T2TRG Research Group
Internet-Draft
Intended status: Informational
Expires: 30 November 2026

M. Richardson
Sandelman Software Works

29 May 2026

A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors
draft-irtf-t2trg-taxonomy-manufacturer-anchors-18

Abstract

This document provides a taxonomy of methods used by manufacturers of silicon and devices to secure private keys and public trust anchors. This deals with two related activities: how trust anchors and private keys are installed into devices during manufacturing, and how the related manufacturer held private keys are secured against disclosure.

This document does not evaluate the different mechanisms, but rather just serves to name them in a consistent manner in order to aid in communication.

This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG).

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-irtf-t2trg-taxonomy-manufacturer-anchors/.

Discussion of this document takes place on the t2trg Research Group mailing list (mailto:t2trg@irtf.org), which is archived at https://mailarchive.ietf.org/arch/browse/t2trg/. Subscribe at https://www.ietf.org/mailman/listinfo/t2trg/.

Source for this draft and an issue tracker can be found at https://github.com/t2trg/draft-irtf-t2trg-taxonomy-manufacturer-anchors.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

This Internet-Draft will expire on 30 November 2026.

Copyright Notice

Table of Contents

1. Introduction

An increasing number of protocols derive a significant part of their security by using trust anchors [RFC4949] that are installed by manufacturers into a device during manufacturing. Disclosure of the list of trust anchors does not usually cause a problem, but changing them in any way does. This includes adding, replacing or deleting anchors.

The document [RFC6024] deals with how trust anchors are managed in the device which uses them. This document deals with how the PKI associated with such a trust anchor is managed.

Many protocols also leverage manufacturer installed identities. These identities are usually in the form of [ieee802-1AR] Initial Device Identity certificates (IDevID). The 3GPP Vendor Base Station Certificate [3GPP.33.310] is another form that uses a similar design pattern. The identity has two components: a private key that must remain under the strict control of a trusted part of the device, and a public part (the certificate), which (ignoring, for the moment, personal privacy concerns) may be freely disclosed.

There also situations where identities are tied up in the provision of symmetric shared secrets. A common example is the SIM card [_3GPP.51.011]. Provisioning of a physical SIM card is generally considered a one-touch operation. A virtual SIM (an eSIM) could be factory provisioned.

It is further not unusual for many devices (particularly smartphones) to also have one or more group identity keys. This is used, for instance, in [fidotechnote] to make claims about being a particular model of phone. The key pair that does this is loaded into large batches of phones for privacy reasons: If a single key was used, then this would allow tracking. The shared key pair can not be used to identify a specific device.

The trust anchors are used for a variety of purposes. For instance, trust anchors are used to verify:

* the signature on a software update (as per [RFC9019]),

* a TLS Server Certificate, such as when setting up an HTTPS connection,

* the [RFC8366] format voucher that provides proof of an ownership change.

Device identity keys are used when performing enrollment requests (in [RFC8995], and in some uses of [RFC9140].) The device identity certificate is also used to sign Evidence by an Attesting Environment (see [RFC9334]).

These security artifacts are used to anchor other chains of information such as: an Entity Attestation Token (EAT) [RFC9711] Claim as to the version of software/firmware running on a device [I-D.birkholz-suit-coswid-manifest], an EAT claim about legitimate network activity (via [I-D.ietf-iotops-mud-rats], or embedded in the IDevID in [RFC8520]).

Known software versions lead directly to vendor/distributor signed Software Bill of Materials (SBOM), such as those described by [RFC9393] and the NTIA/SBOM work [ntiasbom] and CISQ/OMG SBOM work underway [cisqsbom].

In order to manage risks and assess vulnerabilities in a Supply Chain, it is necessary to determine a degree of trustworthiness in each device. A device may mislead audit systems as to its provenance, about its software load or even about what kind of device it is (see [RFC7168] for a humorous example).

In order to properly assess the security of a Supply Chain it is necessary to understand the kinds and severity of the threats which a device has been designed to resist. To do this, it is necessary to understand the ways in which the different trust anchors and identities are initially provisioned, are protected, and are updated.

To do this, this document details the different trust anchors and identities (IDs) found in typical devices. The privacy and integrity of the trust anchors and IDs is often provided by a different, superior artifact. This relationship is examined.

While many might desire to assign numerical values to different mitigation techniques in order to be able to rank them, this document does not attempt to do that, as there are too many other (mostly human) factors that would come into play. Such an effort is more properly in the purview of a formal processes such as [ISO27001].

This document is a product of the IRTF Thing-to-Thing Research Group (T2TRG) and represents the consensus of the research group.

1.1. Terminology

This document is not a standards track document, and it does not make use of formal requirements language.

This document defines a number of hyphenated terms, and they are summarized here:

birth-certificates: another term for IDevID

device-generated: a private or symmetric key that is generated on the device

infrastructure-generated: a private or symmetric key that is generated by some system, likely located at the factory that built the device

IDevID-certificates: an initial device identity certificate as per [ieee802-1AR]

key-executives: the people who are entrusted with pieces (shares) of a split secret Section 5.3.1

pkilevel: the number of Certification Authorities in a public key infrastructure, counting the root, or trust-anchor as the first level, and not counting the End-Entity certificates Section 5.1

manufacturer-installed-certificates: or MIC, see idevid

mechanically-installed: when a key or certificate is programmed directly into non-volatile storage by an out-of-band mechanism such as JTAG [JTAG]

network-transferred: when a key or certificate is transferred into a

system via private interface, such as serial console, JTAG managed mailbox, or other physically private interface

network-transferred: when a key or certificate is transferred into a system using a network interface which would be available after the device has shipped. This applies even if the network is physically attached using a bed-of-nails [BedOfNails].

device/infrastructure-co-generated: when a private or symmetric key is derived from a secret previously synchronized between the silicon vendor and the factory using a common algorithm.

TPM: a Trusted Platform Module, such as the set of devices standardized by the Trusted Computing Group (TCG). See [TPM20spec].

In addition, Section 4.1 introduces three primary private key generation techniques named _arbitrarily_ after three vegetables (avocado, broccoli, and carrot) and two secondary ones named after two more (squash and spinach). The two secondary ones refer to methods where a secure element is involved, and mnemonically start with the same letter: S.

2. Applicability Model

There is a wide variety of devices to which this analysis can apply (see [I-D.ietf-iotops-7228bis].) This document will use a J-group processor as a sample. This class is sufficiently large to experience complex issues among multiple CPUs, packages and operating systems, but at the same time, small enough that this class is often deployed in single-purpose IoT-like uses. Devices in this class often have Secure Enclaves, and can include silicon manufacturer controlled processors in the boot process. Note that access to the secure enclave is often not available to system integrators. A very common IoT platform, the Raspberry PI, has a secure enclave with the GPU, but access to it is not available.

Almost all larger systems (servers, laptops, desktops) include a Baseboard Management Controller (BMC), which ranges from an M-Group Class 3 MCU, to a J-Group Class 10 CPU (see, for instance [openbmc] which uses a Linux kernel and system inside the BMC). As the BMC usually has complete access to the main CPU’s memory, I/O hardware and disk, the boot path security of such a system needs to be understood first as being about the security of the BMC.

