Certificate and CRL Profiles

Candidate Version 1.1 – 15 Jun 2004

Open Mobile Alliance

OMA-Security-CertProf-V1_1-20040615-C

Continues the Technical Activities
Originated in the WAP Forum

Contents

1. Scope

This document specifies Open Mobile Alliance (OMA) WAP Certificate and CRL profiles. It is based on work done within IETF's PKIX working group ([15]). The term "WAP server" used here is not limited to WAP gateways but may include third party servers and content/service provider servers processing certificates conforming to this specification.

2. References

2.1 Normative References

[1] American National Standard X9.62, "*Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm,*" Accredited Standards Committee X9F1, 1999.

[2] T. Berners-Lee, L. Masinter, M. McCahill, "*Uniform Resource Locators (URL)*," IETF RFC 1738, December 1994. URL: ftp://ftp.isi.edu/in-notes/rfc1738.txt.

[3] S. Bradner, "*Key words for use in RFCs to Indicate Requirement Levels*," IETF RFC 2119, March 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2119.txt.

[4] Dierks, T., C. Allen, "*The TLS Protocol Version 1.0*," IETF RFC2246, January 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2246.txt.

[5] ITU-T Recommendation X.500 (1997) | ISO/IEC 9594-1:1998, "*Information Technology – Open Systems Interconnection – The Directory: Overview of concepts, models and services*."

[6] ITU-T Recommendation X.501 (1997) | ISO/IEC 9594-2:1998*, "Information Technology – Open Systems Interconnection – The Directory: Models*."

[7] ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1998, "*Information Technology – Open Systems Interconnection – The Directory: Authentication Framework*."

[8] ITU-T Recommendation X.509 (1997)/Cor. 1 (2000E) | ISO/IEC 9594-8:1998/Cor. 1: 2000(E), "*Technical Corrigendum 1*."

[9] ITU-T Recommendation X.520 (1997) | ISO/IEC 9594-6:1998, "*Information Technology – Open Systems Interconnection – The Directory: Selected Attribute Types*."

[10] ITU-T Recommendation X.680 (1997) | ISO 8824-1:1998, "*Information Technology — Abstract Syntax Notation One (ASN.1): Specification of basic notation*."

[11] ITU-T Recommendation X.681 (1997) | ISO/IEC 8824-2:1998, "*Information Technology – Abstract Syntax Notation One (ASN.1): Information Object Specification*."

[12] ITU-T Recommendation X.682 (1997) | ISO/IEC 8824-3:1998, "*Information Technology – Abstract Syntax Notation One (ASN.1): Constraint Specification*."

[13] ITU-T Recommendation X.683 (1997) | ISO/IEC 8824-4:1998, "*Information Technology – Abstract Syntax Notation One (ASN.1): Parameterization of ASN.1 Specifications*."

[14] ITU-T Recommendation X.690 (1997) | ISO/IEC 8825-1:1998, "*Information Technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).*"

[15] R. Housley, W. Ford, W. Polk, D.Solo, "*Internet X.509 Public Key Infrastructure – Certificate and CRL Profile*," IETF RFC 3280, April 2002. URL: ftp://ftp.isi.edu/in-notes/rfc2459.txt.

[16] RSA Laboratories*, "PKCS #1: RSA Cryptography Standard*," Version 2.1, June 2002. URL: http://www.rsalabs.com/pkcs.

[17] "*Wireless Transport Layer Security Specification*," WAP Forum™, WAP-261-WTLS. URL: http://www.openmobilealliance.org/.

[18] "*Wireless Identity Module Specification – Part: Security*," Open Mobile Alliance™, OMA-WAP-WIM-v1_1. URL: http://www.openmobilealliance.org/.

[19] "*WMLScript Crypto Library*," WAP Forum™, WAP-161-WMLScriptCrypto. URL: http://www.openmobilealliance.org.

[20] "*Specification of WAP Conformance Requirements*," WAP Forum™, WAP-221-CREQ. URL: http://www.openmobilealliance.org/.

[21] "*Wireless Telephony Application Specification*," WAP Forum™, WAP-266-WTA. URL: http://www.openmobilealliance.org/.

[22] "*Signed Content*," Open Mobile Alliance™, Open Mobile Alliance™, OMA-Security-SignedContent-v1_0. URL: http://www.openmobilealliance.org

2.2 Informative References

[23] L. Bassham, R. Housley, W. Polk, "*Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL profile*," IETF RFC 3279, April 2002. URL: ftp://ftp.isi.edu/in-notes/rfc3279.txt.

[24] S. Kille, M. Wahl, A. Grimstad, R. Huber, S. Sataluri, "*Using Domains in LDAP/X.500 Distinguished Names*," IETF RFC 2247, January 1998. URL: ftp://ftp.isi.edu/in-notes/rfc2247.txt.

[25] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams*, "Internet X.509 Public Key Infrastructure – Online Certificate Status Protocol – OCSP*," IETF RFC 2560, June 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2560.txt.

[26] RSA Laboratories, "PKCS #9: Selected Object Classes and Attribute Types," Version 2.0, February 2000. URL: http://www.rsalabs.com/pkcs.

[27] M. Wahl, T. Howes, S. Kille, "*Lightweight Directory Access Protocol (v3)*," IETF RFC 2251, December 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2251.txt.

[28] M. Wahl, S. Kille, T. Howes, *"Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names*," IETF RFC 2253, December 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2253.txt.

[29] "*Wireless Application Protocol Architecture Specification*," WAP Forum™, WAP-210-WAPArch. URL: http://www.openmobilealliance.org/.

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [3].

All sections and appendixes, except “Scope” and “Requirements and Assumptions”, are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

The following are terms and conventions used throughout this specification.

In this document, the term “Recognize” stands for “object is parsed as needed, and its value may thereafter be processed, displayed or ignored.” The term “Process” means “Understand and act in accordance with a given semantics.”

3.3 Abbreviations

ASN.1	Abstract Syntax Notation One, as defined in [10], [11], [12] and [13]
CA	Certification Authority
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules, as defined in [14]
EC	Elliptic Curve
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
LDAP	Lightweight Directory Access Protocol
ME	Mobile Equipment
OCSP	Online Certificate Status Protocol
OMA	Open Mobile Alliance
RSA	Rivest-Shamir-Adleman public key algorithm
TLS	Transport Layer Security
URL	Uniform Resource Locator
WAP	Wireless Application Protocol
WIM	WAP Identity Module
WTLS	Wireless Transport Layer Security

4. Requirements and Assumptions

4.1 Assumptions on the WAP Environment

For the purposes of this specification, the WAP environment may be characterized in the following way:

- Limited bandwidth between WAP clients and WAP servers
- Limited computational capabilities in WAP clients
- Limited memory resources in WAP clients

Further, a reasonable assumption is that WAP servers are, in many cases, connected to, and inter-operating with, the Internet. A certificate profile for use in this environment should therefore, to the extent possible, take this characterization into account.

4.2 General Requirements on WAP Certificate Profiles

This section specifies general requirements on certificates, which are to be used in the WAP environment (as characterized in Section 4.1) and which may be transmitted in WAP protocols.

4.2.1 Reduced Footprint

When defining a certificate profile for WAP, the storage requirements for the certificate need to be reduced, but without loss of the functionality that makes the certificate meaningful and useful. The certificate must, within an appropriate context, identify the holder of the public key. It must also provide a secure binding between the key and its holder.

4.2.2 Limited Processing Requirements

When developing software, it is often possible to reduce the memory requirements of an application by increasing the processing time. This type of time-memory tradeoff cannot be used extensively for WAP certificates, since WAP clients (and in particular the WIM [18] card) have not only restricted memory size but also relatively limited processing power.

In addition to the burden on the constrained WAP client, additional computation requirements may also cause problems for WAP servers that interact with them. While WAP servers in general are free of the memory and processing limitations of WAP clients, they must perform operations on behalf of a large number of clients in a small amount of time. Any significant increase in the time required to process a certificate could impair the server's ability to process transactions at the needed rate.

Some additional processing, above that required for existing certificate formats, may be required. In particular, a more computationally intensive data encoding method may be used. This additional processing required for encoding should be small in comparison to that required for public or private key operations.

4.2.3 Security

Public key certificates are used to provide a secure binding between an entity and its public key. Any modification to the certificate format must provide at least the same level of security as existing certificates.

