Crypto Object for the ECMAScript Mobile Profile

Candidate Version 1.0 – 15 Jun 2004

Open Mobile Alliance

OMA-WAP-ECMACR-V1_0-20040615-C

Continues the Technical Activities
Originated in the WAP Forum

Contents

1. Scope

WAP defines a set of protocols in transport, session and application layers. For additional information on the WAP architecture, refer to [WAPARCH].

This document specifies an object for cryptographic functionality of the ECMAScript Mobile Profile [ESMP].

2. References

2.1 Normative References

[ASN1] ITU-T Recommendation X.680 (1997) | ISO/IEC 8824-1:1998, Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation.

[IOPPROC] "OMA Interoperability Policy and Process", Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, URL:http://www.openmobilealliance.org/

[DER] 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).

[ECMA262] Standard ECMA-262, "ECMAScript Language Specification – Edition 3", December 1999. URL: ftp://ftp.ecma.ch/ecma-st/Ecma-262.pdf

[ESMP] "ECMAScript – Mobile Profile", OMA-WAP-ESMP-V1_0, Open Mobile Alliance™, URL:http://www.openmobilealliance.org/

[PKCS1] PKCS #1: RSA Encryption Standard", version 1.5, RSA Laboratories, November 1993.

[PKCS15] PKCS #15 v1.1: Cryptographic Token Information Syntax Standard", RSA Laboratories, June 6, 2000. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-15/pkcs-15v1_1.pdf

[PKCS9] PKCS #9: Selected Attribute Types, version 2.0, RSA Laboratories, February 2000.

[RFC1521] "MIME (Multipurpose Internet Mail Extensions), Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies", N. Borenstein, et al, September 1993. URL: ftp://ftp.isi.edu/in-notes/rfc1521.txt

[RFC1738] "Uniform Resource Locators (URL)", T. Berners-Lee, et al., December 1994. URL: ftp://ftp.isi.edu/in-notes/rfc1738.txt

[RFC2119] "Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2119.txt

[RFC2253] "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", M. Wahl, et al., December 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2253.txt

[RFC2585] "Internet X.509 Public Key Infrastructure, Operational Protocols: FTP and HTTP", R. Housley, et al., May 1999. URL:ftp://ftp.isi.edu/in-notes/rfc2585.txt

[RFC2630] "Cryptographic Message Syntax", R. Housley, June 1999. URL: ftp://ftp.isi.edu/in-notes/rfc2630.txt

[RFC2634] "Enhanced Security Services for S/MIME", RFC 2634, Hoffman, P., Editor, June 1999

[TLS-EXT] "Transport Layer Security (TLS) Extensions", S. Blake-Wilson et al., July 2002. URL: http://www.ietf.org/internet-drafts/draft-ietf-tls-extensions-05.txt

[WAPCert] "WAP Certificate and CRL Profiles", WAP-211-WAPCert, WAP Forum Ltd. URL:http://www.openmobilealliance.org/

[WAPWIM] "WAP Identity Module", WAP-260-WIM, WAP Forum Ltd. URL:http://www.openmobilealliance.org/

[X9.62] "The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62 – 1998 (Approved January 1999).

2.2 Informative References

[JavaScriptSign] Signing Text from JavaScript. URL:http://developer.netscape.com/docs/manuals/security/sgntxt/index.htm

[RFC2068] "Hypertext Transfer Protocol - HTTP/1.1", RFC2068, R. Fielding, et al., January 1997. URL: ftp://ftp.isi.edu/in-notes/rfc2068.txt

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

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

[WAPTLS] "TLS Profile and Tunneling Specification", WAP-219-TLS, WAP Forum Ltd. URL: URL:http://www.openmobilealliance.org/

[WAPWPKI] "Wireless Public Key Infrastructure Specification", WAP-217-WPKI, WAP Forum Ltd. URL: URL:http://www.openmobilealliance.org/

[WAPWTLS] "Wireless Transport Layer Security", WAP-261-WTLS, WAP Forum Ltd. URL: URL:http://www.openmobilealliance.org/

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

[XML] "Extensible Markup Language (XML), W3C Proposed Recommendation 10-February-1998, REC-xml-19980210", T. Bray, et al, February 10, 1998. URL: http://www.w3.org/TR/REC-xml

2.3 How to Read this Document

This section is informative.