2.1. A reference manufacturing/boot process

In order to provide for immutability and privacy of the critical trust anchors and IDs, many CPU manufacturers will provide for some kind of private memory area which is only accessible when the CPU is in certain privileged states. See the Terminology and Architecture sections of [RFC9397], notably Trusted Execution Environment (TEE), Rich Execution Environment (REE), and Trusted Application Manager (TAM).

The private memory that is important is usually non-volatile and rather small. It may be located inside the CPU silicon die, or it may be located externally. If the memory is external, then it is usually encrypted by a hardware mechanism on the CPU, with only the key kept inside the CPU.

The entire mechanism may be external to the CPU in the form of a hardware-TPM module, or it may be entirely internal to the CPU in the form of a firmware-TPM. It may use a custom interface to the rest of the system, or it may implement the TPM 1.2 or TPM 2.0 specifications (see [TPM20spec].) Those details are important to performing a full evaluation, but do not matter much to this model (see initial-enclave-location below).

During the manufacturing process, once the components have been soldered to the board, the system is usually put through a system-level test. This is often done as a "bed-of-nails" test [BedOfNails], where the board has key points attached mechanically to a test system. A [JTAG] process tests the System Under Test, and then initializes some firmware into the still empty flash storage.

It is now common for a factory test image to be loaded first: this image will include code to initialize the private memory key described above, and will include a first-stage bootloader and some kind of (primitive) Trusted Application Manager (TAM). The TAM is a piece of software that lives within the trusted execution environment.

Embedded in the first-stage bootloader will be a Trust Anchor that is able to verify the second-stage bootloader image.

After the system has undergone testing, the factory test image is erased, leaving the first-stage bootloader. One or more second-stage bootloader images are installed. The production image may be installed at that time, or if the second-stage bootloader is able to install it over the network, it may be done that way instead.

There are many variations of the above process, and this section is not attempting to be prescriptive, but to provide enough illustration to motivate subsequent terminology.

The process may be entirely automated, or it may be entirely driven by humans working in the factory, or a combination of the above.

These steps may all occur on an access-controlled assembly line, or the system boards may be shipped from one place to another (maybe another country) before undergoing testing.

Some systems are intended to be shipped in a tamper-proof state, but it is usually not desirable that bed-of-nails testing be possible without tampering, so the initialization process is usually done prior to rendering the system tamper-proof. An example of a one-way tamper-proof, weather resistant treatment might to mount the system board in a case and fill the case with resin.

Quality control testing may be done prior to, as well as after, the application of tamper-proofing, as systems which do not pass inspection may be reworked to fix flaws, and this should ideally be impossible once the system has been made tamper-proof.

3. Types of Trust Anchors

Trust Anchors are fundamentally public keys with authorizations implicitly attached through the code that references them.

They are used to validate other digitally signed artifacts. Typically, these are chains of PKIX certificates leading to an End-Entity certificate (EE).

The chains are usually presented as part of an externally provided object, with the term "externally" to be understood as being as close as untrusted flash, to as far as objects retrieved over a network.

There is no requirement that there be any chain at all: the trust anchor can be used to validate a signature over a target object directly.

The trust anchors are often stored in the form of self-signed certificates. The self-signature does not offer any cryptographic assurance, but it does provide a form of error detection, providing verification against non-malicious forms of data corruption. If storage is at a premium (such as inside-CPU non-volatile storage) then only the public key itself need to be stored. For a 256-bit ECDSA key, this is 32 bytes of space.

The following subsections explain a number of different kinds of trust anchors that a manufacturer can include into a device. Each kind of trust anchor has different kinds of trust associated with it based on what authorization is associated with the key. That is, these different anchors do different things, and so the risk associated with the anchor depends upon what those things that can be done are.

There are some characteristics associated with each trust anchor that depend upon the associated risk.

1. There is a risk around whether or not a trust anchor can be replaced or modified so that it's a different anchor, (with the associated private key) controlled by a different entity. This is not a risk associated with loss of the original manufacturer's private key, but with integrity of the public key.

2. There is a risk that multiple trust anchors could be added to the device. This would not replace the manufacturer's anchor, but augment it with additional trust anchors not authorized by the manufacturer.

3. There is a risk that a trust anchor could be removed from the device, or rendered unusable. For instance, it might be impractical to replace a trust anchor, but it might be possible via fault injection or high-energy radiation for an attacker to corrupt one or two bytes of a trust anchor, rendering the anchor useless.

4. There is a risk associated with compromise of the associated private key that goes with the (public key) of the trust anchor. While this is the most obvious concern expressed: that an attacker gets control of the key, the above three risks may be more practical for some attackers.

5. There is a risk that the manufacturer will lose access to the private key, without the private key being compromised. No attacker possesses the private key, but neither can the manufacturer use the key anymore.

Should any of the above things occur, the manufacturer will be in a position where they might have to deploy new trust anchors to every device. For some trust anchors, they may be safely replaced with an over-the-air update. For the trust anchor that authorizes over-the-air updates, the manufacturer might be in a position where every single device has to be recalled and serviced with specialized hardware capable of updating the firmware. For some devices, the hardware will have to be entirely replaced, typically at great cost.

This has already happened, see for example: [leakedmsikey] and [hyundaiexample].

3.1. First-Stage Bootloader Trust Anchor

This anchor is part of the first-stage bootloader, and it is used to validate a second-stage bootloader which may be stored in external flash.

This is called the first-stage bootloader trust anchor. In most cases this trust anchor is burnt into fuses in the CPU, often within the die along with the first-stage bootloader itself. It can not be changed without replacement of the entire device. Replacement, removal or modification of this trust anchor is improbable.

The anchor could be rendered unusable if additional fuses can be blown. Some fuse implementations allow for bits to be changed from 1 (unblown), to 0 (blown). Access to blow the fuses is usually restricted. On some devices, it requires voltages not normally present, making this impossible to do by software. However, it might be possible for an attacker to blow fuses using an external high voltage probe or via injection of gamma ray. The likely result however is that the device would no longer boot.

3.2. Software Update Trust Anchor

This anchor is used to validate the main application (or operating system) load for the device.

It can be stored in a number of places. First, it may be identical to the First Bootloader Trust Anchor.

Second, it may be stored in the second-stage bootloader, and therefore its integrity is protected by the First Bootloader Trust Anchor.

Third, it may be stored in the application code itself, where the application validates updates to the application directly (update in place), or via a double-buffer arrangement. The initial (factory) load of the application code initializes the trust arrangement.

In this situation the application code is not started in a secured boot path. The second-stage bootloader does not validate the application/operating system before starting it, but it may still provide measured boot mechanism (see [measuredboot], section 2.3.)

3.3. Trusted Application Manager Anchor

This anchor is the key used by the [RFC9397] Trusted Application Manager (TAM). Code which is signed by this anchor will be given execution privileges as described by the manifest which accompanies the code.

The TAM software _itself_ will typically be verified by the first-stage bootloader, so it needs to be signed by the First Bootloader Trust Anchor. The TAM could also be a component of the first or second stage bootloader.

This privilege may include updating anchors that are present within the TAM, or elsewhere in the Trusted Execution Environment.

