The Identity for Network Access use-case requires the network equipment to authenticate itself to the upstream network's authentication service, and may require that the upstream network prove its identity to the particular networking device[7].

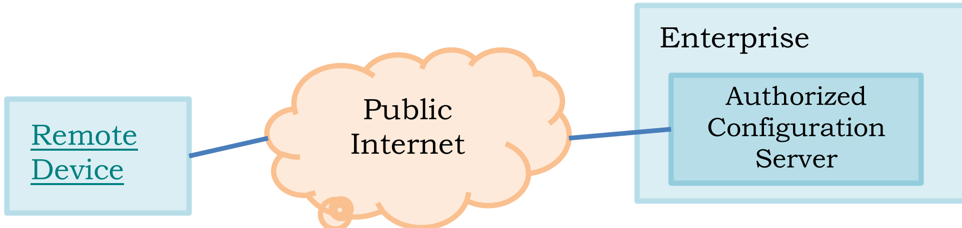
3.2 Secure Zero Touch Provisioning

Business Drivers: There are many cases where a networking device may be shipped with no unique configuration, but must be configured before it can be used within a network.

In these cases, Zero Touch Provisioning (ZTP) may be desirable, requiring a mechanism where a newly-installed device initiates communication through a public network, but only enough to obtain the configuration information that would specify policy for operational use. As an example, downloaded configuration and keys might enable access to a corporate VPN, or might authorize access to restricted content.

Because the result of loading configuration on to the remote device could be access to an enterprise's private network, careful design is required to ensure that only authorized devices are able to download a configuration.

[7] For example, in the process of establishing an RFC 5246 Transport Layer Security (TLS) session, downstream and upstream devices might exchange DevID certs to prove identity.



Part of this use-case may require proof that the device is actually in the hands of an authorized user before its configured, perhaps through the use of out-of-band authorization information.

How It Works: ZTP can be made more secure with several additions:

- The networking device should use a Manufacturer-installed credential to authenticate itself to the ZTP server.
- The networking device can check a credential supplied by the ZTP server to ensure that it's authorized to serve configuration.
- The ZTP server can use Remote Attestation (Sections 3.7 and 5.7**Error! Reference source not found.**) to verify that the device is running authorized software before completing the configuration.

3.3 Securing Secrets

Business Drivers: Network equipment often contains sensitive information such as traffic logs or cryptographic keys (e.g., shared secrets, passwords, VPN keys, SSL keys, and stored data encryption keys). Disclosure of these secrets could result in disclosure of confidential network traffic and privacy-sensitive information or even enable malicious tampering with the network. Network operators (especially Service Providers and Enterprises) must protect these secrets against disclosure to keep their networks secure and reliable and also to meet regulatory or customer requirements for confidentiality and privacy.

How It Works: Because network equipment is often located in insecure locations and always subject to network attacks and exploitation of software vulnerabilities, protection must be provided against physical attacks, network attacks, and software compromise. Many techniques can be used to protect keys or shared secrets, depending on the operations that must be performed with them, especially throughput requirements.

There are two sub-cases that can be considered, depending on scaling expectations:

- Keys may be stored in the TPM and used in the TPM, providing substantial protection against loss, but signing throughput limited by the type of key used, and the TPM's low-power crypto engine.
- Alternately, the TPM may protect a key that's used externally by a high-throughput crypto engine. In this case, the key is well protected by the TPM "at rest", but must be released by the TPM before it can be used, increasing the risk that it can be snooped.

Using a Virtual Private Network gateway device as an example:

- A remote customer-premise device may log in to a corporate VPN across the internet. In this case, a small number of authentication operations may suffice, and keys may be kept and authenticated entirely within the TPM.
- But the centralized VPN gateway may be performance-optimized for high-throughput authentication from thousands of remote devices; in this case, keys may be held in encrypted storage, but released for use by platform software.

This use-case covers the way these secrets can be managed.

3.4 Protection of Configuration Data

Business Drivers: Network Equipment usually requires configuration, often involving many parameters stored in a variety of files. The equipment Owner may wish to retain control over changes to configuration files on the equipment, with the goal of ensuring that unauthorized configuration changes don't compromise their network.

How It Works: In this case, the Owner may wish to sign and encrypt configuration files so each file can only be used on the intended device, and the device will only accept configuration from the authorized Owner. Signing or Attestation may also be used to ensure configuration integrity, i.e. to assure the equipment Owner that the configuration has not been modified.

In some cases, it may be desirable to prevent rollback to previous versions of configuration files.

Some Network Equipment could also benefit from full-disk encryption to protect data at rest, to discourage cloning. The document *TCG Storage Opal Integration Guidelines* [17] gives an overview of the Trusted Computing Group suite of specifications for self-encrypting storage devices.

3.5 Remote Device Management

Business Drivers: Network equipment Owners with a large number of devices often want to manage those devices remotely, including the ability to monitor devices and reconfigure them dynamically. Remote management and reconfiguration is especially important in modern, flexible computing environments that implement Software-Defined Networking (SDN) or Network Function Virtualization (NFV).

Remote Device Management may also include the need for trustworthy decommissioning or refurbishment (See Section 3.11).

How It Works: To perform remote device management securely, the device and management system must generally establish a secure communications channel with mutual authentication and confidentiality and integrity protection. This secure channel is used by the management system to monitor device status and send reconfiguration commands.

3.6 Software Inventory

Business Drivers: Most Network devices rely on complex embedded software to enable basic features as well as to enforce security policies. This software is often updated on devices already in the field, using releases and patches usually supplied by the device manufacturer.

Network Administrators are left with the task of keeping track of which devices have been updated to what revision level, sometimes tracking many independent components on a single complex device. Automating the process could save money and improve security by making it harder to overlook out-of-date releases.

How It Works: Mechanisms can be implemented to allow the Administrator to query devices to find which revision level of which components are installed on each network device in their network. The ISO specification *ISO/IEC 19770-2 Information Technology - Software Asset Management - Part 2: Software Identification Tag* [6], (also known as "SWID Tags"), coupled with the TCG protocol suite *Platform Trust Service (*PTS*),* can collect verified software image information from remote devices.

3.7 Attestation of Boot Integrity for Network Devices ("Health Check")

Business Drivers: One extension to remote device management enabled by TCG technology is an ability to monitor the authenticity of software versions and configurations running on each device. This allows owners and auditors to detect deviation from approved software and firmware versions and configurations, potentially identifying infected devices.

How It Works: Networking equipment requires some extension from conventional PC-based Attestation due to more varied platform architecture, including redundancy, in-service software updates, and hot-swappable field replaceable components. In addition, the focus on overall system functionality calls for an extension of the conventional attestation plan to encompass more objects in PCRs, for example, hashes of application executables, OS daemons and configuration files.[8]

Attestation may be performed by network management systems. Networking Equipment is often highly interconnected, so it's also possible that Attestation could be performed by neighboring devices.

Attestation depends on a process sometimes called Measured Boot; See Section 4.1 for a comparison of boot security procedures.

3.8 Inventory of Composite Devices

Business Drivers: Many network devices are composed of one or more control or management units plus optional components like line processing units, feature-processing units and other kinds of field replaceable units (FRUs). The definition of a composite system and its internal organization, whether it's homogenous or heterogeneous, hierarchical or flat,

[8] Following a 'chain of trust' model, conventional TCG practice would be to use TPM measurements to prove that a reliable policy engine is running in the device, and then use that engine to verify application resources. However, for embedded devices like Network Equipment, the system designer has quite a bit of latitude in determining which software components are part of the trusted computing base.

is up to the system designer, but the behavior of the network device is based upon the composite behavior of individual components. The security posture of the network device is therefore only accurately represented by a composite measure that includes the posture of sub-components. Any change to one of these sub-components or even an undetected exchange endangers the security of the user's data on the network. A secure inventory allows determination of the components used and their state.

Many network devices allow FRUs to be replaced without triggering a complete system restart (often called 'hot swap'); for these devices, system-level reboots may be very rare, and the system's security posture must be re-evaluated every time an individual unit is inserted or removed from the system.

How It Works: Mechanisms may be implemented to provide proof of identity and configuration for each modular component, summarized and reported by the management process for the device.

3.9 Integrity-Protected Logs

Business Drivers: Various processes in the day-to-day operation of network equipment are based on information gathered from the system status of servers, routers and sensors. SACM, SIEM or even legal interception are based on state information of various components. Tampering with this information, mostly instantiated as log files, can impact the security protection (e.g. by suppressing intrusion-detection (IDS) data) or impact the integrity of information delivered by the legal interception interface.

Integrity-protected log files can be used by the management or external entities by providing information on the authenticity and integrity of the file. An integrity-protected log file can show on which device it was created, plus evidence that it has not been subject to unauthorized modification, and other details dependent on the use case.

This use case does not address the specifics of a SACM, SIEM or legal interception system and does not specify any part of such a processing entity, but shows the applicability of Trusted Computing for the protection of log files and other data used in later processes.

How It Works: A device's internal log file or system wide log server receives events and preserves these events for subsequent analysis. The data stored in the log file can be protected using TPM-based signatures expressing the origin of the data, along with the time of creation and the state of the device that created the data. All this can later be used in subsequent analysis to determine the trustworthiness of the data collected.

Confidentiality of log information is not covered in this use-case.

3.10 Entropy Generation

Business Drivers: Businesses and governments have a need to ensure that traffic that passes across untrusted network connections is reliably protected by strong cryptographic keys.

How It Works: Many networking protocols such as SSH and IPsec have a need for cryptographic-quality random numbers to avoid the generation of predictable session keys.

In addition, the TCP stack for Network Equipment should use good-quality randomness for the TCP window starting point as well as in the selection of ephemeral ports. These help to mitigate SYN and RST attacks against the device.

Network Equipment also often requires good-quality random numbers to generate Cryptographic Security Parameters (CSPs) such as RSA key-pairs.

Entropy sources used in Network Equipment may have to comply with standards such as *NIST Special Publication 800-90 A/B/C* [14] or BSI AIS-31, *A Proposal for Functionality Classes for Random Number Generators* [15] [17].

The TPM can act as a source for cryptographic-quality entropy.

3.11 Deprovisioning

Business Drivers: Networking Devices often contain information that's considered sensitive by the Administrator, such as customer configurations or routing policies. Once the device is taken out of service, this information must be reliably destroyed.

How It Works: Confidential information can include TPM keys themselves, or information encrypted by TPM keys. The TPM mechanisms for deleting keys can ensure that the confidential information will become inaccessible.

4. Building Blocks

4.1 Chain of Trust

Security of Network Equipment Devices is critically dependent on the integrity of the software running on the device. Software integrity is usually assured with a chain-of-trust model, with each stage during startup checking the next stage before execution, as shown in Figure 3. While that model is extensible to as many stages as needed, there's clearly a special case for the first step in the process, the one before which nothing exists to do any checking. This stage is referred to as the Root of Trust, and code that runs in the Root of Trust must be trusted implicitly.

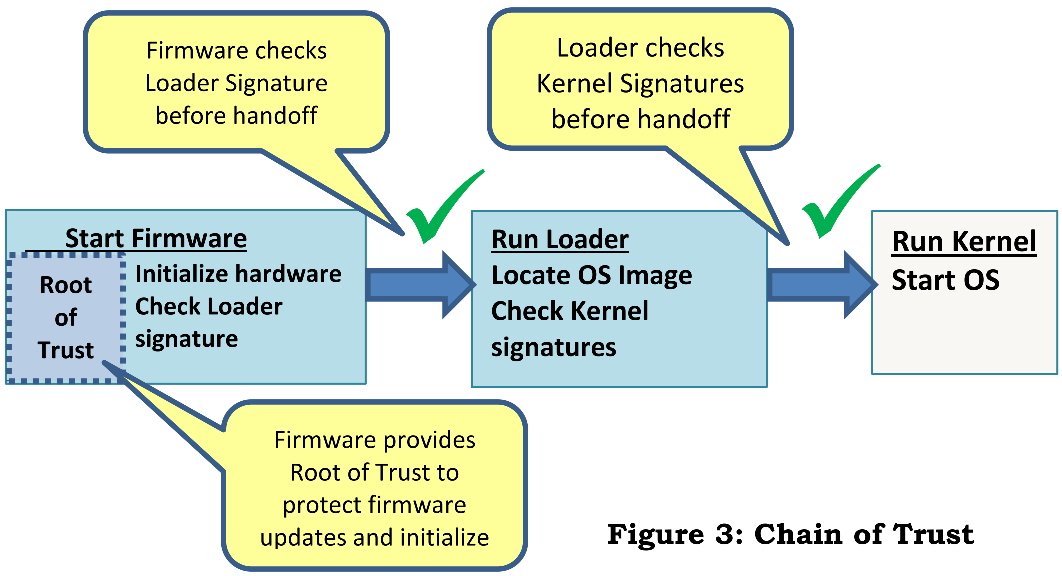


Figure 3: Chain of Trust

Underpinning essentially all of the security mechanisms outlined in this document is a reliable Root of Trust, often implemented in code called a Basic Input Output System (BIOS), and usually stored in non-volatile memory so it's available to deliver the very first instructions to the processor for execution after reset. The root of trust has several functions, among which are initializing the TPM and doing the first measurement, and ensuring that firmware is not changed without authorization,

Although some systems are able to use an immutable root of trust, which cannot be modified or hacked, most current devices need a capability to upgrade the software that forms the root of trust, and must rely on some hardware assistance to lock the flash[9], as well as code in the root of trust to protect the device from unauthorized updates or code implants.