This specification draws heavily upon a number of existing standards, and assumes familiarity with:

- The ECMAScript Language Specification [ECMA262]
- WAP ECMAScript Mobile Profile [ESMP]
- Basic cryptography concepts

This specification is not written as a tutorial, but examples may be given. The examples are not exhaustive, and are generally informative only.

In all cases where there may be a question or ambiguity in the specification, source standards always take precedent, unless explicitly noted otherwise.

2.4 Acknowledgement

The signText method is based on [JavaScriptSign].

3. Definitions and Abbreviations

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 [RFC2119].

All sections and appendixes, except “Scope” (Section 1) and “Introduction” (Section 4) are normative, unless they are explicitly indicated to be informative.

3.2 Definitions

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

Content - subject matter (data) stored or generated at an origin server. Content is typically displayed or interpreted by a user agent in response to a user request.

Device - a network entity that is capable of sending and receiving packets of information and has a unique device address. A device can act as both a client and a server within a given context or across multiple contexts. For example, a device can service a number of clients (as a server) while being a client to another server.

ECMAScript-MP - a scripting language used to program the mobile device. ECMAScript-MP is an extended subset of the JavaScript™ scripting language.

JavaScript - a *de facto* standard language that can be used to add dynamic behaviour to HTML documents. JavaScript is one of the originating technologies of ECMAScript.

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.

Resource - a network data object or service that can be identified by a URL. Resources may be available in multiple representations (e.g. multiple languages, data formats, size and resolutions) or vary in other ways.

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.

User - a user is a person who interacts with a user agent to view, hear or otherwise use a rendered content.

User Agent - a user agent (or content interpreter) is any software or device that interprets markup language such as XHTML, script language, such as ECMAScript or resources. This may include textual browsers, voice browsers, search engines, etc.

Web Server - a network host that acts as an HTTP server.

WML - the Wireless Markup Language is a hypertext markup language used to represent information for delivery to a narrowband device, e.g. a phone.

3.3 Abbreviations

CA	Certification Authority
CMS	Cryptographic Message Syntax
DER	Distinguished Encoding Rules
DN	Distinguished Name
ECMA	European Computer Manufacturer Association
ESMP	ECMAScript – Mobile Profile
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
MIME	Multipurpose Internet Mail Extensions
OID	Object Identifier
PKCS	Public-Key Cryptography Standards
RFC	Request For Comments
RSA	Rivest Shamir Adleman public key algorithm
SHA	Secure Hash Algorithm
TLS	Transport Layer Security
UI	User Interface
URL	Uniform Resource Locator
UTF	UCS Transformation Format
W3C	World Wide Web Consortium
WAP	Wireless Application Protocol
WIM	WAP Identity Module
WMLS	Wireless Markup Scripting Language
WTLS	Wireless Transport Layer Security
WWW	World Wide Web
XHTML-MP	XHTML Mobile Profile

4. Introduction

The WAP Forum has recognized that convergence between the wired web and wireless devices, as targeted by the WAP, is an important step toward bringing wireless devices into the mainstream. As a part of the convergence process, WAP Forum has redefined the scripting language that is to be used by WAP devices, the ECMAScript-MP [ESMP].

The Crypto Object provides access to cryptographic features of the User Agent, such as digital signing. Application developers may take advantage of this functionality in addition to the functionality provided by transport layer security ([RFC2246], [WAPTLS], [WAPWTLS]).

WAP ECMAScript Crypto Object is specified to be as much a possible compatible with [JavaScriptSign].

Specific differences between WMLScript crypto library [WMLSCRYPT] and ECMAScript Crypto, and between ECMAScript Crypto and [JavaScriptSign] are detailed in Appendix C and Appendix D.

5. Object Definition

5.1 Crypto Object

The Crypto object provides cryptographic functionality.

5.1.1 Properties

No properties are defined.

