Workgroup: T2TRG Research Group
Internet-Draft:
draft-richardson-t2trg-idevid-
considerations-06
Published: 3 February 2022
Intended Status: Informational
Expires: 7 August 2022
Authors: M. Richardson
Sandelman Software Works

A Taxonomy of operational security considerations for manufacturer installed keys and Trust Anchors

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.

RFCEDITOR: please remove this paragraph. This work is occurring in https://github.com/mcr/idevid-security-considerations

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 7 August 2022.

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. 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. [RFC6024] deals with how trust anchor stores are managed, while this document deals with how the associated PKI which is 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 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]), it now comes as a virtual SIM, but which is usually not provisioned at the factory. The provision of an initial, per-device default password also falls into the category of symmetric shared secret.

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 (see [I-D.richardson-rats-usecases]). The key pair that does this is loaded into large batches of phones for privacy reasons.

The trust anchors are used for a variety of purposes. Trust anchors are used to verify:

* the signature on a software update (as per [I-D.ietf-suit-architecture]),

* 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 [I-D.ietf-emu-eap-noob]. The device identity certificate is also used to sign Evidence by an Attesting Environment (see [I-D.ietf-rats-architecture]).

These security artifacts are used to anchor other chains of information: an EAT 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.birkholz-rats-mud], 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 [I-D.ietf-sacm-coswid] 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 (TrAnc) and identities (IDs) found in typical devices. The privacy and integrity of the TrAncs 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 ISO9001 process such as ISO14001.

1.1. Terminology

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

This section will be expanded to include needed terminology as required.

The words Trust Anchor are contracted to TrAnc rather than TA, in order not to confuse with [I-D.ietf-teep-architecture]'s "Trusted Application".

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

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

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

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

mechanically-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.

2. Applicability Model

There is a wide variety of devices to which this analysis can apply. (See [I-D.bormann-lwig-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 (such as the "Grapeboard"), and can include silicon manufacturer controlled processors in the boot process (the Raspberry PI boots under control of the GPU).

Almost all larger systems (servers, laptops, desktops) include a Baseboard Management Controller (BMC), which ranges from a 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 TrAnc 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 section of [I-D.ietf-teep-architecture], notably TEE, REE, and TAM, and also section 4, Architecture.

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. 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 stage one 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 be 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 (TrAnc) 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.

When evaluating the degree of trust for each trust anchor there are four aspects that need to be determined:

*can the trust anchor be replaced or modified?

*can additional trust anchors be added?

*can trust anchors be removed?

*how is the private key associated with the trust anchor, maintained by the manufacturer, maintained?

The first three things are device specific properties of how the integrity of the trust anchor is maintained.

The fourth property has nothing to do with the device, but has to do with the reputation and care of the entity that maintains the private key.

Different anchors have different authorizations associated with them.

These are:

3.1. Secured First Boot Trust Anchor

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

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 Secure Boot Trust Anchor.

Second, it may be stored in the second-stage bootloader, and therefore its integrity is protected by the Secured First Boot 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 in a secured boot situation, as the second-stage bootloader does not validate the application/operating system before starting it, but it may still provide measured boot mechanism.

3.3. Trusted Application Manager anchor

This anchor is the secure key for the [I-D.ietf-teep-architecture] 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. This privilege may include updating anchors.

3.4. Public WebPKI anchors

These anchors are used to verify HTTPS certificates from web sites. These anchors are typically distributed as part of desktop browsers, and via desktop operating systems.

The exact set of these anchors is not precisely defined: it is usually determined by the browser vendor (e.g., Mozilla, Google, Apple, Safari, 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 securing 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 some period of years. That upon being reconnected, that the device would be unable to do DNSSEC validation. This failure would result in them being unable to obtain operating system updates that would then include the updates to the DNSSEC key.

3.6. What else?

TBD?

4. Types of Identities

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 at the least, 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 expert operator. A number of derived protocols such as [I-D.ietf-anima-brski-async-enroll], [I-D.ietf-anima-constrained-voucher], [I-D.richardson-anima-voucher-delegation], [I-D.friel-anima-brski-cloud] extend this in a number of ways.