Techniques for assuring protection of BIOS or other boot firmware can be found in NIST SP 800-147, BIOS Protection Guidelines.

[9] Unlike ordinary software, the root of trust must be protected against attacks from the privileged run-time operating system. This usually results in some kind of hardware feature in the processor to lock flash against OS software.

Although TCG technology is not required to secure the root of trust, NIST SP 800-155 does specify the use of a TPM as part of a comprehensive software integrity strategy, linking to the Health Check features covered in Section 5.7 of this document.

4.2 Secure Boot vs Measured Boot

There are generally two approaches to ensuring that code running on a Networking Device is authorized, and has not been subject to illicit modification.

- **Secure Boot** (also known as Verified Boot) is a process by which each stage of the boot process checks a cryptographic signature on the next stage before executing it. In a typical system, there might be a BIOS that does a check of an OS Loader's signature, which in turn might do a check of the OS kernel before launching it.
- **Measured Boot** (also known as Attested or Authenticated Boot) is a process by which each stage of the boot "measures" or computes and stores the hash of components in the next stage prior to launching it.

Secure and Measured Boot can be extended to run-time software through mechanisms such as the Linux Integrity Measurement Architecture [19]. See Section 5.7.1 of this document for more on the use of IMA.

Secure Boot is a subset of an overall *Secure Computing* architecture, which dictates a platform's behavior, as opposed to *Trusted Computing*, which enables the Administrator to deduce a platform's behavior. While implementations of secure boot are relatively common, complete Secure Computing or Trusted Computing environments are still difficult to design and implement.

Secure Boot and Measured Boot have some commonality and some differences:

- While Measured Boot depends critically on the TPM to safely store *measurements*[10] (See Section 5.7), Secure Boot can be implemented without the use of TPM technology.
- Both techniques rely on a Root of Trust in the initial BIOS, although Secure boot could be used to reliably instantiate the Roots of Trust required by Trusted Computing.
- While measured boot won't stop a corrupted system from starting, it is able to verify the state of a broader range of components in the boot path (e.g. configuration files that might impact system security, in addition to executables).
- Secure Boot and Measured Boot have different risk profiles. Secure boot can convert an innocent mistake such as a key mismatch into a network outage by refusing to start, while Measured boot can allow a corrupted network component to continue running, risking further damage.
- "Sealing", or encrypting data such that it can only be decrypted when a platform is in a known measured state, is a cost-effective way to protect data-at-rest, even with secure computing.

[10] A *measurement* is the cryptographic hash of an object such as an executable file

- A single system could reasonably execute both Secure Boot and Measured Boot at the same time. Measured boot and attestation enhance confidence in a computer's behavior even when it's not practical to cryptographically sign each and every object accessed as the system starts up.[11]

4.3 802.1AR Device Identification

[This Informative summary of 802.1AR appeared first in TCG Infrastructure WG *TPM Keys for Platform Identity for TPM 1.2*, May 2015 [4]; it's been edited in this section for wording and for applicability to TPM 2.0]

The IEEE 802.1 working group defined the IEEE 802.1AR Standard for Local and Metropolitan Area Networks Secure Device Identity and it was published on 22 December 2009 [1]. This standard defines a per-device unique identity installed at manufacturing time and used subsequently in device-to-device authentication exchanges. This standards-based device identity can also be coupled in multiple ways with user identification.

The 802.1AR standard defines a secure device identifier (DevID) as "a cryptographic identity that is bound to a device and used to assert the device's identity". It further specifies:

- the DevID is an X.509 credential
- globally unique per-device identifiers and the cryptographic binding of a device to its identifiers
- the relationship between an initial identity installed during manufacturing and subsequent locally-significant identities installed by the Administrator
- interfaces and methods for use of DevIDs with existing, and new, provisioning and authentication protocols

The initially installed identity is defined as an IDevID ("I" for initial) and subsequently locally defined identities are LDevIDs ("L" for local). The IDevID is installed at manufacturing time. Since the "TPM Keys for Platform Identity" specification allows for an on-premise creation of the device identity, it will use the term DevID, whether the DevID is an IDevID or an LDevID. An IDevID will be created at manufacturing time and provides proof that this device has been manufactured by a certain manufacturer. An LDevID is created on the Administrator's premises and provides proof that this device is owned by a certain enterprise (or individual).

There is another dimension to the IDevID/ LDevID discussion with TPMs prior to and including version 1.2. In order to create an IDevID, the Device Manufacturer needs to take ownership of the TPM. However, commands that change ownership make all previously created keys unusable. Section 5.1.1.4 of this document outlines precautions that must be taken by the Device Manufacturer to ensure sure that the IDevID cannot easily be made unusable by the TPM. More on this topic can be found in *TPM Keys for Platform Identity for TPM 1.2* [4], Section 6.

[11] E.g., a Device Manufacturer can't sign a customer-mutable configuration file, but the Administrator may know exactly what should be in the file on a fielded machine. Measurement can report a fingerprint of the contents without requiring a signature on each file.

The 802.1AR specification DevID module (covering IDevID and LDevID) contains a service interface, storage holding a DevID secret and credential, secure hashing functions, a random number generator (RNG) and asymmetric cryptography functions. These functions exist in the TPM and calls by middleware (such as the TPM Software Stack (TSS) or the TPM API) can be used to meet interface requirements outlined in 802.1AR.

The IEEE Standard 802.1AR-2009 [1] can be used together with TPM-based keys and certificates. The TPM acts partly as the Secure Device Identifier Module (DevID Module) which the standard defines as “a logical security component that will secure, store and operate on one or more DevID Secret(s) [(private key)] and associated DevID credentials”. The document *TPM Keys for Platform Identity for TPM 1.2* [4] addresses usage of the TPM after provisioning has occurred and leverages TCG-specified X.509 certificate extensions to prove TPM residency of the keys, using the Subject Key Attestation Evidence (SKAE) extension [10]. The document *TPM Keys for Platform DevID for TPM 2* defines how TPM 2.0 key attributes and residency are proven for use as Device Identifiers.

5. Implementation Guidance

This section outlines approaches for the use of TCG technology to satisfy the Use Cases outlined above. These sections are not normative, but rather suggest ways that existing TCG technology can be put to use.

5.1 Device Identity

Device Identity requirements can be satisfied by the use of IEEE 802.1AR Device ID certificates (See Section 4.1 above). Appendix B of *IEEE Std 802.1AR-2009* [1] specifies implementation of DevIDs using a TPM, including a MIB and some required behavior (enable/disable for example) that is not specifically addressed in TCG documents.

Figure 4 shows the relationship of credentials installed in the TPM during its lifetime:

- The EK and its Cert are added by the TPM manufacturer to show that the TPM is authentic.
- Initial Device ID (IDevID) credentials are installed by the device manufacturer to identify the device's manufacturer and serial number.
- The device manufacturer may also install an Initial Attestation key (IAK) and certificate to allow attestation without requiring the administrator set up a CA (See Section 5.1.1.3).
- The device manufacturer can also provide a mechanism to allow an Administrator to configure Local Device ID's (LDevIDs) and their own Attestation keys (LAK) (Section 5.1.2).

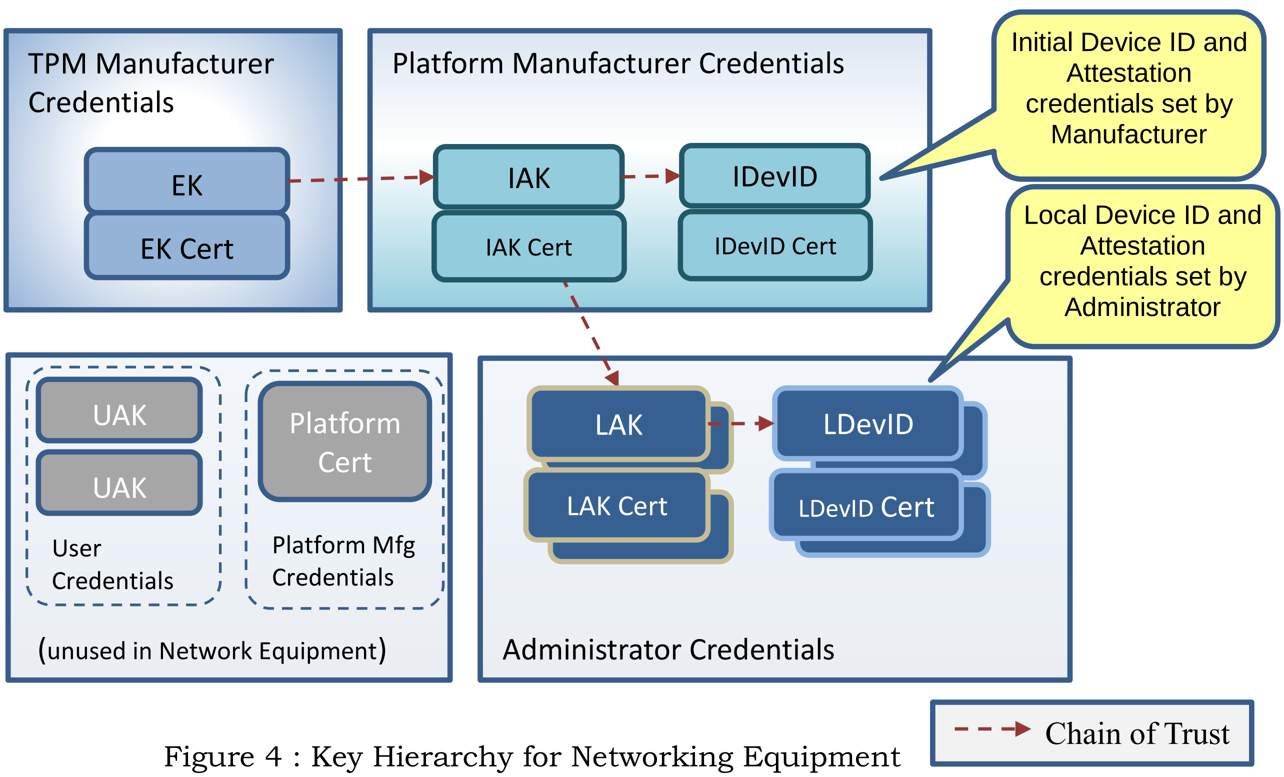


Figure 4 : Key Hierarchy for Networking Equipment

There are several TCG documents that are relevant to the use of TPMs to implement 802.1AR certificates:

- TCG Infrastructure WG: *TPM Keys for Platform Identity for TPM 1.2.* [2] This document gives details useful for DevIDs in TPM1.2, including a number of use-cases for DevID keys, and techniques for provisioning these keys.
- TCG Infrastructure WG: *TPM Keys for Platform DevID for TPM 2* [5] gives analogous details for TPM 2.0.
- TCG Infrastructure WG: *TCG TPM v2.0 Provisioning Guidance* [8]. This document gives guidance on TPM 2.0 hierarchies, etc, setting out expectations for TPM manufacturer, platform manufacturer and Administrator in the general case of TPMs in many kinds of equipment.

To support Device Identity requirements, the Device Manufacturer must configure the TPM:

- Either the TPM Manufacturer must ensure that the TPM is configured with an EK and EK certificate when the chip is manufactured, or the Device Manufacturer must generate the EK and EK Cert.
- Although this happens automatically with TPM2, a TPM1.2 must be configured as "always on", i.e., Enabled and Activated, when the Device Manufacturer's product is built.
- The Device Manufacturer must ensure that the TPM is configured with an IDevID key and an IDevID Certificate before the Device leaves the manufacturing facility.

