WAP Signed Content
WAP-233-SCONT-20010531-t
Prototype
Version 31-May-2001

Wireless Application Protocol
Signed Content Specification

Disclaimer:

"This document is not a WAP Forum™ specification. This document is subject to revision or removal without notice. No part of this document may be used to claim conformance or interoperability with the WAP Forum™ specifications."
You may use this document or any part of the document for internal or educational purposes only, provided you do not modify, edit or take out of context the information in this document in any manner. You may not use this document in any other manner without the prior written permission of the WAP Forum™. The WAP Forum™ authorizes you to copy this document, provided that you retain all copyright and other proprietary notices contained in the original materials on any copies of the materials and that you comply strictly with these terms. This copyright permission does not constitute an endorsement of the products or services offered by you.

The WAP Forum™ assumes no responsibility for errors or omissions in this document. In no event shall the WAP Forum™ be liable for any special, indirect or consequential damages or any damages whatsoever arising out of or in connection with the use of this information.

This document is not a WAP Forum™ specification. This document is subject to revision or removal without notice. No part of this document may be used to claim conformance or interoperability with the WAP Forum™ specifications.
Contents

1. SCOPE ........................................................................................................................................... 3

2. DOCUMENT STATUS ................................................................................................................... 4
   2.1 DOCUMENT HISTORY ............................................................................................................. 5
   2.2 ERRATA .................................................................................................................................. 5
   2.3 COMMENTS .............................................................................................................................. 5

3. REFERENCES .................................................................................................................................. 6
   3.1 NORMATIVE REFERENCES ...................................................................................................... 6
   3.2 INFORMATIVE REFERENCES .................................................................................................. 7

4. DEFINITIONS AND ABBREVIATIONS ....................................................................................... 8
   4.1 DEFINITIONS .................................................................................................................. 8
   4.2 ABBREVIATIONS ..................................................................................................................... 8
   4.3 DOCUMENT CONVENTIONS .................................................................................................. 8

5. INTRODUCTION ............................................................................................................................ 10

6. TRUST MANAGEMENT AND POLICIES ....................................................................................... 11
   6.1 BEHAVIOUR CONSIDERATIONS .............................................................................................. 11
   6.2 IDENTIFICATION OF TRUSTED CONTENT PROVIDERS .......................................................... 12
   6.3 SIGNATURE VALIDATION ......................................................................................................... 13
   6.4 TIME STAMPING .................................................................................................................... 14

7. CONTENT ENCODING AND SIGNATURE FORMATS .................................................................. 15
   7.1 CONTENT CODING .................................................................................................................. 15
   7.2 S/MIME SIGNATURE FORMAT ................................................................................................ 15
   7.3 XML SIGNED CONTENT ........................................................................................................... 17

8. IDENTIFICATION OF TRUSTED SIGNERS ON THE WIM .......................................................... 18
   8.1 FRAMEWORK ......................................................................................................................... 18
   8.2 IDENTIFICATION OF TRUSTED USAGE .................................................................................. 18

9. STATIC CONFORMANCE REQUIREMENTS .................................................................................. 19
   9.1 GENERAL REQUIREMENTS ..................................................................................................... 19
   9.2 CLIENT (ME) OPTIONS .............................................................................................................. 19

ANNEX A SIGNEDDATA USING ECDSA ....................................................................................... 23
   A.1 FIELDS OF THE SIGNEDDATA ............................................................................................... 23
   A.2 ACTIONS OF THE SENDING AGENT ...................................................................................... 23
   A.3 ACTIONS OF THE RECEIVING AGENT .................................................................................... 23

ANNEX B EXAMPLES ...................................................................................................................... 24
   B.1 EXAMPLE OF SIGNATURE ON A WML CARD ...................................................................... 24

© 2001, Wireless Application Protocol Forum Ltd. All rights reserved
1. Scope

The Wireless Application Protocol (WAP™) is a result of continuous work to define an industry wide specification for developing applications that operate over wireless communication networks. The scope for the WAP Forum is to define a set of specifications to be used by service applications. The wireless market is growing very quickly, reaching new customers and providing new services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation, and fast/flexible service creation, WAP selects and defines a set of open, extensible protocols and content formats as a basis for interoperable implementations.

The objectives of the WAP Forum are:

− To bring Internet content and advanced data services to digital cellular phones and other wireless terminals.
− To create a global wireless protocol specification that will work across differing wireless network technologies.
− To enable the creation of content and applications that scale across a very wide range of bearer networks and device types.
− To embrace and extend existing standards and technology wherever appropriate.

This specification defines formats and generic management procedures for signed content sent in WAP protocols.
2. Document Status

2.1 Document History
Document: WAP Signed Content
Document Identifier: WAP-233
Base Specification Approval Date: TBD
SINs Incorporated in this baseline document: 0

<table>
<thead>
<tr>
<th>SIN Approval Date</th>
<th>SIN Approval Date</th>
</tr>
</thead>
</table>

2.2 Errata
Known problems associated with this document are published at http://www.wapforum.org/.

2.3 Comments
Comments regarding this document can be submitted to the WAP Forum in the manner published at http://www.wapforum.org/.
3. References

3.1 Normative references


3.2 Informative references


4. Definitions and Abbreviations

4.1 Definitions

The following are terms and conventions used throughout this specification.