* [I-D.ietf-rats-architecture] depends upon a key provisioned into the Attesting Environment to sign Evidence.

* [I-D.ietf-suit-architecture] may depend 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 ECIES can solve this. There may be other legal considerations why the IDevID might not be used, and a second key provisioned.

* TBD

4.1.1. Operational Considerations for Manufacturer IDevID Public Key Infrastructure

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 document will overview.

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 is a fourth situation where the IDevID is provided as part of a Trusted Platform Module (TPM), in which case the TPM vendor may be making the same tradeoffs.

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.

4.1.2. Key Generation process

4.1.2.1. On-device private key generation

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 verified random number generator. [factoringrsa] is an example of a successful attack on this scenario.

There are a number of options of how to get the public key securely from the device to the certification authority.

This transmission must be done in an integral manner, and must be securely associated with the assigned serial number. The serial number goes into the certificate, and the resulting certificate needs to be loaded into the manufacturer's asset database.

One way to do the transmission is during a factory Bed of Nails test (see [BedOfNails]) or Boundary Scan. When done via a physical connection like this, then this is referred to as a *device-generated / mechanically-transferred* method.

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 *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.

After the key generation, the device needs to set a flag such that it no longer will generate a new key / will accept a new IDevID via the factory connection. This may be a software setting, or could be as dramatic as blowing a fuse.

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, unless the network initialization also permits an attacker to load or replace trust anchors at the same time.

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.

4.1.2.2. Off-device private key generation

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

If the device does not come with a serial number in silicon, then one should be assigned and placed into a certificate. The private key and certificate could be programmed into the device along with the initial bootloader firmware in a single step.

Aside from the change of origin for the randomness, a major advantage of this mechanism is that it can be done with a single write to the flash. The entire firmware of the device, including configuration of trust anchors and private keys can be loaded in a single write pass. Given some pipelining of the generation of the keys and the creation of certificates, it may be possible to install unique identities without taking any additional time.

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.

As in the previous case, the transfer may be done via physical interfaces such as bed-of-nails, giving the *infrastructure-generated / mechanically-transferred* method.

There is also the possibility of having a *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.

4.1.2.3. Key setup based on 256 bit secret seed

A hybrid of the previous two methods leverages a symmetric key that is often provided by a silicon vendor to OEM manufacturers.

Each CPU (or a Trusted Execution Environment [I-D.ietf-teep-architecture], or a TPM) is provisioned at fabrication time with a unique, secret seed, usually at least 256 bits in size.

This value is revealed to the OEM board manufacturer only via a secure channel. Upon first boot, the system (probably within a TEE, or within a TPM) will generate a key pair using the seed to initialize a Pseudo-Random-Number-Generator (PRNG). The OEM, in a separate system, will initialize the same PRNG and generate the same key pair. The OEM then derives the public key part, signs it and turns it into a certificate. The private part is then destroyed, ideally never stored or seen by anyone. 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.

This method appears to have all of the downsides of the previous two methods: the device must correctly derive its own private key, and the OEM has access to the private key, making it also vulnerable. The secret seed must be created in a secure way and it must also be communicated securely.

There are some advantages to the OEM however: the major one is that the problem of securely communicating with the device is outsourced to the silicon vendor. The private keys and certificates may be calculated by the OEM asynchronously to the manufacturing process, either done in batches in advance of actual manufacturing, or on demand when an IDevID is demanded. Doing the processing in this way

permits the key derivation system to be completely disconnected from any network, and requires placing very little trust in the system assembly factory. Operational security such as often incorrectly presented fictionalized stories of a "mainframe" system to which only physical access is permitted begins to become realistic. That trust has been replaced with a heightened trust placed in the silicon (integrated circuit) fabrication facility.

The downsides of this method to the OEM are: they must be supplied by a trusted silicon fabrication system, which must communicate the set of secrets seeds to the OEM in batches, and they OEM must store and care for these keys very carefully. There are some operational advantages to keeping the secret seeds around in some form, as the same secret seed could be used for other things. There are some significant downsides to keeping that secret seed around.