5.1.1 OEM Device Identity and Counterfeit Protection

Device Identity and Counterfeit protection can be provided by an IEEE 802.1AR Initial Device Identifier (IDevID) certificate, signed by the Device Manufacturer and installed in the device while it's still under the manufacturer's control. This certificate provides cryptographic evidence that the specific physical device was manufactured by the specified manufacturer.

IEEE 802.1AR certificates include the device serial number and traceability to the manufacturer's CA. The combination of manufacturer and Device serial numbers are expected to be unique.

An individual unit of Network Equipment (e.g. a large router) may comprise many field replaceable modules (FRUs). In this case, the serial number in the certificate should reflect the serial number of the FRU which houses the TPM. For example, in equipment with redundant control plane processors, the IDevID should contain the serial number of the control plane processor module itself, and will change if that processor unit is replaced for any reason in a fielded installation.

5.1.1.1 Configuring Initial Identity Credentials

To support Device Identity requirements, the Device Manufacturer must configure a certificate and keys in the TPM. There are differences between TPM1.2 and TPM2.0 in this process; this section outlines common elements, and the subsequent sections give the family-specific differences.

In all cases, the Device Manufacturer must ensure that the TPM is configured with an EK, EK Certificate, an IDevID key and certificate, before the network equipment leaves the manufacturing facility. The manufacturer may also want to configure an Initial Attestation Key (IAK) and cert (see Section 5.1.1.3).

To do this, the Device Manufacturer will create the IDevID key-pair in the TPM, issue a signing request containing the device serial number and other information to its own CA, and sign the certificate. Per 802.1AR, the signature should use SHA256. (See Section 5.1.1.4 for how to do this with TPM1.2). DevID keys must be created with modes that prevent duplication of the key from one TPM to another, to ensure that the identity of a Device can't be moved from one instance to another. (See Sections 5.1.1.4 and 5.1.1.6.)

TCG EK Credential Profile For TPM Family 2.0 [9] offers the following suggestion for EK credential lifetime, and the same advice should be considered for IDevID, or other immutable certificates. To quote Section 2.2.3:

> ***2.2.3*** *EK Credential Lifetime*
>
> *An EK Credential contains fields that express the validity period of the credential. The validity period is at the discretion of the manufacturer. The credential is not expected to expire during the normal life expectancy of the platform in which it resides. The lifetime can vary widely between different types of platforms (e.g. while a typical validity period for a PC Client platform is 5-10 years, non-user device TPMs are expected to operate indefinitely into the future in which case the value 99991231235959Z should be used as expiration date). The credential lifetime can also depend on the lifetime of the TPM device and the algorithm type of the Endorsement Key. The time frame during which the security strength of the EK is acceptable SHOULD be taken into account by the manufacturer when determining the credential lifetime (e.g. see SP800-57[10]).*

It is not believed that there is any Personally Identifiable Information in Network Equipment PCRs, but that determination must be made by the Device manufacturer. (See Section 2.4.1)

5.1.1.2 Signing and Encryption Key Types

There are two types of keys that could be used for DevIDs in a TPM, as defined in *TPM Keys for Platform Identity for TPM 1.2* [2] Section 2.4, and *TPM Keys for Platform DevID for TPM2.0*. [5] Section 2.3:

- Signing keys, which are restricted to signing, and allow no other operations
- General Purpose or Combined Keys (aka Legacy keys) which can be used for signing as well as encryption and decryption

The key type used for DevIDs should be selected based on the application and security needs for the equipment in question.

General-Purpose keys are required in applications that implement different TLS variants that may use RSA or Diffie-Helman to convey the shared session secret during setup. Keys capable of decryption are also required for features such as Protection of Configuration Data, described in Section 5.4 in this document. Although 802.1AR does not impose a requirement, Section 7.2.13 of *IEEE 802.1AR-2009 - Secure Device Identity* [1] does suggest *not* imposing restrictions on DevID key usage.

Applications that are restricted to variants of TLS using Diffie-Hellman or Elliptic Curve Diffie-Hellman (DH or ECDH), and are particularly concerned about attacks on keys in the TPM, may want to use Signing keys for DevID, and provision separate Encryption DevID key-pairs for other functions that need encryption with identity keys.

The trade-off is summarized in Table 1:

General Purpose Keys	Separate Signing and Encryption Keys
Fewer Keys to Keep Track Of	Better Security
Use General-Purpose (signing and encryption) keys for DevID	Use separate DevID keys for signing and encryption • DevID-Signing for non-RSA variants of TLS • DevID-Encryption for RSA variants of TLS and other applications such as Protection of Configuration Data
Requires less TPM space	More state to be stored in, or restored to, the TPM

Table 1: DevID Key Types

NIST Special Publication 800-57 Part 1, Rev 4 Section 5.2 gives additional guidance on use of keys.[12]

5.1.1.3 Identity for Attestation

TPM mechanisms are arranged so that in privacy-sensitive situations, Remote Attestation can be done either without knowing the exact identity of the target system, or by trusting the correlation between platform and attestation identity to a special Privacy CA (also known as Attestation CA; see Chapter 7 of *IWG Reference Architecture for Interoperability (Part 1) [20]*)

For Networking Equipment, where clear identity of the devices in question is usually a critical requirement, a simpler approach can be used. In this case, the device manufacturer should provision an *Initial Attestation Key* (IAK) and certificate that parallels the IDevID, with the same device ID information as the IDevID certificate (i.e., the same Subject Name and Subject Alt Name, even though the key pairs are different). This allows a quote from the device, signed

[12] See http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf

by the IAK, to be linked directly to the device that provided it, by examining the corresponding IAK certificate.[13]

Inclusion of an IAK does not preclude a mechanism whereby an Administrator can define Local Attestation Keys (LAK's) if desired (Section 5.1.2.2).

5.1.1.4 Initial Identity with TPM1.2

TPM1.2 has relatively limited mechanisms for managing Ownership:

- For TPM1.2, the Device Manufacturer must configure TPM Ownership to make an SRK, so that an IDevID may be created as a child of the SRK.[14]
- To ensure that the IDevID credentials are not rendered invalid by deletion of the SRK, Ownership of the TPM must be locked.

In order to prevent the SRK from being deleted, rendering any keys stored under it useless, TPM Ownership can be locked with the non-volatile TPM_DisableOwnerClear() ordinal. If the TPM supports physical presence, the volatile TPM_DisableForceClear() ordinal may be used to block ForceClear on each startup.

The detailed structure of the 802.1AR certificate for TPM1.2 is given in *TPM Keys for Platform Identity for TPM1.2* [2]

The 802.1AR document requires the use of SHA256 for signature generation, although the TPM1.2 can only perform SHA1 signatures using its built-in functions. However, the TPM 1.2 allows the host to compute the hash, prepend the appropriate ASN.1 encoding, and to submit this to the TPM for SHA256 RSA signature generation using Distinguished Encoding Rules (DER).[15]

TCG TPM1.2 documents do not define handles that should be used to identify DevIDs, so these must be assigned by the Device Manufacturer.

Identity keys in TPM1.2 should be created as *non-migratable*, or a Certified Migration Key (CMK), to eliminate the chance that a device identity could be improperly copied from one device to another.[16]

5.1.1.5 Key Types in TPM2.0

In TPM 1.2, identity keys must be descendants of the Storage Root Key, but in TPM 2.0, there are a number of options.

---

[13] As long as the IDevID and IAK are signed by equally-reputable CAs (or more simply, the same CA) and contain the same Subject Name and Subject Alt Name, this approach eliminates the Asokan Attack, https://tools.ietf.org/html/rfc6813

[14] For TPM1.2, this is an unavoidable overlap of administrative domains; normally, Ownership would be taken by the Administrator, not the Manufacturer, but that's incompatible with a manufacturer-supplied IDevID

[15] See TPM1.2 Part I Design Principles, Section 31.2.2 TPM_SS_RSASSAPKCS1v15_DER

[16] In TPM1.2, it's possible to use a Certified Migration Key (CMK) with suitable Migration Selection Authority (MSA) and Migration Authority (MA) to allow a key to be moved under controlled conditions. The point, of course, is to make sure that an unauthorized copy of a DevID cannot be made.

- Secondary, Internally-Generated keys are like DevID keys in TPM 1.2; the key pair is generated in the TPM, and can be configured so the private key is never released. A Secondary key is always the child of a parent key in the hierarchy.
- Primary keys are unique to TPM 2.0. These keys are also generated in the TPM, but they are generated using a deterministic Key Derivation Function that starts with a generic Template, and uses a primary seed in the TPM to generate the key pair. Primary keys can be deleted and regenerated from the template as long as the seed value is unchanged.
- TPM 2.0 also includes a specialized mechanism where an external entity can generate the key pair and wrap it so it can only be imported to a specific TPM. This can be used to circumvent the non-deterministic time taken to generate an RSA key inside the TPM, or it can be used to meet unique security requirements where an external key generator must be used. Imported key pairs cannot be Primary keys, and they cannot be marked as non-duplicable (i.e., they can't be marked as fixedParent). (Listed in the table below as "Secondary Imported")

Table 2 shows the different properties of the key generation approaches.

<table>
<tr><th>Primary Key</th><th>Secondary
Internally-Generated</th><th>Secondary
Imported</th></tr>
<tr><td>Must be created from a template; cannot be Imported</td><td>Generated internally</td><td>Key generated externally, wrapped with the parent key</td></tr>
<tr><td colspan="2">RSA key generation is non-deterministic, and could take a minute or more of elapsed time</td><td>Importing keys is deterministic, as they're wrapped with a symmetric key</td></tr>
<tr><td colspan="2">DevIDs should be Created as non-Duplicable (FixedTPM, FixedParent)</td><td>Imported keys can't be marked as Fixed, and require a Policy to block export or migration.</td></tr>
<tr><td>Key probably needs to be made Persistent to avoid long regeneration each power-on</td><td>Key can be saved, wrapped by its parent, then Loaded on power-up or use.</td><td>Importing the key produces a version wrapped by the parent, which can be quickly reloaded when needed.</td></tr>
<tr><td colspan="2">Certs can only be generated with a multi-step Online process using Endorsement Credential and TPM2_ActivateCredential()[17]</td><td>Keys and certs can be generated externally and installed off-line across an air-gap if necessary.</td></tr>
</table>

[17] Since the key-pair is generated in the TPM, it's not known in advance, implying the need for a multi-step procedure to generate the key and then incorporate the public half in a matching certificate. Conversely, the public key of an imported key pair is known in advance, so a matching certificate can be delivered along with the wrapped key for import.

Key can be re-generated from a generic template in case of TPM2_Clear()	Keys are lost in case of a TPM2_Clear()	Wrapped key can be re-imported after a TPM2_Clear() (Assuming the parent is restored, and the Wrapped key is not lost)

Table 2: Key Generation Options for TPM 2.0

For the remainder of this section, we assume the following:

- In most applications, IDevID keys would be Primary keys installed by the device manufacturer. The Clear operation may be used to remove user identity and keys from a device, but the IDevID keys can be regenerated from a generic template after a TPM2_Clear() operation.
- LDevID keys are Secondary, internally-generated keys, and are intended to be deleted by a TPM2_Clear() operation.
- Manufacturers can also support Imported DevID keys if they have high-throughput manufacturing processes where the indeterminate wait for RSA key generation would be unacceptable, or for specialized customers that want to manage their own key generation.

In TPM 1.2, special Restricted Attestation keys must be used to sign internally-generated structures such as quotes, to prevent an external actor from spoofing a quote. The TPM 2.0 retains this capability with Restricted Signing Keys, which can sign an internally-generated quote or external data, as long as the external data doesn't contain the fixed pattern that matches an internally-generated structure. (See Trusted Platform Module Library Family 2.0 Part 1 [2], Section 25.1.2)

Thus, a Restricted Signing Key could be used for both Identity and Attestation, with the microscopic possibility that the TPM will refuse to sign a nonce that happens to look like a quote.

For the remainder of this section, we assume that Device IDs (IDevID, LDevID) are separate keys from Attestation keys (IAK, LAK).

5.1.1.6 Initial Identity with TPM2.0

To support Device Identity requirements, the Device Manufacturer must configure keys in the TPM as described in the *TPM v2.0 Provisioning Guidance* [8] document.

IDevID keys must be signing keys. If they are also Restricted, then they can serve as attestation keys. However due to complexities of using restricted keys with off-the-shelf crypto software stacks such as OpenSSL[18], platform vendors may opt to provision IDevID keys as

[18] Signing external data with a Restricted key requires a Ticket, and there's some concern that managing tickets would require an involved OpenSSL rewrite. Not impossible, but real work.

unrestricted signing keys. Doing so would necessitate provisioning a separate restricted signing key pair for attestation purposes (identified as an IAK in Figure 3).