**Client**
A device (or application) that initiates a request for a connection with a server.

**Origin Server**
The server on which a given resource resides or is to be created. Often referred to as a web server or an HTTP server.

**Proxy Server**
The server on which a given resource neither resides nor is to be created. To complete the client request, the proxy server must get the resource from the related origin server(s).

**Server**
A device (or application) that passively waits for connection requests from one or more clients. A server may accept or reject a connection request from a client.

4.2 Abbreviations

For the purposes of this specification, the following abbreviations apply.

- **ASN.1**
  Abstract Syntax Notation One, as defined in [8], [9], [10] and [11].

- **CA**
  Certification Authority

- **CRL**
  Certificate Revocation List

- **BER**
  Basic Encoding Rules, as defined in [12]

- **DER**
  Distinguished Encoding Rules, as defined in [12]

- **ECDSA**
  Elliptic-Curve Digital Signature Algorithm

- **ME**
  Mobile Equipment

- **MIME**
  Multipurpose Internet Mail Extension(s)

- **OCSP**
  Online Certificate Status Protocol

- **OID**
  Object Identifier

- **RSA**
  Rivest-Shamir-Adleman public key algorithm

- **SHA-1**
  Secure Hash Algorithm

- **TSA**
  Time Stamp Authority

- **URL**
  Uniform Resource Locator

- **WAP**
  Wireless Application Protocol

- **WIM**
  WAP Identity Module

- **WINA**
  WAP Interim Naming Authority

- **WML**
  Wireless Markup Language

- **WTLS**
  Wireless Transport Layer Security

4.3 Document conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this specification are to be interpreted as described by [3].
This specification presents ASN.1 notation in the **bold Helvetica** typeface. When ASN.1 types and values are referenced in normal text, they are differentiated from normal text by presenting them in the **bold Helvetica** typeface.
5. Introduction

This section is informative.

The capability to distribute integrity-protected content is a fundamental building block for information security. Examples of uses include signed emails, distribution of (trustworthy!) software upgrades, authenticated web pages, reliable 3rd party software downloads, etc.

This specification describes methods to provide integrity, message authentication, and identification of the content signer for content of any type sent in WAP protocols, or stored in WAP terminals, through the use of digital signatures. In addition, it describes a generic framework for trust management related to signed content, and generic rules for WAP client behavior upon receiving such content. One advantage of signed content over other, protocol-based, security mechanisms such as WTLS [22], is the ability for a third party to securely provide information to a recipient regardless of the server providing the content.

Due to the WAP gateway’s performance-enhancing functionality, special care has to be taken to ensure that signed content does not get modified on its way from the origin server to the WAP client. Section 7 deals with content encoding and methods of delivery.

Annex B contains examples.
6. Trust Management and Policies

6.1 ME behaviour considerations

When a client processes signed content, it MUST verify the signature and MUST be able to provide information on the signer and the result of the verification. In a user-oriented environment clients MUST be able to display this information. Optionally, the WIM MAY be used to perform the signature verification. (See [27].)

6.1.1 Display Considerations

The client MUST be able to provide the following information regarding successful signature verifications:

- signer’s name (from the associated certificate),
- signer’s CAs’ names (recursively, including the name of the trusted root CA), and
- the type of signed content (e.g. WMLScript [20], WTA [21], Provisioning [25]).

The client MUST also be able to provide the following information regarding unsuccessful signature verifications:

- whether the content contained a signature,
- whether the signature was corrupt (i.e. whether the signature verified using the key contained in the referenced certificate),
- whether the associated certificate was valid,
- whether the certificate was trusted, and
- whether the certificate was authorized to download signed content.

In a user-oriented environment the client MUST be able to display this information to a user, and it MUST be displayed upon the first receipt of signed content from a particular signer. The user SHOULD be given the opportunity to choose not to view this information when subsequently receiving content of the same type from this signer.

6.1.2 Acceptance of Signed Content

To operate effectively and securely, the client must be configured to require content signing for designated content types (as specified in the inner “Content-Type” header, see Section 7) or purposes. This section will recommend client behavior regarding the acceptance of signed content.

When trusted signed content (i.e., containing a valid signature of a trusted signer) is processed and the results displayed to a user, the display and the result of the verification SHOULD be presented in a manner that is distinctive from content received unprotected or untrusted.

If the signed content is any form of executable content, in the absence of alternative rules specified for a particular form of content, the content SHOULD only be executed as trusted content on the client if the signature verification was successful and the signer has been authorized to download the given type of executable content. If the content was not signed when a signature was expected then the content should be considered untrusted or not be executed. If the signer has not been authorized to download the given type of executable content, the user MAY be informed of the signer’s identity and given the opportunity to approve execution of the content. The user SHOULD be given the opportunity to choose not to be informed when subsequently receiving content of the same type from this signer.

After validation, the content is passed to the designated processing application.