In any new format, the security of the data representation must be examined as well. Signatures must be properly padded to avoid possible forgery, and parameter specifications must have sufficient integrity protection to avoid substitution attacks.

4.2.4 Compatibility with Existing Infrastructure

To the extent possible, certificates issued in conformance with this specification should work interchangeably with other X.509 [7] certificates in certificate-processing Internet applications in order to leverage the existing infrastructure. Any new format that requires major changes to the installed base of certificate-processing products and CA infrastructure is unlikely to be easily adopted.

5. Certificate Profiles

5.1 General

This section defines WAP certificate profiles. The profiles are, unless otherwise mentioned, based on the Internet Certificate Profile [15], which in turn is based on the format defined in [7]. For full implementation of this section implementers are required to consult the underlying format and semantics defined in [7] and [15]. This specification provides, for each certificate type discussed, additional details regarding the contents of some individual fields in the certificate. Certificates issued in conformance with recommendations and requirements in this section will be reasonably compact, and MEs MUST be able to process certificates of size up to at least 700 bytes, while other certificate-processing entities MUST be able to process certificates of size up to at least 2000 bytes. MEs that support X.509-based server authentication or signed content MUST be able to process end-entity certificates of size up to at least 1000 bytes and CA certificates of size up to at least 2000 bytes, in addition to requirements listed in Section 5.4 and/or 5.5, and SHOULD be able to process longer certificates (NOTE: Because many TLS server certificates currently in use have a length over 1000 bytes it is highly recommended that MEs have the ability to process certificates of at least 1500 bytes to ensure broad interoperability with existing servers). Certificate-processing clients MUST support a certificate chain depth of at least three (i.e., two subordinate CA certificates between the end-entity certificate and the CA root certificate in the chain). A client that encounters a certificate or certificate chain that does not conform to this profile must not fail the certificate processing in an uncontrolled manner. In addition, certificate-processing servers must also support a chain depth of at least three.

ASN.1 definitions relevant for this section that are not supplied by the normative references can be found in Appendix A.

5.2 User Certificates for Authentication

This certificate type is intended for client authentication, e.g. in WTLS ([17]) or TLS ([4]). The certificate profile is intended for certificates stored in WAP clients such as handsets and WIM cards. Since it will not always be the case that the identity of the cardholder is known at the time of certificate issuance, some certificates will bind a public key to some distinguishable entity (e.g. a particular WIM card), rather than to a specific subscriber.

5.2.1 Certificate Serial Number

CAs should avoid using serial numbers longer than 8 bytes (63 bits, topmost bit cannot be set to **1**).

5.2.2 Signature (Algorithm)

The only signature algorithms defined for use with this profile are **sha1WithRSAEncryption** and **ecdsa-with-SHA1**. Certificate-processing applications MUST be able to verify certificates signed with one of these algorithms (certificate-processing applications need only support one of them).

5.2.3 Issuer (Name)

Applications MUST recognize all required distinguished name attributes listed in Section 4.1.2.4 in [15] (i.e. attributes **countryName**, **organizationName**, **organizationalUnitName**, **stateOrProvinceName**, **commonName** and **domainComponent**) and SHOULD recognize all other attributes listed in Section 4.1.2.4 in [15]. Further, they MUST recognize the **serialNumber** attribute defined in Section 7 in this document. CAs are not required to issue certificates with the **serialNumber** attribute, but they should be able to do so. When including attributes with **DirectoryString** syntax, CAs should use the **UTF8String** choice, and must do so for certificates issued after December 31, 2003 (see [15]). This field must not be left empty.

5.2.4 Subject (Name)

As for the Issuer field, applications MUST recognize all required distinguished name attributes listed in Section 4.1.2.4 in [15] (see above), and SHOULD recognize all other attributes listed in Section 4.1.2.4 in [15]. Further, they MUST recognize the **serialNumber** attribute defined in Section 7 in this document. CAs are not required to issue certificates with the **serialNumber** attribute, but are encouraged to do so, especially for initial certificates stored in WAP clients. In those cases, this specification also recommends that it be the only attribute in subject names. Such practice will result in compact certificates. Whatever the practice in those cases is, the certificate must bind the public key to a distinguishable entity (e.g. a

particular WIM card or other key storage). When including attributes with **DirectoryString** syntax, CAs should use the **UTF8String** choice, and must do so for certificates issued after December 31, 2003. This field must not be left empty.

5.2.5 Subject Public Key

The only public key types defined for use with this specification are **rsaEncryption** and **id-ecPublicKey** (see Section 8). RSA keys should be 1024 bits or longer. EC public keys should be 160 bits or longer. Certificate-processing applications are not required to handle keys longer than 2048 (RSA) or 163 (EC) bits.

NOTE – While using 2048-bit RSA keys in conjunction with SHA-1-based signatures does not add any security over 1024-bit RSA keys, the requirement is included here for legacy reasons

5.2.6 Certificate Extensions

Certificate-processing applications MUST recognize the following standard extensions: **keyUsage**, **extKeyUsage**, **certificatePolicies**, **subjectAltName**, and **basicConstraints**. Further, they SHOULD recognize the **nameConstraints**, **policyConstraints**, **authorityKeyIdentifier** and **subjectKeyIdentifier** extensions. If the **keyUsage** extension is included, it shall have the **digitalSignature** bit set if the public key is an RSA key. If the public key is an EC-DH key, it shall have the **keyAgreement** bit set. For RSA keys, the extension may also have the **keyEncipherment** bit set. Other bits must not be set. The **keyUsage** extension should be marked as critical.

NOTE 1 – The choice between the **keyEncipherment** bit and the **keyAgreement** bit depends on the particular public key algorithm; for RSA, it shall be **keyEncipherment**, for EC, it shall be **keyAgreement**.

NOTE 2 – This specification recommends that the **certificatePolicies** extension only consist of one **CertPolicyId**. Conforming certificate-processing applications are not required to recognize **policyQualifiers**.

CAs should not include the **basicConstraints** extension.

CAs should, if including the **subjectKeyIdentifier** and/or **authorityKeyIdentifier** extension, use the **KeyIdentifier** field, and calculate the value of that field in accordance with the procedure defined in Section 9.4.4 of [18].

Further, certificate-processing applications SHOULD recognize the **domainInformation** extension defined in Section 9. CAs may include this extension in issued certificates.

NOTE – A CA, which does not include any extensions in issued certificates, shall set the certificate version to 1 (i.e. the default value).

5.3 User Certificates for Digital Signatures

This certificate type is intended for verifications of digital signatures, created for example by *signText()* in WMLScript [19]. The certificate profile is intended for certificates stored in WAP clients such as handsets and WIM cards.

The requirements on this certificate type are identical to the type in Section 5.2, except that if the **keyUsage** extension is present (recommended), the only bits allowed to be set are the **digitalSignature** bit and/or the **nonRepudation** bit.

5.4 X.509-Compliant Server Certificates

5.4.1 Scope

This certificate type is intended for server authentication, e.g. in WTLS ([17]) or TLS ([4]). The certificate profile is intended for certificates sent over the air in WAP protocols

5.4.2 Certificate Serial Number

CAs must avoid using serial numbers longer than 20 bytes (159 bits since top-most bit must be set to **0**). Clients MUST be able to recognize serial number values up to 20 bytes long.

NOTE – A client, which does not perform certificate status checking, only needs to parse this field.

5.4.3 Signature (Algorithm)

The only signature algorithms defined for use with this profile are **sha1WithRSAEncryption** and **ecdsa-with-SHA1** (see Section 8). Clients MUST support at least one of these algorithms. Clients that support server-authenticated TLS sessions MUST support **sha1WithRSAEncryption**. Clients MUST be able to process certificates signed with keys up to and including 2048 bits (RSA) or 233 bits (EC).

NOTE – Because many TLS server certificates currently in use are signed with the **md5WithRSAEncryption** signature algorithm, it is highly recommended that MEs have the ability to process server certificates signed with this algorithm to ensure broad interoperability with existing servers.