3.4. Public WebPKI Anchors

These anchors are used to authenticate HTTPS certificates from websites. These anchors are typically distributed as part of desktop browsers, and via desktop operating systems. On IoT devices with specific purposes [RFC8520], a limited number of connections is expected, so a limited number of trust anchors is usually distributed.

The exact set of these anchors is not precisely defined: it is usually determined by the browser vendor (e.g., Mozilla, Google, Apple, Microsoft), or the operating system vendor (e.g., Apple, Google, Microsoft, Ubuntu). In most cases these vendors look to the CA/Browser Forum [CABFORUM] for inclusion criteria.

3.5. DNSSEC root

This anchor is part of the DNS Security extensions. It provides an anchor for integrity protection of DNS lookups. Secure DNS lookups may be important in order to get access to software updates. This anchor is now scheduled to change approximately every 3 years, with the new key announced several years before it is used, making it possible to embed keys that will be valid for up to five years.

This trust anchor is typically part of the application/operating system code and is usually updated by the manufacturer when they do updates. However, a system that is connected to the Internet may update the DNSSEC anchor itself through the mechanism described in [RFC5011].

There are concerns that there may be a chicken-and-egg situation for devices that have remained in a powered off state (or disconnected from the Internet) for a period of many years. Upon being reconnected, the device would be unable to do DNSSEC validation because the trust anchor within the device would be too old.

This could be fixed with a software/firmware update. However, it could also be that in such a situation, that the device would be unable to validate the DNSSEC names required in order to obtain the operating system update.

3.6. Private/Cloud PKI anchors

It is common for many IoT and network appliances to have links to vendor provided services. For instance, the IoT device that calls home for control purposes, or the network appliance that needs to validate a license key before it can operate. (This may be identical to, or distinct from a Software Update anchor. In particular, the device might call home over HTTPS to learn if there is a software update that needs to be done, but the update is signed by another key.)

Such vendor services can be provided with public certificates, but the very short lifetime for public certificates [CABForum90day] precludes their use in many operational environments. Instead, a private PKI anchor is included. This can be in the form a multi-level PKI (as described in Section 5.1), or degenerate to a level-1 PKI: a self-signed certificate.

A level-1 PKI is very simple to create and operate, and there are innumerable situations where there is just a call to the "curl" utility [curl], with a "--pinnedpubkey" option to specify the anchor.

3.7. Onboarding and other Enrollment anchors

[RFC8995], [RFC8572], and [RFC8366] specify mechanisms for onboarding of new devices. The voucher artifact is transferred to the device by various means, and the device must verify the signature on it. This requires a trust anchor to be built-in to the device, and some kind of private PKI be maintained by the vendor (or its authorized signing designate). [I-D.ietf-anima-masa-considerations] provides some advice on choices in PKI design for a MASA. The taxonomy presented in this document applies to describing how this PKI has been designed.

3.8. Onboarded network-local anchors

[RFC7030], [RFC8995] and [RFC9641] provide mechanisms by which new trust anchors may be loaded by a device during an onboarding process. The trust anchors involved are typically local to an enterprise and are used to validate connections to other devices in the network.

This typically includes connections to network management systems that may also load or modify other trust anchors in the system. [I-D.ietf-anima-masa-considerations] provides some advice in the BRSKI [RFC8995] case for appropriate PKI complexity for such local PKIs.

Note that this trust anchor is that of the network operator, and it is not related to the trust anchor described in the previous section that is used to validate an ownership transfer.

4. Types of Device Identities

Device identities are installed during manufacturing time for a variety of purposes.

Identities require some private component. Asymmetric identities (e.g., RSA, ECDSA, EdDSA systems) require a corresponding public component, usually in the form of a certificate signed by a trusted third party.

This certificate associates the identity with attributes.

The process of making this coordinated key pair and then installing it into the device is called identity provisioning.

4.1. Manufacturer installed IDevID certificates

[ieee802-1AR] defines a category of certificates that are installed by the manufacturer which contain a device unique serial number.

A number of protocols depend upon this certificate.

* [RFC8572] and [RFC8995] introduce mechanisms for new devices (called pledges) to be onboarded into a network without intervention from an operator. A number of derived protocols such as [RFC9733], [I-D.ietf-anima-constrained-voucher], [I-D.ietf-anima-brski-cloud] extend this in a number of ways.

* [RFC9334] depends upon a key provisioned into the Attesting Environment to sign Evidence. The IDevID may be used for this.

* [RFC9019] depends upon a key provisioned into the device in order to decrypt software updates. Both symmetric and asymmetric keys are possible. In both cases, the decrypt operation depends upon the device having access to a private key provisioned in advance. The IDevID can be used for this if algorithm choices permit. ECDSA keys do not directly support encryption in the same way that RSA does, for instance, but the addition of HPKE [RFC9180] allows for use of other key types. There may be other legal considerations why the IDevID might not be used, and a second key provisioned.

The manufacturer has the responsibility to provision a key pair into each device as part of the manufacturing process. There are a variety of mechanisms to accomplish this, which this section details.

There are three fundamental ways to generate IDevID certificates for devices:

1. generating a private key on the device, creating a Certificate Signing Request (or equivalent), and then returning a certificate to the device.

2. generating a private key outside the device, signing the certificate, and the installing both into the device.

3. deriving the private key from a previously installed secret seed, that is shared with only the manufacturer.

There are additional variations where the IDevID is provided as part of a Trusted Platform Module (TPM) or Secure Element (SE). The vendor may purchase such devices from another vendor, and that vendor often offers provisioning of a key pair into the device as a service.

The document [I-D.moskowitz-ecdsa-pki] provides some practical instructions on setting up a reference implementation for ECDSA keys using a three-tier mechanism.

The names of the five methods below are intended to be mnemonic.

* Avocados have big seeds inside of them.

* Broccoli has a seed on the top (which is eaten).

* Carrots have a complicated seed process involving multiple years and many leaves.

* The two methods involving Secure Elements are named with the letter S.

4.1.1. Avocado method: On-device private key generation

In this method, the device generates a private key on the device. This is done within some very secure aspect of the device, such as in a Trusted Execution Environment, and the resulting private key is then stored in a secure and permanent way. The permanency may extend beyond use of on-CPU flash, and could even involve blowing of one-time fuses.

Generating the key on-device has the advantage that the private key never leaves the device. The disadvantage is that the device may not have a verifiable random number generator. The use of a pseudo-random number generator needs to be well seeded as explained in [RFC4086]. [factoringrsa] is an example of a random number generation failure that was visible through survey of public keys, and led to a successful attack on the private keys.

There are a number of options of how to get the public key from the device to the certification authority. As it is a public key, privacy is less of a concern, and the focus is on integrity. (However, disclosing the public key may have impacts on trackability of the device.)

So, transmission must be done in an integral manner, and must be securely associated with an assigned serial number. The serial number goes into the certificate, and the resulting certificate needs to be loaded into the manufacturer’s asset database, and returned to the device to be stored as the IDevID certificate.

One way to do the transmission is during a factory Bed of Nails test (see [BedOfNails]) or JTAG Boundary Scan. There are two ways this can be done. In the _device-generated_ / _mechanically-installed_ method, the public key or certificate signing request (CSR) is retrieved by doing JTAG operations directly with an EEPROM or RAM. The CPU is not involved, and the memory accessed is likely not encrypted. (The private key might be in the same place, or it could be safely elsewhere).