Note – Certain content types may require a more fine-grained authorization decision, e.g. the signer may also have to be identified as belonging to a certain trust domain corresponding to a particular permission set before the content is accepted. While this Specification certainly allows for such scenarios/architectures, it does not specify methods for them, since the scope of this document is on providing management procedures and data formats rather than trust and privilege architectures.
6.2 Identification of Trusted Content Providers

For certain designated content types the signer of the content must be recognized as an authorized content producer. There are three recommended approaches to recognizing authorized content producers:

− Certification by a Trusted CA;
− Direct Trust; or
− Explicit User Acceptance.

Support of the Direct Trust and Explicit User Acceptance methods is optional, but the Certification by a Trusted CA method is mandatory.

Note – Support of the optional methods (Direct Trust and Explicit User Acceptance) may cause a terminal not to comply with the 3GPP MExE specification [28].

6.2.1 Certification by a Trusted CA

Clients that support the download of signed content MUST be able to recognize authorized content producers using the method described in this section.

Certain CAs may be recognized by the client as being trusted to authorize content signers for the download of designated content types. For example, one CA may be trusted to authorize content signers for WTA content, while another may be trusted to authorize content signers for Provisioning content. In order for a CA to be trusted to authorize content signers it MUST:

− be provisioned in the client or WIM at manufacture time or downloaded using one of the secure root download techniques specified in WPKI [24] Sections 7.1.3 and 7.1.4, and
− be securely marked as being trusted for the authorization of content signers for certain designated content types.

The signed content MUST only be considered trusted if the certificate used to verify the signature on the content has been authorized by a CA trusted for the designated content type.

One method of marking CAs on the WIM as being trusted for designated content types is specified in Section 8. This method SHOULD be used whenever the CA certificate is being stored on a WIM and the designated content is executable.

Clients MUST be able to recognize which end entities have been granted the privilege of signing content by the trusted CA. A distinction is made between executable and non-executable content.

6.2.1.1 Certification of signers of executable content

This option identifies authorized content signers by way of an indication within the signer’s issued certificate. This method MUST be used whenever “Certification by a Trusted CA” is used and the content is executable. CAs that support this option must have a method of indicating within issued certificates whether individual end entities have been granted the privilege of signing content.

If WTLS [22] Certificates are being used for the download of WTA content, the “T=wta” naming attribute may be used to identify end users capable of signing WTA. This mechanism is described in [26], Section 8.2. (Editor’s Note: [26] is actually the March 9, 2000 Proposed version of the Cert Profile. This attribute will not be defined in Section 8.2 until the July 200 release. We will then have to update the reference.)

If X.509 [7] certificates are being used for the download of executable code, then the Extended Key Usage extension must be used to identify end users capable of signing code. This extension should be marked critical and must contain the object identifier \texttt{id-kp-codeSigning} defined in [13] Section 4.2.1.13, or some other WAP-defined object identifier designated for a particular purpose. Clients that support the support the download of signed executable code and that also process X.509 certificates MUST be able to recognize and process the Extended Key Usage extension and the \texttt{id-kp-codeSigning} OID.

This method may also be used for non-executable content.
Other methods of indicating within the issued certificate whether individual end entities have been granted the privilege of signing content may be used. Other specifications may define such methods.

### 6.2.1.2 Certification of other content signers

End entities certified by a trusted CA MAY be considered capable of signing non-executable content regardless of whether an explicit indication exists within the certificate.

Non-executable content may use this method. For such content, any trusted root on the client (or WIM) may be used when verifying the content.

### 6.2.2 Signers Trusted Directly

The client MAY directly trust certain content signers to sign designated content types. If such a scheme is being used, then the client MUST securely store an identifier that can be used to identify the signer’s certificate. The client MUST also securely store the content type for which the signer is trusted. The client MUST use one of the following identifiers to identify the signer’s certificate. The client MUST support derivation of all of these identifiers from a certificate which is not trusted by any of the other methods in this specification and is being used to verify a signature on signed content.

- X.509 certificate Issuer and Serial Number, or
- hash of X.509 certificate Issuer and Serial Number (as defined in Section 6.1.4 of [17]).

One method of securely storing a list of trusted content signers on the WIM is specified in Section 8. This method SHOULD be used whenever this information is being stored on the WIM.

The signed content MUST only be considered trusted if the signature verifies correctly and the certificate used to verify the signature on the content matches one included on the list of content signers trusted directly. If the content is executable, then the signer must be explicitly designated as trusted for this particular content type, as defined in Section 8.

The list of identifiers and associated content types SHOULD be provisioned on the client or WIM at the time of manufacture.

### 6.2.3 Explicit User Acceptance

If certain signed content has been verified (see Section 6.3) using a certificate that is not trusted directly for signed content and that does not chain to a trusted CA root that is trusted to authorize content signers (as described in the previous two sections), then the client MUST either discard the content, regard it as untrusted, or query the end user to determine whether the signer is trusted to sign the content. In the latter situation, the client MUST display at least the following information when querying the user:

- signer’s name (from the associated certificate),
- signer’s root CA’s name, and
- the type of signed content (e.g. WTA, Provisioning, or WMLScript).