5.4.4 Issuer (Name)

Clients MUST recognize all required distinguished name attributes listed in Section 4.1.2.4 in [15] (i.e. attributes **countryName**, **organizationName**, **organizationalUnitName**, **stateOrProvinceName**, **commonName**, and **domainComponent**) and SHOULD recognize all other attributes listed in Section 4.1.2.4 in [15]. Further, they MUST recognize the **serialNumber** attribute defined in Section 8 in this document. Clients MUST be able to process certificates (e.g. chain building) even if naming attributes are unknown. CAs are not required to issue certificates with the **serialNumber** attribute, but they should be able to do so. When including attributes with **DirectoryString** syntax, CAs should use the **UTF8String** choice, and must do so for certificates issued after December 31, 2003 (see [15]). This field must not be left empty.

5.4.5 Subject (Name)

As for the Issuer field, clients MUST recognize all required distinguished name attributes listed in Section 4.1.2.4 in [15] (see above), and SHOULD recognize all other attributes listed in Section 4.1.2.4 in [15]. Further, they MUST recognize the **serialNumber** attribute defined in Section 8 in this document. Clients MUST be able to process certificates (e.g. chain building) even if naming attributes are unknown. CAs are not required to issue certificates with the **serialNumber** attribute, but should be able to do so. When including attributes with **DirectoryString** syntax, CAs should use the **UTF8String** choice, and must do so for certificates issued after December 31, 2003. This field must not be left empty.

5.4.6 Subject Public Key

The only public key types defined for use with this specification are **rsaEncryption** and **id-ecPublicKey** (see Section 8). RSA keys should be 1024 bits or longer. EC public keys should be 160 bits or longer.

5.4.7 Certificate Extensions

Clients MUST recognize the following standard extensions, which may appear in server certificates: **keyUsage**, **extKeyUsage**, **authorityKeyIdentifier**, and **subjectAltName**,. Further, they SHOULD recognize the **certificatePolicies** and **authorityInfoAccess** extensions. Client certificate processing MUST NOT fail due to the presence of unrecognized, but non-critical, extensions.

For the **extKeyUsage** extension, clients MUST recognize the **id-kp-serverAuth** object identifier.

For the **authorityKeyIdentifier** extension, clients MUST recognize the **keyIdentifier** field.

For the **subjectAltName** extension, clients MUST recognize the **dNSName** and the **iPAddress** choices of the **GeneralName** type.

For the **certificatePolicies** extension, clients SHOULD recognize the **CPSuri** qualifier and the **UserNotice** qualifier, defined in [15], and SHOULD be able to process (i.e. retrieve and display) information conveyed in them.

Server certificates should contain the **authorityKeyIdentifier, keyUsage**, **extKeyUsage**, and **subjectAltName** extensions.

Server certificates that are intended for use in TLS sessions and containsthe **keyUsage** extension, must have key usage bits set in accordance with [4], Section 7.4.2. CAs should mark the **keyUsage** extension as critical.

Further, server certificates must include the **id-kp-serverAuth** object identifier in the **extKeyUsage** extension, when present, and are recommended to use the **dNSName** choice in the **subjectAltName** extension.

CAs should, if including the **subjectKeyIdentifier** and/or **authorityKeyIdentifier** extension, use the **KeyIdentifier** type, and calculate the value of that type in accordance with the procedure defined in Section 9.4.4 of [18].

5.5 Content Signing Certificates

5.5.1 Scope

This section specifies a profile for certificates to be used by content signers in conjunction with the WAP Signed Content specification [22]. This profile is general and is not specific to any particular content type.

5.5.2 Basic Certificate Requirements

The requirements for content signing certificates (except certificate extensions) are identical to those in Sections 5.4.1 - 5.4.6. The requirements for certificate extensions in content signing certificates are specified in the next section.

5.5.3 Certificate Extensions

Clients MUST recognize the following standard extensions: **keyUsage**, **extKeyUsage**, **authorityKeyIdentifier**, and **subjectAltName**. Further, they SHOULD recognize the **nameConstraints**, **certificatePolicies**, **policyConstraints**, **authorityInfoAccess**, **basicConstraints** and **subjectKeyIdentifier** extensions. Client certificate processing MUST NOT fail due to the presence of unrecognized, but non-critical, extensions.

For the **extKeyUsage** extension, clients MUST recognize the **id-kp-codeSigning** object identifier.

For the **authorityKeyIdentifier** extension, clients MUST recognize the **keyIdentifier** field.

For the **subjectAltName** extension, clients MUST recognize the **dNSName** choice of the **GeneralName** type. Clients SHOULD also be able to compare this name against the expected name of the origin server or content signer, if available. User-oriented devices SHOULD display a warning if the names do not match.

For the **certificatePolicies** extension, clients SHOULD recognize the **CPSuri** qualifier and the **UserNotice** qualifier, defined in [15], and SHOULD be able to process (i.e. retrieve and display) information conveyed in them.

Content signing certificates should contain the **authorityKeyIdentifier, keyUsage**, **extKeyUsage**, and **subjectAltName** extensions.

Content signing certificates that contain the **keyUsage** extension must have the **digitalSignature** bit set. CAs should mark the **keyUsage** extension as critical.

Further, content signing certificates should include the **id-kp-codeSigning** object identifier in the **extKeyUsage** extension when the subject is authorized to signed exectuable content, and are recommended to use the **dNSName** choice in the **subjectAltName** extension.

CAs should, if including the **subjectKeyIdentifier** and/or **authorityKeyIdentifier** extension, use the **KeyIdentifier** type, and calculate the value of that type in accordance with the procedure defined in Section 9.4.4 of [18].

5.5.4 Certificate Chain Requirements

If an end entity content signing certificate contains the **id-kp-codeSigning** object identifier in the **extKeyUsage** extension, then clients MUST only accept the certificate (for the purpose of verifying signature on executable content) if all CA certificates in the certificate path validating the end entity certificate (except the self-signed root certificate) also contain the **extKeyUsage** extension with the **id-kp-codeSigning** object identifier.

Content signing certificates issued in conformance with this profile which contain the **id-kp-codeSigning** object identifier in the **extKeyUsage** extension must validate with a certificate chain in which all CA certificates (except the self-signed root certificate) also contain the **extKeyUsage** extension with the **id-kp-codeSigning** object identifier

5.6 Authority Certificates

This section defines a profile for CA certificates. The certificate profile is intended for certificates stored in WAP clients such as handsets and WIM cards, or sent over-the-air in WAP protocols.

NOTE – When a WTLS Server Certificate has been issued by a CA, whose public key only exist in the form of an X.509-certificate, a certificate-processing application may use the following procedure to find the corresponding CA certificate:

- If the `WTLSCertificate.ToBeSignedCertificate.issuer.identifier_type` is `x509_name`, the application shall assume that the field contains a full, DER-encoded distinguished name from a corresponding subject field in an X.509-certificate, and use this as a basis for further search. If no matching certificate can be found, the chaining fails.
- If the `WTLSCertificate.ToBeSignedCertificate.issuer.identifier_type` is `key_hash_sha`, the application shall assume that the field contains a value to be found in the corresponding X.509-certificate's **subjectKeyIdentifier** extension or in the **CommonCertificateAttributes.identifiers** field for the CA certificate in question, if it is stored in a WIM. If no such certificate can be found, the chaining fails.

In all other cases (i.e. when the `issuer.identifier_type` is of type `text` or `binary`) the chaining procedure is undefined.

5.6.1 Certificate Serial Number

Same requirements as in Section 5.2.

5.6.2 Signature Algorithm

Same requirements as in Section 5.2.

5.6.3 Issuer (Name)

Same requirements as in Section 5.2.

5.6.4 Subject (Name)

In self-signed certificates, the subject name shall be the same as the issuer name. Apart from this, same requirements as in Section 5.2.

5.6.5 Subject Public Key

Same requirements as in Section 5.4. Keys must be at least 1024 (RSA) or 160 (EC) bits long. Clients MUST be able to process keys up to and including 2048 bits (RSA) or 233 bits (EC).

5.6.6 Certificate Extensions

Same requirements as in Section 5.2.6, except for the following differences:

- The **keyUsage** extension should be present, and must have at least the **keyCertSign** bit set if present.
- The **basicConstraint** extension must be present, and shall be critical. The **cA** component shall be set to **TRUE**, and the **pathLenConstraint** component need not be present.

Clients supporting X.509-based server authentication or content signing MUST be able to process the **basicConstraint** extension as well as the **subjectKeyIdentifier** extension

5.7 Other Certificates

This profile does not put any requirements on other certificates used in the WAP environment (i.e. certificates not transmitted in WAP protocols and not stored in WAP clients), but recommends that these certificates be issued in conformance with the profile in [15].