A different JTAG mechanism would involve reading/writing to some other area of the CPU, such as an on-CPU static RAM or debug area. In this case, the CPU is involved. When done via a physical connection like this, then this is referred to as a _device-generated_ / _mechanically-transferred_ method. The key difference is that CPUs usually have much better defenses against JTAG use after manufacturing, often via one time fuses that can blown.

There are other ways that could be used where a certificate signing request is sent over a special network channel when the device is powered up in the factory. This is referred to as the _avocado device-generated_ / _network-transferred_ method.

Regardless of how the certificate signing request is sent from the device to the factory, and how the certificate is returned to the device, a concern from production line managers is that the assembly line may have to wait for the certification authority to respond with the certificate. This is inherently a synchronous process, as the process can not start until the private key is generated and stored.

After the key generation, the device needs to set a flag such that it no longer will generate a new key, and will not accept a new IDevID via the factory network connection. This may be a software setting, or could involve blowing a one-time fuse.

Devices are typically constructed in a fashion such that the device is unable to ever disclose the private key via an external interface. This is usually done using a secure-enclave provided by the CPU architecture in combination with on-chip non-volatile memory.

The risk is that if an attacker with physical access is able to put the device back into an unconfigured mode, then the attacker may be able to substitute a new certificate into the device. It is difficult to construct a rationale for doing this as the attacker would not be able to forge a certificate from the manufacturer's CA. Other parties that rely on the IDevID would see the device as an imposter if another CA was used. However, if the goal is theft of the device itself (without regard to having access to firmware updates), then use of another manufacturer identity may be profitable. Stealing a very low value item, such as a light bulb, makes very little sense. Stealing medium value items, such as appliances, or high-value items such as cars, yachts or even airplanes, would make sense. Replacing the manufacturer IDevID permits the attacker to also replace the authority to transfer ownership in protocols like [RFC8995].

4.1.2. Broccoli method: Off-device private key generation

In this method, a key pair is generated in the factory, outside of the device. The factory keeps the private key in a restricted area, but uses it to form a Certificate Signing Request (CSR). The CSR is passed to the manufacturer's Certification Authority (CA), and a certificate is returned. Other meta-data is often also returned, such as a serial number.

Generating the key off-device has the advantage that the randomness of the private key can be better ensured. As the private key is available to the manufacturing infrastructure, the authenticity of the public key is well known ahead of time.

The private key and certificate can be programmed into the device along with the initial bootloader firmware in a single step.

As the private key can be known to the factory in advance of the device being ready for it, the certificate can also be generated in advance. This hides the latency to talk to the CA, and allows for the connectivity to the CA to be less reliable without shutting down the assembly line. A single write to the flash of the device can contain the entire firmware of the device, including configuration of trust anchors and private keys.

The major downside to generating the private key off-device is that it could be seen by the manufacturing infrastructure. It could be compromised by humans in the factory, or the equipment could be compromised. The use of this method increases the value of attacking the manufacturing infrastructure.

If private keys are generated by the manufacturing plant, and are immediately installed, but never stored, then the window in which an attacker can gain access to the private key is immensely reduced. But, the process then becomes more synchronous, negating much of the advantage of such a system.

As in the previous case, the transfer may be done via physical interfaces such as bed-of-nails, giving the _broccoli infrastructure-generated_ / _mechanically-transferred_ method.

There is also the possibility of having a _broccoli infrastructure-generated_ / _network-transferred_ key. There is a support for "server-generated" keys in [RFC7030], [RFC8894], and [RFC4210]. All methods strongly recommend encrypting the private key for transfer. This is difficult to comply with here as there is not yet any private key material in the device, so in many cases it will not be possible to encrypt the private key. Still, it may be acceptable if the device is connected directly by a wired network and unroutable addresses are used. A private wired network can often be made as secure as a bed-of-nails interface.

4.1.3. Carrot method: Key setup based on secret seed

In this method, a random symmetric seed is generated by a component manufacturer. This is typically the manufacturer of the CPU, often a system on a chip (SOC). In this section there are two suppliers involved: the first is the familiar one that is responsible for the entire device (the device supplier), and the second one is the vendor of silicon chip. This silicon chip is where a symmetric seed key has been provisioned.

In this process, the silicon chip supplier provisions a unique secret into each device shipped. This is typically at least 256 bits in size, but often is larger. Note that the silicon chip supplier has to create an inventory specific for each client.

This can be via fuses blown in a CPU's Trusted Execution Environment [RFC9397], a TPM, or a Secure Element that is provisioned at its fabrication time.

This seed value is revealed to the board/system manufacturer only via a secure channel. Upon first boot, the system (within a TEE, a TPM, or Secure Element) will generate a key pair using this seed to initialize a Pseudo-Random-Number-Generator (PRNG). The manufacturer, in a separate system, will initialize the same PRNG and, thus generate the same key pair. The manufacturer then derives the public key part, and signs it with their certification authority (CA), which inserts this public key into a certificate. The private part is then destroyed by the manufacturer, ideally never stored or seen by anyone. This same mechanism is used by the TCG for manufacturer provisioning of EK certificates in TPMs, as explained by Section 2.1.1 of [TPMEK].

The certificate (being public information) is placed into a database, in some cases it is loaded by the device as its IDevID certificate, in other cases, it is retrieved during the onboarding process based upon a unique serial number asserted by the device.

In some ways, this method appears to have all the tradeoffs of the previous two methods: the device must correctly derive its own private key, and the manufacturer has access to the private key, making it also vulnerable. But, the device does not depend upon any internal source of random numbers to derive its key.

The manufacturer does all the private key manipulation in a secure place, probably offline, and need never involve the actual physical factory. The manufacturer can even do this in a different country.

The security of the process rests upon the difficulty in extracting the seed provided by the silicon chip supplier. While the silicon chip supplier must operate their factory in a much more secure fashion, which has a much higher cost, the exposure for this facility can be much better controlled. The device manufacturing factory, which has many more components as input, including device testing, can operate at a much lower risk level as a result.

Additionally, there are some other advantages to the device manufacturer: The private keys and certificates may be calculated by the manufacturer asynchronously to the manufacturing process, either done in batches in advance of actual manufacturing, or on demand when an IDevID is requested.

There are, however, additional downsides of this method for manufacturer: the cost of the silicon is often higher, and may involve additional discrete parts. The higher cost is the result of outsourcing the security risk to the silicon manufacturer supplier.

The weakest link in this process is that the resulting seeds must be communicated to the device manufacturer securely, usually in batches. This will often be done by physical courier, but other arrangements are possible. The device manufacturer must store and care for these seeds very carefully, but as soon as a certificate has been generated, the seeds can be destroyed.

4.1.4. Squash method: on-device generation with Secure Element

In this method, a key-pair is generated by the device using a secure element. (It may be a discrete TPM, but when TPM is used to generate keys, that method is considered to fall into the avocado category)

The secure element provides additional assurance that the private key was properly generated. Secure elements are designed specifically so that private keys can not be extracted from the device, so even if the device is attacked in a sophisticated way, using hardware, the private key will not be disclosed.