Except for the case of externally-generated keys noted in Section 5.1.1.5, IDevID keys should be Primary keys, stored in the Endorsement Hierarchy[19].

The Device Manufacturer must ensure that the TPM is configured with an EK, EK Certificate, an IDevID Key and IDevID certificate before the network equipment leaves the manufacturing facility.

- The Device Manufacturer must provision the Endorsement Hierarchy authorization.
- The IDevID certificate (if it is stored in the TPM) should be write-locked to prevent it from being deleted.
- The IDevID Template does not need to be stored in the TPM, as it is generic and not specific to a unique device.

In order to prevent the EPS (Endorsement Primary Seed) from being changed, rendering any keys stored under it useless, the TPM2_ChangeEPS() command must be blocked. This command is optional[20], and ideally it would *not* be implemented by the TPM vendor. If it is implemented and enabled, then the Device Manufacturer must implement strict control, provisioning the Platform Authentication to a random value or un-fulfillable policy upon every platform reset.

TPM2_Clear() resets the endorsement hierarchy authorization, clears the endorsement hierarchy (including IDevID keypair if stored here), and removes NV-defined indexes which are not protected by the TPMA_NV_PLATFORMCREATE flag. If the IDevID certificate is stored in the TPM, it should be placed in the Platform NV hierarchy, by setting its TPMA_NV_PLATFORMCREATE flag, so it's not lost by a TPM2_Clear().[21]

To prevent permanent loss of the IDevID keypair, the Device Manufacturer could either block execution of this command or recreate the IDevID keypair from a generic template.

The exact format of an IDevID certificate for TPM2.0 is defined in Section 7 of TCG specification *TPM Keys for Platform DevID for TPM2.0*. [5]

---

[19] TPM2.0 embodies three independent key hierarchies: a "platform hierarchy" for platform protection, independent of specific users, an "endorsement hierarchy" for exposing information related to identity and a "storage hierarchy" for cryptographic usage related to specific users or applications.
IDevID keys are analogous to the EK, but without the privacy restrictions placed on the EK; as such, they should be stored using the same rules as the EK (i.e., in Endorsement). The Platform hierarchy could be used, but generally objects in the platform hierarchy should not be required after the platform OS has launched.

[20] The 'optional choice' is exercised by the TPM Manufacturer; if the TPM Manufacturer implements the ordinal, the Device Manufacturer needs to figure out how to block it.

[21] There is no Endorsement hierarchy for NV RAM, so the only two choices for the IDevID cert is Platform (which does survive TPM2_Clear) or Storage (which does not). The EK Cert is stored in Platform; the IDevID cert should be too.

TCG TPM v2.0 Provisioning Guidance [8], Section 7.8 gives defined handles for IDevID credentials in TPM 2.0:

Description	Reserved Handles
IDevID Key	0x81020000
IDevID Certificate	0x01C90000

It must not be possible to move identity keys from one TPM to another. In TPM 2.0, the mechanism to prevent keys from being duplicated depends on the method of creation:

- If keys are generated within the TPM, FixedParent and FixedTPM must be set (see TPM 2.0 *Trusted Platform Module Library, Part 2: Structures*, Section 8.3 [3]).
- Imported keys can't be marked as Fixed, so they must be accompanied by a Policy that prevents export.

5.1.1.7 Loss of IDevID Credentials

OEMs should recognize that loss of IDevID credentials in a fielded device could be hard to remedy.

Although the IDevID key-pair must be protected by the TPM to ensure that the private key remains secret, it's up to the Device Manufacturer to store the IDevID certificate (and possibly a wrapped IDevID key) somewhere secure, but not necessarily secret, in the platform. OEM's *could* lock the IDevID certificate and wrapped key into NVRAM in a TPM, although there's no requirement to do so, and NV space is often at a premium.

For devices that don't have space in the TPM NVRAM to store Initial identity credentials, there are a couple of alternatives:

- If the manufacturer doesn't track device id information in its own database, loss of these credentials would require return to factory where Initial identity keys could be regenerated.
- If the manufacturer does keep track of EK's, it would be possible to re-create an IDevID in a fielded device using enrollment based on TPM Certify mechanisms, even if Ownership is lost. (See *TPM Identity Keys for TPM 1.2* [4] or *TPM Platform DevID for TPM 2.0* [5].)
- For TPM 1.2, as long as Ownership is maintained, the wrapped IDevID key and matching certificate could be replaced from a manufacturer's database. See Sections 5.1.1.4, 5.1.1.6 on locking Ownership.

As noted in Section 2.4.1, manufacturers may want to support upgrade of cryptographic algorithms in devices with long lives. Cryptographic algorithms cannot be changed in TPM1.2, but TPM2.0 offers a mechanism to add algorithms, although doing so in fielded devices might require replacement of any 'permanent' identification.

5.1.1.8 Compromise of Keys

Private keys associated with the EK and IDevID are immutable; once manufactured, these keys are not easily changed in a fielded device. Furthermore, compromise is unlikely, as they are protected by the TPM.

But manufacturers do need to take care to protect intermediate keys stored in CAs in their manufacturing facility, since exposure of a signing key could compromise many devices, with little recourse to re-secure them short of return-to-factory.

5.1.2 Local Device Identity

IEEE 802.1AR also includes provision for Local DevID's, customer-specific credentials which can be installed by the Administrator of the Network Equipment, during or after installation. LDevID certificates will be signed by the Administrator's CA.

While the LDevID can contain anything the Administrator wants, they're probably best thought of as enterprise "asset tags", providing cryptographic evidence that the Device in question is one that was legitimately configured for use within the Administrator's enterprise. Multiple LDevID tags might be applied to a single device, depending on its organizational role(s).

The Device Manufacturer must provide a mechanism to remove Administrator-installed LDevID keys without deleting IDevID.

Modular Network Equipment may contain multiple field replaceable units; as with IDevID certificates, the basic LDevID can identify the FRU, but not necessarily the overall Device.

5.1.2.1 How to Install a Local Device ID

An Administrator may want to install LDevID credentials remotely, when the device is located off-site (as would be the case for Service-Provider's Customer-Premise Equipment, see Figure 1). As such, care must be taken that the LDevID is being installed in the right device.

The following steps can be used to install an LDevID into a Device that's already equipped with an IDevID and IAK. These steps assume Enrolment over Secure Transport (EST; see IETF RFC 7030).

- Log into the Device using a TLS session that can't be easily hijacked, authenticated by the IDevID. Use that TLS session for all of the following steps:
    - Use of the IDevID certificate for TLS authentication ensures that the connection terminates on the right device by showing the serial number assigned by the device manufacturer.
    - The Device's identity as expressed by the manufacturer's serial number can be trusted if:
        - The certificate can be traced to the Device Manufacturer.
        - The Device can respond to a challenge, proving it holds the IDevID private key.

  - Check the Manufacturer’s serial number against a bill-of-sale to ensure it’s the right device.
  - These steps provide proof that there’s a secure connection to the exact device into which the LDevID should be placed.
- The device should generate the LDevID key pair and a CSR containing the public key and the asset information selected by the Administrator.
  - The CSR may also contain TPM_CERTIFY_INFO signed by the IAK key to prove that the LDevID key is in the same TPM as the IDevID[22],[23].
- The CSR should be returned to the Enterprise CA.
  - Since an IDevID key must be configured to prevent duplication from one TPM to another, the enterprise CA can use the TPM_CERTIFY_INFO to convince itself that the IDevID and LDevID are in the same TPM, and that TPM is in the intended Device.
- The Enterprise CA can then create the LDevID certificate with OIDs showing TPM residency, sign it and send it back to the device.

This procedure is defined in more detail in *TPM Keys for Platform DevID for TPM2.0.* [5].

An analogous procedure can be used for cases where an externally-generated key pair is desired for LDevID (i.e., LDevID doesn’t have to be generated in the TPM for those cases where an externally-generated key would be more appropriate).[24] This case is also covered in *TPM Keys for Platform DevID for TPM2.0.* [5], Section 6.2.

Chapter 21 of the book *A Practical Guide to TPM 2.0* (W. Arthur & D. Challener) [13] also outlines techniques that can be used to remotely provisioning DevID credentials.

5.1.2.1.1 Conveying TPM Certify Information

While there could be a number of ways to get the TPM_CERTIFY_INFO from the TPM to the CA responsible for signing a DevID, this document suggests uniform practice between TPM 1.2 and TPM 2.0.

---

[22] This proof is simple if the IAK contains the same device serial number as the IDevID, and they’re signed by the same issuing authority. Other mechanisms must be used if the IAK is created independently. See Section 5.1.1.3.

[23] Use of TPM Certify yields a trustworthy mechanism to get the LDevID into the same TPM as the corresponding IDevID, even if the control plane software on the device can’t be trusted. If TPM Certify is not used, there’s a possibility of variations of the Asokan attack, where the Device might have a valid IDevID, but do something that misuses the LDevID, e.g. not putting it a TPM at all, or passing it off to some other device. TPM Certify must be verified by a CA that puts the TCG OID indicating TPM key residency into the resulting certificate.
Note that TPM Certify info in TPM2.0 (TPMS_CERTIFY_INFO) is quite different from TPM Certify info for TPM1.2 (TPM_CERTIFY_INFO and TPM_CERTIFY_INFO2), but they both satisfy the requirement of proving co-residency.

[24] We note that externally-generated keys don’t necessarily have to risk exposure. An HSM (or another TPM) could generate an LDevID key pair, and wrap it with a key known only to the TPM destined to receive the LDevID.

- For TPM 1.2, TPM_CERTIFY_INFO is added to the Certificate Signing Request as an X.509 extension. This extension is defined in Subject Key Attestation Evidence (SKAE) [10].
- The TPM2_Certify structures are more complex, but a new CSR extension is [will be] defined in *TPM Keys for DevID for TPM 2.0* [5].

5.1.2.2 Attestation with Local Identity

Administrators who wish to use LDevIDs for device identity, using their own credentials to identify devices, may want to provision "local" Attestation Keys (LAK's) as well, allowing the Attestation system to reliably identify the device within the Administrator's context. Use of a Local AK in addition to the LDevID can prove that Attestation has not been compromised by an Asokan attack (RFC 6813).

5.1.3 Identity for Network Authentication

When presenting identity for network access, the Device may use its IDevID or one of its configured LDevID certificates, at the discretion of the Administrator. The appropriate certificate should be chosen by methods controlled by the Device's operating system software.

IETF's Transport Layer Security (RFC-5246) Section 7.4.4 defines the mechanism by which a server can negotiate trust anchors, by, for example, specifying a list of the distinguished names of acceptable certificate authorities.

5.1.3.1 Proving a Link to the TPM

DevIDs may be used to authenticate access to various kinds of network services, in which case the receiving service must decide how much trust can be placed in the credential.

DevID private keys that can be shown to be held by a TPM can presumably be judged more likely to be credible, and less likely to have been compromised.

There are two techniques to demonstrate to a network service that a particular DevID key is stored in a TPM:

- Sign the DevID cert with a CA that will only be used to sign certificates that are known by the Administrator to originate in a TPM. This may include a specific TCG OID corresponding to the Certificate Practices Statement (CPS), referenced below in this section.
- Extend the service's login authentication to parse an additional x.509 extension field called a Subject Key Attestation Extension (SKAE, TPM1.2 only).

The use of SKAE to send proof to a CA that a DevID key is stored in a TPM1.2 device is described in *TCG Infrastructure Workgroup: Subject Key Attestation Evidence Extension [10].*

The CA can certify that the key is in a TPM using the mechanism described in *TCG Infrastructure WG TPM Keys for Platform Identity for TPM 1.2 [2]*

To quote Section 3.1.1.1:

> *In order to assert that the CA verified TPM residency for the key used in a DevID certificate, a certificate policy OID unique to the TCG (and for this purpose) MUST be included in the certificate policy extension of the DevID certificate. The CA MUST perform verification of the AIK signature over the TPM_CERTIFY_INFO or TPM_CERTIFY_INFO2 prior to including the certificate policy TCG OID.*
>
> *The TCG Policy OID is: 2.23.133.6.1.2*

The analogous method to certify that keys are resident in a TPM is described in the TCG document *TPM Keys for Platform DevID for TPM 2 [5].*

5.1.3.2 Why Not Just Use EK and Platform Cert for Identity?

While the Endorsement Key (EK) must be present in every TPM, and must be unique, that doesn't mean it's a good candidate for device identity.

- The conventional use of the EK credential is to prove the provenance of the TPM itself, by linking to a chain of certificates rooted at the TPM manufacturer's CA.
- But beyond that, the EK can't be used directly as a signing key to prove possession of the EK secret key; to protect privacy in privacy-sensitive applications, that proof can only be done indirectly through other keys using TPM Certify information.