6. CRL Profiles

As it is not expected that X.509 CRLs will be sent over the air in WAP protocols, or stored (or processed) by MEs, this specification does not put any requirements on the format of these data structures, but recommends that they be issued in conformance with the profile in [15].

7. Attributes

7.1 Distinguished Attributes

This specification defines one certificate attribute for use in relative distinguished names. Applications MUST recognize this attribute. CAs are not required to include it in issued certificates, but are, as mentioned in Section 5, encouraged to do so under certain circumstances.

7.1.1 The serialNumber Attribute

The **serialNumber** attribute is intended to carry locally unique names (values). Use of this attribute provides for short subject names, while maintaining distinguished name requirements. In addition, some business and processing advantages may present themselves if the identifier for an entity, once assigned, remains invariant over a CA's lifetime. For example, this enables certificate-processing systems to identify a particular entity even when keys or other distinguished name attributes change for the entity. Privacy concerns must also be taken into account when considering such usage, however. CAs including this attribute in the subject name of an entity must not reuse the attribute value in certificates issued to other entities. When included in CA names, it can be a representation of the hash of the CA's key or some other unique value. The attribute is defined in [9] and reproduced here for reference purposes:

```
serialNumber ATTRIBUTE ::= {
        WITH SYNTAX PrintableString
        EQUALITY MATCHING RULE caseIgnoreMatch
        ID id-at-serialNumber
        }
```

7.2 WTLS Certificate reserved naming attribute types and values

This section lists WTLS certificate ([17], Section 10.5.2) naming attribute types and values that are "reserved," in the sense that other specifications MAY have special processing for PKI entities using them. CAs issuing certificates (or naming themselves) MUST ONLY use these reserved naming attribute values in accordance with the referenced specification.

- WTA entity: WTLS Certificate naming attribute "`T=wta`" ("`T=wta`" is the X.520 naming attribute **title** with value "**wta**"). Governing specification: [21].
- Certificate Authority: WTLS Certificate naming attribute "`T=ca`" ("`T=ca`" represents the X.520 naming attribute **title** with value "**ca**"). Governing specification: [17].

8. Signature Algorithms and Public-Key Types

Algorithms are represented in X.509-certificates with the following ASN.1 type

```
AlgorithmIdentifier {ALGORITHM:AlgorithmSet} ::= SEQUENCE {
        algorithm               ALGORITHM.&id ({AlgorithmSet}),
        parameters              ALGORITHM.&Type ({AlgorithmSet}{@algorithm}) OPTIONAL
        }

ALGORITHM ::= TYPE-IDENTIFIER
```

A reference to parameterized type **AlgorithmIdentifier{}** tightly binds a set of **algorithm** object identifiers to their associated **parameters** types.

8.1 Signature Algorithms

The following information object set may be augmented to meet local requirements. Note that deleting members of the set, or adding members to it, may prevent interoperability with conforming implementations.

```
SupportedSignatureAlgorithms ALGORITHM ::= {
        {NULL IDENTIFIED BY sha1WithRSAEncryption} |
        {NULL IDENTIFIED BY ecdsa-with-SHA1},
        ... -- For future (or local) extensions
        }
```

The object identifier value for **sha1WithRSAEncryption** can be found in [16]. When this algorithm is used, the signing and formatting shall be done in accordance with the procedure defined in [16].

The object identifier value for **ecdsa-with-SHA1** can be found in [1] (or [23]). When this algorithm is used, the signing and formatting shall be done in accordance with the procedure defined in [1]. Parameters for the used Elliptic Curve shall be included in the **subjectPublicKeyInfo** field of the certificate for the authority issuing the certificate in question.

8.2 Public-Key Types

The following information object set may be augmented to meet local requirements. Note that deleting members of the set, or adding members to it, may prevent interoperability with conforming implementations.

```
SupportedPublicKeyAlgorithms ALGORITHM ::= {
        {NULL IDENTIFIED BY rsaEncryption} |
        {Parameters IDENTIFIED BY id-ecPublicKey},
        ... -- For future extensions
        }
```

The object identifier value for **rsaEncryption** can be found in [16]. The format of the public key shall in this case be in accordance with [16].

The object identifier value for **id-ecPublicKey** can be found in [1]. The format of the public key shall in this case be in accordance with [1]. The **Parameters** type is defined in [1], and is repeated here for reference purposes:

```
Parameters ::= CHOICE {
        ecParameters            ECParameters,
        namedCurve              CURVES.&id({CurveNames}),
        implicitlyCA            NULL
        }
```

When EC parameters are inherited, the **parameters** field shall contain **implictlyCA**, which is the ASN.1 value **NULL**. This is the preferred option. When EC parameters are specified by reference, the **parameters** field shall contain the **namedCurve** choice, which is an object identifier. When EC parameters are explicitly included, they shall be encoded in the ASN.1 structure **ECParameters**. Whatever the choice, only those values consistent with the curves defined in [17] are allowed. For example, the basic curve assigned number 5 in [17] shall be identified as follows:

```
ansi-x9-62 OBJECT IDENTIFIER ::= {iso(1) member-body(2) us(840) 10045}

-- WTLS basic curve assigned number 5

parameter Parameters ::= namedCurve : {ansi-x9-62 curves(3) characteristic-two(0) 1}
```

9. Certificate Extensions

9.1 The domainInformation Extension

This extension carries information that pertain to the usage of this certificate and the domain in which it has been issued, such as:

- whether on-line status requests, for example using the OCSP ([25]) protocol, are required before using this certificate;
- the name of the domain root Certification Authority (optional); and
- an (optional) URL ([2]) pointing to a resource containing a DER-encoded ([14]) value of type **Extensions**, carrying more (non-critical) extensions linked to this certificate. The hash included in the certificate protects the integrity of this externally stored value.

This extension shall not be critical.

```
domainInformation EXTENSION ::= {
        SYNTAX                  DomainInformation
        IDENTIFIED BY           wap-ce-domainInformation
        }

DomainInformation ::= SEQUENCE {
        domainInfoFlags         DomainInfoFlags DEFAULT {onLineStatusRequest},
        domainAuthorityIdentifier   Name {{SupportedNamingAttributes}} OPTIONAL,
        otherExtensions         [0] ExtensionReference OPTIONAL,
        ... -- For future extensions
        } (CONSTRAINED BY {-- Critical extensions may not be referenced but must be explicitly included --})

DomainInfoFlags ::= BIT STRING {
        onLineStatusRequest     (0)
        }

ExtensionReference ::= SEQUENCE {
        url     IA5String, -- URL in accordance with [2] pointing to a DER-encoded value of type Extensions
        digest  Digest
        }

Digest ::= SEQUENCE {
        digestAlgorithm         AlgorithmIdentifier {{DigestAlgorithms}} DEFAULT sha1,
        digest                  OCTET STRING (SIZE(8..wap-ub-digest))

}
```

The **domainInfoFlags.onLineStatusRequest** bit, when set, indicates that a certificate-processing application should perform some on-line checking of the certificates' status before using it.

The **domainAuthorityIdentifier** field will, when present, contain the distinguished name of the domain root CA. Inclusion of this in an end-entity certificate may simplify policy-checking, certificate chain traversal, etc.

The **otherExtensions** field will, when present, contain a URL pointing to other (non-critical) extensions pertaining to this certificate, and a **digest** of those extensions (the digest shall be calculated on the whole DER-value, including tag and length, of an **Extensions** PDU).

10. Object Classes

This specification defines one new object class, **wapEntity**, in support of the attributes defined in Section 7. The purpose of this object class is to enable usage (e.g. storage, lookup) of those attributes in Directory-enabled environments. As an alternative, the **naturalPerson** object class, defined in [26], can be used.

