Device Identity with DICE and RIoT:

Keys and Certificates

*(Draft for Review)*

*Ronald Aigner, Paul England, Kevin Kane, Andrey Marochko, Dennis Mattoon,*

*Rob Spiger, Stefan Thom, and Greg Zaverucha*

*Microsoft Corporation*

Abstract

This specification describes a cryptographic device identity and attestation scheme based on the TLS protocol and X.509 client certificates. The protocol and certificate formats can be implemented by any type of security processor, but are well suited to DICE+RIoT security architectures. Devices without hardware-based security can also implement the protocol in software, although the resultant identity and attestations will be of lower assurance.

Contents

[2 Terms and Definitions 2](#_Toc495921986)

[3 Overview 4](#_Toc495921987)

[3.1 Protocol Overview 4](#_Toc495921988)

[3.2 DeviceID and FWID Encodings 5](#_Toc495921989)

[3.3 TLS Handshake and Certificate Validation 5](#_Toc495921990)

[3.4 Attestation 6](#_Toc495921991)

[3.5 Bugs and Patching 6](#_Toc495921992)

[3.6 Example DICE+RIoT Software Architecture 7](#_Toc495921993)

[3.7 RIoT Core Operations 8](#_Toc495921994)

[3.8 Layer 1 Behavior 9](#_Toc495921995)

[3.9 Extending to Multiple Layers 9](#_Toc495921996)

[3.10 RIoT Implementation Notes 10](#_Toc495921997)

[3.11 Alternative Implementations 11](#_Toc495921998)

[4 X.509 Certificate Profile Normative Requirements 11](#_Toc495921999)

[4.1 Composite Identity Extension Encoding 11](#_Toc495922000)

[4.2 Serial Number Generation 12](#_Toc495922001)

[4.3 Certificate Lifetime 13](#_Toc495922002)

[4.4 Subject Name and Issuer Name 13](#_Toc495922003)

[4.5 Alias Key Certificate 13](#_Toc495922004)

[4.6 Non-Vendor Certified DeviceID Certificate 13](#_Toc495922005)

[4.7 Vendor Certified DeviceID Certificates 14](#_Toc495922006)

[4.8 DeviceID Certificate Signing Requests 14](#_Toc495922007)

[4.9 Recommended Algorithms for Interoperability 14](#_Toc495922008)

[Appendix A – Privacy Considerations 14](#_Toc495922009)

[References 14](#_Toc495922010)

# Terms and Definitions

This section provides definitions for terms used in this specification.

**Alias Key, Credential, and Certificate**Alias Keys are asymmetric key pairs created by a device; new Alias Keys are created for each new firmware revision (and perhaps more frequently.) The public part of an Alias Key pair is certified by the device’s DeviceID key to form an Alias Key Certificate. The key and certificate together form the Alias Credential.

**Attestation**Attestation is cryptographic reporting of the security configuration of a device. In this specification, attestation information is encoded in a cryptographic hash value called a Firmware ID (FWID).

**DeviceID Key, Credential, and Certificate**The Device Identity (DeviceID) is an asymmetric key pair that serves as a long-term identifier for a device. This specification assumes that a DeviceID never changes. If a vendor provides a mechanism to change a DeviceID, either remotely using cryptographic protocols or during remanufacturing, then the protocols in this specification will identify it as a different device.

If the DeviceID key pair is created using DICE+RIoT mechanisms, then the DeviceID will reflect the combined hardware identity of the device and the RIoT Core component. RIoT Core is assumed to be invariant, so the DeviceID will not change.

DeviceID public keys may be self-certified or certified by the device vendor to form DeviceID Certificates.

**Device Identity Composition Engine (DICE)**DICE, or Device Identity Composition Engine, is a hardware/firmware capability that generates a cryptographically unique value, called the Compound Device Identity, for the lowest-layer of software that boots on the device [[[1]](#endnote-2)].

DICE-compliant platforms support RIoT-style security architectures [2].

**Composite Identity**

The Composite Identity is a data structure that encodes the hardware identity of the device (the DeviceID) as well as the firmware that it is running (the FWID.)

**Compound Device Identity (CDI)**The Compound Device Identity is a secret value that is unique to a device and the cryptographic identity (e.g. the hash) of the RIoT Core layer that it booted.