5.1.2 Methods

5.1.2.1 signText()

5.1.2.1.1 Introduction

Many kinds of applications, e.g., electronic commerce, require the ability to provide persistent proof that someone has authorised a transaction. Although transport layer security ([RFC2246], [WAPTLS], [WAPWTLS]) provides transient client authentication for the duration of a connection, it does not provide persistent authentication for transactions that may occur during that connection. One way to provide such authentication is to associate a digital signature with data generated as the result of a transaction, such as a purchase order or other financial document.

To support this requirement, the User Agent provides the Crypto.signText method, that asks the user to sign a string of text. A call to the signText method displays the exact text to be signed and asks the user to confirm that. After the data has been signed and both the signature and the data have been sent across the network, the server can extract the digital signature and validate it, and possibly store it for accountability purposes.

The User Agent SHOULD use special signature keys that are distinct from authentication keys used for transport layer security. A WIM [WAPWIM] MAY be used for private key storage and signature computation.

5.1.2.1.2 Syntax

Syntax:	**`resultString = [window.]crypto.signText(stringToSign, options, [caNameString1, [caNameString2, . . . ]])`**
Argument List:	stringToSign - The string that you want the user to sign. This will be presented to the user, so it should be human-readable. options – Contains several options, as described in 5.1.2.1.3 caNameString - A string that specifies the DN for a CA whose certificates you trust for signing purposes. You should provide a caNameString parameter for each CA that you trust for the transaction involved. The DN is formatted according to [RFC2253].
Description:	returns the signature as a string value as described in 5.1.2.1.5.
Return Value Type:	string If the user approves the operation, the signText method returns a base-64-encoded CMS [RFC2630] SignedData value (see Format of Result String).
Errors or Exceptions:	Following error codes (as strings) may be returned: • **`error:noMatchingCert`** - The user did not have a certificate issued by a CA specified by one of the caNameString parameters. • **`error:userCancel`** - The user cancelled the operation. • **`error:internalError`** - An internal error such as an out-of-memory or decoding error occurred.
Example(s):	**`var foo = crypto.signText("Bill of Sale\n--------------------\n3 Tires $300.00\n1 Axle $795.00\n2 Bumpers $500.00\n--------------------\nTotal Price $1595.00", "ask");`**
Reference	-

5.1.2.1.3 Signing Options

The options parameter includes several options relevant for the signing process. Options are encoded as strings, separated with a space character. The User Agent MUST ignore options it does not recognise.

The following options are defined in the current version of this specification:

CA option - One of two strings:

- "ask" indicates that you want the User Agent to display a dialog asking the user to select a certificate to use for signing. The dialog lists the certificates signed by the CAs listed in the caNameString parameters. If no caNameString parameters are provided, the dialog lists all certificates installed in the certificate database that signText can use for signing. The User Agent is REQUIRED to support this option.
- "auto" indicates that you want the User Agent to select a signing certificate automatically from those available in the certificate database. If one or more caNameString parameters are provided, the User Agent chooses a certificate signed by one of the specified CAs. If no caNameString parameters are provided, the User Agent selects a certificate from the entire set of available certificates that signText can use for signing. The User Agent MAY support this option, or if not, treat it as if it was "ask". The `signingCertificate` signed attribute SHOULD NOT be included if the certificate (or a certificate label) is not shown to the user.

Certificates option

- "nocert" indicates that the certificate(s) should not be included in the result. Supporting this option is OPTIONAL. By default, certificates are included.

5.1.2.1.4 Description

This section is informative.

The `signText` method requests that a user digitally signs a text string. The calling script provides the text to sign (`stringToSign`), a string indicating various signing options like the CA option indicating a preference for manually or automatically selecting one of the certificates in the certificate database that can be used for signing, and (optionally) a list of CA DNs (`caNameString` parameters). If the CA option is set to `"auto"`, `signText` automatically selects a certificate signed by a CA specified by one of the `caNameString` parameters. If the CA option is set to `"ask"`, `signText` displays all certificates in the certificate database that are signed by a CA identified by one of the `caNameString` parameters and invites the user to select one of them. If the CA option is set to `"ask"` but no `caNameString` parameters are provided, `signText` displays all the certificates in the certificate database that can be used for signing.