10.1 The wapEntity Object Class

This auxiliary object class is intended to hold WAP-specific attributes for entities. It has been designed for use within directory services based on the LDAP protocol ([27]) and the X.500 family of protocols ([5], [6]), where support for WAP-defined attributes is considered useful.

```
wapEntity OBJECT-CLASS          ::=             {
        SUBCLASS OF { top }
        KIND auxiliary
        MAY CONTAIN { WAPEntityAttributeSet }
        ID wap-oc-wapEntity
        }

WAPEntityAttributeSet ATTRIBUTE ::= {
        serialNumber,
        … -- For future extensions
        }
```

Appendix A. ASN.1 module (Normative)

This annex includes all of the ASN.1 type, value, and information object class definitions contained in this specification, in the form of the ASN.1 module **WAPCertificateProfiles (**external type, value and class definitions are imported from modules in normative references)**.** To simplify strict type checking and adherence to profiles in this specification, the module contains a definition of the **WAPCertificate** type (compatible with the **Certificate** type defined in [7]) as well. This module could be input to an ASN.1 compiler for the purpose of verifying syntactic correctness or automatic generation of ASN.1-handling code.

```
WAPCertificateProfiles {joint-iso-itu-t(1) identified-organizations(23) wap(43) modules(0) certificate-profiles(1)}

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

-- EXPORTS All --

-- All types and values defined in this module are exported for use in other ASN.1 modules.

IMPORTS

informationFramework, selectedAttributeTypes, authenticationFramework, certificateExtensions
        FROM UsefulDefinitions {joint-iso-itu-t(2) ds(5) module(1) usefulDefinitions(0) 3}

ATTRIBUTE, OBJECT-CLASS, top
        FROM InformationFramework informationFramework

countryName, organizationName, organizationalUnitName, stateOrProvinceName, commonName,
        serialNumber
        FROM SelectedAttributeTypes selectedAttributeTypes

SIGNED, ALGORITHM, EXTENSION, Version, Validity
        FROM AuthenticationFramework authenticationFramework

keyUsage, extKeyUsage, certificatePolicies, subjectAltName, basicConstraints, nameConstraints,
        policyConstraints, authorityKeyIdentifier, subjectKeyIdentifier
        FROM CertificateExtensions certificateExtensions

rsaEncryption, sha1WithRSAEncryption
        FROM PKCS-1 {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-1(1) modules(0) pkcs-1(1)}

id-ecPublicKey, ecdsa-with-SHA1, Parameters
        FROM ANSI-X9-62 {iso(1) member-body(2) us(840) 10045 module(4) 1}

id-sha1
        FROM OIWSECSIGAlgorithmObjectIdentifiers {iso(1) identified-organization(3) oiw(14) secsig(3)
        oiwsecsigalgorithmobjectidentifiers(1)};

-- Upper bounds --

wap-ub-depth                    INTEGER ::= 5

wap-ub-width                    INTEGER ::= 1

wap-ub-extensions               INTEGER ::= 255

wap-ub-attributes               INTEGER ::= 255

wap-ub-publicKey                INTEGER ::= 5000

wap-ub-digest                   INTEGER ::= 255

-- Object Identifiers --

wap             OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) identified-organizations(23) 43}

wap-at          OBJECT IDENTIFIER ::= {wap 2} -- Attributes branch

wap-ce          OBJECT IDENTIFIER ::= {wap 3} -- Certificate extensions branch

wap-oc          OBJECT IDENTIFIER ::= {wap 4} – Object class branch

wap-ce-domainInformation        OBJECT IDENTIFIER ::= {wap-ce 1}
```

```
wap-oc-wapEntity                OBJECT IDENTIFIER ::= {wap-oc 1}


-- WAP certificate syntax --

WAPCertificateInfo ::= SEQUENCE {
        version                 [0] EXPLICIT Version DEFAULT v1,
        serialNumber            CertificateSerialNumber,
        signature               AlgorithmIdentifier {{SupportedSignatureAlgorithms}},
        issuer                  Name {{SupportedNamingAttributes}},
        validity                Validity,
        subject                 Name {{SupportedNamingAttributes}},
        subjectPublicKeyInfo    SubjectPublicKeyInfo {{SupportedPublicKeyAlgorithms}},
        extensions              [3] EXPLICIT Extensions {{SupportedExtensions}} OPTIONAL
                                -- if present, version must be v3
        }

WAPCertificate ::= SIGNED {WAPCertificateInfo}

CertificateSerialNumber ::= INTEGER -- Recommended to be less than 8 bytes

AlgorithmIdentifier {ALGORITHM:AlgorithmSet} ::= SEQUENCE {
        algorithm               ALGORITHM.&id ({AlgorithmSet}),
        parameters              ALGORITHM.&Type ({AlgorithmSet}{@algorithm}) OPTIONAL
        }

SupportedPublicKeyAlgorithms ALGORITHM ::= {
        {NULL IDENTIFIED BY rsaEncryption} |
        {Parameters IDENTIFIED BY id-ecPublicKey},
        ... -- For future extensions
        }

SupportedSignatureAlgorithms ALGORITHM ::= {
        {NULL IDENTIFIED BY sha1WithRSAEncryption} |
        {NULL IDENTIFIED BY ecdsa-with-SHA1},
        ... -- For future extensions
        }

Name {ATTRIBUTE: IOSet} ::= CHOICE {
        rdnSequence SEQUENCE SIZE (1..wap-ub-depth) OF RelativeDistinguishedName {{IOSet}}
        }

RelativeDistinguishedName {ATTRIBUTE : IOSet} ::= SET SIZE(1..wap-ub-width) OF
        AttributeTypeAndValue {{IOSet}}

AttributeTypeAndValue {ATTRIBUTE : IOSet} ::= SEQUENCE {
        type    ATTRIBUTE.&id ({IOSet}),
        value   ATTRIBUTE.&Type ({IOSet}{@type})
        }

SupportedNamingAttributes ATTRIBUTE ::= {
        countryName |
        organizationName |
        organizationalUnitName |
        stateOrProvinceName |
        commonName |
        domainComponent |
        serialNumber,
        ... -- For future extensions
        }

-- Defined here for easier access (see [24])

domainComponent ATTRIBUTE ::= {
        WITH SYNTAX IA5String
        ID      { 0 9 2342 19200300 100 1 25}
        }

SubjectPublicKeyInfo {ALGORITHM: IOSet} ::= SEQUENCE {
```

```
        algorithm                   AlgorithmIdentifier {{IOSet}},
        subjectPublicKey            BIT STRING (SIZE(1..wap-ub-publicKey))
        }

Extensions {EXTENSION : IOSet} ::= SEQUENCE SIZE (1..wap-ub-extensions) OF Extension {{IOSet}}

Extension {EXTENSION : IOSet} ::= SEQUENCE {
        extnId                      EXTENSION.&id ({IOSet}),
        critical                    BOOLEAN DEFAULT FALSE,
        extnValue                   OCTET STRING
        }(CONSTRAINED BY {-- extnValue must contain a DER-encoded value of type EXTENSION.&Extntype
        -- for the object identified  by extnId --})

domainInformation EXTENSION ::= {
        SYNTAX  DomainInformation
        IDENTIFIED BY               wap-ce-domainInformation
        }

DomainInformation ::= SEQUENCE {
        domainInfoFlags             DomainInfoFlags DEFAULT {onLineStatusRequest},
        domainAuthorityIdentifier   Name {{SupportedNamingAttributes}} OPTIONAL,
        otherExtensions             [0] ExtensionReference OPTIONAL,
        ... -- For future extensions
        } (CONSTRAINED BY {-- This extension should not be critical --})

DomainInfoFlags ::= BIT STRING {
        onLineStatusRequest         (0)
        }

ExtensionReference ::= SEQUENCE {
        url         IA5String, -- URL in accordance with pointing to a DER-encoded value of type Extensions
        digest      Digest
        }

Digest ::= SEQUENCE {
        digestAlgorithm             AlgorithmIdentifier {{DigestAlgorithms}} DEFAULT sha1,
        digest                      OCTET STRING (SIZE(8..wap-ub-digest))
        }

DigestAlgorithms ALGORITHM ::= {
        {NULL IDENTIFIED BY id-sha1},
        ... -- For future extensions
        }

sha1 AlgorithmIdentifier {{DigestAlgorithms}} ::= {algorithm id-sha1, parameters SHA1Parameters : NULL}

SHA1Parameters ::= NULL

SupportedExtensions EXTENSION ::= {
        domainInformation |
        keyUsage |
        extKeyUsage |
        certificatePolicies |
        subjectAltName |
        basicConstraints |
        nameConstraints |
        policyConstraints |
        authorityKeyIdentifier |
        subjectKeyIdentifier,
        ... -- For future extensions
        }

wapEntity OBJECT-CLASS              ::=                 {
        SUBCLASS OF { top }
        KIND auxiliary
        MAY CONTAIN { WAPEntityAttributeSet }
        ID wap-oc-wapEntity
        }
```

```
WAPEntityAttributeSet ATTRIBUTE ::= {
        serialNumber,
        … -- For future extensions
        }
END
```

Appendix B. Static Conformance Requirements (Normative)

The notation used in this appendix is specified in [20].