**Firmware ID (FWID)**The Firmware Identity (FWID) is the hash of the Firmware Security Descriptor (FSD).

**Firmware Security Descriptor (FSD)**The Firmware Security Descriptor is a vendor-defined machine-readable data structure that describes the identity of the Layer 1 code (and perhaps other security-relevant data.) RIoT Core uses the FSD to validate the Layer 1 code that it loads.

Examples of Firmware Security Descriptors include:

1. The Layer 1 firmware image itself,
2. A manifest that cryptographically identifies authorized Layer 1 firmware,
3. A firmware version number for signed firmware in the context of a secure boot implementation,
4. Security-critical configuration settings for the device

This specification neither defines what state is security critical, nor how it is represented in the Firmware Security Descriptor. However, the FSD should be machine-readable, and there should be a canonical representation that can be hashed to form the Firmware ID or FWID.

For DICE+RIoT architectures, and in the context of this specification, the RIoT Core layer is assumed to be unchanging. Hence, the Firmware Security Descriptor does *not* include the identity of the RIoT Core layer. (Future specifications may allow for secure update of the RIoT Core layer.)

**Layer 0, 1, 2…**

Many computer systems boot in stages, with each layer authenticating and loading the next layer and providing increasingly sophisticated runtime services.

In this specification, Layer zero is the first code started by the processor. In DICE+RIoT systems, Layer zero is also called RIoT Core, and provides foundational security services. Simple devices may package all functionality in Layer 1. More complex devices may include more layers.

**Name-Encoded DeviceID**The Name Encoded DeviceID is an optional string encoding of the hash of the DeviceID public key in the *Issuer Name* or *Subject Name* fields of an X.509 certificate.

**RIoT**

RIoT (Robust Internet of Things) is an architecture for providing foundational trust services to computing devices [[[2]](#endnote-3)].

**RIoT Core**

The name RIoT Core is given to the code that is measured by the processor to create the Compound Device Identity (CDI). In turn, RIoT Core uses the CDI to create the DeviceID and other keys. While this document is primarily concerned with Device Identity, a full featured RIoT implementation may also provide features for data-at-rest protection (sealing), secure firmware updates, and other services. RIoT Core implementations are generally device/vendor specific.

This specification assumes that RIoT Core never changes.

**Security Processor**Dedicated hardware for protecting cryptographic keys and operations.

# Overview

The following sections provide an overview of the protocol and certificate formats defined in this specification. The protocols in this section can be implemented by software or a hardware security processor. Later sections describe how the keys and certificates can be created in RIoT-based systems.

## Protocol Overview

Devices authenticate themselves and establish secure communications using a Transport Level Security (TLS) session with client certificates [[[3]](#endnote-4)]. This specification defines X.509 client-certificate profiles and certificate chaining rules that allow devices to authenticate themselves and their security configuration (i.e., their identity and, for example, the firmware they are currently running) [[[4]](#endnote-5)].

All conforming devices are provisioned with, or can generate, an asymmetric key pair called the DeviceID. This specification assumes that the DeviceID remains the same for the life of the device. Vendors may allow the DeviceID to change (for example during remanufacture, or using other cryptographic protocols) but relying parties that use the protocol in this specification will then identify it as a different device.

Devices also generate, or are provisioned with, a second family of keys called Alias Keys. In contrast to DeviceID credentials, which do not change, new Alias Keys and credentials are created more often - *e.g.* when device firmware is updated or if the owner changes.

Alias Keys and (optionally) the DeviceID Keys are certified with X.509-format certificates. The primary function of the DeviceID key is to establish device identity and certify Alias Keys. Fields in the generated Alias Key certificate can also specify the Firmware ID (e.g. the patch level or the hash of the device firmware) to support attestation.

DeviceID public keys themselves can be certified by the vendor or, if no vendor certification is desired or required, they may be self-certified, or remain uncertified. In the former case, the DeviceID acts as a subordinate CA, and in the latter case, the DeviceID is a root CA. Devices may support more than one option to serve different relying parties. See Figure 1.



Figure 1: Certificate chaining for vendor-certified, self-certified, and uncertified (bare) DeviceID keys.

