Document title
X.509 Certificate Profiles
Date
2022-02-02
Author
Emanuel Palm
Contact
emanuel.palm@pinterop.se

Document type
PD
Version
1.0
Status
Proposal
Page
1 (30)

Eclipse Arrowhead

X.509 Certificate Profiles

Profile Description (PD)

Abstract

This document specifies how X.509 certificates are to be configured, issued and validated to facilitate secure identification and communication within and between *Eclipse Arrowhead* local clouds, when such security is desired and the kind of certificate is relevant.

This is an unrefined conversion from an earlier document written in Markdown. It is not formatted or hyperlinked properly, but will be in the future. Its contents are still representative, however.

Contents

1 Introduction

In this document, we describe how X.509 certificates must be configured if used in conjunction with Eclipse Arrowhead. X.509 is a certificate standard produced by the International Telecommunication Union - Telecommunication Standardization Sector (ITU-T) and is famously used by the TLS and DTLS protocols, the former of which is used to establish secure connections on the World Wide Web. In brief, an X.509 certificate represents the identity of its owner. It records required inputs to a secure key exchange algorithm, as well as how the identity it represents is endorsed by a hierarchy of issuers. In the context of Eclipse Arrowhead, X.509 certificates are used to manage what IoT devices, and other entities, are to be trusted and what cryptographic algorithms are to be used to establish their identities.

The rest of this document is organized as follows. This section continues by listing key terminology and outlining some linguistic conventions. Section 2 outlines the X.509 certificate format and the 8 core Arrowhead X.509 profiles. Sections 3, 4, 5 and 6 consider secure algorithm inputs, certificate identification, certificate life-cycle management, and using certificates as input to authorization procedures, respectively.

1.1 Relation to the IETF RFC 5280 X.509 Profile

All certificate profiles specified in this document are *required* to be strict subsets of the RFC 5280 X.509 profile of the Internet Engineering Task Force (IETF).

1.2 Significant Terminology

The following subsections represent technical domains with particular bearing on this document. Each subsection briefly describes a domain and defines relevant terms and abbreviations.

1.2.1 Eclipse Arrowhead

Service-oriented communication architecture for Industry 4.0 automation.

- **Device**: A physical machine that could be capable of hosting Arrowhead *systems*.
- **Local Cloud**: A physical protected network consisting of communicating *systems*.
- **Service**: An explicitly defined network application interface accessible to authorized *systems*.
- **System**: A software application providing Arrowhead-compliant *services* that runs on a *device*.

1.2.2 X.509

Certificate standard for establishing trust between devices over untrusted computer networks.

- **Certificate Authority (CA)**: Entity issuing (signing) other certificates to endorse their validity.
- **Certificate Chain**: A chain consisting of an *end entity* certificate, its *issuer*'s certificate, that *issuer*'s *issuer*'s certificate, and so on up to the *root CA*'s certificate.
- **Certificate Revocation List (CRL)**: A list identifying certificates that no longer are to be considered valid even though they are yet to expire.
- **Certificate Signing Request (CSR)**: A request for a certificate to be issued by a receiving CA.
- **End Entity**: Entity having but not issuing certificates.
- **Entity**: Any thing or being potentially able to hold and use an X.509 certificate.
- **Fingerprint**: The hash of the DER form of an X.509 certificate, produced using a cryptographic *hash algorithm*.

- **Hash Algorithm**: A function that produces an arbitrary fixed-size output for any given arbitrarily sized input. The same input always produces the same output.
- **Intermediary CA**: CA that *did not* issue its own certificate and, therefore, can be trusted by explicitly trusting another certificate further up its issuance hierarchy.
- **Issuer**: The CA that signed a given certificate.
- **Public Key Infrastructure (PKI)**: The structure of entities, each having a certain role, required to facilitate secure certificate distribution.
- **Root CA**: CA that *did* issue (self-sign) its own certificate and, therefore, must be explicitly trusted.
- **Subject**: The entity owning and being described by a given certificate.
- **Trust Anchor**: Another name for *root CA*.

1.2.3 X.501

Naming schema for X.500 directories. The standard is used to name the subjects and issuers of X.509 certificates.

- **Distinguished Name (DN)**: A hierarchical naming format composed consisting of RDNs. An example of a DN could be ‘O=My Company,CN=Robert Robertson+E=robert@mail.com‘. The ‘O‘ RDN is at the highest hierarchical level, while the ‘CN+E‘ RDN is at the level below it. The comma character ‘,‘ is used to delimit the RDNs in this example.
- **Relative Distinguished Name (RDN)**: A list of attribute/value pairs belonging to the same hierarchical level in a DN. Examples of RDNs could be ‘O=My Company‘ and ‘CN=Robert Robertson+E=robert@mail.com‘. The first RDN consists of a single pair while the second consists of two delimited by ‘+‘.

1.2.4 ASN.1

Interface description language for describing messages that can be sent or received over computer networks using several associated encodings. The standard is used to describe the structure of X.509 certificates, which *must* be encoded using ASN.1 DER, as defined below.