4.1.5. Spinach method: Secure Element factory generation

In this method, a key-pair is generated by a Secure Element silicon chip supplier in their factory. This method is essentially identical to the carrot method, but it occurs in a different factory!

As a result the choice of which certification authority (CA) gets used may be very different. It is typical for the silicon chip supplier to operate a CA themselves. There are a few options:

1. they may put IDevIDs into the device which are generic to the silicon chip supplier

2. they may operate a CA on behalf of the device manufacturer,

3. they may even connect over a network to the device manufacturer's CA.

The device manufacturer receives the secure element devices in batches in a similar way that they receive other parts. The secure elements are placed by the device manufacturer's plant into the devices.

Upon first boot the device manufacturer firmware can read the IDevID certificate that has been placed into the secure elements, and can ask the secure element to perform signing operations using the private key contained in the secure element. However, the private key can not be extracted!

Despite the increased convenience of this method, there may be a risk if the secure elements are stolen in transport. A thief could use them to generate signatures that would appear to be from device manufacturer devices. To deal with this, there is often a simple activation password that the device manufacturer firmware must provide to the secure element in order to activate it. Note that this password is probably stored in the clear in the device manufacturer's firmware: it can't be encrypted, because the secure source of decryption keys would be in the secure element, creating a cyclic dependency.

Note that the Secure Element silicon vendor has to create an inventory specific for each client.

5. Public Key Infrastructures (PKI)

This section covers the attributes of the Public Key Infrastructure that a manufacturer must operate in order to provision IDevID certificates to devices.

[RFC5280] describes the format for PKIX certificates. Numerous mechanisms of enrollment have been defined (including: EST [RFC7030], CMP [RFC4210], SCEP [RFC8894]).

[RFC5280] provides mechanisms to deal with multi-level certification authorities, but it is not always clear what operating rules apply.

The certification authority (CA) that is central to [RFC5280]-style public key infrastructures can suffer three kinds of failures:

1. disclosure of the private key,

2. loss of access to the private key,

3. inappropriate/unauthorized signing of a certificate or artifact by an unauthorized actor.

A PKI which discloses one or more private certification authority keys is no longer secure.

An attacker can create new identities, and forge certificates connecting existing identities to attacker controlled public/private key pairs. This can permit the attacker to impersonate any specific device.

In case 3, a failure occurs if the CA is convinced to sign (or issue) a certificate which it is not authorized to sign. See for instance [ComodoGate]. This is an authorization failure, and while a significant event, it does not result in the CA having to be re-initialized from scratch as no private keys were disclosed. In ecosystems where CRLs or OCSP is used, then the unauthorized certificate can be revoked.

This is distinguished from when access to the private key is lost, but the key has not been disclosed. Any signatures that have already been made continue to be trustworthy, however, no new signature can be made. This renders the CA useless. If this concerns the firmware signing key, it likely results in a recall of all products that need to use this trust anchor. (Note: that there are some situations where a trust anchor will be created, it will then be used to sign a number of subordinate CAs, enough for the anticipated lifespan of the anchor, and then the private key intentionally destroyed.)

If the PKI uses Certificate Revocation Lists (CRL), then an attacker that has access to the private key can also revoke existing identities.

In the other direction, a PKI which loses access to a private key can no longer function. This does not immediately result in a failure, as existing identities remain valid until their expiry time (notAfter). However, if CRLs or OCSP are in use, then the inability to sign a fresh CRL or OCSP response will result in all identities becoming invalid once the existing CRLs or OCSP statements expire.

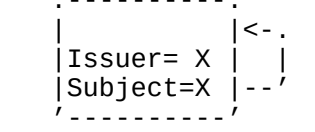
This section details some nomenclature about the structure of certification authorities.

5.1. Number of levels of certification authorities (pkilevel)

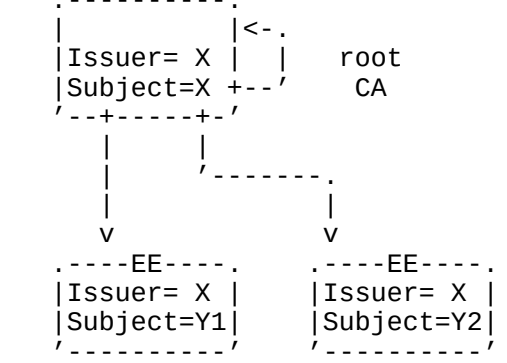
Section 6.1 of [RFC5280] provides a Basic Path Validation. In the formula, the certificates are arranged into a list.

The certification authority (CA) starts with a Trust Anchor. This is counted as the first level of the authority.

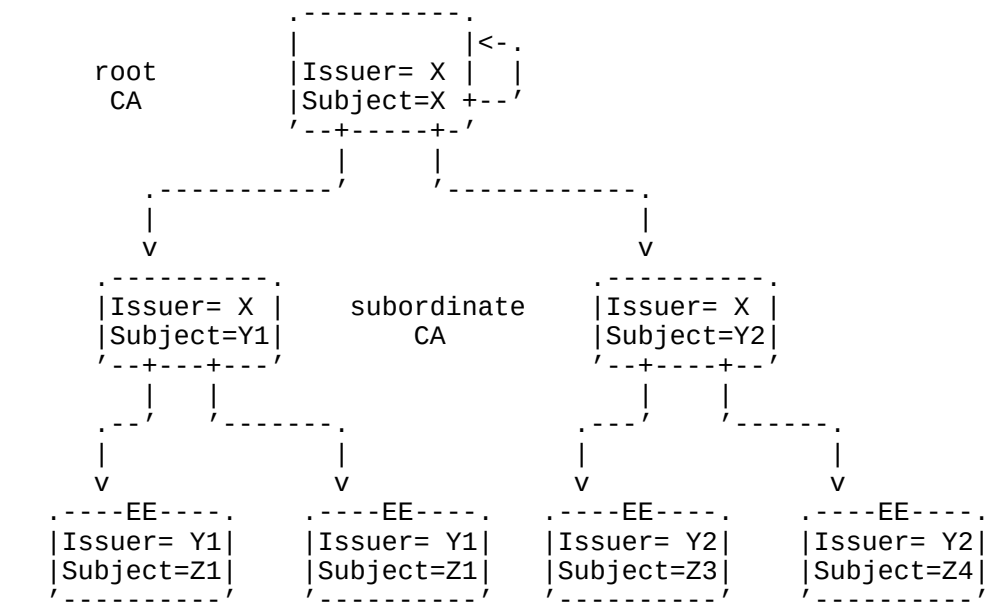
In the degenerate case of a self-signed certificate, this constitutes a one level PKI.



The private key associated with the Trust Anchor signs one or more certificates. When this first level authority signs only End-Entity (EE) certificates, then this is a two level PKI.



When this first level authority signs subordinate certification authorities, and those certification authorities sign End-Entity certificates, then this is a three level PKI.



In general, when arranged as a tree, with the End-Entity certificates at the bottom, and the Trust Anchor at the top, then the pkilevel is defined to be where the deepest EE certificates are, counting from one.

