PKCS #6:
Extended-Certificate Syntax Standard

RSA Data Security, Inc.
Redwood City, CA

June 3, 1991

Version 1.4[1]

1. Scope

This standard describes a syntax for extended certificates. An extended certificate consists of an X.509 public-key certificate [1] as revised by RFC 1114 [3] and a set of attributes, collectively signed by the issuer of the X.509 public-key certificate. Thus the attributes and the enclosed X.509 public-key certificate can be verified with a single public-key operation, and an ordinary X.509 certificate can be extracted if needed, e.g., for Privacy-Enhanced Mail [2,3].

The intention of including a set of attributes is to extend the certification process beyond just the public key to include other information about a given entity, such as electronic mail address. A non-exhaustive list of attributes is given in PKCS #9 [4].

The preliminary intended application of this standard is in the digital signature and envelope syntax standard [5], but it is expected that other applications will be developed.

2. References

[1] CCITT. *Recommendation X.509: The Directory—Authentication Framework.* 1988.

[2] J. Linn. *RFC 1113: Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures.* August 1989. See also J. Linn. *RFC [1113D]: Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encryption and Authentication Procedures.* Internet draft, March 1991.

---

[1] Versions 1.0–1.3 are superseded by this version. The earlier versions were distributed to participants in RSA Data Security, Inc.'s Public-Key Cryptography Standards meetings in February and March 1991.

[3] S. Kent and J. Linn. *RFC 1114: Privacy Enhancement for Internet Electronic Mail: Part II -- Certificate-Based Key Management.* August 1989. See also S. Kent. *RFC [1114B]: Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management.* Draft, February 1991.

[4] RSA Data Security, Inc. *PKCS #9: Selected Attribute Types.* Version 1.0, June 1991.

[5] RSA Data Security, Inc. *PKCS #7: Cryptographic Message Syntax Standard.* Version 1.4, June 1991.

[6] CCITT. *Recommendation X.501: The Directory—Models.* 1988.

[7] CCITT. *Recommendation X.208: Specification of Abstract Syntax Notation One (ASN.1).* 1988.

[8] CCITT. *Recommendation X.209: Specification of Basic Encoding Rules for Abstract Syntax Notation One (ASN.1).* 1988.

[9] CCITT. *Recommendation X.500: The Directory—Overview of Concepts, Models and Services.* 1988.

[10] NIST. *Special Publication 500-183: Stable Implementation Agreements for Open Systems Interconnection Protocols.* Version 4, Edition 1, Part 11, December 1990.

[11] RSA Data Security, Inc. *PKCS #1: RSA Encryption Standard.* Version 1.4, June 1991.

3. Definitions

For the purposes of this standard, the following definitions apply.

`AlgorithmIdentifier`: A type that identifies an algorithm (by object identifier) and any associated parameters. This type is defined in X.509 [1].

`Attribute`: A type that contains an attribute type (specified by object identifier) and one or more attribute values. This type is defined in X.501 [6].

ASN.1: Abstract Syntax Notation One, as defined in X.208 [7].

BER: Basic Encoding Rules, as defined in X.209 [8].

`Certificate`: A type that binds an entity's distinguished name to a public key with a digital signature. This type is defined in X.509 [1] as revised by RFC 1114 [3]. This type also contains the distinguished name of the certificate issuer (the signer), an issuer-specific serial number, the issuer's signature algorithm identifier, and a validity period. Appendix A gives more information. Throughout this standard, the term "X.509 certificate" refers to the revised syntax in RFC 1114 [3].

DER: Distinguished Encoding Rules for ASN.1, as defined in X.509 [1], Section 8.7.

`Name`: A type that uniquely identifies or "distinguishes" objects in a X.500 directory [9]. This type is defined in X.501 [6]. In an X.509 certificate, the type identifies the certificate issuer and the entity whose public key is certified.

4. Symbols and abbreviations

No symbols or abbreviations are defined in this standard.

5. General overview

The next section specifies extended-certificate syntax. An appendix reviews the meaning of X.509 certificates.

This standard exports one type, `ExtendedCertificate`.

6. Extended-certificate syntax

This section gives the syntax for extended certificates.