- **Basic Encoding Rules (BER)**: Binary ASN.1 encoding that appends basic type and length information to each encoded value, which means that decoding a given message does not require knowledge of its original ASN.1 description. Defined in X.690.
- **Distinguished Encoding Rules (DER)**: A subset of BER that guarantees canonical representation, which is to say that every pair of structurally equivalent ASN.1 messages can be represented in DER in exactly one way. Must be used when encoding X.509 certificates. Defined in X.690.
- **Object Identifier (OID)**: A hierarchical and universally unique identifier, useful for identifying parts of ASN.1 messages.
- **Octet**: An 8-bit byte.

1.2.5 TLS and DTLS

Transport Layer Security (TLS) and *Datagram Transport Layer Security* (DTLS) are IETF (Internet Engineering Task-Force) standards for establishing secure connections over untrusted transports. Both can use X.509 to perform authentication when establishing secure connections.

- **Authentication Algorithm**: An asymmetric, or *public key*, encryption algorithm used to establish a degree of confidence in the identity of a peer.

- **Cipher Suite**: A four-part set consisting of a *key exchange*, *authentication*, *encryption* and *hash* algorithm. Such a suite must be agreed upon for a TLS connection to be possible to establish. The *authentication* and *hash* algorithms form a *signature suite*.
- **Encryption Algorithm**: A symmetric encryption algorithm, typically used to make data opaque in transit between a sender and a recipient. Can also be referred to as a *stream* or *block* cipher, depending on its mode of operation.
- **Key Exchange Algorithm**: An algorithm used by parties for exchanging *public keys* as part of preparing for secure communication.
- **Hash Algorithm**: A function that produces an arbitrary fixed-size output for any given arbitrarily sized input. The same input always produces the same output. Used to reduce the size of public key signatures, among other things.
- **Peer**: Either end of a two-way communication.
- **Public key**: The public component of a public/private key pair. Knowledge of the public key allows for messages to be encrypted such that only the possessor of the private key can decrypt them, and vice versa. Used to produce *signatures* and to *share secrets*.
- **Shared Secret**: The input to an *encryption algorithm* that has been secretly shared between two parties.
- **Signature**: The private key encryption of a hashed object, which most significantly can be an X.509 certificate. Knowledge of the corresponding public key, hashing algorithm and hashed object can is sufficient to verify that the signature was created by the possessor of the private key and that the object has not been tampered with.
- **Signature Suite**: A two-part set consisting of an *authentication* and *hash* algorithm. Used to produce and validate *signatures*. Can be a subset of a *cipher suite*.

1.3 Conventions

The words **must**, **must not**, **required**, **should**, **should not**, **recommended**, **may**, and **optional** in this document are to be interpreted as follows: **must** and **required** denote absolute requirements that must be adhered to for a certificate to be considered compliant to a given profile; **must not** denotes an absolute prohibition; **should**, **should not** and **recommended** denote recommendations that should be deviated from only if special circumstances make it relevant; and, finally, **may** and **optional** denote something being truly optional.

2 Certificate Profiles

There are eight certificate profiles defined in this document, depicted in the following diagram:

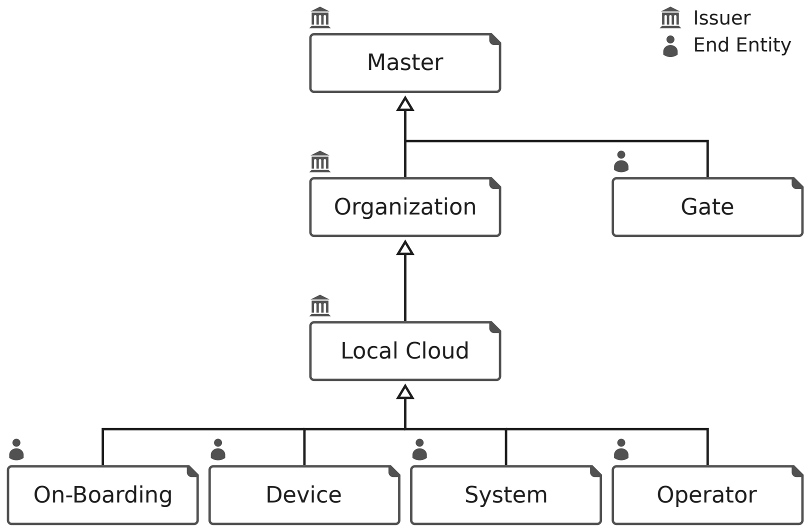

Figure 1: Hierarchy of X.509 Profiles

Each diagram box represents a profile. The arrows in the diagram are to be read as ”issued by”, meaning that the every certificate adhering to the profile from which the arrow extends must be issued (signed) by a certificate with the profile pointed to.

In this section, we begin by considering the X.509 format itself, after which we outline the Master, Gate, Organization, Local Cloud, On-Boarding, Device, System and Operator profiles. Each profile is described in terms of its (a) purpose, (b) issuer, (c) subject name and (d) extensions. The details included in this section are intended to be enough to allow for the correct parameterization of certificates, but are unlikely to be sufficient for implementing software for handling them. If the latter is relevant, please refer to RFC 5280, X.509, X.501, X.690 and ASN.1.