It is quite common to have a three-level PKI, where the root (level one) of the CA is stored in a Hardware Security Module (HSM) in a way that it cannot be continuously accessed ("offline"), while the level two subordinate CA can sign certificates at any time ("online").

5.2. Protection of CA private keys

The private key for the certification authorities must be protected from disclosure. The strongest protection is afforded by keeping them in an offline device, passing Certificate Signing Requests (CSRs) to the offline device by human process.

For examples of extreme measures, see [kskceremony]. There is however a wide spectrum of needs, as exampled in [rootkeyceremony]. The SAS70 audit standard is usually used as a basis for the ceremony, see [keyceremony2].

This is inconvenient, and may involve latencies of days, possibly even weeks to months if the offline device is kept in a locked environment that requires multiple keys to be present.

There is therefore a tension between protection and availability. Convenient and timely access to sign new artifacts is not something that is just nice to have. If access is inconvenient then it may cause delays for signing of new code releases, or it may incentivize technical staff to build in workarounds in order that they can get their job done faster. To address the conflicting needs, some levels of the PKI be offline, and some levels of the PKI be online.

5.3. Preservation of operationally critical private keys

A public key (or certificate) is installed into target device(s) as a trust anchor. It is there in order to verify further artifacts, and it represents a significant investment. Trust anchors must not be easily replaced by attackers, and securing the trust anchor against such tampering may involve burning the trust anchor into unchangeable fuses inside a CPU.

Replacement of the anchor can involve a physical recall of every single device. It is therefore important that the trust anchor is usable for the entire lifetime of every single one of the devices.

The protections discussed in the previous section aim to thwart attacks. The attacker wants to get access to the private key material, or to convince the infrastructure to use the private key material to their bidding. Such an event, if it was undetected, would be catastrophic. It would render almost every device useless (or potentially dangerous) until the anchor could be replaced.

There is a different situation, however, which would lead to a similar result. If the legitimate owner of the trust anchor infrastructure loses access to the private keys, then an equally catastrophic situation occurs.

Code signing is usually done with a keypair that is attached to an End-Entity (EE) Certificate, from a trust anchor that has been assigned code signing authorization. The private key of this EE also needs to be treated with significant care. However, as long as the code signing CA’s trust anchor and private key are preserved, it is usually preferable to lose access to the code signing EE certificate private key rather than risk disclosure. A new code signing EE certificate can be created if needed. This loss is not hassle-free; access to the code signing key can be critical path to getting fixes deployed.

There are many situations that could lead to this. The most typical situation would seem to be some kind of physical damage: a flood, a fire. Less obvious situations could occur if CA key staff lose access to authentication secrets, such as a password, or if the human with the password(s) dies, or becomes incapacitated.

Backups of critical material are routinely done. Storage of backups offsite addresses the risk of physical damage, and in many cases the organization maintains an entire set of equipment at another location.

The question then becomes how the backups are unlocked or activated. Why attack the primary site physically if an attacker can target the backup site, or the people whose job it is to activate the backup site?

Consider the situation where a hurricane or earthquake takes out all power and communications at an organizations' primary location, and it becomes necessary to activate the backup site. What does it take to do that?

Typically, the secrets will be split using [shamir79] into multiple pieces, each piece being carried with a different trusted employee.

In [kskceremony], the pieces are stored on smartcards that are kept in a vault, and the trusted people carry keys to the vault.

One advantage of this mechanism is that if necessary, the doors to the vault can be drilled out. This takes some significant time and leaves significant evidence, so it can not be done quietly by an attacker. In the case of the DNSSEC Root, a failure of the vault to open actually required this to be done.

In other systems the digital pieces are carried on the person themselves, ideally encrypted with a password known only to that person.

[shamir79] allows for keys to be split up into many components (n of them), where only some smaller number of them, k, need to be present to reconstruct the secret. This is known as a (k, n) threshold scheme.

5.3.1. Secret sharing, k-of-n

In this document, each of the people who hold a piece of the secret are referred to as Key-Executives.

The choice of n, and the choice of k is therefore of critical concern. It seems unwise for an organization to publish them, as it provides some evidence as to how many Key-Executives would need to be coerced.

The identities of the n Key-Executives should also be confidential. The list of who they are should probably be limited to the members of the board and executive. There does not seem to be any particular reason for the Key-Executives to be members of the board, but having a long term relationship with the enterprise seems reasonable, and a clear understanding of when to use the piece.

The use of secret sharing, as opposed to providing each Key-Executive a complete backup copy (k=1), is that it means that no individual Key-Executive can perform operations on their own. Instead, a minimum of k Key-Executives is needed to do any operation. This protects the organization against corruption and coercion of any one Key-Executive. As the Key-Executives can not operate on their own, there is also a lower risk to them.

The minimum number of people (k) needed should also remain confidential.

A number that can be published is the difference between k and n, which represents the number of redundant Key-Executives that exist. The publication of this number is an assurance to the organizations' customers that there is a business continuity plan.

An enterprise that has operations in multiple places may be better positioned to survive incidents that disrupt travel. For instance, an earthquake, tsunami, or pandemic not only has the possibility to incapacitate Key-Executives but it could also damage smartcards or USB key that the shares are stored on. [shamir79] suggests that n=2k-1, which implies that a simple majority of Key-Executives are needed to reconstruct the secret.

However, there are other values of k with some different tradeoffs: requiring a majority of Key-Executives might be difficult when travel is impossible. For instance, a value of k set to be less than a simple majority. This enables the situation where Key-Executives are split between two or more continents (with each continent having at least k Key-Executives) would allow either continent to continue operations without the other group.

Secret sharing may also be a good way to manage a code signing or firmware update signing key, even when that is an End-Entity Certificate. Split it among development groups in three time zones (eight hours apart), such that any of those development groups have

enough shares, and can therefore issue an emergency security patch. Another way would be to have three End-Entity certificates that can sign code, and have each time zone sign their own code. That implies that there is at least a level two PKI around the code signing process, and that any bootloaders that need to verify the code being starting it are able to do PKI path operations.

5.4. Supporting provisioned anchors in devices

IDevID-type Identity (or Birth) Certificates which are provisioned into devices need to be signed by a certification authority maintained by the manufacturer. During the period of manufacture of new product, the manufacturer needs to be able to sign new Identity Certificates.

During the anticipated lifespan of the devices the manufacturer needs to maintain the ability for third parties to validate the Identity Certificates. If there are Certificate Revocation Lists (CRLs) involved, then they will need to re-signed during this period. Even for devices with a short active lifetime, the lifespan of the device could be very long if devices are kept in a warehouse for many decades before being activated.

Trust anchors which are provisioned in the devices will have corresponding private keys maintained by the manufacturer. The trust anchors will often anchor a PKI which is going to be used for a particular purpose. There will be End-Entity (EE) certificates of this PKI which will be used to sign particular artifacts (such as software updates), or messages in communications protocols (such as TLS connections). The private keys associated with these EE certificates are not stored in the device, but are maintained by the manufacturer. These need even more care than the private keys stored in the devices, as compromise of the software update key compromises all the devices, not just a single device.

6. Evaluation Questions

This section recaps the set of questions that may need to be answered. This document does not assign valuation to the answers.