Devices authenticate themselves using the current Alias Key and Certificate as client certificates in a TLS session. Depending on the authentication scenario, devices may also send the Vendor-Certified DeviceID certificate chain or the self-certified DeviceID certificate during the TLS handshake. If the TLS handshake succeeds, the server can determine the public DeviceID from the DeviceID certificate and the Firmware ID from the Alias Certificate (additional considerations for validating the certificate chain are presented later in this document.)

Devices should also validate server TLS certificates, but this is outside the scope of this specification.

## DeviceID and FWID Encodings

This specification defines a *Composite Identity* data structure that contains the public DeviceID (*i.e.* the hardware identity) as well as the FWID (*i.e.* the software identity.) The Composite Identity is included as an extension field of the Alias Certificate.

## TLS Handshake and Certificate Validation

Devices implementing this specification authenticate themselves (their DeviceID) and their security-critical configuration (their FWID) though the TLS handshake. Servers wishing to authenticate clients must request client certificates. Conforming devices will respond with either:

1. The certificate chain rooted in the vendor CA,
2. The certificate chain rooted in the DeviceID, or
3. The bare Alias Certificate

See figure 1.

Servers should also provide server certificates during the TLS handshake. A device may terminate the connection if the server certificate chain is not valid or is not trusted.

It is outside the scope of this specification to define the policies that relying parties should use to authenticate and authorize devices. However, the following considerations may be relevant:

1. **All Certificate Chain Processing[[5]](#footnote-2)**
   1. Normal chain processing rules defined in RFC 5280 [3] should be implemented. For example, all certificates should:
      1. Be properly signed, and be within their validity date-range
      2. Have matching *Issuer Name* and *Subject Name* fields through the chain
      3. Have their revocation status checked
      4. Check pathLengthConstraint rules are not violated
      5. Check that cA is asserted for all certificates apart from the Alias Certificate
   2. The DeviceID encoded in the DeviceID certificate should match the DeviceID claimed in the RIoT extension of the Alias Certificate
2. **Non-Vendor-Certified DeviceIDs**
   1. The DeviceID Certificate should be self-signed, and the DeviceID should be recognized.
3. **Vendor-Certified DeviceIDs**
   1. The DeviceID certificate should chain back to the vendor CA.
4. **Bare Alias Certificates**
   1. The Alias Certificate signature must be signed by the DeviceID encoded in The Composite Identity

If these checks are successful, the DeviceID and FWID can be extracted from the Composite Identity extension of the Alias Certificate.

## Attestation

Attestation, in the context of this specification, is cryptographic reporting of the security configuration of a device via its Firmware ID. The FWID is defined to be the hash of a machine-readable data structure, called the Firmware Security Descriptor, that captures all security-critical information that the vendor deems relevant.

The format of the Firmware Security Descriptor is not specified here: In the simplest case, the Firmware Security Descriptor is simply the Layer 1 firmware on the device. In that case, the FWID will be the hash of the updatable firmware image. More complex examples of Firmware Security Descriptors include manifests that identify authorized firmware, version numbers included in digital certificates used to authenticate firmware, and Platform Configuration Register-style secure logs [[[6]](#endnote-6)].

Relying party determination of valid FWIDs is outside the scope of this specification.

## Bugs and Patching

This specification defines:

1. The DeviceID Credential, which does not change for the life of the device
2. The Alias Credential, which changes more frequently (at least as often as firmware is updated.)

Devices implementing this specification should employ the very highest levels of protection for the private DeviceID key. Ideally, this will involve DICE+RIoT mechanisms or other hardware-based protection. This specification does *not* provide any means to remediate devices if the DeviceID private key is exposed or used to issue malicious certificates (future specifications may address this capability.)

It is assumed that Alias Private Keys are less well protected: for example, Alias Private Keys may be managed by device firmware rather than protected by hardware. If the device Firmware has an exploitable bug, then the Alias private keys may be revealed or misused. In either case, security can only be re-established when device firmware is updated. In DICE+RIoT systems, updating firmware naturally results in the creation of a new Alias Key and Credential. If other hardware security modules are used, then a similar re-keying process should be adopted.