2.1 Certificate Format

The ASN.1 syntax of the third version of the X.509 certificate format is defined in as follows in RFC 5280:

```
Certificate ::= SEQUENCE {
    tbsCertificate       TBSCertificate,
    signatureAlgorithm   AlgorithmIdentifier,
    signatureValue       BIT STRING
}

TBSCertificate ::= SEQUENCE {
    version          [0]  EXPLICIT Version DEFAULT v1,
    serialNumber          INTEGER,
    signature             AlgorithmIdentifier,
    issuer                Name,
    validity              Validity,
    subject               Name,
    subjectPublicKeyInfo  SubjectPublicKeyInfo,
    issuerUniqueID   [1]  IMPLICIT UniqueIdentifier OPTIONAL,
    subjectUniqueID  [2]  IMPLICIT UniqueIdentifier OPTIONAL,
    extensions       [3]  EXPLICIT Extensions OPTIONAL
}
```

The `Certificate` structure itself consists only of a `TBSCertificate`, a signature algorithm identifier and a concrete signature. The signature is meant to be produced by the issuer of the certificate and serves to protect the integrity of the certificate under the assumption that the issuer is trusted. All data protected by the signature is collected in the `TBSCertificate`, the fields of which are described in the following subsections.

2.1.1 `version`

X.509 version of the certificate. Only `v3(2)` supports certificate extensions, which *must* be used by all profiles described in this document. Supporting `v1(0)` and `v2(1)` at all is *optional*.

2.1.2 `serialNumber`

A positive integer of no more than 20 octets uniquely assigned by the issuer of the certificate.

Security recommendation. The serial number should be exactly 20 octets long and be produced by a cryptographic random number generator. This serves both to (A) make the certificate more resistant to collision attacks and (B) to prevent the serial number from leaking important details about the certificate issuer, such as how many certificates it has issued, how long the signing process took, and so on. If certificate size would be a major concern, for example when certificates are used by significantly constrained devices, fewer octets *may* be used. Note that the serial number is signed and must be positive, which means that the most significant bit of its first ASN.1 BER `INTEGER` octet must be 0.

2.1.3 `signature`

Identifies the signature suite used to produce the `signatureValue` in the enclosing `Certificate`, as well as any parameters made relevant by the suite. Examples of suites could be *RSA with SHA256* or *ED25519*. This field must contain the same suite as the `signatureAlorithm` field in the enclosing `Certificate`, as described in the beginning of Section 2.1.

See Section 3 for information about selecting cryptographic algorithms.

2.1.4 `issuer`

The DN of the issuer of the certificate in question. More specifically, this field contains an exact copy of the `subject` field of Section 2.1.6 from the certificate of its issuer, or a copy of the `issuer` field of the same certificate if it is self-signed.

2.1.5 `validity`

The period during which the certificate *may* remain active and its issuer is bound to know whether or not the certificate has been revoked or not. Denoted as a duration between two timestamps, `notBefore` and `notAfter`, as show below.

```
Validity ::= SEQUENCE {
    notBefore   Time,
    notAfter    Time
}

Time ::= CHOICE {
    utcTime     UTCTime,
    generalTime GeneralizedTime
}
```

For each of these two dates, the date in question *must* be encoded as a `UTCTime` if its year is less than or equal to 2049, or as a `GeneralizedTime` if the year is equal to or greater than 2050. See Section 4.1.2.5 of RFC 5280 for more details.

The time span denoted by this field *should* be shorter than the expected lifetime of the entity owning it, typically significantly shorter, if provisions exist for renewing it during its lifetime. More specific recommendations will be published in other documents of the Eclipse Arrowhead project.

2.1.6 `subject`

The DN used to describe the owner of the certificate. The field is defined as follows:

```
Name ::= CHOICE {
    rdnSequence RDNSequence
}

RDNSequence ::= SEQUENCE OF RelativeDistinguishedName

RelativeDistinguishedName ::= SET SIZE (1..MAX) OF AttributeTypeAndValue

AttributeTypeAndValue ::= SEQUENCE {
    type  AttributeType,
    value AttributeValue
}

AttributeType ::= OBJECT IDENTIFIER

-- Concrete type determined by associated \texttt{AttributeType}.
-- In our case that type will always be \texttt{DirectoryString}.
AttributeValue ::= ANY

DirectoryString ::= CHOICE {
     teletexString   TeletexString   (SIZE (1..MAX)),
     printableString PrintableString (SIZE (1..MAX)),
     universalString UniversalString (SIZE (1..MAX)),
     utf8String      UTF8String      (SIZE (1..MAX)),
     bmpString       BMPString       (SIZE (1..MAX))
}
```

The below ‘AttributeType‘s *must* be supported by compliant software implementations, as required by RFC 5280. Supporting them means that that their OIDs are known to be associated with values of type `DirectoryString`, as defined above. Failing to support them means that the certificate validation procedure of RFC 5280, Section 6, cannot be executed. See RFC 5280 Section 4.1.2.4 for more attributes that *should* be supported.