An extended certificate consists of three parts: "extended certificate information," a signature algorithm identifier, and a digital signature on the extended-certificate information. The extended certificate information consists of an X.509 certificate (already signed by an issuer) and a set of attributes providing other information about the entity whose public key is certified in the X.509 certificate. The issuer that signs the extended certificate is the same as the one that signs the X.509 certificate.

The process by which an extended certificate is constructed involves the following steps:

1. An `ExtendedCertificateInfo` value containing an X.509 certificate and a set of attributes is constructed by a certificate issuer.

2. The `ExtendedCertificateInfo` value is signed with the certificate issuer's private key.

3. The `ExtendedCertificateInfo` value, a signature algorithm identifier, and the certificate issuer's signature are collected together into an `ExtendedCertificate` value, defined below.

This section is divided into two parts. The first part describes the extended-certificate-information type `ExtendedCertificateInfo`, and the second part describes the top-level type `ExtendedCertificate`.

Notes.

1. In applications where an extended certificate or an X.509 certificate can be processed, the following syntax is recommended (but not required):

```
ExtendedCertificateOrCertificate ::= CHOICE {
  certificate Certificate, -- X.509/RFC 1114 certificate
  extendedCertificate [0] IMPLICIT ExtendedCertificate }
```

   The `certificate` alternative has the same BER encoding as an ordinary X.509 certificate, and the values resulting from the two alternatives can be distinguished because of the context-specific tag 0 on the `extendedCertificate` alternative.

2. There are at least four reasons that extended certificates are built on top of X.509 certificates, rather than in place of them:

   - changes to X.509 certificate syntax are easily followed
   - an X.509 certificate can be extracted for compatibility with other standards (e.g., Privacy-Enhanced Mail [2,3], or X.509 itself)
   - both the X.509 certificate and the extended certificate can be verified with a single public-key operation, since they are signed together by the same certificate issuer
   - there is little redundancy—having an extended certificate in place of an X.509 certificate would require that the extended certificate contain much of the information already in the X.509 certificates, such as issuer and subject names

6.1 ExtendedCertificateInfo

Extended-certificate information shall have ASN.1 type `ExtendedCertificateInfo`:

```
ExtendedCertificateInfo ::= SEQUENCE {
  version Version,
  certificate Certificate,
  attributes Attributes }

Version ::= INTEGER

Attributes ::= SET OF Attribute
```

The fields of type `ExtendedCertificateInfo` have the following meanings:

- `version` is the version number, for compatibility with future revisions of this standard. It shall be 0 for this version of the standard.
- `certificate` is an X.509 certificate.
- `attributes` is a set of attributes. These attributes are additional information about the subject of the certificate. Some attribute types that might be useful here are defined in PKCS #9 [4].

6.2 ExtendedCertificate

An extended certificate shall have ASN.1 type `ExtendedCertificate`:

```
ExtendedCertificate ::= SEQUENCE {
  extendedCertificateInfo ExtendedCertificateInfo,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature Signature }

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier

Signature ::= BIT STRING
```

The fields of type `ExtendedCertificate` have the following meanings:

- `extendedCertificateInfo` is the "extended certificate information." It is the value being signed.
- `signatureAlgorithm` identifies the signature algorithm (and any associated parameters) under which the extended-certificate information is signed. Examples include PKCS #1's `md2WithRSAEncryption` and `md4WithRSAEncryption` [10].
- `signature` is the result of signing the extended-certificate information with the certificate issuer's private key.

The signature process consists of two steps:

1. The value of the `extendedCertificateInfo` field is DER encoded, yielding an octet string.
2. The result of step 1 is signed with the certificate issuer's private key under the specified signature algorithm, yielding a bit string, the signature.

Note. The syntax for `ExtendedCertificate` could equivalently be written with the X.509 `SIGNED` macro:

```
ExtendedCertificate ::= SIGNED ExtendedCertificateInfo
```

Appendix A. X.509 public-key certificates

Public-key certificates such as X.509 certificates determine the skeletal structure of trust within a distributed public-key cryptosystem. By signing a certificate, a certificate issuer binds together an entity's public key with the entity's name and other information. By verifying the signature on the certificate, someone who trusts the certificate issuer can develop trust in the entity's public key.

Because certificates are the most important part of an interoperable public-key standard, PKCS has adopted the use of X.509 certificates. This approach maintains compatibility with other users of the X.509 standard—in particular, with Internet Privacy Enhanced Mail.