As a result of these restrictions on EK use, a separate set of credentials traceable to the device manufacturer is used to identify the device.

Similarly, some Device Manufacturers may supply a Platform Certificate including a manufacturer's serial number (see *TCG Credential Profiles For TPM Family 2.0* [12]). While that's similar to a DevID, again, they're not the same:

- The Platform certificate is an *Authorization Certificate*, not a Public Key Certificate (See RFC 5755).
- An Authorization Certificate isn't bound to a key pair, so it can't be used for proof of identity. The Platform credential does identify the associated EK credential, but has no key of its own for signing.

In some cases, the platform credential might be used as a link in the chain to create and install an IDevID after the device has left the manufacturer's premises, although that use is not described in this document.

5.2 Secure Zero Touch Provisioning

There are many cases where a networking device may be shipped with no unique configuration, but must be configured before it can be used with a network.

In these cases, Zero Touch Provisioning (ZTP) may be desirable, requiring a mechanism where the device communicates through the network as soon as it's activated, to obtain the configuration information that would specify policy for operational use. As an example, downloaded configuration might enable access to a corporate VPN, or might authorize access to restricted content.

Part of this use-case may require proof that the device is actually in the hands of an authorized user before it is configured, perhaps through the use of an out-of-band authorization code.

The IETF *Zero Touch Provisioning for NETCONF or RESTCONF based Management* and IETF *Autonomic Networking Integrated Model and Approach (anima)* bootstrapping work provide possible frameworks in which the configuration process can work.[25] See:

https://datatracker.ietf.org/doc/draft-ietf-netconf-zerotouch/

https://datatracker.ietf.org/doc/draft-ietf-anima-bootstrapping-keyinfra/

TCG technologies can improve the security of ZTP in two ways:

- A DevID key stored in the TPM (Section 5.1) can provide cryptographic assurance of the identity of the device attempting to register for configuration. One way to do this is described in IETF RFC 7030, "*Enrollment over Secure Transport*".
- The Trusted Network Connect IF-M protocol suite can be used to offer assurance to the central configuration system that the Device is running authorized software. This mechanism is described in IETF RFC 5792, "*PA-TNC: A Posture Attribute (PA) Protocol Compatible with Trusted Network Connect (TNC)*". See also Section 5.7 below.

5.3 Securing Secrets

Network Equipment may need to manage many different kinds of secrets, and it may not be possible to use a single mechanism to secure all of them.

The same mechanism used to create and store LDevID private keys inside the TPM can be used for other uses of asymmetric keys, as long as the rate at which challenges must be signed is modest (around five per second for many hardware TPMs[26]). An advantage of keeping these keys inside the TPM is that no amount of bus-snooping will reveal the secret.

In cases where higher throughput is needed (e.g. a VPN Gateway that must validate many thousands of sessions, at a rate of hundreds of sessions per second), or applications where shared-secret symmetric keys are used, it will be necessary to pass secrets to cryptographic functions outside of the TPM. In that case, secret keys can be stored in the TPM using a mechanism called Sealing, where keys or other secrets are stored in the Device's file system in an encrypted file that can only be decrypted with keys released from the TPM when pre-defined criteria are met.

Sealing is a mechanism provided by TPMs to encrypt small data objects (typically symmetric keys) using a key that's kept in the TPM. Sealing and Unsealing offer a couple of capabilities:

[25] As of May 2017, these works are still in Draft format.

[26] See *Waltzing the Bear, or: A Trusted Virtual Security Module*, Ronald Toegl, Florian Reimair, and Martin Pirker, pg 155 for some measurements of hardware TPMs. https://online.tugraz.at/tug_online/voe_main2.getvolltext?pCurrPk=67562

- The object to be encrypted by the TPM is unstructured; that is to say, the TPM regards the object simply as a string of bytes. In TCG documents, this data object is referred to as a 'blob'.
- The encrypted object is not stored in the TPM; it's sent to the TPM along with keying and access information, using a "Sealing" operation, and the TPM returns the object encrypted, where it can be stored in the host platform's file system.
- The sealed object can be unsealed when returned to the TPM, assuming a number of conditions are met:
    - The relevant asymmetric key must be resident in the TPM.
    - Authorization to use the parent key must be present.
    - In addition, the sealed object would usually require that one or more of the PCRs have a particular value, indicating that the platform is in a known state.[27]

If the conditions are met, the TPM will return the decrypted blob, which the system designer can use as a structure containing symmetric keys for decrypting a larger object. Alternately, the decrypted blob could contain keying material for use in high-throughput accelerators or other applications.

The system designer has complete freedom in choosing which, if any, PCRs should be consulted prior to unsealing the object. Here are two examples:

1. Some secrets need to be accessed just once, early in the boot sequence of a platform. In that case, the system designer could allocate one PCR specifically for this use, and seal the secret against this PCR with a value of zero (the value after reset). After retrieving the secret, the system designer could extend that PCR with an arbitrary value, preventing the object from being unsealed again by a hacker.
2. The system designer could seal the secret against a number of PCRs that represent the exact versions and configurations of software booted on the box, preventing the secret from being revealed unless the OS is in the exact right configuration. This approach needs to be used with caution, as a software update that's not flawlessly staged could render the secret permanently unintelligible.

5.4 Protection of Configuration Data

Network Equipment often has many configurable settings that must all be coordinated to achieve secure and reliable network connectivity.

Organizations wishing to enforce network policies may want to create configuration files centrally, and distribute them to routers in a way that they can't be changed or moved from one router to another.

Transport-level security (e.g. a TLS session authenticated with a DevID) would normally be used with network-connected devices to ensure that the right configuration goes to the right

[27] Technically, an object can be sealed without requiring reference to any PCR, but that would have the same effect as a simpler encrypt/decrypt operation.

device, but an alternate approach to this that can work across an air-gap is to encrypt and sign configuration files so they can only be decrypted and used on the intended router.

This can be accomplished if the organization managing the network devices creates a configuration file for each device, and encrypts each one using a symmetric key, which in turn can be encrypted using the public portion of an IDevID or LDevID key corresponding to the desired device.[28] Authenticity can be assured by signing the package.

Once installed on the device, the configuration file can be decrypted using the TPM_UnBind() (for TPM1.2) or TPM2_Decrypt() (for TPM2.0) to decrypt and execute the file.

Administrators must build in enough agility in the provisioning process to encrypt configuration files with new keys when fielded units must be replaced.

5.5 Remote Device Management

A management station should be able to perform checks at any time to verify the identity, software and configuration of each device. Classic issues like missed updates and deviations in configuration parameters can be detected in an automated way, making established management systems more secure in the following ways:

- Trusted Computing allows monitoring and verifying the state of the devices against expected values documented in the management system. Deviations can be reported automatically to the Administrator using Remote Attestation and the PTS protocol suite (see *TCG Attestation PTS Protocol: Binding to TNC IF-M* [11]), reducing operational costs to the customer.
- Remote Attestation can also increase confidence in the supply chain, by proving that devices have the correct software loaded.
- Using a DevID to authenticate the device tightens the ability of the management station to check the inventory of devices deployed. Unauthorized or changed devices can be detected without manual checks of the physical devices.

5.6 Software Inventory

The International Standards Organization (ISO) has published document ISO/IEC 19770-2:2009 *Information technology -- Software asset management -- Part 2: Software identification tag* [6] describing the format for Software Identification tags (SWID tags), stored in XML files on a device. SWID tags encode version information and expected hashes for each software component, and should be updated by the device manufacturer's software update process each time a new image or patch is installed.

SWID tags can be retrieved from a device to determine which software was last installed on the device. Although techniques beyond the scope of this document must be used to determine if the software installed is the version expected, comparison of the hashes in the

[28] Some customers may prefer to use separate signing and encryption keys, and not have a single DevID that must be configured for both. See Section 5.1.1.2 for tradeoffs.

SWID tags with hashes retrieved from the TPM would provide assurance that the installed software matches a valid release from the manufacturer.[29]

While SWID tags are simply files and can be retrieved with any number of different mechanisms, the TCG Trusted Network Communications workgroup has incorporated SWID tag retrieval into the TNC protocol suite. Specified in *SWID Message and Attributes for IF-M*, the TNC protocol allows a management station to determine whether each network device is loaded with the expected release.

The presence of a particular SWID tag indicates which software version is probably installed. For purposes of attestation, the SWID tags must be signed by the network device manufacturer, or potentially by the Administrator of the device or other organizations, providing authority that the hashes claimed by SWID values are the ones that should be present on the device.

SWID tags contain hashes of the software modules they describe; with careful alignment by the network device manufacturer, these hashes in SWID tags can be used in some cases as "known good values" for Attestation, and can be compared to what was recorded in the TPM using a measured-boot process.

This allows a degree of automatic checking as part of the Attestation procedure described in Section 5.7:

- Use the TNC protocol described in *SWID Message and Attributes for IF-M* to retrieve signed SWID values showing what should be installed on the device.
- Use TNC protocol *TCG Attestation PTS Protocol: Binding to TNC IF-M* [11] to retrieve signed PCR values from the TPM, along with boot logs, to determine what was actually loaded during the boot process.

Signatures, module names and hashes can then be compared by the external Attestation Server to detect tampering.

The following documents give details on SWID tags:

- *ISO/IEC 19770-2:2009(en) Information technology — Software asset management — Part 2: Software identification tag* [6]
- *Guidelines for the Creation of Interoperable Software Identification (SWID) Tags*; NIST IR 8060 (http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8060.pdf)
- *TCG Trusted Network Connect: SWID Message and Attributes for IF-M*, Version 1.0, Revision 27, 2 June 2015

  http://www.trustedcomputinggroup.org/files/resource_files/F47E6340-1A4B-B294-D0B3F7FF6497F489/SWID_Messages_For_IFM_v1r29.pdfhttp://www.iso.org/iso/catalogue_detail.htm?csnumber=53670

[29] Of course the definition of what's being hashed and the hash algorithms have to align to be useful.

5.7 Attestation of Integrity for Network Devices ("Health Check")

Attestation is a mechanism by which *measurements* stored in a TPM can be retrieved by a remote attestation server to determine the configuration of the equipment at boot time as well as at runtime. In the case of Network Equipment, remote attestation might be invoked by a management station (called the Challenger in TCG documents), causing a networking device to report its boot (and runtime) configuration. Alternatively, two peer devices might do Mutual Attestation to prove to each other that their configurations are acceptable.

Attestation of both boot time and run time configuration can include a wide range of objects that might have a bearing on the security posture of a device.

Boot time objects typically include:

- The initial firmware loaded (such as UEFI code)
- Options of the initial firmware
- Boot loader
- OS kernel

When extended to run time, Attestation can include:

- Executables launched by the OS
- Shared libraries loaded into memory for execution
- Policy or configuration files
- Scripts
- Cryptographic key material and credentials used by the OS, or other relevant sources of state information.

As a result of an attestation process, the Challenger will receive a cryptographically signed "quote" and log from the Device indicating which versions of which modules were loaded and run as the equipment started up.

Attestation requires that the Device implements "Measured Boot" (Section 4.1), in which, starting at a Root of Trust for Measurement (RTM), each stage computes a hash of the next stage and stores the result in a TPM Platform Configuration Register (PCR) before launching it. Each stage of the process can also measure any ancillary files that may be of interest in determining the device's security state, as listed above. Along with extending measurements into PCRs, device software is expected to keep a log file which contains the name and version of the object along with the hash. The log file does not need to be secure; the management station can confirm its contents by computing the hash of the hashes and comparing the result to the signed PCR values. While the contents of the log file are not specified, the log data structure used by PTS is defined in Section 3.25 of *TCG Attestation, PTS Protocol: Binding to TNC IF-M* [11], and could serve as a template.

To obtain attestation information, the Challenger must establish the identity of the Device, and then request the signed quote plus log information. This can be done using the TCG Platform Trust Services (PTS) protocols, defined in *TCG Attestation PTS Protocol: Binding to TNC IF-M*.

Several components are needed for Attestation:

- The Device must extend the hash of each boot object into the appropriate PCR, and keep the corresponding log entries. If runtime integrity must be attested as well, the

Device must extend the hash of each runtime object into the appropriate PCR, and also keep the corresponding log entries.

- The Device must have an Attestation Key (either IAK or LAK) configured, to allow it to sign the quote.
- Either a Privacy CA must be used to certify an AK, or the AK should be otherwise traceable to the device's identity.[30] (See Sections 5.1.1.3 and 5.1.2.2.)
- The device must implement the PTS protocol (or equivalent) to deliver attestation information.