No existing attestation scheme can provide authoritative reporting of the security configuration of a device in the face of zero-day privilege escalation bugs or unknown firmware vulnerabilities. This is because, if firmware is exploitable, there is no way to know what code is *actually* running. However, the attestation scheme described in this specification *can* prove that the device is running the most recent firmware version, as opposed to old or unknown software. This means that attestation is *at least* useful to force updates of old firmware because old firmware will be attested as such and can never impersonate newer firmware (at least until more bugs are found and exploited).

## Example DICE+RIoT Software Architecture

This section describes an implementation of a key and certificate generation procedure using DICE+RIoT mechanisms.

DICE, or Device Identity Composition Engine, is standardized processor or platform functionality that provides software with a secret, called the *Compound Device Identity* (CDI), that depends upon the hardware device, and the identity of the booting software. RIoT (Robust IoT) is family of techniques for using a CDI as the basis for device identity, protected storage, and other security capabilities [1,2].

Figure 2 illustrates an architecture in which a DICE-enabled processor runs a first-stage bootloader called *RIoT Core,* which in turn boots the remaining device firmware called *Layer 1*. DICE functionality in the processor creates a secret value called the Compound Device Identity (CDI) and passes it to RIoT Core. The CDI is typically a 32-byte value produced using a cryptographic one-way function based on a per-device secret and the hash of RIoT Core. RIoT Core uses the CDI to derive additional keys and secrets, as described in this section, and then deletes the CDI from RAM and registers. This specification assumes that RIoT Core never changes, so the CDI remains the same for the life of the device.

The identity of the code and security-critical configuration data comprising Layer 1 is represented by its FWID. The RIoT Core layer is responsible for ensuring that Layer 1 is compliant with the Firmware Security Descriptor (FDS) that determines the FWID.

In addition to the Alias Key certificate generation, the RIoT core layer an optionally generate a self-signed DeviceID certificate and a self-signed Certificate Signing Request (CSR) for the DeviceID public key.

The following section describes one way that the RIoT Core layer can produce keys and certificates that are compliant with this specification.

Figure 2: DICE + RIoT architecture for device Identity (non-vendor-certified).

DevID  
CSR

DeviceID CSR Generation

Processor / ROM (DICE)

Per-Device Secret

CPU

Alias Key Certificate Generation

DeviceID Certificate Generation

DeviceIDprivate

DeviceIDpublic

Aliaspublic

Aliasprivate

CDI

Alias Key Deterministic Asymmetric Key Generation

Updatable Device Firmware  
(or Second-Stage Boot Loader)

RIoT  
Core

Layer 1

DevID  
Cert

Alias  
Cert

Firmware Security Descriptor (FSD)

DeviceID  
Deterministic Asymmetric Key Generation

## RIoT Core Operations

The RIoT Core performs the following operations on each boot:

1. **FWID Calculation**
   1. RIoT Core establishes the FWID by direct measurement (e.g., hash of the image) or by indirect measurement (e.g., certificate and/or manifest validation via the Firmware Security Descriptor or FSD).
   2. If the FSD is invalid or missing (or if other unexpected circumstances occur), then further key generation is aborted, and Cleanup is performed (step 7.)

(The definition and representation of the FSD is outside the scope of this specification.)

1. **DeviceID Creation**
   1. RIoT Core creates an asymmetric DeviceID key pair using a deterministic key generation function seeded with the CDI. In this specification, RIoT Core is assumed not to change so the same DeviceID key pair is created on each boot.
   2. The public DeviceID key is exported to Layer 1.
   3. The private DeviceID key is used in step 4(b) and 5(b), below.
2. **Alias Key Creation**
   1. RIoT Core creates the Alias Key pair using a deterministic key generation function seeded with the CDI (secret) and the FWID (public).
   2. Both public and private parts of the Alias Key are exported to Layer 1.
   3. The public part of the Alias Key is certified in step 4(b) below.
3. **Alias Key Certificate Generation**
   1. RIoT Core creates a certificate template structure for the Alias Key containing the encoded Composite Identity as an extension.
   2. RIoT Core signs the certificate template with the private DeviceID key to form the Alias Certificate.
   3. The Alias Certificate is exported to Layer 1.
4. **DeviceID Self-Signed Certificate Generation (Optional)**
   1. RIoT Core creates a certificate structure containing the DeviceID public key.
   2. RIoT Core signs the certificate using the DeviceID private key.
5. **DeviceID-signed Certificate Signing Request (Optional)**
   1. RIoT Core creates a CSR structure (e.g. a PKCS10 request) for the DeviceID public key, signed by the DeviceID private key.
6. **Cleanup**
   1. All RAM and register state that is not explicitly exported in the steps above is deleted. This will include the CDI and private DeviceID key.