Attribute Type	Abbreviation	OID
Common Name	`CN`	`2.5.4.3`
Country	`C`	`2.5.4.6`
Distinguished Name Qualifier	`DN`	`2.5.4.46`
Domain Component	`DC`	`0.9.2342.19200300.100.1.25`
Organization	`O`	`2.5.4.10`
Organizational Unit	`OU`	`2.5.4.11`
Serial Number	`SN`	`2.5.4.5`
State or Province Name	`ST`	`2.5.4.8`

Every certificate *must* contain at least one `subject` *Distinguished Name Qualifier* (`DN`). It *should* only contain one. It *should* be encoded as a `PrintableString`. The rightmost (i.e. least significant) such identifies the profile of the certificate. The identifiers are as follows:

X.509 Profile	Distinguished Name Qualifier (`DN`)
Master	`ma`
Gate	`ga`
Organization	`or`
Local Cloud	`lo`
On-Boarding	`on`
Device	`de`
System	`sy`
Operator	`op`

In addition, each certificate *must* contain at least one `subject` *Common Name* (`CN`) that is a valid DNS label (https://datatracker.ietf.org/doc/html/rfc1035#section-2.3.1) of no more than 62 characters. It *should* only contain one `CN`. It *should* be encoded as a `PrintableString`. The rightmost (i.e. least significant) such contains the name, or *alias*, meant to be used by humans when referring to the certificate. While names *may* use up to 62 characters, we *recommend* the use of 10 to 20 character long lowercase identifiers, such as `51e41vjoxq`, produced with secure random number generators. Randomized identifiers hide details that otherwise would become accessible to adversaries with certificate copies, such as how many certificates have been issued, the type of machines they are associated with, and so on. The `CN` field value *must*, furthermore, be unique among all certificates issued by the same CA with same Distinguished Name Qualifier (`DN`). It *should not* be derived from a portion of the `serialNumber` of the same certificate.

The `CN` is meant to help humans refer to specific certificates, they *should not* be used as primary references by machines. See Section 4 for a discussion about machine certificate references.

2.1.7 `subjectPublicKeyInfo`

The public key of the subject, as well as the identity of the algorithm associated with it. The field is defined as follows.

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm        AlgorithmIdentifier,
    subjectPublicKey BIT STRING
}
```

See Section 3 for information about selecting cryptographic algorithms.

2.1.8 `issuerUniqueID` and `subjectUniqueID`

Optional identifiers uniquely associated with the issuer or subject of the certificate.

These fields *must not* be used. See Section 4 for a discussion about certificate identification.

2.1.9 `extensions`

A list of certificate extensions, as defined below.

```
Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension

Extension  ::= SEQUENCE {
    extnID     OBJECT IDENTIFIER,
    critical   BOOLEAN DEFAULT FALSE,
    extnValue  OCTET STRING
}
```

RFC 5280 explicitly outlines 17 different X.509 extensions, listed by category below.

Extension	`extnID` (OID)	Brief Description
Identifier Extensions		
Authority Key Identifier	`2.5.29.35`	Identifier uniquely associated with certificate issuer.
Subject Key Identifier	`2.5.29.14`	Identifier uniquely associated with certificate subject.
Key Usage Extensions		
Key Usage	`2.5.29.15`	Public key usage restrictions.
Extended Key Usage	`2.5.29.37`	Additional public key usage restrictions.
Policy Extensions		
Certificate Policies	`2.5.29.32`	List of policies accepted by the certificate subject.
Policy Mappings	`2.5.29.33`	List of policy equivalence relations.
Policy Constraints	`2.5.29.36`	Policy validation constraints (e.g. policy X must be accepted).
Inhibit `anyPolicy`	`2.5.29.54`	Policy validation constraint related to use of `anyPolicy`.
Name Extensions		
Subject Alternative Name	`2.5.29.17`	Additional names associated with the certificate subject.
Issuer Alternative Name	`2.5.29.18`	Additional names associated with the certificate issuer.
Name Constraints	`2.5.29.30`	List of subject name restrictions for issued certificates.
CRL Extensions		
CRL Distribution Points	`2.5.29.31`	List of where relevant CRLs can be found.
Freshest CRL	`2.5.29.46`	List of where delta CRLs can be found.
Information Extensions		
Authority Information Access	`1.3.6.1.5.5.7.1.1`	List of where issuer information and services can be found.
Subject Information Access	`1.3.6.1.5.5.7.1.11`	List of where subject information and services can be found.
Other Extensions		
Subject Directory Attributes	`2.5.29.9`	Additional subject identification attributes (e.g. nationality).
Basic Constraints	`2.5.29.19`	Description of where a certificate belongs in its CA hierarchy.

Each category of extensions is described in the following subsections.

Identifier Extensions The **Authority Key Identifier** and **Subject Key Identifier** extensions are used to identify the public key of the issuer and subject of a given certificate, respectively. The extensions are defined as follows:

```
-- Required for all but self-signed CA certificates (root CAs).
AuthorityKeyIdentifier ::= SEQUENCE {
    keyIdentifier             [0] KeyIdentifier           OPTIONAL, -- Required.
    authorityCertIssuer       [1] GeneralNames            OPTIONAL, -- Should not be used.
    authorityCertSerialNumber [2] CertificateSerialNumber OPTIONAL  -- Should not be used.
}