TCG specifications allow the Challenger to obtain attestation information from the Device, but it's the Challenger's task to determine if the hashes are acceptable. Acceptable hashes for software modules are called Reference Integrity Metrics (RIMs) in TCG documents[31], and:

- May be obtained directly from the OEM.
- Could be learned from 'known-good' systems.
- Could be retrieved from the device under attestation, assuming the reference measurements are signed by the entity that's authorized to provide software for the product.

Extending Measured Boot into the OS requires additional OS kernel functionality, allowing for recording of measurements before software execution or reading[32] of files identified as security-critical. Each file identified for analysis is measured and recorded in the TPM, plus a measurement log. Unlike earlier stages of the boot process though, the OS measurements may continue as long as the system runs, resulting in incremental changes to logs and PCRs as different objects are accessed through the system's life.

5.7.1 Linux Integrity Measurement Architecture (IMA)

In Linux, a part of the security sub-system, called the Integrity Measurement Architecture (IMA), carries out the measurement process. IMA extends the principle of Measured Boot into the operating system by providing means to measure selected applications started by the OS. System designers and administrators can benefit from the following:

---

[30] Devices in which the AK and DevID are not tightly bound are subject to the "Asokan Attack" (RFC 6813) where an infected device tricks a clean device into attesting on its behalf. Mechanisms such as TPM_Certify and SKAE can be used to provide proof that the AK and DevID are resident in the same TPM

[31] For TPM1.2, see *TCG Infrastructure Working Group Reference Manifest (RM) Schema Specification*, Version 2.0, https://www.trustedcomputinggroup.org/tcg-reference-manifest-rm-schema-specification/.

[32] A examples of simple file reads that could have integrity implications might be configuration files or interpreted-language scripts such as shell or Python. These aren't strictly executable, but they may be able to trigger unauthorized execution.

- Measurement of any software executable on the system as it is loaded into memory for execution, including shared libraries.
- Measurement of all files read by a particular user, including programs running with permissions of that user.
- A system designer may benefit from that by measuring all files of specific programs, such as database systems or web servers, typically running under a dedicated user account.

The system designer must decide on the OS and application components to be measured. Executable objects would typically be measured, but measurement may also include any other kind of object such as those listed in Section 5.7 above. IMA maintains a measurement log, referred to as Integrity Measurement Log (IML) in TCG documents[33].

IMA has many configuration options:

- IMA log formats are configurable using *IMA templates*, allowing a system designer to adapt the log format to his/her needs.
- IMA is configurable using a policy, which to a certain extent allows for definition of files to be measured. IMA uses that policy to decide whether a file is to be measured or not according on each file Open(). The policy can be hard-coded and compiled into the Linux kernel or it can be set dynamically on every boot by writing it to a file in the Linux *securityfs* filesystem (*/sys/kernel/security/ima/policy*). It is advisable to always set the policy as early as possible in the boot process, although Linux kernels of version 4.7 and later allow the policy to be updated after initial boot.
- IMA also supports policies that are based on Linux Security Modules (LSM) rules. LSM provides an interface for security technologies to extend the Linux kernel's capabilities with more security-related features. It is used by technologies such as SELinux, Simplified Mandatory Access Control Kernel (SMACK), TOMOYO, AppArmor, Yama, and IMA, as well as others[34]. IMA is usually used to measure immutable files, so to exclude all log files—which are, by definition, mutable—the SELinux attribute "logfile" can be used to tag these, allowing the definition of a policy rule to exclude files that are tagged with that attribute. Of course, an administrator must create a policy rule to set that attribute on all log files. The same applies to scripts and other interpreted executables. Using LSM rules an administrator can improve the measurement process and define more fine-grained IMA policies by including and excluding files depending on their expected usage.

However, system designers need to be aware of some limitations to IMA, and some impact on the overall system, including the following:

- Files that are accessed by IMA for measuring may cause measurement violations due to concurrent exclusive file accesses, called the Time of Measure Time of Use (ToMToU)

[33] also known as Stored Measurement Log (SML) in some documents

[34] See https://wiki.gentoo.org/wiki/Extended_Verification_Module.

problem[35]. Whenever IMA cannot gain read access to a file a measurement violation is indicated by a special log entry, containing a file hash of all zeros, but, in turn, the PCR gets extended with a hash of all ones. System designers may wish to flag that situation when analyzing log files.

- Binding the platform state to an IMA-maintained PCR might not make a lot of sense in most cases. As program execution order, even during boot, is non-deterministic on most modern operating systems, the sequence of measurements in the log is very likely not to be the same across boot cycles. This will result in unpredictable PCR values, even if the set of executed programs has not changed.
- Verification of IMA measurements would require an enhanced verification to a management station. Instead of just comparing a single PCR value to a known-good one, the management station needs to calculate the accumulated hash over the Integrity Measurement Log and compare it to the reported hash from the TPM Quote. Furthermore, the management station needs to compare all entries of the measurement log to previously defined reference measurements, RIMs. That is, it must hold hundreds or even tens of thousands of reference values for a particular system configuration. Typically, a search-optimized (indexed) database is used.
- As the IMA measurement process continues operation, even during the retrieval of a “TPM Quote”, there might be some mismatch between the last recorded entry in the PCR and the last entry in the log, when retrieving it. The log might have advanced while the “quote” operation was running. Accordingly, the appropriate log entry—the last one that was incorporated in the PCR—must be identified to allow the Challenger to verify the log. The process for identifying that log entry can look like the following:
    1. Retrieve the log for the first time, L1 (with the number of entries |L1|).
    2. Generate the TPM Quote.
    3. Retrieve the log again, L2 (with the number of entries |L2|).
    4. Check whether the log has advanced (|L2| > |L1|[36]). If so, then the log entry that has been used for the TPM Quote must be identified, which can either happen on the device under attestation itself or on the management station, which, in turn, needs either both of the log files or just the second one (L2) and additionally the number of entries in L1 (|L1|). The process would look something like what follows:[37]
        a. Calculate the accumulated hash from the whole first log (L1).

[35] If a file that is opened exclusively for use/writing (at the time of use), the measurement may fail, as the measurement process cannot also open the file for reading in order to measure it (at the time of measurement).

[36] |Lx| indicates the number of entries in the log.

[37] While the order of entries in the log may be non-deterministic, we do assume that the log file contains entries in *exactly* the same order as they were measured into the TPM.

   b. Compare the accumulated hash against the hash from the TPM Quote. If it matches, then return the full log up to the current entry. Otherwise, continue.

   c. Update the accumulated hash by incorporating the next log entry from list L2. Then continue with step (b).

- In an OS with multiple concurrent processes, the TSS Access Broker should be used to manage concurrent access from multiple processes to a single TPM.[38]
- While the file hashes are collected, none of the common file metadata is logged by IMA, so there's no record of user ID or group ID, either of the file itself or the process accessing it, or file attributes.
- IMA cannot distinguish between text files, log files, executable scripts, and other files. IMA can just detect access to files other than executables. Affected are all kinds of interpreted languages, such as Java, Python, Ruby, Lua, shell scripts, etc. IMA's policy language is insufficient for that purpose. Depending on the number of files involved, IMA's facilities for measuring every file can be used. The Challenger then is responsible to identify relevant files, such as scripts. As IMA keeps an internal hash table to avoid adding duplicate entries to the log, measuring a lot of files may cause that hash table to consume a notable amount of kernel memory and may cause out-of-memory errors. LSM rules in an IMA policy, as mentioned above, can be used to restrict measurement only to files that have security significance.

IMA is subject to continuous improvement, and may offer additional desirable features after this document has been published.

5.7.1.1 IMA Appraisal

IMA Appraisal is the equivalent of Secure Boot extending into the operating system. In contrast to IMA measurements, IMA Appraisal requires a file to match a known-good measurement in order to allow access.

Known-good measurements can be managed several ways.

One approach is to simply learn the hash of each file from a clean system. For that purpose, a particular boot argument to the Linux kernel must be passed (*ima_appraise=fix*), instructing IMA to record the known-good measurements of accessed files. On any subsequent boot, IMA appraisal can be activated to enforce access of a file only if it matches the known-good measurement, by passing the corresponding kernel boot argument (*ima_appraise=enforce*).

By default, IMA allows access to files if the measurements match and will update the known-good measurement if the file is modified after having passed the check on the initial file-open.[39]

IMA also supports *immutable* files utilizing digital signatures on the known-good measurements, generated and signed when the software components are built, prior to

[38] For example, see http://www.trustedcomputinggroup.org/wp-content/uploads/TSS-TAB-and-Resource-Manager-00-91-PublicReview.pdf

[39] https://wiki.gentoo.org/wiki/Integrity_Measurement_Architecture#Registering_the_file_hashes_for_the_system

installation on a particular machine. Private keys stay with the build system; only the signer's public key(s) must be available in the IMA keyring (currently *.ima_root_ca*), although they must be loaded into the keyring early in the boot process, typically in an *initramfs*.

Digital signatures provide protection against online tampering with the files. Offline tampering would still be possible by generating a malicious key-pair, re-signing all the files with the malicious private key, and then replacing the public key with the malicious one. It's possible to protect against offline changes to extended file attributes in general, including the IMA Appraisal known-good measurements, with the Linux kernel Extended Verification Module (EVM)[40]. With EVM, IMA can use a cryptographic hash (HMAC) or digital signature, with a key loaded at boot time, to verify the extended file attributes on any access. EVM keys can be protected by the TPM using Sealing, or in applications where there's an interactive user, a password can be used to unlock the EVM key.

A system designer may utilize IMA Appraisal whenever a system must run only approved software that is known prior to device deployment. While systems which use signatures generated by the device manufacturer's software build system can readily be updated in the field (assuming keys don't change), the software update mechanism on systems that require re-measurements of software on the system itself is out of scope for this document and is left to be defined by the system designer.

5.7.2 PCR Definitions

The allocation of various functions to specific PCRs depends a lot on the underlying software architecture of the Network Equipment.

There are two definitions available at the time of this writing to serve as starting points for Network Equipment:

- UEFI-based *TCG_PC Client Implementation for BIOS.pdf* Section 3.2.3
- Mobile Platforms: *TPM2.0 Mobile Common Profile* [14] Section 2.5

Guidelines for information to be collected in PCRs can also be found in the following:

- *NIST SP800-155 BIOS Integrity Measurement Guidelines*; http://csrc.nist.gov/publications/drafts/800-155/draft-SP800-155_Dec2011.pdf

5.8 Composite Networking Devices

In modular Network Equipment, there's usually a control plane that manages field replaceable units (FRUs) like line cards, each of which may contain complex control processors with TPMs themselves. FRUs, including control plane processors, can usually be "hot swapped", that is, inserted and removed while the rest of the system continues to run,

[40] See https://wiki.gentoo.org/wiki/Extended_Verification_Module

yielding a system where components may come and frequently, even though the system as a whole rarely goes down or starts up all at once. This results in several challenges:

- Each time an FRU is inserted or removed, a control plane process typically notes the change in configuration, initializes the new unit and may alert other components in the system of the new configuration. These new components should not be admitted to the system unless they can show that they are in a known-good state.
- The conventional definition of attestation may need to be extended with the goal of reporting not just the attested state of one control plane processor, but including also the state of the subsidiary FRUs, each with their own processor state.

TCG offers a number of mechanisms that are useful to the design of such systems:

- **Integrity-Protected logs** could be used to keep a tamper-evident record of modules that were added or subtracted during the life of the system.
- **Identity** - The system designer must set the rules and procedures for determining membership in the composite device, but Device Identity (e.g., IDevID or LDevID) could provide a solid basis for determining if specific devices are eligible to join a composite or not.
- **Internal Attestation** among elements – System designers may want elements of a system to prove not only their identity, but also that they're running authorized software.[41]
    - In some cases, control processes may already know what versions of software they'll accept on subsidiary units (e.g. a router control plane may only work with line cards with certain software revisions, and if the software isn't up to rev, the control plane may force an upgrade.)
    - In other cases, units may need to provide evidence that software versions being run are authentic (e.g. signed SWID tags). This might be applicable to peer nodes in a distributed system, where there's no obvious leader to dictate expected software versions.
    - Except for the most aggressively distributed systems, these mechanisms would function inside the composite "device", and would be designed to suit the manufacturer's software environment, without the need for interoperability specifications. As a result, extensions to standards are not likely needed for this application.
- **External Attestation** – Composite systems will usually elect a leader, either through hardware or software means, to communicate with the outside world and represent that state of the system (i.e. provide a management plane and management interface). One of the functions that might be desired is Remote Attestation, where an external management device can identify the system, its components and the authenticity of software running on the system, including all the software running on subsidiary units which may not be visible to an external management agent.