The signed content MUST only be considered trusted if the signature verifies correctly and the end user agrees to trust the content signer. Since many users may not feel qualified to make this trust decision and/or may find the UI confusing, this method is not recommended for executable content.

The user SHOULD be given the opportunity to choose not to be informed when subsequently receiving content of the same type from this signer.

### 6.3 Signature Validation

Content signers MUST be identified using either an X.509 certificate or a WTLS Certificate. Content signers SHOULD use an X.509 certificate as profiled in the WAP Certificate and CRLs Profile [26] Section 6.5 (Content Signing Certificates), but MAY use WTLS Certificates. Clients supporting signed content SHOULD be able to process X.509 certificates as profiled in the WAP Certificate and CRLs Profile Section 6.5 and be able to validate certificate paths as specified in [7] but subject to the limitations of the WAP Certificate and CRLs Profile Section 6.5.
Clients supporting the download of signed content SHOULD provide a method of checking the revocation status of the signer’s certificate, but are not required to do so. One possible method is the OCSP protocol, described in [29]. Another possibility is the inclusion of the SignedData.crls field within the S/MIME signature to transport Certificate Revocation Lists [13] to the client. Alternatively, the use of short-lived certificates may be used to reduce the effect of certificate compromise. If short-lived certificates are used, however, then it is RECOMMENDED that time stamping also be used (see Section 6.4) to allow the signed content to be verified beyond the expiry of the certificate.

Signed content MUST be verified using a certificate trusted by the client. In order for the certificate to be trusted, it MUST chain to a trusted root CA certificate or a certificate trusted directly by the client (as specified in 6.2.1, 6.2.2 and 6.2.3). The latter certificate MUST either be provisioned on the client or WIM at manufacture time or securely downloaded using one of the methods specified in WPKI Sections 7.1.3 and 7.1.4. Depending on the application environment, in order to be trusted, signed content may need to satisfy additional requirements. One of the methods in Section 6.2 SHOULD be included in these requirements.

6.4 Time Stamping

According to [13], Section 6, an X.509 certificate that has expired cannot be successfully validated. Thus, in order to allow validation of signed content after the signer’s certificate has expired, a Time Stamping Authority (TSA) MAY be used. A TSA is a Trusted Third Party that creates a token attesting to the existence of data (the signed content) at a particular point in time.

If a valid Time Stamp Token for a particular signed content exists and can be verified, then the time contained within the token MUST be used as the time input to the certificate path validation algorithm.

Clients supporting signed content MAY support processing of Time Stamp Tokens according to [28]. Content signers should include a Time Stamp Token within the signed content as an unsigned attribute if that content is expected to be used beyond the normal expiry time of the signing certificate. The Time Stamp Token attribute is defined in [28] Annex A. For this attribute the time stamp must be computed over the signature field in SignerInfo.

In order for a Time Stamp Token to be valid, it MUST satisfy the requirements in [28]. In addition, the TSA MUST have a certificate that can be validated using a trusted root CA certificates either provisioned on the client or WIM at manufacture time or securely downloaded using one of the methods specified in WPKI Sections 7.1.3 and 7.1.4.

Note – If clients do not support the use of timestamps, it will be necessary for an organization to resign ALL of their signed content each time their signed content certificate expires. This applies both to signed content that may or may not contain a time stamp. If the signed content is not timestamped then it must be re-signed when the corresponding certificate expires, regardless of whether or not clients support timestamping.
7. Content Encoding and Signature Formats

7.1 Content Encoding

Due to limitation in WAP clients, content of the following content types may have to be encoded by the origin server before the signature operation:

- WML.
- WMLScript.
- WTA.
- Provisioning content.

It is up to the origin server to decide whether or not any given piece of content requires encoding before the signature operation in order to be processed by any given clients. Encoding rules are specified in [18], [19], [20], and [25].

7.2 S/MIME signature format

Regardless of the type of the content, it shall always be encapsulated within a MIME ([4], [5], and [14]) object, with an appropriate “Content-Type” setting. The following MIME headers are allowed within the MIME object encapsulating the content itself:

- Content-Type
  - Allowed parameters: name, charset
- Content-Language
- Content-Length
- Content-Transfer-Encoding
- Content-Description
- Content-Disposition
  - Allowed parameters: filename. Note security issues discussed in [30].

Note that clients MUST be able to parse MIME headers, and MUST recognize and process the Content-Type and Content-Transfer-Encoding headers in order to handle signed content correctly. Clients MUST NOT fail processing of signed content only due to the presence of un-recognized MIME headers.

The constructed MIME object is then signed and enveloped in another MIME object in accordance with procedures described in [6] and [15]. The following restrictions apply to values of type SignedData:

- SignedData.version shall be set to 1, unless SignerInfo.version is 3, see below.
- SignedData.digestAlgorithms shall contain one algorithm identifier value, the sha1Identifier from PKCS #1 [16].
- SignedData.encapContentInfo.contentType shall be set to id-data.
- SignedData.encapContentInfo.eContent shall contain the MIME object containing the encoded content. Nested signatures are not permitted. Thus, this field shall consist of unsigned content only.
- SignedData.certificates may be present but may only contain one or more X.509-compatible certificates. Attribute certificates are not permitted.
- SignedData.crls may be present.
SignedData.signerInfos:

- SignerInfo.version shall be set to 1, unless the SignerInfo.sid is the subjectKeyIdentifier choice, see below, in which case both SignerInfo.version and SignedData.version shall be set to 3.