B.1 ME Options

B.1.1 General Certificate Options

This table specifies generic certificate-processing requirements for MEs. In the table, "M" stands for "Mandatory to implement" and "O" stands for "Optional."

NOTE – This subsection does not apply to ME implementations that never handles (receives, stores, etc.) certificates profiled in accordance with this document

Item	Function	Reference	Status	Requirements
Cert-Gen-C-001	General X.509 Certificate support - Parsing of fields as needed for functionality outlined below	5	M	
Cert-Gen-C-002	General X.509 Certificate support - Able to handle client certificates at least up to 700 bytes long	5	M	
Cert-Gen-C-003	Issuer Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.2, 5.3, 5.4	M	
Cert-Gen-C-004	Issuer Name - Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.2, 5.3, 5.4	O	
Cert-Gen-C-005	Issuer Name - Capable of displaying **PrintableString**, **UTF8String** and **NumericString** values	5.2, 5.3, 5.4	M	
Cert-Gen-C-006	Issuer Name - Recognize the **serialNumber** attribute	5.2, 5.3, 5.4	M	
Cert-Gen-C-007	Subject Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.2, 5.3, 5.4	M	
Cert-Gen-C-008	Subject Name -Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.2, 5.3, 5.4	O	
Cert-Gen-C-009	Subject Name - Capable of displaying **PrintableString, UTF8String** and **NumericString** values	5.2, 5.3, 5.4	M	
Cert-Gen-C-010	Subject Name - Recognize the **serialNumber** attribute	5.2, 5.3, 5.4	M	

B.1.2 X.509 Server Certificate options

This table specifies certificate-processing requirements for MEs that support X.509-based server authentication.

Item	Function	Reference	Status	Requirements
Cert-SrvA-C-001	General X.509 Certificate support - Parsing of all fields	5.1	M	
Cert-SrvA-C-002	General X.509 Certificate support - Able to process server certificates at least up to 1000 bytes long (CA certificates 2000 bytes)	5.4.1	M	
Cert-SrvA-C-003	General X.509 Certificate support - Capable of processing certificates with unknown distinguished name attributes (e.g. needed for chain building)	5.4.4 5.4.5	M	
Cert-SrvA-C-004	General X.509 Certificate support - Capable of processing certificates with unknown, non-critical certificate extensions	5.4.7	M	
Cert-SrvA-C-005	Verification - Certificate path processing as defined in [7] (and [8]), but subject to limitations in Section 5.1 and 5.4	5.1, 5.4	M	
Cert-SrvA-C-006	Serial Number - Handling of serial numbers up to 20 bytes long	5.4.2	M	
Cert-SrvA-C-007	Issuer Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.4.4	M	
Cert-SrvA-C-008	Issuer Name - Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.4.4	O	
Cert-SrvA-C-009	Issuer Name - Recognize the **serialNumber** attribute	5.4.4	M	
Cert-SrvA-C-010	Subject Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.4.4, 5.4.5	M	
Cert-SrvA-C-011	Subject Name - Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.4.4, 5.4.5	O	
Cert-SrvA-C-012	Subject Name - Recognize the **serialNumber** attribute	5.4.4, 5.4.5	M	
Cert-SrvA-C-013	Extensions - Recognize and process extensions as specified in this document: **keyUsage, subjectAltName, extKeyUsage, authorityKeyIdentifier**. For CA certificates, must also process the **basicConstraints** and **subjectKeyIdentifier** extension.	5.4.7 5.6.6	M	

Item	Function	Reference	Status	Requirements
Cert-SrvA-C-014	Extensions - Recognize and process extensions as specified in this document: **certificatePolicies, authorityInfoAccess**	5.4.7	O	
Cert-SrvA-C-015	Signature Algorithms - Capable of processing certificates signed with at least one of the algorithms specified in this document	5.4.3	M	Cert-SrvA-C-016 OR Cert-SrvA-C-017
Cert-SrvA-C-016	Signature Algorithms - Capable of verifying signatures made with RSA keys up to and including 2048 bits	5.4.3	O	
Cert-SrvA-C-017	Signature Algorithms - Capable of verifying signatures made with EC keys up to and including 233 bits	5.4.3	O	

B.1.3 TLS Certificate options

This table specifies further certificate-processing requirements for those MEs that support server-authenticated TLS sessions.

Item	Function	Reference	Status	Requirements
Cert-TLS-C-001	Signature Algorithms - Capable of verifying signatures made with RSA keys up to and including 2048 bits	5.4.3	M	

B.1.4 X.509 Content Signing Certificate options

This table specifies certificate-processing requirements for MEs that support X.509-based content signing.

Item	Function	Reference	Status	Requirements
Cert-ContSig-C-001	General X.509 Certificate support - Parsing of all fields	5.1	M	
Cert-ContSig-C-002	General X.509 Certificate support - Able to process server certificates at least up to 1000 bytes long (CA certificates 2000 bytes)	5.1	M	
Cert-ContSig-C-003	General X.509 Certificate support - Capable of processing certificates with unknown distinguished name attributes (e.g. needed for chain building)	5.5.2 5.4.4 5.4.5	M	
Cert-ContSig-C-004	General X.509 Certificate support - Capable of processing certificates with unknown, non-critical certificate extensions	5.5.3	M	
Cert-ContSig-C-005	Verification - Certificate path processing as defined in [7] (and [8]), but subject to limitations in Section 5.1, 5.4, and 5.5.	[7], 5.1, 5.4, 5.5	M	
Cert-ContSig-C-006	Serial Number -Handling of serial numbers up to 20 bytes long	5.5.2 5.4.2	M	

Item	Function	Reference	Status	Requirements
Cert-ContSig-C-007	Issuer Name -Recognize the following required RFC 2459 attributes: **countryName**, **organizationName**, **organizationalUnitName**, **commonName**, **stateOrProvinceName**, **domainComponent**	5.5.2 5.4.4	M	
Cert-ContSig-C-008	Issuer Name - Recognize all recommended RFC 2459 attributes: **localityName**, **title**, **surname**, **givenName**, **initials**, **generationQualifier**	5.5.2, 5.4.4	O	
Cert-ContSig-C-009	Issuer Name - Recognize the **serialNumber** attribute	5.5.2, 5.4.4	M	
Cert-ContSig-C-010	Subject Name - Recognize the following required RFC 2459 attributes: **countryName**, **organizationName**, **organizationalUnitName**, **commonName, stateOrProvinceName**, **domainComponent**	5.5.2, 5.4.4, 5.4.5	M	
Cert-ContSig-C-011	Subject Name - Recognize all recommended RFC 2459 attributes: **localityName**, **title**, **surname**, **givenName**, **initials**, **generationQualifier**	5.5.2, 5.4.4, 5.4.5	O	
Cert-ContSig-C-012	Subject Name - Recognize the **serialNumber** attribute	5.5.2, 5.4.4, 5.4.5	M	
Cert-ContSig-C-013	Extensions - Recognize and process extensions as specified in this document: **keyUsage**, **subjectAltName**, **extKeyUsage**, **authorityKeyIdentifier**. For CA certificates, must also process the **basicConstraints** and **subjectKeyIdentifier** extension.	5.5.3, 5.6.6	M	
Cert-ContSig-C-014	Extensions - Recognize and process extensions as specified in this document: **nameConstraints, certificatePolicies, policyConstraints, authorityInfoAccess, basicConstraints** and **subjectKeyIdentifier**	5.5.3	O	
Cert-ContSig-C-015	Signature Algorithms - Capable of processing certificates signed with at least one of the algorithms specified in this document	5.5.2, 5.4.3	M	Cert-ContSig-C-016 OR Cert-ContSig-C-017
Cert-ContSig-C-016	Signature Algorithms - Capable of verifying signatures made with RSA keys up to and including 2048 bits	5.5.2. 5.4.3	O	
Cert-ContSig-C-017	Signature Algorithms - Capable of verifying signatures made with EC keys up to and including 233 bits	5.5.2, 5.4.3	O	
Cert-ContSig-C-018	Certificate Chain Processing - Recognize and process the **id-kp-codeSigning** OID in the **extKeyUsage** extension in certificate path validation as specified in this document.	5.5.4	M	

B.2 Certificate-processing application Option

This section specifies requirements on certificate processing WAP applications not located in the ME, e.g. WTLS servers.