[41] For example, a Device shouldn't use a crypto-accelerator FRU unless it can prove that it's not been tampered with.

5.9 Integrity-Protected Logs

Use cases like Security Information and Event Management (SIEM), IETF *Security Automation and Continuous Monitoring* (SACM) or even Lawful Interception rely on log files created by devices deployed in the field outside the direct, physical administration of the operator or owner. Supporting these use cases requires integrity-protected or non-repudiable log files created by the device. Protecting the integrity requires evidence of the authenticity, order, and time of the log file entries.

An integrity-protected log can be utilized in cases like Lawful Interception proving the correct operation of a particular device or link at the time of data interception, for later forensic or court-ordered evaluation.

5.9.1 Functional Requirements

Integrity-Protected logged data must be protected against undetected manipulation such as deletion, insertion of modification of log events, and can additionally provide evidence of the state of the system at the time of the logged event.

Each entry in the log file provides the following information:

- Type of event
- Date and time
- Order relative to other events

The TPM-issued signature covers the following information, showing it has not been modified after being signed:

- Identity of the log device, based on the DevID
- A non-volatile monotonic sequence number showing the order of events and proving that the current event immediately follows the previous event, so that causal relationships can be determined, and deletion of events can be detected
- (optional) time of signature
- (optional) state of device during signature operation

Network Devices may reboot unexpectedly, or they may operate without a reboot for years; as such, logging mechanisms must typically meet two additional goals.

- Log files must persist through a reboot, and retain integrity through the reboot.
- But at the same time, devices may produce a lot of logging information, so mechanisms that “trim” the logs, discarding old entries, are often provided.

The need to discard old log information implies that there can be no expectation of a permanent log that goes all the way back to the first use of the Device; it’s the Administrator’s responsibility to collect and archive log information periodically if a long-term record is needed. The intent of the mechanism described here is to allow the Administrator to demonstrate that a set of log files provides a complete record.

5.9.2 Integrity-Protected Log Implementation

Implementation of Integrity-Protected Logs uses a number of TPM mechanisms:

- The TPM time-keeping mechanism can be used to generate tamper-resistant time stamps on log entries.
- The Monotonic Counter mechanism can be used to sequence log entries and to detect missing or added entries, even across boot cycles.
- The TPM signing mechanism can be used to sign log entries so that any changes in an individual entry can be detected.

5.9.2.1 Timekeeping in TPM 1.2

While the TPM does not keep a real-time clock, it does provide a mechanism that keeps track of elapsed time since the last initialization. This relative time measure is called a Tickstamp (*TPM1.2 Part 1 Design Principles* Chapter 20).

The tick counter increments at a specified rate, but cannot be "set", and does not relate to any external real time, so there is usually a requirement to establish the correlation between the monotonic tick-counter of the TPM and a trustworthy external real time clock.

The correlation process works by measuring the difference between an authorized external time source and the local Tickstamp timer. A Tickstamp is taken prior to the request to an external source, and then again after the response has been processed, yielding a bounded uncertainty in time correlation. The internal tick counter of the TPM is subject to drift, so correlation may have to be repeated periodically to maintain adequate sync. (See *TPM 1.2 Part 1 Design Principles* Section 20.3 for a detailed description.)

Figure 5 shows the process:

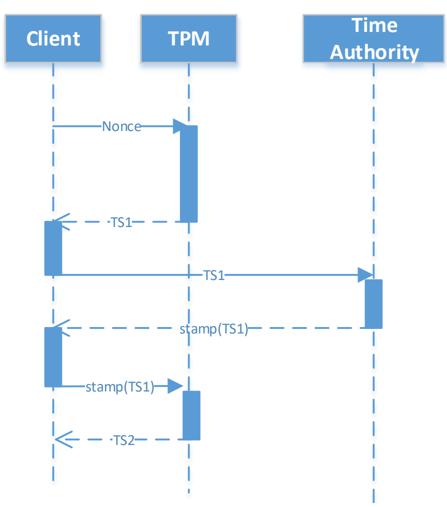


Figure 5 : Time Correlation

The TPM supports the generation of signed tick stamps that include a signature over user-supplied data associated with the current tick count value, the tick increment rate and tick session nonce. Every time the TPM is reset a new tick session nonce is generated. (See *TPM 1.2 Part 1 Design Principles* Section 20.1 for a detailed description.)

5.9.2.2 Timekeeping in TPM 2.0

TPM 2.0 offers additional timekeeping facilities, described in *TPM 2.0 Part 1 Architecture* Chapter 36.7. *Time* contains the time in milliseconds since the last startup of the TPM. *resetCount* is a counter that increments every time the TPM is successfully reset. Its purpose is, among other things, to indicate possible discontinuity in *Clock*. *restartCount* is used to provide an indication of discontinuities due to (1) TPM Resume, (2) TPM Restart, or (3) *_TPM_Hash_Start*; all allowing TPM *Time* to fall behind real time (*TPM 2.0 Part 1 Architecture* Chapter 36.5).

Clock is a volatile value that increments in memory each millisecond. The non-volatile value *NV Clock* is updated periodically from that value. On every reboot the value of *Clock* is set to the value of *NV Clock*. However, in case of unexpected power loss the value of *Clock* may be reported twice as the final *Clock* value might not have been written to *NV Clock*. The TPM flag *safe* of the *TPMS_CLOCK_INFO* structure is used to indicate such a situation of unexpected power loss. Mitigations are described in *TPM 2.0 Part 1 Architecture* Chapter 36.2. In contrast, an orderly shutdown of the TPM is possible using the TPM2_Shutdown() command, which tells the TPM to preserve an appropriate state.

Neither the TPM 1.2 nor the TPM 2.0 has a real-time clock that is able to advance time in the power-off state. The TPM1.2 clock retains no state through a power cycle, but in TPM 2.0, *Clock* is non-volatile, and can be set forward (never backward, though, except by installing a new owner) by external software managing the TPM, allowing for correlation to an external real-time clock. As the TPM may be driven by an imprecise frequency source, the clock is subject to drifting. To compensate, the TPM 2.0 allows external software to adjust the rate of advancement by +/- 15 %.

Attestation data in TPM 2.0 includes the *resetCount* value, although it is obfuscated to preserve privacy (*TPM 2.0 Part 1 Architecture* Chapter 36.7). Nevertheless, an attesting management station can detect a reboot, and hence a possible discontinuity in time, by checking whether the *resetCount* value has changed. TPM 2.0 also allows for usage of timing data in policies, e.g. to limit the usage of keys for a certain amount of time.

5.9.2.3 Assembling the Log

While this document does not specify the exact format of Integrity-Protected log files, the logs can be assembled using TPM Monotonic Counters for sequencing, TPM signing for integrity, and Tickstamps for temporality.

To detect addition or deletion of events in the log, TPM Monotonic Counters (*TPM1.2 Design Principles* Chapter 17, *TPM Rev 2.0 Part 1 – Architecture* Section 37.2.4) can be used. These counters can be incremented by a TPM user, but can never be decremented or cleared. This mechanism allows log entries to be created where each entry has a sequence identifier that's exactly one larger than the previous entry, allowing deletions or additions to be easily detected. Old log entries may be deleted, but in the remaining logs, the monotonic count should increment uniformly from the oldest entry through each intermediate entry to the newest.

For high-value, low-rate[42] log information, a simple mechanism can be used:

- Each new log entry can be assembled with a Tickstamp or Timestamp to show when it happened, and the next value from a non-volatile Monotonic Counter.
- The log entry can be signed with an IDevID or LDevID key, and appended to the log file.
- Reboots of the system and restarts of the TPM can be detected using the tick session nonce of a Tickstamp (TPM 1.2 and TPM 2.0) or the values *resetCount* and *restartCount* (TPM 2.0 only).

Any subsequent tampering will invalidate signatures, or show a gap in the sequence numbers.

It should be noted that the TPM also incorporates a mechanism which can be used to audit the command codes which it has been called upon to execute; this mechanism works by keeping an internal digest of commands, which can be compared to an external log file of what commands were executed.

High-rate logs would require a more complex block-signing mechanism, something beyond the scope of this document.

5.10 Entropy Generation

Many network protocols rely on a dependable source of random numbers for correct operation. For security related protocols, random numbers may be used by a cryptographic element. For other protocols, this might be a TCP sequence identifier, or a source port number, or some other nonce used to identify a particular session or traffic flow.

The TCG TPM definition includes a requirement for a Random Number Generator, which can be accessed after the TPM self-test sequence is complete, simply by invoking the TPM_GetRandom() or TPM2_GetRandom() ordinals.

TCG specifications allow TPM vendors considerable flexibility in implementation choices. A common design approach, not mandated by TCG, would be a mechanism where a physical noise source of some sort (a True Random Number Generator, or TRNG) can be used to seed a Deterministic Random Bit Generator (DRBG), ensuing predictable statistical properties.

For many Network Equipment vendors, compliance to standards such as NIST Special Publication 800-90A/B/C [14] or BSI AIS-31, *A Proposal for Functionality Classes for Random Number Generators* [15], is a requirement; in this case, the OEM designer should consult the TPM vendor for compliance certifications.

There are two aspects of the standards that OEM designers should consider:

a) The standards require continuous testing of the components of the RNG, in addition to a self-test at startup. These tests must be done inside the TPM, as they require

[42] Average rate of a few log entries per hour; TPMs can't sign quickly, and the Monotonic Counter in TPM1.2 will wrap and/or wear out if it's incremented more than once every few minutes.

access to internal state. If there's a failure of the self-test at any point, the TPM will report a failure in response to further Requests.

b) The standards require that entropy from a physical source (TRNG) should be added periodically to the DBRG, so that repeated reads will not exhaust the entropy supply.

A compliant TPM is required to monitor and replenish the entropy level as needed, and may return fewer bytes of random numbers than requested should requests outpace generation of new entropy.

Rates at which new entropy is generated within the TPM is not specified by TCG or the standards, so it's up to the OEM designer and the TPM vendor to ensure that the TPM in consideration generates entropy at an adequate rate for the application.

The TPM has a mechanism with which external entropy can be added to the TPM's DRBG output (TPM_StirRandom() and TPM2_StirRandom()). As long as the TPM vendor certifies that the device generates entropy on its own at an adequate rate, the OEM designer is not likely to need this ordinal.[43]

It's always good practice for the OEM's operating system to collect and merge entropy from as many sources as are available. For example, the Linux OS rng-tools can be configured to merge entropy from a number of sources such as arrival time of network traffic, or disk seek times, with the entropy obtained from the TPM to supply /dev/random for OS and application needs.

5.11 Deprovisioning

Networking equipment often contains sensitive information that an Administrator would want to protect from disclosure, and must be destroyed when the device is deprovisioned.

The Device Manufacturer should provide a mechanism to enable deprovisioning.

Upon deprovisioning, the device:

- Should delete any LDevID, LAK or other customer-generated keys from the TPM.
- Should *not* erase IDevID or IAK keys, since these indicate the manufacturer's device identity, not the Administrator's.

Any installation-specific configuration files should also be deleted.

Deletion of files from modern flash-based memory systems is not easy; an approach to ensure the deletion is reliable would be:

- Create a local storage key as a child of the SRK.
- Store all configuration files in a partition using disk-encryption technology.

Upon deprovisioning, the TPM can be instructed to delete the local storage key, rendering the partition unintelligible.

[43] Assuming a platform has additional sources of entropy, StirRandom can also be used to defend against a hidden failure of the TPM's internal entropy source.

6. References

[1] IEEE Standard for Local and Metropolitan Area Networks
Borza, Mike (ed.) and Max Pritikin (ed.), “802.1AR-2009 – IEEE Standard for Local and Metropolitan Area Networks - Secure Device Identity”, IEEE Computer Society, New York, New York, December 10, 2009,

http://www.ieee802.org/1/pages/802.1ar.html

[2] Trusted Computing Group, *Trusted Platform Module Library, Part 1: Architecture, Family “2.0”*, Level 00 Revision 01.16

[3] Trusted Computing Group, *Trusted Platform Module Library, Part 2: Structures, Family “2.0”*, Level 00 Revision 01.16

[4] Trusted Computing Group, *TPM Keys for Platform Identity for TPM 1.2,* Version 1.0, Revision 2, 12 May 2015

[5] Trusted Computing Group, *TPM Keys for Platform DevID for TPM 2,* Version ??, Revision ??, [Not published as of July, 2017] [http://members.trustedcomputinggroup.org/apps/org/workgroup/infra_wg/download.php/31089/TPM%20Keys%20for%20Platform%20DevID%20for%20TPM%202.0%20--%20Current%20Draft]

[6] The International Organization for Standardization/International Electrotechnical Commission (ISO/IEC), *Information Technology - Software Asset Management - Part 2: Software Identification Tag*, ISO/IEC 19770-2, November 2009 https://www.iso.org/obp/ui/#iso:std:iso-iec:19770:-2:ed-1:v1:en

[7] Registry of Reserved TPM 2.0 Handles and Localities
Trusted Computing Group, "Registry of Reserved TPM 2.0 Handles and Localities," Trusted Computing Group, Beaverton, OR, October 11, 2013.