- SignerInfo.sid shall be the issuerAndSerialNumber choice if the signer’s certificate is an X.509 certificate. If the signer’s certificate is a WTLS certificate, this field shall be the subjectKeyIdentifier choice and shall contain the 160-bit SHA-1 hash of the byte string representation of the public modulus [16] if the public key is an RSA key or the 160-bit SHA-1 hash of the byte string representation of the x-coordinate of the elliptic curve point (see also Section 10.5.1 of [22]).

- SignerInfo.digestAlgorithm shall contain one algorithm identifier value, the sha1Identifier from PKCS #1 [16].

- SignerInfo.signedAttrs MAY be present. Clients MUST be able process the following attributes from [6]:
  - messageDigest
  - contentType

Further, clients SHOULD be able to process the following attributes from [6]:

- signingTime

- SignerInfo.signatureAlgorithm shall be set to rsaEncryption or ecdsa-with-SHA1 (see [1] or Annex A).

- SignerInfo.unsignedAttrs MAY be present.
  - Clients MUST be able process the wtlsCertificates attribute defined in Section 7.2.1. Note that this requirement implies clients MUST be able to parse WTLS certificates.
  - The Time Stamp Token attribute (defined in [28] Annex A) MAY be present but there MUST NOT be more than one such attribute.

- The SignedData structure shall be BER ([12]) encoded before transmission

- Within the SignedData structure, the signedAttributes structure must be DER encoded (this is in accordance with [6]).

With regards to the SignedData.certificates field, clients MUST support certificate processing as specified in [26].

When SignerInfo.signedAttrs are present, issuers must include the id-messageDigest and id-contentType attributes.

Clients MUST NOT fail signature validation due only to the presence of un-recognized attributes (signed or unsigned).

The ContentType field for the outer MIME object shall be application/pkcs7-mime, and the rest of the outer MIME object shall be created in accordance with Section 3.2 and 3.4.2 of [15]. A WAP gateway may transform this content type to the WSP assigned content type number 39 and binary encode any of its associated headers.

Note – Annex B contains an example of the signature procedure.

### 7.2.1 The wtlsCertificates attribute

In cases where a signed content server only has access to WTLS certificates, it MAY supply these certificates to recipients in the wtlsCertificates attribute. Other means of transferring certificates needed for verification of signed content are also possible, e.g. by use of an underlying WTLS connection.

The wtlsCertificates attribute MAY be sent as an unsigned or as a signed attribute.

```plaintext
wtlsCertificates ATTRIBUTE ::= {
  WITH SYNTAX WTLSCertificates
  ID wap-at-wtlsCertificates
}
```

```plaintext
wap OBJECT IDENTIFIER ::= {joint-iso-itu-t(2) identified-organizations(23) 43}
wap-at OBJECT IDENTIFIER ::= {wap 2} -- Attributes branch
```
wap-at-wtlsCertificates OBJECT IDENTIFIER ::= {wap-at 1}

WTLSCertificates ::= OCTET STRING

Values of type WTLSCertificates shall be a sequence of ordinary WTLS Certificates encoded in accordance with [22], i.e., of the following WTLS type:

\begin{verbatim}
struct {
  WTLSCert Certificate certificate_list<0..2^16-1>;
} Certificates;
\end{verbatim}

The value is enclosed in an ASN.1 OCTET STRING.

### 7.3 XML signed content

This version of this specification does not specify a content-encoding procedure for XML signed data.
8. Identification of trusted signers on the WIM

8.1 Framework

This section describes a method for using the WIM to identify entities trusted for the download of certain designated content types or purposes. This method SHOULD be used whenever trusted CA certificates or other signer certificates will be stored on the WIM. Clients MUST support this method of trusted signer identification if the client supports the WIM.

The CommonCertificatesAttributes.trustedUsage field from PKCS #15 [17] shall be used to store a sequence of object identifiers that represent the privileges associated with the holder of the certificate in question. This field is of type Usage, which is a sequence of an optional keyUsage and an optional extKeyUsage field. For the purposes of marking trusted signers for the download of signed content, only the extKeyUsage field shall be present. This field is a sequence of object identifiers (OIDs).

8.2 Identification of trusted usage

For a CA certificate certifying signers of non-executable content, the first OID shall be either wap-explicitIndication or wap-implicitIndication depending upon whether the method in Section 6.2.1.1 or Section 6.2.1.2 (respectively) has been chosen.

```plaintext
wap-wsg OBJECT IDENTIFIER ::= {wap 1}
wap-signedContent-indications {wap-wsg 5}
wap-explicitIndication OBJECT IDENTIFIER ::= {wap-signedContent-indications 0}
wap-implicitIndication OBJECT IDENTIFIER ::= {wap-signedContent-indications 1}
```

If the wap-explicitIndication OID is present then explicit indications of end-entity signing privileges MUST be present in an entity’s certificate which chains back to this CA before a client may accept signed content signed by that entity. If the wap-implicitIndication OID is present then all end-entities with certificates signed by this CA shall be assumed to have been granted the privilege of signing designated types of content.