B.2.1 General Certificate Options

This table specifies generic certificate-processing requirements. In the table, "M" stands for "Mandatory to implement" and "O" stands for "Optional.

Item	Function	Reference	Status	Requirements
Cert-Gen-S-001	General X.509 Certificate support - Parsing of all fields	5	M	
Cert-Gen-S-002	General X.509 Certificate support - Able to handle certificates at least up to 2000 bytes long	5	M	
Cert-Gen-S-003	General X.509 Certificate support - Capable of processing certificates with unknown distinguished name attributes (e.g. needed for chain building)	5	M	
Cert-Gen-S-004	Verification - Certificate path processing as defined in [7] (and [8]).	5.1	M	
Cert-Gen-S-005	Issuer Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.2, 5.3	M	
Cert-Gen-S-006	Issuer Name - Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.2, 5.3	O	
Cert-Gen-S-007	Issuer Name - Recognize the **serialNumber** attribute	5.2, 5.3	M	
Cert-Gen-S-008	Subject Name - Recognize the following required RFC 2459 attributes: **countryName, organizationName, organizationalUnitName, commonName, stateOrProvinceName, domainComponent**	5.2, 5.3	M	
Cert-Gen-S-009	Subject Name - Recognize all recommended RFC 2459 attributes: **localityName, title, surname, givenName, initials, generationQualifier**	5.2, 5.3	O	
Cert-Gen-S-010	Subject Name - Recognize the **serialNumber** attribute	5.2, 5.3	M	
Cert-Gen-S-011	Extensions - Recognize and process extensions as specified in this document	5	M	
Cert-Gen-S-012	Extensions - Recognize and process the **domainInformation** extension	9	O	
Cert-Gen-S-013	Signature Algorithms - Capable of processing certificates signed with at least one of the algorithms specified in this document	8	M	Cert-Gen-S-014 OR Cert-Gen-S-015
Cert-Gen-S-014	Signature Algorithms - Capable of verifying signatures made with RSA keys up to and including 2048 bits	5.6.5	O	

Item	Function	Reference	Status	Requirements
Cert-Gen-S-015	Signature Algorithms - Capable of verifying signatures made with EC keys up to and including 233 bits	5.6.5	O	
Cert-Gen-S-016	Chain Processing - Process certificate chains of at least 3	5.1	M	

Appendix C. Certificate Examples (Informative)

ACKNOWLEDGEMENT – These examples have been developed with the help of the OSS ASN.1 compiler.

C.1 Example of a client certificate for authentication

The certificate is shown both as a dump of the **WAPCertificateInfo** contents and as a hexadecimal dump of the whole certificate.

```
WAPCertificateInfo SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 270
  serialNumber SerialNumber INTEGER: tag = [UNIVERSAL 2] primitive; length = 4
    1234567890
  signature SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
    algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 9
      { 1 2 840 113549 1 1 5 }
    parameters OpenType
      0x0500
  issuer SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 38
    SET OF: tag = [UNIVERSAL 17] constructed; length = 18
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 16
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 10 } -- OrganizationName
        value OpenType
          0x0c0941434d4520496e632e -- "ACME Inc."
    SET OF: tag = [UNIVERSAL 17] constructed; length = 16
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 14
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 3 } -- commonName
        value OpenType
          0x0c0754657374204341 -- "Test CA"
  validity Validity SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 30
    notBefore Time CHOICE
      utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
        000101110000Z
    notAfter Time CHOICE
      utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
```

```
        001101100000Z
  subject SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 15
    SET OF: tag = [UNIVERSAL 17] constructed; length = 13
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 11
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 5 } -- serialNumber
        value OpenType
          0x130431303031 -- "1001"
  subjectPublicKeyInfo SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 157
    algorithm SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
      algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 9
        { 1 2 840 113549 1 1 1 }
      parameters OpenType
        0x0500
    subjectPublicKey BIT STRING: tag = [UNIVERSAL 3] primitive; length = 139
     0x0030818702818100b8488400d4b6088be48ead459ca19ec717aaf3d1d4ee3ecca49612...
```

Hexadecimal dump of the signed certificate:

```
308201A5 3082010E 02044996 02D2300D 06092A86 4886F70D 01010505 00302631
12301006 0355040A 0C094143 4D452049 6E632E31 10300E06 03550403 0C075465
73742043 41301E17 0D303030 31303131 31303030 305A170D 30303131 30313130
30303030 5A300F31 0D300B06 03550405 13043130 30313081 9D300D06 092A8648
86F70D01 01010500 03818B00 30818702 818100B8 488400D4 B6088BE4 8EAD459C
A19EC717 AAF3D1D4 EE3ECCA4 96128A13 597D16CC 8B85EB37 EFCE110C 63B01E68
4E5CF632 291EAC60 FD153C26 6EAAC36A D4CEA923 19F9BFDD 261AD2BF E41EAB4E
17FE6783 41EE52D9 A0A8B4DE C07B7ACC 76762514 045CEE99 94E0CF37 BAE05F8D
E33B35FF 98BCE777 42CE4B12 273BD122 137FE902 0105300D 06092A86 4886F70D
01010505 00038181 00202BB7 D273C08B 9A0BF4D0 3B314FEB 2A30BC4E 4929DC30
A6CB3EA1 4760D991 A3C083A8 C59E33D8 A5A866F0 E94A33B0 92FA6A31 95D7E8C5
FD9B5E4E 673F2C1C 6ECF5C0D 511A905E C300F672 61774275 084DA194 FBF4F01C
BD9DABB7 CD32044F 350B0DC6 1081E68B 10AC2ACD 9E526312 D737B665 08FC48B0
A0074516 9E7FEC12 1E
```

C.2 Example of a CA certificate

The certificate can be used to verify the signature of the client certificate in the previous section.

```
WAPCertificateInfo SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 318
  version : tag = [0] constructed; length = 3
    Version INTEGER: tag = [UNIVERSAL 2] primitive; length = 1
      2
  serialNumber SerialNumber INTEGER: tag = [UNIVERSAL 2] primitive; length = 1
    1
  signature SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
    algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 9
      { 1 2 840 113549 1 1 5 }
    parameters OpenType
      0x0500
  issuer SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 38
    SET OF: tag = [UNIVERSAL 17] constructed; length = 18
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 16
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 10 } -- organizationName
        value OpenType
          0x0c0941434d4520496e632e -- "ACME Inc."
    SET OF: tag = [UNIVERSAL 17] constructed; length = 16
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 14
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 3 } -- commonName
        value OpenType
          0x0c0754657374204341 -- "Test CA"
  validity Validity SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 30
    notBefore Time CHOICE
      utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
        000101100000Z
    notAfter Time CHOICE
      utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
```

```
      001111100000Z
  subject SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 38
    SET OF: tag = [UNIVERSAL 17] constructed; length = 18
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 16
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 10 } -- organizationName
        value OpenType
          0x0c0941434d4520496e632e -- "ACME Inc."
    SET OF: tag = [UNIVERSAL 17] constructed; length = 16
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 14
        type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 4 3 } -- commonName
        value OpenType
          0x0c0754657374204341 "Test CA"
  subjectPublicKeyInfo SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 159
    algorithm SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
      algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 9
        { 1 2 840 113549 1 1 1 }
      parameters OpenType
        0x0500
    subjectPublicKey BIT STRING: tag = [UNIVERSAL 3] primitive; length = 141
      0x0030818902818100ad1f35964b3674c807b9f8a645d2c8174e514b69a4b46a7382915a...
  extensions : tag = [3] constructed; length = 19
    SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 17
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 15
        extnId OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 29 19 }-- basicConstraints
        critical BOOLEAN: tag = [UNIVERSAL 1] primitive; length = 1
          TRUE
        extnValue OCTET STRING: tag = [UNIVERSAL 4] primitive; length = 5
          0x30030101ff
```