6.1. Integrity and Confidentiality of on-device data

initial-enclave-location: Is the location of the initial software trust anchor internal to the CPU package? Some systems have a software verification public key which is built into the CPU package, while other systems store that initial key in a non-volatile device external to the CPU.

initial-enclave-integrity-key: If the first-stage bootloader is external to the CPU, it will often need integrity protection. To verify that, another key is needed. It could be a keyed hash. Where is the key for this hash located?

initial-enclave-confidentiality-key: If the first-stage data is external to the CPU, is it kept confidential by use of encryption?

first-stage-exposure: The number of people involved in the first stage initialization. An entirely automated system would have a number zero. A factory with three 8-hour shifts might have a number that is a multiple of three. A system with humans involved may be subject to bribery attacks, while a system with no humans may be subject to attacks on the system which are hard to notice.

first-second-stage-gap: how far and long does a board travel between being initialized with a first-stage bootloader to where it is locked down so that changes to the bootloader can no longer be made. For many situations, there is no distance at all as they occur in the same factory, but for other situations boards are manufactured and tested in one location, but are initialized elsewhere.

6.2. Integrity and Privacy of device identity infrastructure

For IDevID provisioning, which includes a private key and matching certificate installed into the device, the associated public key infrastructure that anchors this identity must be maintained by the manufacturer.

identity-pki-level: referring to Section 5.1, the level number at which End-Entity certificates are present.

identity-time-limits-per-subordinate: how long is each subordinate CA maintained before a new subordinate CA key is generated? There may be no time limit, only a device count limit.

identity-number-per-subordinate: how many identities are signed by a particular subordinate CA before it is retired? There may be no numeric limit, only a time limit.

identity-anchor-storage: how is the root CA key stored? An open description that might include whether an HSM is used, or not, or even the model of it.

identity-shared-split-extra: referring to Section 5.3.1, where a

private key is split up into n components, of which k are required to recover the key, this number is equal to n-k. This is the number of spare shares. Publishing this provides a measure of how much redundancy is present while not actually revealing either k or n.

identity-shared-split-continents: the number of continents on which the private key can be recovered without trans-continental travel by any of the secret shareholders.

6.3. Integrity and Privacy of included trust anchors

For each trust anchor (public key) stored in the device, there will be an associated PKI. For each of those PKI the following questions need to be answered.

pki-level: how deep is the EE that will be evaluated, based upon the criteria in Section 5.1

pki-algorithms: what kind of algorithms and key sizes can actively be used with the device.

pki-lifespan: what is the timespan for this anchor. Does it get replaced at some interval, and if so, by what means is this done?

pki-level-locked: (a Boolean) the level where the EE cert will be found locked by the device, or can levels be added or deleted by the PKI operator without code changes to the device.

pki-breadth: how many different non-expired EE certificates is the PKI designed to manage?

pki-lock-policy: can any EE certificate be used with this trust anchor to sign? Or, is there some kind of policy OID or Subject restriction? Are specific subordinate CAs needed that lead to the EE?

pki-anchor-storage: how is the private key associated with this trust anchor stored? How many people are needed to recover it?

7. Security Considerations

This entire document is about security considerations.

8. IANA Considerations

This document makes no IANA requests.

9. Acknowledgements

Robert Martin of MITRE provided some guidance about citing the SBOM efforts. Carsten Bormann and Sini Ruohomaa provided many editorial suggestions.

10. Changelog

11. References

11.1. Normative References

[ieee802-1AR]
IEEE Standard, "IEEE 802.1AR Secure Device Identifier", 2018, <https://standards.ieee.org/ieee/802.1AR/6995/>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/rfc/rfc5280>.

11.2. Informative References

[3GPP.33.310]
3GPP and J. Anja, "Network Domain Security (NDS); Authentication Framework (AF)", 30 September 2011, <http://www.3gpp.org/ftp/Specs/archive/33_series/33.310/33310-970.zip>.

[BedOfNails]
Wikipedia, "Bed of nails tester", 30 December 2024, <https://en.wikipedia.org/wiki/Bed_of_nails_tester>.

[CABFORUM] CA/Browser Forum, "CA/Browser Forum Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates, v.1.7.3", October 2020, <https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.7.3.pdf>.

[CABForum90day]
CA/Browser Forum, "CA/Browser Forum Latest Baseline Requirements, version 2.1.5", n.d., <https://cabforum.org/working-groups/server/baseline-requirements/requirements/>.

[cisqsbom] CISQ/Object Management Group, "TOOL-TO-TOOL SOFTWARE BILL OF MATERIALS EXCHANGE", 1 July 2020, <https://www.it-cisq.org/software-bill-of-materials/index.htm>.

[ComodoGate] "Comodo-gate hacker brags about forged certificate exploit", 28 March 2011, <https://www.theregister.com/2011/03/28/comodo_gate_hacker_breaks_cover/>.

[curl] Stenberg, D., "CURL", 28 May 2025, <https://curl.se/download.html>.

[factoringrsa] Bernstein, D. J., Chang, Y.-A., Cheng, C.-M., Chou, L.-P., Heninger, N., Lange, T., and N. van. Someren, "Factoring RSA keys from certified smart cards: Coppersmith in the wild", 16 September 2013, <https://eprint.iacr.org/2013/599>.

[fidotechnote] FIDO Alliance, "FIDO TechNotes: The Truth about Attestation", 19 July 2018, <https://fidoalliance.org/fido-technotes-the-truth-about-attestation/>.

[hyundaiexample] The Register, "Software developer cracks Hyundai car security with Google search", 17 August 2022, <https://www.theregister.com/2022/08/17/software_developer_cracks_hyundai_encryption/>.

[I-D.birkholz-suit-coswid-manifest] Birkholz, H., "A SUIT Manifest Extension for Concise Software Identifiers", Work in Progress, Internet-Draft, draft-birkholz-suit-coswid-manifest-00, 17 July 2018, <https://datatracker.ietf.org/doc/html/draft-birkholz-suit-coswid-manifest-00>.

[I-D.ietf-anima-brski-cloud] Friel, O., Shekh-Yusef, R., and M. Richardson, "Bootstrapping Remote Secure Key Infrastructure (BRSKI) Cloud Registrar", Work in Progress, Internet-Draft, draft-ietf-anima-brski-cloud-19, 9 September 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-brski-cloud-19>.

[I-D.ietf-anima-constrained-voucher]
Richardson, M., Van der Stok, P., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (cBRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-30, 27 February 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-constrained-voucher-30>.

[I-D.ietf-anima-masa-considerations]
Richardson, M., Werner, T., and P. C. Liu, "Operational Considerations for Voucher infrastructure for BRSKI MASA", Work in Progress, Internet-Draft, draft-ietf-anima-masa-considerations-02, 26 March 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-anima-masa-considerations-02>.

[I-D.ietf-iotops-7228bis]
Bormann, C., Ersue, M., Ker辰nen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-ietf-iotops-7228bis-08, 15 May 2026, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-7228bis-08>.

[I-D.ietf-iotops-mud-rats]
Birkholz, H., Richardson, M., and P. C. Liu, "MUD-Based RATS Resources Discovery", Work in Progress, Internet-Draft, draft-ietf-iotops-mud-rats-02, 28 November 2025, <https://datatracker.ietf.org/doc/html/draft-ietf-iotops-mud-rats-02>.