The remaining OIDs in the sequence shall indicate the types of signed content for which the CA is trusted to authorize signers. This applies to both executable and non-executable content. It is left to other specifications to define object identifiers for certain content types or purposes. New object identifiers in the WAP object identifier tree shall be assigned by WINA.

For certificates other than CA certificates, OIDs in the sequence shall indicate the types of signed content for which the certificate holder is directly trusted.
## 9. Static Conformance Requirements

This static conformance clause defines a minimum set of features that should be implemented to ensure interoperability. A feature can be optional (O), or mandatory (M) [23].

### 9.1 General Requirements

This section applies to all the clients and servers that conform to this specification.

<table>
<thead>
<tr>
<th>Item</th>
<th>Function</th>
<th>Sub-Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContSign-All-01</td>
<td>Certificates</td>
<td>Content signers identified by either an X.509 or a WTLS certificate.</td>
<td>6.3</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-All-02</td>
<td></td>
<td>Content signers identified by a WAP profiled X.509 certificate.</td>
<td>6.3</td>
<td>O</td>
<td>[26]</td>
</tr>
<tr>
<td>ContSign-All-03</td>
<td></td>
<td>Content signers identified by a WTLS certificate.</td>
<td>6.3</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ContSign-All-04</td>
<td>Signature Format</td>
<td>Content encapsulated and signed as specified in Section 7.2.</td>
<td>7.2</td>
<td>M</td>
<td>[4], [5], [6], [14], [15]</td>
</tr>
</tbody>
</table>

### 9.2 Client (ME) Options

This section applies to all the clients that conform to this specification.

<table>
<thead>
<tr>
<th>Item</th>
<th>Function</th>
<th>Sub-Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContSign-C01</td>
<td>Verify Signature</td>
<td>Verify signature on signed content.</td>
<td>6.1</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C02</td>
<td>Signature Information</td>
<td>Be able to provide the following information regarding successful signature verifications: signer’s name, signer’s CAs’ names, type of signed content.</td>
<td>6.1.1</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C03</td>
<td></td>
<td>Be able to provide the following information regarding unsuccessful signature verifications: whether a signature was present, whether the signature was corrupt, whether the certificate was valid, whether the certificate was trusted, whether the signer was authorized to sign the type of content.</td>
<td>6.1.1</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C04</td>
<td></td>
<td>Be able to display information on successful and unsuccessful signature verifications.</td>
<td>6.1.1</td>
<td>O</td>
<td>ContSign-C-02 AND ContSign-C-03</td>
</tr>
<tr>
<td>ContSign-C-05</td>
<td>Display signed content and the result of verification in a manner distinctive from texts received unprotected.</td>
<td>6.1.2</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContSign-C-06</td>
<td>Give the user the option not to be informed when subsequently receiving content from a signer.</td>
<td>6.1.2</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContSign-C-07</td>
<td>WIM Support</td>
<td>Use WIM to perform signature verification.</td>
<td>6.1</td>
<td>O</td>
<td>WIM-115 AND (WIMME-044 OR WIMME-045)</td>
</tr>
<tr>
<td>ContSign-C-08</td>
<td>Executable Content</td>
<td>Only execute signed content if signature verification successful and signer authorized to download the given type of content.</td>
<td>6.1.2</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-09</td>
<td>If executable content signer is not authorized inform user and give opportunity to approve execution.</td>
<td>6.1.2</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContSign-C-10</td>
<td>Identification of Trusted Content Providers</td>
<td>Recognize authorized content signers using the “Certification by a Trusted CA” method as described in Section 6.2.1.</td>
<td>6.2.1</td>
<td>M</td>
<td>ContSign-C-24</td>
</tr>
<tr>
<td>ContSign-C-11</td>
<td>Recognize authorized content signers using the “Signers Trusted Directly” method as described in Section 6.2.2.</td>
<td>6.2.2</td>
<td>O</td>
<td>ContSign-C-24</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-12</td>
<td>Recognize authorized content signers using the “Explicit User Acceptance” method as described in Section 6.2.3.</td>
<td>6.2.3</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContSign-C-13</td>
<td>Certificates</td>
<td>Be able to process X.509 certificates as specified in [26].</td>
<td>6.3</td>
<td>O</td>
<td>WAPCert:MCF</td>
</tr>
<tr>
<td>ContSign-C-14</td>
<td>Be able to check certificate revocation status.</td>
<td>6.3</td>
<td>O</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ContSign-C-15</td>
<td>Trusted Roots</td>
<td>Validate certificates using only roots provisioned on the client or WIM or downloaded securely using one of the methods specified in WPKI.</td>
<td>6.3</td>
<td>M</td>
<td>WPKI:MCF</td>
</tr>
<tr>
<td>ContSign-C-16</td>
<td>Time Stamps</td>
<td>Be able to process a Time Stamp Token and correctly validate the time stamped signature as described in Section 6.4.</td>
<td>6.4</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-17</td>
<td>Use the time value inside the Time Stamp Token as the time input for the certificate path validation algorithm, when a token exists and can be verified by the device.</td>
<td>6.4</td>
<td>M</td>
<td>ContSign-C-16</td>
<td></td>
</tr>
<tr>
<td>Item</td>
<td>Function</td>
<td>Sub-Function</td>
<td>Reference</td>
<td>Status</td>
<td>Requirements</td>
</tr>
<tr>
<td>-----------------</td>
<td>---------------------------------</td>
<td>------------------------------------------------------------------------------</td>
<td>-----------</td>
<td>--------</td>
<td>--------------</td>
</tr>
<tr>
<td>ContSign-UOE-C-1</td>
<td>Display of Signature Information</td>
<td>Be able to display information on successful and unsuccessful signature verifications.</td>
<td>6.1.1</td>
<td>M</td>
<td>ContSign-C-02 AND ContSign-C-03</td>
</tr>
</tbody>
</table>