[8] Trusted Computing Group, TCG TPM v2.0 Provisioning Guidance, Version 1.0, Revision 1.0, Mar 15, 2017

[9] EK Credential Profile 2.0
Trusted Computing Group, *TCG EK Credential Profile For TPM Family 2.0, revision 14*, Trusted Computing Group, Beaverton, OR, 2014.

[10] Trusted Computing Group, *TCG Infrastructure Workgroup Subject Key Attestation Evidence Extension*, Specification Version 1.0, Revision 7, 16 June 2005

[11] Trusted Computing Group, *TCG Attestation PTS Protocol: Binding to TNC IF-M*, Specification Version 1.0, Revision 28, August 24, 2011

[12] Trusted Computing Group, *TCG Credential Profiles for TPM Family 2.0; Level 2*, https://members.trustedcomputinggroup.org/apps/org/workgroup/infra_wg/download.php/30794/Credential_Profile_Platform_Att_V1.0-000-17-03-28-Draft.docx [not published yet]

[13] *A Practical Guide to TPM 2.0,* W. Arthur & D. Challener, Apress; 1st Edition, January 28, 2015

[14] National Institute of Standards and Technology, *Recommendation for Random Number Generation Using Deterministic Random Bit Generators,* Revision 1, June 2015, http://dx.doi.org/10.6028/NIST.SP.800-90Ar1

[15] National Institute of Standards and Technology, *Report on Post-Quantum Cryptography, NISTIR 8105* http://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf

[16] Bundesamt für Sicherheit in der Informationstechnik (BSI) AIS-31, *A Proposal for Functionality Classes for Random Number Generators,* Version 2.0, 18 September 2011, https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Zertifizierung/Interpretationen/AIS_31_Functionality_classes_for_random_number_generators_e.pdf

[17] ISO/IEC 18031:2011 Information technology -- *Security techniques -- Random bit generation,* published on: 2011-11, http://www.iso.org/iso/catalogue_detail.htm?csnumber=54945

[18] Trusted Computing Group, *TCG Storage Opal Integration Guidelines*, Version 1.00, Revision 1.00, March 16, 2016 https://trustedcomputinggroup.org/wp-content/uploads/TCG_Storage_ReferenceDocument_Opal_Integration_Guidelines_v1.00_r1.00.pdf

[19] Linux *Integrity Measurement Architecture*, http://linux-ima.sourceforge.net, /https://wiki.gentoo.org/wiki/Project:Integrity

[20] Trusted Computing Group, *IWG Reference Architecture for Interoperability (Part 1),* Specification Version 1.0, http://www.trustedcomputinggroup.org/resources/infrastructure_work_group_reference_architecture_for_interoperability_specification_part_1_version_10, June 2005.

[21] TCG Specification, *TPM 2.0 Mobile Common Profile, Family "2.0"* Level 00 Revision 31, 21 December, 2015

7. Glossary

There are a few terms that are specific to Network Equipment:

- Device – a unique piece of network equipment
- CPE – Customer Premise Equipment
- FRU – Field Replaceable Unit
- From IEEE 802.1AR:
    - **Initial Secure Device Identifier (IDevID)**: The Secure Device Identifier installed on the device by the manufacturer
    - **Locally Significant Secure Device Identifiers (LDevIDs):** A Secure Device Identifier credential that is unique in the local administrative domain in which the device is used
    - **Secure Device Identifier (DevID):** A device identifier that is cryptographically bound to the device and is composed of the Secure Device Identifier Secret and the Secure Device Identifier Credential

TCG-specific acronyms and abbreviations can be found in the TCG Glossary

https://www.trustedcomputinggroup.org/wp-content/uploads/TCG_Glossary_Board-Approved_12.13.2012.pdf

8. Appendices

8.1 Use-Cases For Further Study

8.1.1 Secure Identity in Virtual Machines

Manufacturers are starting to incorporate Virtual Machine technology into Network Equipment, either as an element of an integrated hardware/software product or as a software-only product running entirely within a virtual machine environment, as in Network Function Virtualization.

In either case, there's often a need to identify the hardware resource underlying the virtual machine, to ensure that the function is actually executing in the expected environment, and to determine if the software is running in a virtual machine or directly on a physical machine.

8.1.2 Distributed Composite Devices

In today's infrastructure, composite devices make use of wired or wireless networks as interconnects to merge management planes or sub-sets of control plane functions. Therefore, the physical components of a composite device can be geographically distributed, but act as a single entity on the management plane. In addition, network interfaces can be associated with more than one composite device; each managed via independent management planes interfaces.

There is a need to take into account the type of composition used to merge management planes or control plane functions of distinguishable physical units. Every distributed composition relies on a specific form of interconnect between physically separated units, which can be used as a basis for corresponding solutions.

8.2 Standardization Work For Further Study

8.2.1 Known Good Hashes

There are currently no existing protocols that are being used to convey known-good hashes from the Network Device Manufacturer to Attestation systems that might use these values.

One suggested approach could be for the manufacturer to provide an "oracle", which could return a signed quote and a log with an entry for every possible object that could be measured, using the PTS protocol; the Attesting system then would use this list as a data base to look up expected values.

8.2.2 Composite Systems

External Attestation of a composite system needs new protocol work. The current attestation mechanism assumes attestation is being carried out on an integrated unit with one TPM. In a composite system, attestation might require multiple steps:

- Attest the control plane element that was elected (either by hardware or software) to represent the device to the outside world. This can be done using conventional PTS.
- Ask the control plane element for a list of devices that it currently has accepted into its configuration. This might be a list of line cards, or a list of stacked switches, or a bunch of servers. The list might be flat or hierarchical, as decided by the system designer. All that the management station needs is a list of the elements and how to identify them.
- The management system might then ask for attestation results from some or all of the units reported. Some units may include a separate TPM, which can be leveraged to enrich the attestation of the composite device or even allow for direct remote attestation of a sub-component.

In most cases, the elements that make up the system (e.g. line cards) will not be directly visible externally, i.e., they won't have routable IP addresses to their own control plane processors, resulting in a need for a mechanism to send a request for Attestation that can be forwarded by the receiving management plane processor to the appropriate internal FRU.

For PTS, this would probably be done by adding an "element id" field to the request/response.

For an SNMP-based mechanism, it might be handled by introducing a layer of hierarchy into the MIB.

8.2.3 Reference Manifests

The document *TCG Infrastructure Working Group Reference Manifest (RM) Schema Specification*, Version 2.0, https://www.trustedcomputinggroup.org/tcg-reference-manifest-rm-schema-specification/ defines the XML schema with which integrity information is communicated between entities for TPM1.2.

There currently is not an equivalent document for TPM2.0

1786
9. Revision History

Name	Description	Date	Revision
S. Hanna & G. Fedorkow	First draft	2015-05-01	v0.1
G. Fedorkow		2015-05-07	v0.2
G. Fedorkow	Integrate TCG template and email outlines	2015-05-12	V0.2, R3
S. Hanna & G. Fedorkow	Elaborate and reorganize use-cases	2015-05-28	V1.0 R4
N. Kuntze	Update on the Use Cases	2015-05-29	V0.2, R5
G. Fedorkow	Merge Nicolai’s updates	2015-06-01	V1.0 R6
N. Kuntze	Minor Comments and additional Use Case	2015-06-01	V1.0 R7
G. Fedorkow	Restructure Implementation section to reflect use-cases	2015-06-07	V1.0 R8
G. Fedorkow	Merged comments from Nicolai, Carsten, Steve H and others		
G. Fedorkow	Many updates at Face-to-Face meeting	2015-06-18	V1.0 R9b
G. Fedorkow	New material in Device ID section, proposed text on Privacy, background on 802.1AR and comments from T Laffey and S Hanna	2015-07-10	V1.0 R10b
Max, Henk, Carsten	New material from Max, Henk and Carsten		V1.0 R11
G Fedorkow	Merged other comments and changes		V1.0 R11b
G Fedorkow	Revised RNG Implementation section, updated DevID after Aug 12 phone conf	2015-08-24	V1.0 R12a
G Fedorkow	Outlined approaches to many implementation sections. Merged in text on license management implementation from Steve Hanna	2015-09-07	V1.0 R12c
G Fedorkow	Revision to License Manager implementation section 6.6 to allow operation on air-gapped networks Revision to Composite Inventory section 6.9 to respond to comments Updates responding to T Laffey’s comments on v11b	2015-09-014	V1.0 R13a
G Fedorkow	Merged in updates from Nicolai - Remote Management - Integrity Logs	2015-10-03	V1.0R13c
G Fedorkow	Merged TPM2 material from Bill Sulzen, and addressed many comments from T Laffey (Thanks Tom!)	2015-10-15	V1.0R14a

G Fedorkow	Restructured Section 6.1.1 to eliminate redundancy	2015-10-18	V1.0R14c
G Fedorkow	Updates after Montreal F2F Added Software Inventory sections	2015-12-8	V1.0R15a
G Fedorkow	Highlighted open issues prior to F2F		V1.0R16
G Fedorkow	Comment resolution following San Francisco F2F Eliminated sections on VM (it's new work, not guidance) Revised Identity section to address many comments Simplified Composite Systems and Integrity Logs based on F2F discussion	2016-03-16	V1.0R17a
G Fedorkow	Cleared change bars Moved some of the 'future' work to Appendix Started to clean the bibliography		V1.0R18a
G Fedorkow	Added "How to Install an LDevID" Added review comments from Bill Sulzen	2016-05-13	V1.0R19b
G Fedorkow	Editorial comments resolved and "accepted"; doc change-tracker shows comments that might need additional review Added a short sub-sub-sub-section on EST approaches for conveying TPM_Certify; this section may need normative work somewhere else (probably IETF)	2016-05-25	V1.0R19c
H. Birkholz	Added details on current practice for Composite Devices, sub-component composition and a corresponding subsection 8.1.3. to "Use-Cases For Further Study"	2016-06-01	V1.0R19d
G Fedorkow	Added comments from D Challener Merged updates from Henk on Composite Systems	2016-06-01	V1.0R19e
G Fedorkow	Merged corrections and comments from Graeme Proudler	2016-06-05	V1.0R19f
G Fedorkow	Merged corrections from W Sulzin Started to reduce TPM1.2 bias	2016-06-05	V1.0R19g
G Fedorkow	Merged corrections from S Hanna		V1.0R19h
G Fedorkow	Merged corrections from Nicolai & Michael Eckel	2016-06-19	V1.0R19i
	Updates from Vienna F2F		
G Fedorkow	Replaced IMA section	2016-10-19	V1.0R21a
G Fedorkow	Editorial updates from Seoul F2F call (hmm, that would be Face-to-Phone…)	2016-10-24	V1.0R21b
G Fedorkow	Change bars removed from R21b	2016-10-24	V1.0R22a
G Fedorkow	Closed more reviewer comments Moved New Work items to appendix	2016-11-03	V1.0R22b

G Fedorkow	Reviewer comments from Graeme, TC, Huawei and Cisco	2016-12-28	V1.0R23a
G Fedorkow	Reviewer comments from Kent Watsen, JNPR	2017-01-21	V1.0R23b
G Fedorkow	Updates after February 2017 F2F with change bars	2017-03-22	V1.0R23d
G Fedorkow	Clean copy with change-bars removed, ready for the next round of revision	2017-03-22	V1.0R24a
G Fedorkow	Removed License Management section; needs more restrictions on use. Many editorial corrections from Graeme (thanks!)	2017-05-12	V1.0R25a
G Fedorkow	Editorial corrections from Rob Spiger (thanks!)	2017-05-21	V1.0R25b
G Fedorkow	Microscopic Editorial corrections	2017-05-23	V1.0R25c
G Fedorkow	No functional changes; cleared change bars, removed comment bubbles that have already been addressed	2017-05-24	V1.0R26a
G Fedorkow	Editorial corrections from review comments – commas, periods and semicolons, some clarification of wording	2017-06-05	V1.0R26b
G Fedorkow	Cleared change bars for review	2017-07-03	V1.0R27a