In all cases the user may choose either to cancel or approve the signing operation. If the user approves the operation, the User Agent requests verification data for the signing key (like the WIM PIN). If the user provides the correct data, `signText` signs the specified string and returns the signed string to the script.

5.1.2.1.5 Format of the Result String

The result string returned by `signText` is a base-64-encoded CMS [RFC2630] `signedData` value wrapped in a `contentInfo` object with a `contentType` of `signedData`. The components of `signedData` have the following values:

Component	Value
`version`	`1`
`digestAlgorithms`	`sha-1`
`encapContentInfo.eContentType`	`id-data`
`encapContentInfo.eContent`	Not present. The data signed is not included in the `signedData` object.
`certificates`	User's signing certificate [WAPCert] and any intermediate CAs required to chain up to one of the trusted CAs listed in the `caNameString` parameters (the trusted CA certificate may be omitted), or not present (if the "nocert" option was set).
`crls`	Not present.
`signerInfos.version`	`1 (or 3, see below)`
`signerInfos.sid .issuerAndSerialNumber`	The issuer and serial number for the certificate used to sign the data. (If this information is not available in the User Agent, `subjectKeyIdentifier` as a form of signer identifier may be used as an alternative. In this case, according to [RFC2630], `version` needs to be `3`.)
`signerInfos.digestAlgorithm`	`sha-1`
`signerInfos.signedAttrs`	Attributes that are REQUIRED or OPTIONAL: • The content type attribute whose value is `id-data`. This attribute is REQUIRED. • The message digest attribute whose value is the message digest of the content. This attribute is REQUIRED. • The signing time attribute, whose value is the time that the object was signed (RECOMMENDED), or random nonce [PKCS9]. One of these attributes is REQUIRED. • The signing certificate attribute [RFC2634] SHOULD be present in case the signing certificate was indicated to the user. Other attributes MAY be present.
`SignerInfos.signatureAlgorithm`	Algorithm used in the signature. Either RSA [PKCS1] or ECDSA [X9.62] MUST be supported by the client. The verifying party (server) is REQUIRED to support RSA and MAY support ECDSA.
`SignerInfos.unsignedAttrs`	The certificate URL (section 5.1.2.1.6) attribute MAY be present.

Several certificates (indicating different identities etc.) may be issued for a single key pair. The signature should protect the integrity of user's choice of signing certificate. This is why the user's signing certificate SHOULD be included in `signedAttrs` as specified in [RFC2634] to avoid replacement attacks. The signing certificate attribute should use the hash of the certificate and OPTIONALLY the `issuerSerial` attribute (since this information is already available for the verifier, duplicating it in signed attributes is not necessary; however, for interoperability reasons, servers are RECOMMENDED to support this attribute).

5.1.2.1.6 Certificate URL Attribute

5.1.2.1.6.1. Introduction

The CMS SignedData structure [RFC2630] allows for the inclusion of certificates associated with the key used to sign the message. When included these certificates are placed in the `SignedData.certificates` field. However, inclusion of certificates in this manner assumes that the client creating the signature has access to the associated certificates. In some environments it is desirable for clients to use references to certificates (i.e. certificate URLs) in place of certificates, so that they do not need to locally store their certificates and can therefore save memory. This section describes an attribute that is to be used to convey certificate URLs in CMS messages.

The concept of certificate URLs are discussed further in [WAPWPKI] and [TLS-EXT]. These specifications allow for certificate URLs to point to either a single DER [DER] encoded X.509 certificate or in some cases a certificate chain defined in [TLS-EXT], Section 8, as a "PkiPath". The definition of this attribute assumes that the client is only aware of the URL or URL's that reference their certificate and certificate path, but not the resource the URL(s) refer to.

5.1.2.1.6.2. OID

The OID for the attribute is as follows (note that this definition is tentative and needs to be confirmed by OMA/WAP Naming Authority).