### 9.2.1 User-Oriented Environment Options

This section applies to all clients operating in a user-oriented environment that conform to this specification.

<table>
<thead>
<tr>
<th>Item</th>
<th>Function</th>
<th>Sub-Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContSign-C-18</td>
<td>MIME Headers</td>
<td>Be able to parse MIME headers.</td>
<td>7.2</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-19</td>
<td></td>
<td>Recognize and process the Content-Type and Content-Transfer-Encoding MIME headers.</td>
<td>7.2</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-20</td>
<td></td>
<td>Not fail processing only due to the presence of an un-recognized MIME header.</td>
<td>7.2</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-21</td>
<td>Signed Attributes</td>
<td>Be able to process the messageDigest and contentType attributes.</td>
<td>7.2</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-22</td>
<td></td>
<td>Be able to process the signingTime attribute</td>
<td>7.2</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-23</td>
<td></td>
<td>Be able to process the wtlsCertificates attribute defined in Section 7.2.1.</td>
<td>7.2</td>
<td>M</td>
<td>WTLS-C060</td>
</tr>
<tr>
<td>ContSign-C-24</td>
<td></td>
<td>Not fail signature validation due only to the presence of un-recognized attributes.</td>
<td>7.2</td>
<td>M</td>
<td></td>
</tr>
<tr>
<td>ContSign-C-25</td>
<td>Identification of Trusted Signers on the WIM</td>
<td>Support storage of trusted signers on the WIM, as specified in Section 8.</td>
<td>8</td>
<td>M</td>
<td>WIM:MCF</td>
</tr>
<tr>
<td>ContSign-C-26</td>
<td></td>
<td>Storage of trusted signers on the WIM.</td>
<td>8</td>
<td>O</td>
<td>ContSign-C-24</td>
</tr>
<tr>
<td>ContSign-C-27</td>
<td></td>
<td>Require explicit indication within end-entity’s certificate if wap-explicitIndication OID is present in corresponding trustedUsage field.</td>
<td>8.2</td>
<td>M</td>
<td>ContSign-C-24</td>
</tr>
</tbody>
</table>

### 9.2.2 Executable Content Options

This section applies to all clients that process signed executable content and that conform to this specification.
<table>
<thead>
<tr>
<th>Item</th>
<th>Function</th>
<th>Sub-Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContSign-EC-C-1</td>
<td>Identification of Trusted Content Providers</td>
<td>Recognize signers of executable content authorized by trusted CAs using method described in Section 6.2.1.1.</td>
<td>6.2.1.1</td>
<td>M</td>
<td>ContSign-C-10</td>
</tr>
</tbody>
</table>

### 9.2.3 Non-Executable Content Options

<table>
<thead>
<tr>
<th>Item</th>
<th>Function</th>
<th>Sub-Function</th>
<th>Reference</th>
<th>Status</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContSign-NEC-C-1</td>
<td>Identification of Trusted Content Providers</td>
<td>Recognize signers of non-executable content authorized by trusted CAs using method described in Section 6.2.1.2.</td>
<td>6.2.1.2</td>
<td>O</td>
<td>ContSign-C-10</td>
</tr>
</tbody>
</table>
Annex A SignedData Using ECDSA

This annex describes how to use the Elliptic Curve Digital Signature Algorithm (ECDSA) with SignedData. ECDSA is specified in [1]. SignedData is defined in [6].

A.1 Fields of the SignedData

When using ECDSA with SignedData the fields of SignerInfo are as in [6], but with the following restrictions:

- **digestAlgorithm** contains the algorithm identifier sha1Identifier (see [16]) which identifies the SHA-1 hash algorithm.
- **signatureAlgorithm** contains the algorithm identifier ecdsa-with-SHA1 which identifies the ECDSA signature algorithm. The associated parameters field contains **NULL**.