-- Required for all but end entity certificates.
SubjectKeyIdentifier ::= KeyIdentifier

KeyIdentifier ::= OCTET STRING
```

RFC 5280 *requires* that `AuthorityKeyIdentifier` extension is included in all CA certificates except for self-signed such. The `keyIdentifier` of the `AuthorityKeyIdentifier` extension of a given certificate *must* match the `SubjectKeyIdentifier` of its issuer's certificate, or *may* be omitted if the certificate is self-signed. Use of the `authorityCertIssuer` and `authorityCertSerialNumber` fields of `AuthorityKeyIdentifier` is *not recommended*. If a self-signed certificate leaves the `authorityCertIssuer` and `authorityCertSerialNumber` fields unspecified, it may omit the `AuthorityKeyIdentifier` extension entirely. RFC 5280 further *requires* that all but end entity certificates use the `SubjectKeyIdentifier` extension. Its value *should* be the the cryptographic hash of the `subjectPublicKey` value (excluding the tag, length, and number of unused bits) of the `subjectPublicKeyInfo` field. See Section 3 for a discussion about what hash algorithms to use. The extensions *must not* be marked as critical.

Key Usage Extensions The **Key Usage** and **Extended Key Usage** extensions are defined as follows:

```
KeyUsage ::= BIT STRING {
    digitalSignature (0),
    nonRepudiation   (1), -- Also known as \texttt{contentCommitment}.
    keyEncipherment  (2),
    dataEncipherment (3),
    keyAgreement     (4),
    keyCertSign      (5),
    cRLSign          (6),
    encipherOnly     (7),
    decipherOnly     (8)
}

ExtKeyUsageSyntax ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId

KeyPurposeId ::= OBJECT IDENTIFIER
```

The `KeyUsage` extension is a set of bit flags indicating various ways in which a certificate *must* only be used. Please refer to RFC 5280 Section 4.2.1.3 for descriptions of each bit flag. The extension *must* be used in all certificates and *must* be marked as critical. How to set it is outlined in the section specific to each particular certificate profile.

The `ExtKeyUsageSyntax` extension has the same purpose, but is open-ended in the sense that it allows for any OID to be used as an indication of what a certain certificate *may* be used for. RFC 5280 lists six such purpose identifiers, out of which two has particular bearing on this document, namely the `serverAuth` (OID `1.3.6.1.5.5.7.3.1`) and `clientAuth` (OID `1.3.6.1.5.5.7.3.2`) identifiers, which indicate that the certificate holder must be able to respond to TLS server and client authorization requests, respectively. The extension *must* be used in all end entity certificates, as well in the certificates of the CAs that handle CSRs directly via network application interfaces. The `serverAuth` and `clientAuth` identifiers *must* be included. The extension *should not* be marked as *critical*. Other purpose identifiers *may* be used.

Policy Extensions In the context of X.509 and RFC 5280, a *policy* is a legal document that binds the owner of a certificate, and, potentially, all certificates issued by that certificate, to certain legal obligations. The four policy extensions defined by RFC 5280 *may* be used to ensure that every certificate in a given certificate chain have accepted some policies of concern. Please refer to RFC 5280 for more details about these extensions.

Name Extensions The **Subject Alternative Name** and **Issuer Alternative Name** allows for additional identities to be associated with a given `subject` or `issuer` name. Such additional identities significantly include DNS names, IP addresses and e-mail addresses. For example, given that some system receives a signed message and the certificate associated with that signature, the system can verify that it received the message via one of the identities listed as subject alternative name in that certificate. The extensions are defined as follows:

```
SubjectAltName ::= GeneralNames
IssuerAltName  ::= GeneralNames

GeneralNames ::= SEQUENCE SIZE (1..MAX) OF GeneralName

GeneralName ::= CHOICE {
    otherName                 [0] OtherName,
    rfc822Name                [1] IA5String,
    dNSName                   [2] IA5String,
    x400Address               [3] ORAddress,
    directoryName             [4] Name,
    ediPartyName              [5] EDIPartyName,
    uniformResourceIdentifier [6] IA5String,
    iPAddress                 [7] OCTET STRING,
    registeredID              [8] OBJECT IDENTIFIER
}
```

The `SubjectAltName` extension *must* be used by all end entity certificates and *must* identify at least one DNS name, IP address or other relevant identifier useful for contacting the certificate owner. The extension *must* also be used for CA certificates that handles CSRs directly via network application interfaces. Use of the `IssuerAltName` is *not recommended*. Both extensions *should* be marked as non-critical.

Security recommendation. All names appearing in a `SubjectAltName` extension *should* be randomized, by which we mean that every name is formulated such that an adversary can derive as little information as possible from it. In the case of an IP address, this means that a larger address space is preferred (e.g. 10.0.0.0/8 rather than 192.168.0.0/16) and that addresses are assigned randomly rather than sequentially. In the case of DNS names, this means that no serially assigned identifiers are used and that no human-readable descriptors of sensitive places or things are part of the name. See Section 7 for a discussion about the use of DNS in the context of Arrowhead.

The **Name Constraints** extension makes it possible for a CA to restrict the set of values allowed for the `subject` and `SubjectAltName` fields in issued certificates. The extension *should not* be used. It *must* be marked as critical if used. Please refer to RFC 5280 Section 4.2.1.10 for more details.

CRL Extensions The CRL extensions facilitate a mechanism for revoking certificates that are still valid and distributing information about those revocations. These extensions *should not* be used to facilitate the revocation of On-Boarding, Device, System and Operator certificates. They *may* be used for revoking certificates of the other profiles.

Eclipse Arrowhead comes with its own certificate revocation procedure via its *Certificate Authority* system. Its use is *recommended* for revoking and verifying the validity of On-Boarding, Device, System and Operator certificates.

Information Extensions The information extensions allows for various types of data sources or services to be associated with the certificate holder. These extensions *should not* be used, unless required by a protocol relevant for the revocation of Master, Organization, Gate and Local Cloud certificates.

Eclipse Arrowhead has its own provisions for metadata distribution and service management, via the *Service Registry* system, *Gatekeeper* system, and other. Those provisions *should* be used.

Other Extensions The **Subject Directory Attributes** allows for arbitrary identification attributes, such as nationality, to be associated with the `subject` of a certificate. It *may* be used. It *must* be marked as non-critical if used.

The **Basic Constraints** extension allows for it to be denoted whether or not a given certificate belongs to a CA, as well as how many intermediary CAs may exist below it in any given certificate chain. The extension is defined as follows:

```
BasicConstraints ::= SEQUENCE {
    cA                  BOOLEAN DEFAULT FALSE,
    pathLenConstraint INTEGER (0..MAX) OPTIONAL
}
```

The extension *must* be used and marked critical by all Arrowhead-compliant certificates. The `pathLenConstraint` *must* be set by all CA certificates, but *must not* be set by end entity certificates.

2.2 Master Profile

A *Master* certificate exists to establish trust between organizations that may want to interconnect their Arrowhead systems. It does this by issuing *Organization* and *Gate* certificates. The former enables organizations to set up their own certificate hierarchies while sharing a common CA with other organizations. The latter kind enables all those organizations to trust a special kind of broker system, which facilitates negotiating connections between organizations.

The Eclipse Arrowhead project *should* maintain an authoritative Master certificate that *may* be used for non-private Arrowhead installations. Other entities *may* setup and maintain their own Master certificates as needed.

2.2.1 `issuer`

May be self-signed or issued by an RFC 5280-compliant CA.

Security notice. In the case of a Master being issued by a widely trusted World Wide Web CA, entities with the Master in their certificate chains could be made able to serve HTTP(S) traffic to regular web browsers without any additional certificate distribution. It could also make it possible for improperly configured systems to establish secure connections with unauthorized systems. Having CAs above a Master provides for new opportunities and new risks at the same time, for which reason it may or may not be desirable.

2.2.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`ma`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.2.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. Omit field if self-signed. See Section 2.1.9.1.
Subject Key Identifier	`2.5.29.14`	No	Hash of subject public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: true, pathLenConstraint: 2`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `keyCertSign(5)` and `cRLSign(6)` *must* be set. See Section 2.1.9.2.

If the certificate will be used to automatically respond to CSRs via a network application interface, the following must also be present:

Extension	OID	Critical	Value
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set in addition to `5` and `6`. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes‘serverAuth‘ and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier to which CSRs can be sent. See Section 2.1.9.4.

2.3 Gate Profile

A *Gate* certificate is associated with a message broker or bus that exists to guarantee delivery of messages between the local clouds of distinct organizations. Its existence means that such messages can sent over a secure transport.

2.3.1 `issuer`

Must be issued by a Master certificate.

2.3.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`ga`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.3.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: false`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes `serverAuth` and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier through which the system can be reached. See Section 2.1.9.4.

2.4 Organization Profile

An *Organization* certificate is maintained by a single organization, allowing it to manage the certificates of its own local clouds.

2.4.1 `issuer`

Must be issued by a Master certificate.

2.4.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`or`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.4.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Subject Key Identifier	`2.5.29.14`	No	Hash of subject public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: true, pathLenConstraint: 1.` See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `keyCertSign(5)` and `cRLSign(6)` *must* be set. See Section 2.1.9.2.

If the certificate will be used to automatically respond to CSRs via a network application interface, the following must also be present:

Extension	OID	Critical	Value
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set in addition to `5` and `6`. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes‘serverAuth‘ and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier to which CSRs can be sent. See Section 2.1.9.4.

2.5 Local Cloud Profile

An *Local Cloud* certificate is maintained by a single local cloud, enabling it to issue end entity certificates for on-boarding and on-boarded devices, as well as for systems and operators.

2.5.1 `issuer`

Must be issued by an Organization certificate.

2.5.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`lo`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.5.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Subject Key Identifier	`2.5.29.14`	No	Hash of subject public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: true, pathLenConstraint: 0`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `keyCertSign(5)` and `cRLSign(6)` *must* be set. See Section 2.1.9.2.

If the certificate will be used to automatically respond to CSRs via a network application interface, the following must also be present:

Extension	OID	Critical	Value
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set in addition to `5` and `6`. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes‘serverAuth‘ and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier to which CSRs can be sent. See Section 2.1.9.4.

2.6 On-Boarding Profile

An *On-Boarding* certificate allows for a device in an Arrowhead local cloud to request a new device certificate. It is used both or either to provision new devices and/or to facilitate renewal of certificates as they are about to expire. Certificates adhering to this profile *must* only be provided to devices known or assumed to be trustworthy.

2.6.1 `issuer`

Must be issued by a Local Cloud certificate.

2.6.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`on`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.6.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: false`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes `serverAuth` and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier through which the owning device can be reached. See Section 2.1.9.4.

2.7 Device Profile

A *Device* certificate allows for a device in an Arrowhead local cloud to request new system certificates. One system certificate is required for each system a given device intends to run. Certificates adhering to this profile *must* only be provided to devices known or assumed to be trustworthy.

2.7.1 `issuer`

Must be issued by a Local Cloud certificate.

2.7.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`de`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.7.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: false`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes `serverAuth` and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier through which the device can be reached. See Section 2.1.9.4.

2.8 System Profile

A *System* certificate allows for a device in an Arrowhead local cloud to provide the services associated with a particular system, and/or to act as a service consumer.

2.8.1 `issuer`

Must be issued by a Local Cloud certificate.

2.8.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`sy`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.8.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: false`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes `serverAuth` and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier through which the system can be reached. See Section 2.1.9.4.

2.9 Operator Profile

An *Operator* certificate allows for a human or computer operator to administer a particular Arrowhead local cloud.

2.9.1 `issuer`

Must be issued by a Local Cloud certificate.

2.9.2 `subject`

The subject field DN *must* contain the following attributes exactly once, either in the same or different RDNs:

Attribute Type	OID	Value
DN Qualifier (DN)	`2.5.4.46`	`op`
Common Name (CN)	`2.5.4.3`	Randomized identifier. See Section 2.1.6.

2.9.3 `extensions`

The following extensions *must* be used and configured as described:

Extension	OID	Critical	Value
Authority Key Identifier	`2.5.29.35`	No	Hash of issuer public key. See Section 2.1.9.1.
Basic Constraints	`2.5.29.19`	Yes	`cA: false`. See Section 2.1.9.7.
Key Usage	`2.5.29.15`	Yes	Bits `digitalSignature(0)` and `keyEncipherment(2)` *must* be set. See Section 2.1.9.2.
Extended Key Usage	`2.5.29.37`	No	Purposes `serverAuth` and `clientAuth` *must* be specified. See Section 2.1.9.2.
Subject Alternative Name	`2.5.29.17`	No	At least one IP address, DNS name or other identifier through which the operator's control system can be reached. See Section 2.1.9.4.

3 Algorithms, Key Lengths and Other Security Details

WChoosing a signature suite for a certificate poorly can lead to severe security vulnerabilities. We *recommend* that credible information security institutes, such as NIST, ENSIA or IETF, be consulted for making choices about algorithms, key lengths and other relevant security details.

Given that no relevant breakthroughs are made or expected in quantum computing, you *may* chose to follow RFC 7525, which recommends the following four TLS cipher suites:

Key Exchange	Authentication	Encryption	Hash
DHE	RSA (2048-bit or 3072-bit)	AES 128 GCM	SHA256
ECDHE	RSA (2048-bit or 3072-bit)	AES 128 GCM	SHA256
DHE	RSA (2048-bit or 3072-bit)	AES 256 GCM	SHA384
ECDHE	RSA (2048-bit or 3072-bit)	AES 256 GCM	SHA384

Only the first of the above cipher suites is required to be supported by all TLS 1.3 implementations (RFC 8446, Section 9.1). Each cipher suite includes a signature suite (the *Authentication* and *Hash* fields). Adherence to RFC 7525 means that RSA (2048-bit or 3072-bit) with SHA256 or SHA384 is used to sign certificates. Given that RFC 7525 is trusted, SHA256 and SHA384 *may* be suitable choices for producing certificate identifiers, as discussed in Section 4.

The above recommendations are *general*, in the sense that no particular assumptions are made about the setting in which the device employing the signature or cipher suite is located. We understand that many Arrowhead installations will involve hardware with limited computational capabilities, which may not be able to handle primitives of the cryptographic strengths we have mentioned. The Eclipse Arrowhead project will publish summaries of recommendations for such and other settings in the future.

4 Identifying Certificates and Their Owners

The X.509 profiles of this document provide two fields and one extension whose values could be used to identify a particular certificate or its owner. These are the the ‘serialNumber‘ and ‘subject‘ fields, as well as the ‘Subject Key Identifier‘ extension. Despite this, they *must not* be used by machines referring to certificates. The reason for this is that they are all set arbitrarily by the creator of each certificate. An adversary with a given certificate could create another certificate with the same values, allowing it to masquerade as the owner of the original certificate without knowledge of its private key.

What *should* be used, rather, is either (1) the cryptographic hash of an entire certificate in its DER form (i.e. its fingerprint) or (2) the cryptographic hash of its ‘subjectPublicKey‘ value (excluding the tag, length, and number of unused bits) of the ‘subjectPublicKeyInfo‘ field, also in its DER form. While the second of these two identifiers will be equal to the ‘Subject Key Identifier‘ for a conformant certificate, it *should not* be generally assumed that a given certificate is conformant. Both of these hashes include the public key of the certificate owner, which means that it becomes harder for an adversary to create a counterfeit certificate. We *recommend* that the certificate hash (fingerprint) be generally used as identifier, as it protects the integrity of the entire certificate as opposed to only one field of it. See Section 3 for details about what hash algorithms to use.

5 Certificate Life-Cycle Management

Certificates must be created, distributed, replaced as they expire and, sometimes, revoked before they expire. If these tasks are handled without care, it can lead to serious security vulnerabilities. To help making this handling as rigorous as possible, the Eclipse Arrowhead project provides the *Certificate Authority* system, which, through some other helper systems, provides an infrastructure for managing the certificate life-cycle within local clouds. We *recommend* that the system be used, or a similarly capable replacement, for all Eclipse Arrowhead installations.

Generally, when certificate life-cycles are managed, we *recommend* that the following be observed:

1. Create each private key on the device that will use it.
2. Use CSRs to avoid moving private keys between devices during certificate issuance.
3. Never make backups or other copies of private keys that can be trivially replaced.
4. Store backups of sensitive private keys offline, if possible.
5. Store active private keys in secure hardware elements, such as TPMs (ISO/IEC 11889).
6. Immediately revoke owned certificates whose private keys are suspected to be compromised.
7. Actively look for and act on revocations in the certificate chains of counter-parties.

The above list is *not* to be considered as being exhaustive. Adhering to it is not a substitute for consulting independent and credible security experts. The list is likely to be revised as more experience is gained related to the security of Arrowhead installations.

6 Authorizing Certificate Owners

While no steps in addition to those in Section 6 of RFC 5280 are mandatory for certificate *validation*, there are additional details made available by our profiles that *should* be used for *authorization* when applicable. These details are (A) profile identifiers and (B) issuer identities. For example, if an Arrowhead system provides a service, it *may* be relevant to ensure that any consumer of that service belongs to the same local cloud and is an operator.

We *recommend* that a procedure practically equivalent to the below algorithm, described in pseudo-code, is used when extracting certificate profile identifiers. The input to the algorithm is an array containing a certificate chain ‘chain‘ of a given peer. The certificate at index 0 of that array is owned by the peer in question, while the remaining certificates represent its issuance hierarchy, in order of issuance from least significant to most.