## Layer 1 Behavior

RIoT Core provides Layer 1 with the following data:

1. The DeviceID public key and certificate
2. Both public and private parts of the Alias Key pair
3. The Alias Key certificate signed by the DeviceID private key
4. The Firmware Security Descriptor (either provided explicitly, or available by other means.)

Layer 1 may load additional modules, assuming they are authorized by the Firmware Security Descriptor. Eventually, Layer 1 (or code loaded by Layer 1) can use the keys and certificates provided by RIoT Core to authenticate itself using TLS.

Layer 1 Firmware should ensure that:

1. It only uses the Alias Key issued to it on this boot. I.e. Level 1 Firmware should *not* use an Alias Credential issued to an earlier firmware version
2. The private portion of the Alias Key is not exposed to external/less-trusted entities
3. The private portion of the Alias Key is not exposed to earlier firmware versions

If these conditions are met, relying parties using the protocols described in this specification can authenticate both the device and its current security configuration.

## Extending to Multiple Layers

Both the protocol and RIoT-implementation extend straightforwardly to multi-layer boot, as well as authentication of multiple programs running on the same host: see Figure 3.



Figure 3: Extending the certificate chain to multiple layers.

For a single RIoT Core layer running monolithic Device Firmware, RIoT core produces an Alias Certificate and key pair for the firmware to use in a TLS session.

If more layers are implemented, then intermediate layers will perform similar functions to the RIoT Core layer. Each additional layer will create a new Alias Key for the next layer, together with an Alias Certificate that encodes the FWID of the layer above. The Alias Certificates issued to the layer above is signed with the Alias Key granted to the underlying layer (rather than the DeviceID key.) In other words, additional layers serve as additional subordinate CAs.

This scheme can be extended all the way to the application layer, where each application gets a unique Alias Key and certificate that encodes it identity.

In these cases, relying parties can walk the Alias Certificate chain to establish both the identity of the top layer and the identity of all the lower layers that comprise the top layer’s trusted computing base.

The Alias Certificate profile defined in this specification allows Alias Certificate chains, with these changes:

1. The Alias Certificate created by the RIoT Core layer, and each additional layer up to the final (leaf) Alias Certificate should assert cA
2. pathLengthConstraint for the DeviceID certificate, any other vendor certificates, and all intermediate Alias Certificates, must allow the desired layering depth

## RIoT Implementation Notes

RIoT-based systems (and other hardware-based security technologies) must restrict implementation complexity to avoid the need for changes because modifying RIoT Core will lead to a different DeviceID. (A future specification will cover secure updates to RIoT Core that can retain device identity.) General purpose X.509 certificate creation and manipulation libraries are often complex. Most of the implementation complexity of X.509 can be avoided by creating the certificate using special purpose code.

The security of RIoT-based systems depends critically on the complexity of inputs. For RIoT, easy-to-parse structures are favored, because these inputs may be controlled by unauthorized entities. The Firmware Security Descriptor structure is the principal complex input to RIoT Core and should therefore be as simple as possible.

Vendors may cache keys and credentials using RIoT-based (or other) protection technologies so that keys only need to be created during initial provisioning and on firmware update. This has the side-effect of improved boot times. However, these optimizations are beyond the scope of this specification.