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

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

wap-at-certificateURL OBJECT IDENTIFIER ::= {wap-at 1}
```

5.1.2.1.6.3. Usage in CMS

This attribute, if present, is included as an unsigned attribute in the CMS message.

5.1.2.1.6.4. Attribute ASN.1 Definition

This attribute is defined as follows in ASN.1 [ASN1]:

```
certificateURL ATTRIBUTE ::= {
     WITH SYNTAX           URLs
     ID                    wap-at-certificateURL
}

URLs ::= SEQUENCE OF URL

      -- A list of one or more URL

URL ::= IA5String
```

```
-- Contains the URL [RFC1738] value and can return either a single

-- X.509 certificate or a chain of certificates represented by

-- a PkiPath
```

Each URL refers to either a single DER-encoded X.509v3 certificate or a DER-encoded certificate chain, using the type PkiPath described in [TLS-EXT], Section 8.

Note that when a list of URLs for X.509 certificates is used, the ordering of URLs is the same as that used in the TLS Certificate message (see TLS [RFC2246], Section 7.4.2), but opposite to the order in which certificates are encoded in PkiPath. In either case, the self-signed root certificate may be omitted from the chain, under the assumption that the server must already possess it in order to validate it.

Servers receiving a certificate URL attribute and supporting this attribute SHALL attempt to retrieve the client's certificate chain from the URLs, and then process the certificate chain as usual. Servers that support this attribute MUST support the http: URL scheme for certificate URLs, and MAY support other schemes.

If the protocol used to retrieve certificates or certificate chains returns a MIME [RFC1521] formatted response (as HTTP does), then the following MIME Content-Types SHALL be used: when a single X.509v3 certificate is returned, the Content-Type is "application/pkix-cert" [RFC2585], and when a chain of X.509v3 certificates is returned, the Content-Type is "application/pkix-pkipath" (see [TLS-EXT], Section 8).

If the signing certificate attribute with a certificate hash is present, then the server MUST check that the hash of the contents of the object retrieved from the URL (after decoding any MIME Content-Transfer-Encoding) matches the given hash. If any retrieved object does not have the correct hash, the server MUST abort certificate processing with an appropriate error.

5.1.2.1.7 Implementation Using the WIM

This chapter describes how to implement the signText function using the WIM [WAPWIM].

In accordance with the recommendation in section 5.1.2.1.1, a non-repudiation key should be used for signing. This implies usage of an authentication object used for this key only, and that the verification requirement cannot be disabled. E.g., in case of a PIN, the PIN must be entered separately for each signature operation.

The certificate issuer name hash (`CredentialIdentifier.issuerNameHash`) [PKCS15] can be used to find a proper certificate. For this, the textual CA DN (signText argument) needs to be converted to a DER encoded format and hashed.

To simplify the user experience, labels, contained in entries that describe private keys and certificates (`commonObjectAttributes.label`) should be used to display options to use for signing.

For a smart card implementation, the procedure is described in [WAPWIM].

Appendix A. Static Conformance Requirements (Normative)

This static conformance requirement [IOPPROC] lists a minimum set of functions that can be implemented to help ensure that implementations will be able to inter-operate. The "Status" column indicates if the function is mandatory (M) or optional (O).

A.1 Client Conformance

Item	Function	Reference	Status	Requirement
ECMACR-C-001	signText	5.1.2.1	M	
ECMACR-C-002	signText options	5.1.2.1.3	M	
ECMACR-C-003	signText option "ask"	5.1.2.1.3	M	
ECMACR-C-004	signText option "auto" processed	5.1.2.1.3	O	
ECMACR-C-005	signText option "auto" recognized	5.1.2.1.3	M	