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

\[\text{ecdsa-with-SHA1 OBJECT IDENTIFIER ::= \{ ansi-x9-62 signatures(4) 1 \}}\]

**signature** contains the DER encoding (as an octet string) of a value of the ASN.1 type **ECDSA-Sig-Value** (see [1]).

\[\text{ECDSA-Sig-Value ::= SEQUENCE \{}\]
\[\text{r INTEGER,}\]
\[\text{s INTEGER }\]

A.2 Actions of the sending agent

When using ECDSA with SignedData, the sending agent uses the message digest calculation process and signature generation process for SignedData that are specified in [6]. To sign data, the sending agent uses the signature method specified in [1], Section 5.3 with the following exceptions:

In [1], Section 5.3.1, the integer \(e\) shall instead be determined by converting the octet string resulting from [6], Section 5.4 to an integer using the data conversion method in [1], Section 4.3.2.

The sending agent encodes the resulting signature using the **ECDSA-Sig-Value** syntax and places it in the **SignerInfo.signature** field.

A.3 Actions of the receiving agent

When using ECDSA with SignedData, the receiving agent uses the message digest calculation process and signature verification process for SignedData that are specified in [6]. To verify SignedData, the receiving agent uses the signature verification method specified in [1], Section 5.4 with the following exceptions:

In [1], Section 5.4.1 the integer \(e\) shall instead be determined by converting the octet string resulting from [6], Section 5.4 to an integer using the data conversion method in [1], Section 4.3.2.

In order to verify the signature, the receiving agent retrieves the integers \(r\) and \(s\) from the **SignerInfo.signature** field of the received message.
Annex B Examples

B.1 Example of signature on a WML card

The content to be signed is in this case of type “text/vnd.wap.wml”. The actual deck is in Figure 1.

![Figure 1 - WML Deck for example 1](data:image/xml;base64,aHR0cHM6Ly93d3cuZnVtYWlsLmNuL3VwbG9hZHMvZG9jdW1lbnQuaHRtbA==)

When encoded to binary WML, the result will be as shown in Figure 2 (in Base64).

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <!-- Template definition for this deck: -->
  <template>
    <do type="ACCEPT" label="Important" name="template">
      <go href="#second"/>
    </do>
  </template>
  <!-- Actual cards in the deck: -->
  <card id="first" title="First Card">
    <p>
      This is an important document. Please read and accept here!
    </p>
  </card>
  <card id="second" title="Second Card">
    <p>
      <do type="ACCEPT" label="Regret" name="template">
        <go href="#first"/>
      </do>
      Press &quot;Regret&quot; if you changed your mind!
    </p>
  </card>
</wml>
```

![Figure 2 - Encoded version of the WML deck](data:image/xml;base64,PGh0bWw+TWFjaGFuZ2Vjcy10Y2hlY2tldC1jb2xvc3QucG5nKz4K)

The resulting MIME message that will be digested and then signed then becomes as shown in Figure 3 (Note: All linefeeds consists of carriage returns (CR) and linefeed (LF) characters. Last line is an empty CRLF line (also included in message digest and signature calculations).)

```
AQRqEEFDQ0VQVAB0ZW1wbGF0ZQ/+/e+g3gwAYA0tccG9ydGFudAAhgwcBq0oDl3NkY29uZAABAQHnVQNmaXJzdAA2A0ZpcnN0IENhcmQAAWADVGhpcyBpcyBhbiBpbXBv
ncRhhbQgZG9jdiW11bnQuIFBSzZWFkIGVzdFZSElAQnVQQWVvbmQgQ2FyZAABYOg3gwAYA1JlZ3dAAAQHnVQNzZWNvbmQgY2FyZAABYOg3gwAYA1JlZ3dAAAQHnV
VQQnZWNvbmQgY2FyZAABYOg3gwAYA1JlZ3dAAAQHnVQcHbQgZG9jdW1lbnQuaHRtbA==
```

© 2001, Wireless Application Protocol Forum Ltd. All rights reserved
The MIME message is digested with SHA-1, yielding the hash shown in Figure 4.

![Figure 3 - MIME object to be signed](image)

The hash is then inserted in the S/MIME `SignedAttributes` structure as a `messageDigest` attribute, together with a `contentType` attribute (value: `id-data`) and a `signingTime` attribute (the signing time is May 7, 2001, 19:49:22 UTC). The resulting, DER-encoded `SignedAttributes` structure is shown in Figure 5.

![Figure 4 - SHA-1 hash of the MIME object (in hex)](image)

This value is then hashed and signed using the algorithms `sha-1` and `rsaEncryption` respectively, and the resulting signature is shown in Figure 6 (lines are wrapped for readability).

![Figure 5 - DER-encoding of the SignedAttributes value (in hex)](image)

For verification purposes, the public key used to sign the message is shown in Figure 7 (in hex).

![Figure 6 - Signature on the SignedAttributes value (in hex)](image)

Finally, a full S/MIME `SignedData` structure is assembled (not shown here), and the resulting object sent through normal means to the client with content type “application/pkcs7-mime”.

![Figure 7 - Public key used for verification of signature](image)