Hexadecimal dump of the self-signed CA certificate:

```
308201D5 3082013E A0030201 02020101 300D0609 2A864886 F70D0101 05050030
```

```
26311230 10060355 040A0C09 41434D45 20496E63 2E311030 0E060355 04030C07
54657374 20434130 1E170D30 30303130 31313030 3030305A 170D3030 31313131
31303030 30305A30 26311230 10060355 040A0C09 41434D45 20496E63 2E311030
0E060355 04030C07 54657374 20434130 819F300D 06092A86 4886F70D 01010105
0003818D 00308189 02818100 AD1F3596 4B3674C8 07B9F8A6 45D2C817 4E514B69
A4B46A73 82915ABB C44ECCED E914DAE8 FCC023AB CEA9C533 80E64179 5CB0DDA6
64B872FC 109F9BBB 852BF42D 994F634C 681608E3 88DCE240 B558513E 5B60027B
D1A07CEF 9C9B6DB3 7C7E1F1A BD238EED 96E4B669 056B260F 55E83F14 E6027127
C9DEB3AD 18AFCD3F 8A5F5BF5 02030100 01A31330 11300F06 03551D13 0101FF04
05300301 01FF300D 06092A86 4886F70D 01010505 00038181 0029E927 40EC957B
313933B1 DD45BB7B 2DA2BC03 A4C40224 5B183E36 E4FE4FB8 948D1155 F47938CC
422C6D77 52F0FAAF FEC11E05 8B6945E3 D7FA8208 B2367A10 6BD546B7 33C235C9
D4DC21F5 B9A1903D B9B19D97 985DBABB 67146949 43C362ED 662872F4 A7C2C859
F6F47752 F25FABD3 E056A7AF E16A96F4 8FC7ADEA 92057A2D 5C
```

C.3 Example of a server certificate for server authentication

The certificate is shown both as an ASN.1 dump of the contents and as a hexadecimal dump of the whole certificate. The certificate contains a (critical) **keyUsage** extension, an **extKeyUsage** extension, a **subjectAltName** extension (**dNSName** alternative), and an **authorityKeyIdentifier** extension. The size of the certificate is 552 bytes.

ASN.1 dump of the certificate information:

```
WAPCertificateInfo SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 397
  version : tag = [0] constructed; length = 3
    Version INTEGER: tag = [UNIVERSAL 2] primitive; length = 1
      2
  serialNumber SerialNumber INTEGER: tag = [UNIVERSAL 2] primitive;
  length = 3
    5678901
  signature SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
    algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 9
      { 1 2 840 113549 1 1 5 } -- sha1WithRSAEncryption
    parameters OpenType -- NULL
      0x0500
  issuer SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 38
    SET OF: tag = [UNIVERSAL 17] constructed; length = 18
```

```
    SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 16
      type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
        { 2 5 4 10 } -- organizationName
      value OpenType -- UTF8String
        0x0c0941434d4520496e632e -- "ACME Inc."
  SET OF: tag = [UNIVERSAL 17] constructed; length = 16
    SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 14
      type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
        { 2 5 4 3 } -- commonName
      value OpenType -- UTF8String
        0x0c0754657374204341 -- "Test CA"
validity Validity SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 30
  notBefore Time CHOICE
    utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
      000101110000Z
  notAfter Time CHOICE
    utcTime UTCTime: tag = [UNIVERSAL 23] primitive; length = 13
      011101100000Z
subject SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 34
  SET OF: tag = [UNIVERSAL 17] constructed; length = 11
    SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 9
      type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
        { 2 5 4 6 } -- countryName
      value OpenType -- PrintableString
        0x13025553 -- "US"
  SET OF: tag = [UNIVERSAL 17] constructed; length = 19
    SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 17
      type OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
        { 2 5 4 10 } -- organizationName
      value OpenType -- PrintableString
        0x130a43657274732052205573 -- "Certs R Us"
subjectPublicKeyInfo SEQUENCE: tag = [UNIVERSAL 16] constructed;
length = 157
```

```
    algorithm SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 13
      algorithm OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive;
      length = 9
        { 1 2 840 113549 1 1 1 } -- rsaEncryption
      parameters OpenType -- NULL
        0x0500
    subjectPublicKey BIT STRING: tag = [UNIVERSAL 3] primitive;
    length = 139
      0x0030818702818100b8488400d4b6088be48ead459ca19ec717aaf3d1d4ee3ecc...
  extensions : tag = [3] constructed; length = 102
    SEQUENCE OF: tag = [UNIVERSAL 16] constructed; length = 100
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 14
        extnId OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 29 15 } -- keyUsage
        critical BOOLEAN: tag = [UNIVERSAL 1] primitive; length = 1
          TRUE
        extnValue OCTET STRING: tag = [UNIVERSAL 4] primitive; length = 4
          0x030205a0 -- digitalSignature, keyEncipherment
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 19
        extnId OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 29 37 } -- extKeyUsage
        extnValue OCTET STRING: tag = [UNIVERSAL 4] primitive; length = 12
          0x300a06082b06010505070301 -- {id-kp-serverAuth}
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 28
        extnId OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 29 17 } -- subjectAltName
        extnValue OCTET STRING: tag = [UNIVERSAL 4] primitive; length = 21
          0x301382117761702e63657274732d722d75732e7573
          -- dNSName : "wap.certs-r-us.us"
      SEQUENCE: tag = [UNIVERSAL 16] constructed; length = 31
        extnId OBJECT IDENTIFIER: tag = [UNIVERSAL 6] primitive; length = 3
          { 2 5 29 35 } -- authorityKeyIdentifier
        extnValue OCTET STRING: tag = [UNIVERSAL 4] primitive; length = 24
```

0x3016801400010203040506070809 0a0b0c0d0e0ffedcba98

Hexadecimal dump of the signed certificate:

30820224 3082018D A0030201 02020356 A735300D 06092A86 4886F70D 01010505
00302631 12301006 0355040A 0C094143 4D452049 6E632E31 10300E06 03550403
0C075465 73742043 41301E17 0D303030 31303131 31303030 305A170D 30313131
30313130 30303030 5A302231 0B300906 03550406 13025553 31133011 06035504
0A130A43 65727473 20522055 7330819D 300D0609 2A864886 F70D0101 01050003
818B0030 81870281 8100B848 8400D4B6 088BE48E AD459CA1 9EC717AA F3D1D4EE
3ECCA496 128A1359 7D16CC8B 85EB37EF CE110C63 B01E684E 5CF63229 1EAC60FD
153C266E AAC36AD4 CEA92319 F9BFDD26 1AD2BFE4 1EAB4E17 FE678341 EE52D9A0
A8B4DEC0 7B7ACC76 76251404 5CEE9994 E0CF37BA E05F8DE3 3B35FF98 BCE77742
CE4B1227 3BD12213 7FE90201 05A36630 64300E06 03551D0F 0101FF04 04030205
A0301306 03551D25 040C300A 06082B06 01050507 0301301C 0603551D 11041530
13821177 61702E63 65727473 2D722D75 732E7573 301F0603 551D2304 18301680
14000102 03040506 0708090A 0B0C0D0E 0FFEDCBA 98300D06 092A8648 86F70D01
01050500 03818100 530D71EC C3F44439 08125646 63709402 19555609 F3ECB411
D39DFD79 9F48A418 92EBC51D 2FF0EB3E 341CC834 B81DDC43 53B5FD4D D34760A7
12ECF610 20C77F0A D387A235 739C1D82 45C049B3 817D32DD 661C67BE A4588A52
68DB4156 669B92B2 DE66A4CE 57C4FDC8 ABDADCC3 5BD3EDDF 6F018B93 ACAD4AE6
E9637EC2 D379B48B

Appendix D. Change History (Informative)

D.1 Approved Version History

Reference	Date	Description
WAP-278-WAPCert-20011112-d	12 Nov 2001	The initial version of this document. Incorporates SCDs: WAP-211_100-WAPcert, WAP-211_101-WAPCert, WAP-211_102-WAPCert, WAP-211_103-WAPCert, WAP-211_104-WAPCert

D.2 Draft/Candidate Version 1.1 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-Security-CertProf-V1_1	16 Sep 2002	see WAP-211_105	OMA template alignment, incorporated WAP-211_105-WAPCert
	03 Mar 2004	All	Updated to latest OMA spec template (OMA-Template-SpecWAP-20040205)
Candidate Version OMA-Security-CertProf-V1_1	15 Jun 2004	n/a	Status changed to Candidate by TP TP ref # OMA-TP-2004-0193-WPKI-V1_0_for-candidate