ECMACR-C-006	signText option "nocert"	5.1.2.1.3	O	
ECMACR-C-010	signText signed attributes	5.1.2.1.5	M	ECMACR-C-011 OR ECMACR-C-012
ECMACR-C-011	signText signed signing time attribute	5.1.2.1.5	O	
ECMACR-C-012	signText signed random nonce attribute	5.1.2.1.5	O	
ECMACR-C-013	signText signed signing certificate attribute	5.1.2.1.5	O	
ECMACR-C-014	signText signed signing certificate attribute with issuerSerial	5.1.2.1.5	O	
ECMACR-C-015	signText unsigned certificate URL attribute	5.1.2.1.6	O	
ECMACR-C-019	signText any other attribute	5.1.2.1.5	O	
ECMACR-C-020	signText hash algorithm SHA-1	5.1.2.1.5	M	
ECMACR-C-030	signText signing algorithm	5.1.2.1	M	ECMACR-C-031 OR ECMACR-C-032
ECMACR-C-031	signText signing algorithm RSA	5.1.2.1	O	
ECMACR-C-032	signText signing algorithm ECDSA	5.1.2.1	O	
ECMACR-C-040	signText use of signature keys that are distinct from authentication keys	5.1.2.1.1	O	
ECMACR-C-041	signText use of WIM	5.1.2.1.7	O	

A.2 Server Conformance

Item	Function	Reference	Status	Requirement
ECMACR-S-001	SignText	5.1.2.1	M	
ECMACR-S-010	signText signed attributes	5.1.2.1.5	M	
ECMACR-S-011	signText signed signing time attribute	5.1.2.1.5	M	
ECMACR-S-012	signText signed random nonce attribute	5.1.2.1.5	M	
ECMACR-S-013	signText signed signing certificate attribute	5.1.2.1.5	M	
ECMACR-S-014	signText signed signing certificate attribute with issuerSerial	5.1.2.1.5	O	
ECMACR-S-015	signText unsigned certificate URL attribute	5.1.2.1.6	O	ECMACR-S-040 AND ECMACR-S-041 AND ECMACR-S-042 AND ECMACR-S-043 AND ECMACR-S-044
ECMACR-S-019	signText any other attribute recognized	5.1.2.1.5	M	
ECMACR-S-020	signText hash algorithm SHA-1	5.1.2.1.5	M	
ECMACR-S-030	signText signing algorithm	5.1.2.1.5	M	
ECMACR-S-031	signText signing algorithm RSA	5.1.2.1.5	M	
ECMACR-S-032	signText signing algorithm ECDSA	5.1.2.1.5	O	
ECMACR-S-040	signText, retrieve client certificate from the URL	5.1.2.1.6.4	O	
ECMACR-S-041	signText, HTTP scheme for certificate URL	5.1.2.1.6.4	O	
ECMACR-S-042	signText, application/pkix-cert	5.1.2.1.6.4	O	
ECMACR-S-043	signText, application/pkix-pkipath	5.1.2.1.6.4	O	
ECMACR-S-044	signText, check certificate hash	5.1.2.1.6.4	O	

Appendix B. Mapping WMLScript Crypto Library Functions to ES-MP Crypto Object Methods (Informative)

Library	Call	Object	Method/Constant	Comment
Crypto	SignText	Crypto	signText()	

Appendix C. Differences between WMLScript Crypto Library and ECMAScript-MP Crypto Object (Informative)

In addition to basic differences in script languages [ESMP], Appendix D, following differences exist:

- In ECMAScript-MP Crypto Object signText, trusted certificates are encoded as textual DN. Multiple authorities are indicated as multiple parameters. In WMLScript signText, trusted certificates are encoded as key identifiers.
- In ECMAScript-MP Crypto Object signText, signing options are encoded as a string. In WMLScript signText, they are encoded as binary.
- In ECMAScript-MP Crypto Object signText, signing certificate may be included as a signed attribute
- Using a key id to indicate signing key is supported in WMLScript signText but not supported in ECMAScript-MP Crypto Object signText

Appendix D. Differences between ECMAScript-MP Crypto Object and JavaScript Crypto Methods (Informative)

ECMAScript-MP Crypto Object signText method supports the following features which are additional to what is supported in JavaScript Crypto

- Additional options may be indicated: "nocert"
- Additional signed attributes are defined

Appendix E. Change History (Informative)

E.1 Approved Version History

Reference	Date	Description
n/a	n/a	No previous version within OMA

E.2 Draft/Candidate Version 1.0 History

Document Identifier	Date	Sections	Description
Draft Versions OMA-WAP-ECMACR-V1_0	25 Sep 2002	n/a	n/a
	14 May 2004	Chapter 2	References to other specifications updated.
Candidate Version OMA-WAP-ECMACR-V1_0	15 Jun 2004	n/a	Status changed to Candidate by TP TP ref # OMA-TP-2004-0193-WPKI-V1_0_for-candidate