```
function getCertificateProfileIdentifier(chain) {
    if (chain.length == 0) { throw "empty chain"; }

    let dnq = getDNQualifier(chain[0]);
    switch (dnq) {
        case "ma":
            break;
        case "or": case "ga":
            requireProfileSequence(chain[1..], ["ma"]); break;
        case "lo":
            requireProfileSequence(chain[1..], ["or", "ma"]); break;
        case "on": case "de": case "sy": case "op":
            requireProfileSequence(chain[1..], ["lo", "or", "ma"]); break;
        default:
            throw "unexpected DNQualifier";
    }
    return dnq;
}

function requireProfileSequence(chain, identifiers) {
    if (identifiers.length == 0) { return; }
    if (chain.length == 0) { throw "expected additional certificate(s)"; }

    if (getDNQualifier(chain[0]) != identifiers[0]) {
        throw "unexpected certificate profile";
    }

    requireProfileSequence(chain[1..], identifiers[1..]);
}
```

The ‘getDNQualifier‘ function is assumed to extract the ‘subject‘ ‘DN‘ field, described in Section 2.1.6, of a given certificate. If several such fields is present in a given certificate, the rightmost (i.e. least significant) must be returned by the function. The ‘[1..]‘ notation is used to describe an array being ”sliced” such that a new array is produced containing all but the first element of the original array. The algorithm asserts that any extracted identifier appears at the expected hierarchical level, apart from extracting the identifier itself.

Testing elements of an issuance hierarchy could be performed by comparing certificate DER representations byte by byte, or by comparing their DER hashes. The following pseudo-code describes how a service provider could assert that a given peer is a system belonging to a certain local cloud:

```
if (getCertificateProfileIdentifier(peer.chain) != "sy" || hash(peer.chain[1]) != hashOfRe
    throw "unauthorized";
}
```

Note that if either of the secure transports TLS or DTLS is used by a given Arrowhead peer, the ‘post_handshake_auth‘ extension (RFC 8446, Section 4.6.2), must be required in order for both peers of a connection to provide and validate certificates. The extension *should* always be required when Arrowhead systems communicate. Use of the ‘post_handshake_auth‘ extension is referred to as *client authentication* by many software libraries and applications, as the initiating peer, or *client*, is also authenticated by the responding peer, or *server*.

7 References

8 Revision History

8.1 Amendments

No.	Date	Version	Subject of Amendments	Author
1				

8.2 Quality Assurance

No.	Date	Version	Approved by
1			