[I-D.moskowitz-ecdsa-pki]
Moskowitz, R., Birkholz, H., Xia, L., and M. Richardson, "Guide for building an ECC pki", Work in Progress, Internet-Draft, draft-moskowitz-ecdsa-pki-10, 31 January 2021, <https://datatracker.ietf.org/doc/html/draft-moskowitz-ecdsa-pki-10>.

[ISO27001] International Organization for Standardization, "ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection -- Information security management systems -- Requirements", 2022.

[JTAG] "Joint Test Action Group", 26 August 2020, <https://en.wikipedia.org/wiki/JTAG>.

[keyceremony2]
Digi-Sign, "SAS 70 Key Ceremony", 4 April 2020, <http://www.digi-sign.com/compliance/key%20ceremony/index>.

[kskceremony]
Verisign, "DNSSEC Practice Statement for the Root Zone ZSK Operator", 18 July 2024, <https://www.iana.org/dnssec/procedures>.

[leakedmsikey]
Binarly efiXplorer Team, "Leaked MSI Source Code with Intel OEM Keys: How Does This Affect Industry-wide Software Supply Chain?", 27 May 2025, <https://www.binarly.io/blog/leaked-msi-source-code-with-intel-oem-keys-how-does-this-affect-industry-wide-software-supply-chain>.

[measuredboot]
Trusted Computing Group, "TCG PC Client Specific Platform Firmware Profile Specification", 4 December 2023, <https://trustedcomputinggroup.org/resource/pc-client-specific-platform-firmware-profile-specification/>.

[ntiasbom] NTIA, "NTIA Software Component Transparency", 1 July 2020, <https://www.ntia.doc.gov/SoftwareTransparency>.

[openbmc] Linux Foundation/OpenBMC Group, "Defining a Standard Baseboard Management Controller Firmware Stack", 1 July 2020, <https://www.openbmc.org/>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <https://www.rfc-editor.org/rfc/rfc4086>.

[RFC4210] Adams, C., Farrell, S., Kause, T., and T. Mononen, "Internet X.509 Public Key Infrastructure Certificate Management Protocol (CMP)", RFC 4210, DOI 10.17487/RFC4210, September 2005, <https://www.rfc-editor.org/rfc/rfc4210>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/rfc/rfc4949>.

[RFC5011] StJohns, M., "Automated Updates of DNS Security (DNSSEC) Trust Anchors", STD 74, RFC 5011, DOI 10.17487/RFC5011, September 2007, <https://www.rfc-editor.org/rfc/rfc5011>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <https://www.rfc-editor.org/rfc/rfc6024>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <https://www.rfc-editor.org/rfc/rfc7030>.

[RFC7168] Nazar, I., "The Hyper Text Coffee Pot Control Protocol for Tea Efflux Appliances (HTCPCP-TEA)", RFC 7168, DOI 10.17487/RFC7168, April 2014, <https://www.rfc-editor.org/rfc/rfc7168>.

[RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <https://www.rfc-editor.org/rfc/rfc8366>.

[RFC8520] Lear, E., Droms, R., and D. Romascanu, "Manufacturer Usage Description Specification", RFC 8520, DOI 10.17487/RFC8520, March 2019, <https://www.rfc-editor.org/rfc/rfc8520>.

[RFC8572] Watsen, K., Farrer, I., and M. Abrahamsson, "Secure Zero Touch Provisioning (SZTP)", RFC 8572, DOI 10.17487/RFC8572, April 2019, <https://www.rfc-editor.org/rfc/rfc8572>.

[RFC8894] Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, DOI 10.17487/RFC8894, September 2020, <https://www.rfc-editor.org/rfc/rfc8894>.

[RFC8995] Pritikin, M., Richardson, M., Eckert, T., Behringer, M., and K. Watsen, "Bootstrapping Remote Secure Key Infrastructure (BRSKI)", RFC 8995, DOI 10.17487/RFC8995, May 2021, <https://www.rfc-editor.org/rfc/rfc8995>.

[RFC9019] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", RFC 9019, DOI 10.17487/RFC9019, April 2021, <https://www.rfc-editor.org/rfc/rfc9019>.

[RFC9140] Aura, T., Sethi, M., and A. Peltonen, "Nimble Out-of-Band Authentication for EAP (EAP-NOOB)", RFC 9140, DOI 10.17487/RFC9140, December 2021, <https://www.rfc-editor.org/rfc/rfc9140>.

[RFC9180] Barnes, R., Bhargavan, K., Lipp, B., and C. Wood, "Hybrid Public Key Encryption", RFC 9180, DOI 10.17487/RFC9180, February 2022, <https://www.rfc-editor.org/rfc/rfc9180>.

[RFC9334] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote ATtestation procedureS (RATS) Architecture", RFC 9334, DOI 10.17487/RFC9334, January 2023, <https://www.rfc-editor.org/rfc/rfc9334>.

[RFC9393] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", RFC 9393, DOI 10.17487/RFC9393, June 2023, <https://www.rfc-editor.org/rfc/rfc9393>.

[RFC9397] Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", RFC 9397, DOI 10.17487/RFC9397, July 2023, <https://www.rfc-editor.org/rfc/rfc9397>.

[RFC9641] Watsen, K., "A YANG Data Model for a Truststore", RFC 9641, DOI 10.17487/RFC9641, October 2024, <https://www.rfc-editor.org/rfc/rfc9641>.

[RFC9711] Lundblade, L., Mandyam, G., O'Donoghue, J., and C. Wallace, "The Entity Attestation Token (EAT)", RFC 9711, DOI 10.17487/RFC9711, April 2025, <https://www.rfc-editor.org/rfc/rfc9711>.

[RFC9733] von Oheimb, D., Ed., Fries, S., and H. Brockhaus, "BRSKI with Alternative Enrollment (BRSKI-AE)", RFC 9733, DOI 10.17487/RFC9733, March 2025, <https://www.rfc-editor.org/rfc/rfc9733>.

[rootkeyceremony] Community, "Root Key Ceremony, Cryptography Wiki", 4 April 2020, <https://cryptography.fandom.com/wiki/Root_Key_Ceremony>.

[shamir79] Shamir, A., "How to share a secret.", 1979, <http://web.mit.edu/6.857/OldStuff/Fall03/ref/Shamir-HowToShareASecret.pdf>.

[TPM20spec]
Trusted Computing Group, "TPM 2.0 Library", March 2025, <https://trustedcomputinggroup.org/resource/tpm-library-specification/>.

[TPMEK] Trusted Computing Group, "TCG Credential Profile EK 2.0", 4 December 2024, <https://trustedcomputinggroup.org/wp-content/uploads/TCG-EK-Credential-Profile-for-TPM-Family-2.0-Level-0-Version-2.6_pub.pdf>.

[_3GPP.51.011]
3GPP and P. L. Thanh, "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface", 15 June 2005, <http://www.3gpp.org/ftp/Specs/archive/51_series/51.011/51011-4f0.zip>.

Author’s Address

Michael Richardson
Sandelman Software Works
Email: mcr+ietf@sandelman.ca