This appendix describes the syntax of X.509 certificates, for reference purposes. To be precise, it describes the modified syntax given in RFC 1114 [3], which has worked its way back to the X.509 standard.

An X.509 certificate consists of three parts: "certificate information," a signature algorithm identifier, and a digital signature on the certificate information. The certificate information consists of an entity's distinguished name, the entity's public key, the distinguished name of the certificate issuer, an issuer-specific serial number, a signature algorithm identifier, and a validity period.

The process by which the certificate is constructed involves the following steps:

1. A `CertificateInfo` value containing certificate information is constructed by the certificate issuer.

2. The `CertificateInfo` value is signed with the certificate issuer's private key.

3. The `CertificateInfo` value, a signature algorithm identifier, and the certificate issuer's signature are collected together into a `Certificate` value, defined below.

The following discussion is divided into two parts. The first part describes the certificate-information type `CertificateInfo`, and the second part describes the top-level type `Certificate`.

A.1 CertificateInfo

Certificate information has ASN.1 type `CertificateInfo`:

```
CertificateInfo ::= SEQUENCE {
  version [0] Version DEFAULT v1988,
  serialNumber CertificateSerialNumber,
  signature AlgorithmIdentifier,
  issuer Name,
  validity Validity,
  subject Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo }

Version ::= INTEGER { v1988(0) }

CertificateSerialNumber ::= INTEGER

Validity ::= SEQUENCE {
  notBefore UTCTime,
  notAfter UTCTime }

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

The fields of type `CertificateInfo` have the following meanings:

- `version` is the version number, for compatibility with future revisions of X.509. Its default value is `v1988`, to which the `Version` type assigns the integer 0. The `[0]` tag on `version` is an explicit tag. This is the default for tags not marked `EXPLICIT` or `IMPLICIT` in the ASN.1 module which defines the `Certificate` type.

- `serialNumber` is the issuer-specific serial number of the certificate. Every certificate for a particular issuer must have a different serial numbers.

- `signature` identifies the issuer's signature algorithm (and any associated parameters).[2]

- `issuer` is the distinguished name of the certificate issuer.

- `validity` is the validity period for the certificate. The validity period specifies the points in time between which the certificate is considered valid.

---

[2]The field name `signature` seems somewhat of a misnomer, and `signatureAlgorithm` would be more appropriate, but this is the way X.509 does it.

- `subject` is the distinguished name of the certificate subject (the entity whose public key is certified).

- `subjectPublicKeyInfo` contains information about the public key being certified. The information identifies the entity's public-key algorithm (and any associated parameters); examples of public-key algorithms include X.509's `rsa` and PKCS #1's `rsaEncryption` [10]. The information also includes a bit-string representation of the entity's public key. For both public-key algorithms just mentioned, the bit string contains the BER encoding of a value of X.509/PKCS #1 type `RSAPublicKey`.

A.2 Certificate

An X.509 certificate has ASN.1 type `Certificate`:

```
Certificate ::= SEQUENCE {
  certificateInfo CertificateInfo,
  signatureAlgorithm AlgorithmIdentifier,
  signature BIT STRING }
```

The fields of type `Certificate` have the following meanings:

- `certificateInfo` is the "certificate information." It is the value being signed.

- `signatureAlgorithm` identifies the signature algorithm (and any associated parameters) under which the certificate information is signed. Examples include NIST/OIW's `md2WithRSA` [10] and PKCS #1's `md2WithRSAEncryption` and `md4WithRSAEncryption` [11]. The value of this field should be the same as the value of the `signature` field of the certificate information.

- `signature` is the result of signing the certificate information with the certificate issuer's private key.

The signature process consists of two steps, as in Section 6.2:

1. The value of the `certificateInfo` field is DER encoded, yielding an octet string.

2. The result of step 1 is signed with the certificate issuer's private key under the specified signature algorithm, yielding a bit string, the signature.

Note. The syntax for `Certificate` is usually written with the X.509 `SIGNED` macro.

```
Certificate ::= SIGNED SEQUENCE {
  version [0] Version DEFAULT v1988,
  serialNumber CertificateSerialNumber,
  signature AlgorithmIdentifier,
  issuer Name,
  validity Validity,
  subject Name,
  subjectPublicKeyInfo SubjectPublicKeyInfo }
```

The `SIGNED` macro has been expanded in this discussion for simplicity.