5. Public Key Infrastructures (PKI)

[RFC5280] describes the format for certificates, and numerous mechanisms for doing 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 a private key,

2. loss of a private key,

3. inappropriate signing of a certificate from an unauthorized source.

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 keypairs. This can permit the attacker to impersonate any specific device.

There is an additional kind of failure when the CA is convinced to sign (or issue) a certificate which it is not authorized to do so. 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.

This is distinguished from when a loss as described above renders the CA completely useless and likely requires a recall of all products that have ever had an IDevID issued from this CA.

If the PKI uses Certificate Revocation Lists (CRL)s, 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.

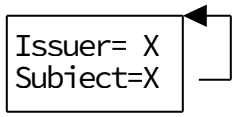
This section details some nomenclature about the structure of certification authorities.

5.1. Number of levels of certification authorities

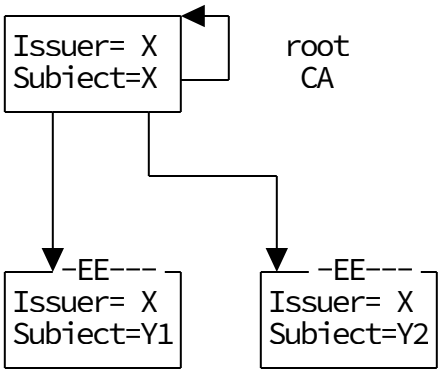
Section 6.1 of [RFC5280] provides a Basic Path Validation. In the formula, the certificates are arranged into a list.

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

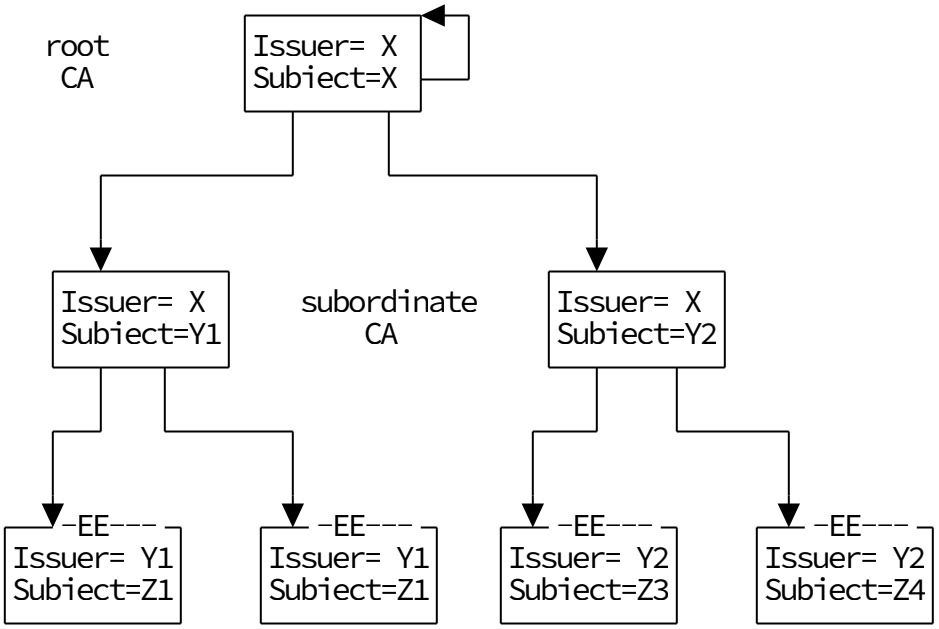
In the degenerate case of a self-signed certificate, then this a one level PKI.



The private key associated with the Trust Anchor signs one or more certificates. When this first level authority trusts 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 level is where the deepest EE certificates are, counting from one.

It is quite common to have a three-level PKI, where the root of the CA is stored in a Hardware Security Module, while the level one subordinate CA is available in an online form.

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 a 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 convenience. This is often mitigated by having some levels of the PKI be offline, and some levels of the PKI be online.