## Alternative Implementations

The protocol defined in this specification does not require a DICE/RIoT system for key generation and certification. Alternative implementations include:

* *Software-Only*: A software-only implementation should employ best-effort protection for the DeviceID private key. For example, the DeviceID private key should be managed by the device’s Trusted Computing Base or Trusted Execution Environment (TEE). Compromise of the Trusted Computing Base may result in key disclosure.
* *Non-DICE Hardware Security Module* (1): Hardware security modules can be designed or adapted to create the Alias Keys and certificates that are compliant with this specification.
* *Non-DICE Hardware Security Module* (2): Hardware security modules that can perform signing, key generation, and firmware measurement, but *cannot* support the attestation as defined in this specification, can use a trusted service to prove the device’s security properties to a trusted service using proprietary protocols. The trusted service can then issue keys and certificates to the device which can be used in the protocol defined in this specification.

# X.509 Certificate Profile Normative Requirements

This section contains the normative requirements for Alias and DeviceID Certificates on conforming devices. This section also contains informational notes on how relying parties should interpret certificates.

*Note: This is a draft specification for review: details may change.*

## Composite Identity ExtensionEncoding

The Composite Identity encodes the DeviceID and the FWID of the device as an extension. It is currently recommended that the extension *not* be marked *critical*, because not all relying parties will process attestation data.

The extension object identifier SHALL be {1,3,6,1,4,1,311,89,3,1}.

RIoTExtension DEFINITIONS ::= BEGIN

RIoTOID OBJECT IDENTIFIER ({1,3,6,1,4,1,311,89,3,1})

CompositeDeviceID ::= SEQUENCE

     {

          version                    INTEGER (1),

          deviceID                   SubjectPublicKeyInfo,

          fwid SEQUENCE

          {

                     hashAlg         OBJECT IDENTIFIER,

                     fwid            OCTET STRING

          }

     }

-- The following definitions are copied from RFC 5280

SubjectPublicKeyInfo ::= SEQUENCE

{

algorithm AlgorithmIdentifier,

subjectPublicKey BIT STRING

}

AlgorithmIdentifier ::= SEQUENCE

{

algorithm OBJECT IDENTIFIER,

parameters ANY

}

END

The SubjectPublicKeyInfo field contains the same key, parameters, and encoding as the SubjectPublicKeyInfo field of the DeviceID certificate. For example, for an EC prime256 key, the DeviceID is [[[7]](#endnote-7)]:

DER Sequence

DER Sequence

ObjectIdentifier(1.2.840.10045.2.1) EC pubkey

ObjectIdentifier(1.2.840.10045.3.1.7) prime256

DER Bit String[65, 0] key value

Supported hash algorithms for the FWID SHALL include (SHA-256):

hashAlg OBJECT IDENTIFIER ({2,16,840,1,101,3,4,2,1})

## Serial Number Generation

Certificate *Serial Numbers* MUST be statistically unique per-Alias Certificate.

If the security processor has an entropy source, an 8-octet (positive) random number MAY be used.

If the security processor has does not have an entropy source, then an 8-octet (positive) *Serial Number* MAY be generated using a cryptographically secure key derivation function based on a secret key, such as those described in SP800-108 [6]. The *Serial Number* MUST be unique for each generated certificate. For the Alias Certificate, this SHOULD be achieved by incorporating the FWID into the key derivation process (e.g. as the *Context* value in SP-800-108) [[[8]](#endnote-8)].

## Certificate Lifetime

Devices with a secure local clock, or secure access to a remote time source, MAY use the clock to set the validity period (*Not Before*, *Not After*) of the device-issued Alias Key certificate, and the optional device-issued Non-Vendor Certified DeviceID Certificate. If a secure clock is not available then devices MAY still issue short-lived certificates, but revocation strategies based on certificate expiry will not be sound.

An alternative strategy is to set the *Not After* time to be the X.509-defined GeneralizedTime value of 99991231235959Z, which indicates that the certificate does not have a well-defined expiration date [4].

## Subject Name and Issuer Name

This specification does not constrain the use of the Subject Name, Issuer Name, or Subject Alternative Name of the Alias Certificate or DeviceID Certificate. Vendors can use these fields for alternate representations of DeviceID and FWID, or for other purposes. Refer to RFC5280 for rules and guidance on the interaction of the Subject and Subject Alternative Name fields when both are present.

## Alias Key Certificate

Compliant devices SHALL create Alias Certificates with the following extensions and constraints. This specification does not include the multiple-layer extension discussed in section 3.9.

|  |  |
| --- | --- |
| **Field Name** | **Contents** |
| *RIoT Extension* | OID: 1 3 6 1 4 1 311 89 3 1  Criticality: Recommend *not critical*  Value: CompositeDeviceID |
| *Subject Public Key Info* | Alias public key and algorithm |
| *Signature Algorithm* and *Signature Value* | DeviceID algorithm and public key |
| *Key Usage* | Key usage appropriate for the cryptographic protocol in use |
| *Extended Key Usage* | id-kp-clientAuth |
| *Basic Constraints* | pathLengthConstraint is absent  cA is absent |

Other fields should follow standard X.509 practice.

## Non-Vendor Certified DeviceID Certificate

Device MAY create self-signed DeviceID certificates with the following fields:

|  |  |
| --- | --- |
| **Field Name** | **Contents** |
| *Subject Public Key Info* | DeviceID public key and algorithm |
| *Key Usage* | keyCertSign is asserted  Key usage appropriate for the cryptographic protocol in use |
| *Basic Constraints* | cA:TRUE  pathLengthConstraint:0 |
| *Signature Algorithm* and *Signature Value* | DeviceID algorithm and public key |

Other fields should follow standard X.509 practice.

## Vendor Certified DeviceID Certificates

Vendors MAY create DeviceID certificates using an external PKI with the following fields:

|  |  |
| --- | --- |
| **Field Name** | **Contents** |
| *Subject Public Key Info* | DeviceID public key and algorithm |
| *Key Usage* | keyCertSign is asserted  Key usage appropriate for the cryptographic protocol in use |
| *Basic Constraints* | cA:TRUE  pathLengthConstraint:0 |
| *Signature Algorithm* and *Signature Value* | Vendor public key and signature |

Other fields should follow standard X.509 practice.

## DeviceID Certificate Signing Requests

Devices MAY create PKCS10 certificate signing requests for DeviceID keys.

## Recommended Algorithms for Interoperability

Devices SHOULD include support for:

1. EC P-256 keys
2. ECDSA signing for creating Alias Certificates (and optionally, DeviceID certificates)
3. SHA256 for certificate generation, and to calculate the FWID

Devices MAY create additional or alternate keys and certificates using other algorithm sets.

# Appendix A – Privacy Considerations

If misuse of the Alias and DeviceID keys and credentials could result in privacy hazards for the device owner or operator, then vendors should ensure that use of these keys is under the control of the owner.

Vendors may provide means to change the DeviceID so that the protocols in this specification identify it as a new and unrelated device: for example, to prevent tracking when the device is resold. If the device is not vendor-certified, then the device can re-create a new DeviceID and DeviceID Certificate, perhaps as part of a factory reset operation. If the device *is* vendor certified, then physical security (e.g. remanufacturing) or additional cryptographic keys and protocols (not specified here) can be used to re-certify the device.

# References

1. *Trusted Platform Architecture Hardware Requirements for a Device Identifier Composition Engine.* Currently in public review at <http://www.trustedcomputinggroup.org/wp-content/uploads/Device-Identifier-Composition-Engine-Rev69_Public-Review.pdf> [↑](#endnote-ref-2)
2. *A Foundation for Trust in the Internet of Things*. <https://www.microsoft.com/en-us/research/publication/riot-a-foundation-for-trust-in-the-internet-of-things/> [↑](#endnote-ref-3)
3. *The Transport Layer Security (TLS) Protocol Version 1.2*. <https://tools.ietf.org/html/rfc5246> [↑](#endnote-ref-4)
4. *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile.* <https://tools.ietf.org/html/rfc5280>. [↑](#endnote-ref-5)
5. See section 3.9 for changes needed to validate longer Alias Certificate chains for multi-layer systems. [↑](#footnote-ref-2)
6. *TPM Library Specification*. <http://www.trustedcomputinggroup.org/tpm-library-specification/> [↑](#endnote-ref-6)
7. *RFC5480: Elliptic Curve Cryptography Subject Public Key Information*. <https://datatracker.ietf.org/doc/rfc5480/?include_text=1> [↑](#endnote-ref-7)
8. *NIST Special Publication 800-108 – Recommendation for Key Derivation Using Pseudorandom Functions.* <http://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-108.pdf> [↑](#endnote-ref-8)