There is usually a need to maintain backup copies of the critical keys. It is often appropriate to use secret splitting technology such as Shamir Secret Sharing among a number of parties [shamir79] This mechanism can be setup such that some threshold k (less than the total n) of shares are needed in order to recover the secret.

5.3. 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 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 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 of 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 Privacy 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, and if it is integrity protected, where is the key used to check the integrity?

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

first-stage-initialization: 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: If a board is initialized with a first-stage bootloader in one location (factory), and then shipped to another location, there may situations where the device can not be locked down until the second step.

6.2. Integrity and Privacy of device identify 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: how deep are the IDevID certificates that are issued?

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? How many people are needed to recover the private key?

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 (the trust root is at level 1)

pki-algorithms: what kind of algorithms and key sizes will be considered to valid

pki-level-locked: (a Boolean) is 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 root stored? How many people are needed to recover it?

7. Privacy Considerations

many yet to be detailed

8. Security Considerations

This entire document is about security considerations.

9. IANA Considerations

This document makes no IANA requests.

10. Acknowledgements

Robert Martin of MITRE provided some guidance about citing the SBOM efforts. Carsten Borman provides many editorial suggestions.

11. Changelog

12. References

12.1. Normative References

[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/info/rfc5280>.

[ieee802-1AR] IEEE Standard, "IEEE 802.1AR Secure Device Identifier", 2009, <http://standards.ieee.org/findstds/standard/802.1AR-2009.html>.

12.2. Informative References

[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/info/rfc8995>.

[I-D.richardson-anima-voucher-delegation] Richardson, M. and W. Pan, "Delegated Authority for Bootstrap Voucher Artifacts", Work in Progress, Internet-Draft, draft-richardson-anima-voucher-delegation-03, 22 March 2021, <https://www.ietf.org/archive/id/draft-richardson-anima-voucher-delegation-03.txt>.

[I-D.friel-anima-brski-cloud] Friel, O., Shekh-Yusef, R., and M. Richardson, "BRSKI Cloud Registrar", Work in Progress, Internet-Draft, draft-friel-anima-brski-cloud-04, 6 April 2021, <https://www.ietf.org/archive/id/draft-friel-anima-brski-cloud-04.txt>.

[I-D.ietf-anima-constrained-voucher] Richardson, M., Stok, P. V. D., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-15, 7 December 2021, <https://www.ietf.org/archive/id/draft-ietf-anima-constrained-voucher-15.txt>.

[I-D.ietf-anima-brski-async-enroll] Fries, S., Brockhaus, H., Oheimb, D. V., and E. Lear, "Support of Asynchronous

Enrollment in BRSKI (BRSKI-AE)", Work in Progress, Internet-Draft, draft-ietf-anima-brski-async-enroll-04, 25 October 2021, <https://www.ietf.org/archive/id/draft-ietf-anima-brski-async-enroll-04.txt>.

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

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/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/info/rfc5011>.

[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/info/rfc8366>.

[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/info/rfc8572>.

[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/info/rfc7030>.

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

[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/info/rfc4210>.

[_3GPP.51.011] 3GPP, "Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface", 3GPP TS 51.011 4.15.0, 15 June 2005, <http://www.3gpp.org/ftp/Specs/html-info/51011.htm>.

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

[BedOfNails] Wikipedia, "Bed of nails tester", 1 July 2020, <https://en.wikipedia.org/wiki/In-circuit_test#Bed_of_nails_tester>.

[pelionfcu] ARM Pelion, "Factory provisioning overview", 28 June 2020, <https://www.pelion.com/docs/device-management-provision/1.2/introduction/index.html>.

[factoringrsa] "Factoring RSA keys from certified smart cards: Coppersmith in the wild", 16 September 2013, <https://core.ac.uk/download/pdf/204886987.pdf>.

[kskceremony] Verisign, "DNSSEC Practice Statement for the Root Zone ZSK Operator", 2017, <https://www.iana.org/dnssec/dps/zsk-operator/dps-zsk-operator-v2.0.pdf>.

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

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

[shamir79] Shamir, A., "How to share a secret.", 1979, <https://www.cs.jhu.edu/~sdoshi/crypto/papers/shamirturing.pdf>.

[nistsp800-57] NIST, "SP 800-57 Part 1 Rev. 4 Recommendation for Key Management, Part 1: General", 1 January 2016, <https://

csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-4/final>.

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

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

[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/>.

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

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

[JTAGieee] IEEE Standard, "1149.7-2009 - IEEE Standard for Reduced-Pin and Enhanced-Functionality Test Access Port and Boundary-Scan Architecture", DOI 10.1109/IEEESTD.2010.5412866, 2009, <https://ieeexplore.ieee.org/document/5412866>.

[rootkeyrollover] ICANN, "Proposal for Future Root Zone KSK Rollovers", 2019, <https://www.icann.org/en/system/files/files/proposal-future-rz-ksk-rollovers-01nov19-en.pdf>.

[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>.

[I-D.richardson-rats-usecases] Richardson, M., Wallace, C., and W. Pan, "Use cases for Remote Attestation common encodings", Work in Progress, Internet-Draft, draft-richardson-rats-usecases-08, 2 November 2020, <https://www.ietf.org/archive/id/draft-richardson-rats-usecases-08.txt>.

[I-D.ietf-suit-architecture] Moran, B., Tschofenig, H., Brown, D., and M. Meriac, "A Firmware Update Architecture for Internet of Things", Work in Progress, Internet-Draft,

draft-ietf-suit-architecture-16, 27 January 2021, <https://www.ietf.org/archive/id/draft-ietf-suit-architecture-16.txt>.

[I-D.ietf-emu-eap-noob] Aura, T., Sethi, M., and A. Peltonen, "Nimble Out-of-Band Authentication for EAP (EAP-NOOB)", Work in Progress, Internet-Draft, draft-ietf-emu-eap-noob-06, 3 September 2021, <https://www.ietf.org/archive/id/draft-ietf-emu-eap-noob-06.txt>.

[I-D.ietf-rats-architecture] Birkholz, H., Thaler, D., Richardson, M., Smith, N., and W. Pan, "Remote Attestation Procedures Architecture", Work in Progress, Internet-Draft, draft-ietf-rats-architecture-14, 9 December 2021, <https://www.ietf.org/archive/id/draft-ietf-rats-architecture-14.txt>.

[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://www.ietf.org/archive/id/draft-birkholz-suit-coswid-manifest-00.txt>.

[I-D.birkholz-rats-mud] Birkholz, H., "MUD-Based RATS Resources Discovery", Work in Progress, Internet-Draft, draft-birkholz-rats-mud-00, 9 March 2020, <https://www.ietf.org/archive/id/draft-birkholz-rats-mud-00.txt>.

[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/info/rfc8520>.

[I-D.ietf-sacm-coswid] Birkholz, H., Fitzgerald-McKay, J., Schmidt, C., and D. Waltermire, "Concise Software Identification Tags", Work in Progress, Internet-Draft, draft-ietf-sacm-coswid-20, 26 January 2022, <https://www.ietf.org/archive/id/draft-ietf-sacm-coswid-20.txt>.

[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/info/rfc7168>.

[I-D.bormann-lwig-7228bis] Bormann, C., Ersue, M., Keranen, A., and C. Gomez, "Terminology for Constrained-Node Networks", Work in Progress, Internet-Draft, draft-bormann-lwig-7228bis-07, 25 October 2021, <https://www.ietf.org/archive/id/draft-bormann-lwig-7228bis-07.txt>.

[I-D.ietf-teep-architecture]
Pei, M., Tschofenig, H., Thaler, D., and D. Wheeler, "Trusted Execution Environment Provisioning (TEEP) Architecture", Work in Progress, Internet-Draft, draft-ietf-teep-architecture-15, 12 July 2021, <https://www.ietf.org/archive/id/draft-ietf-teep-architecture-15.txt>.

Author's Address

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca