Crypto Object for ECMA Script Mobile Profile

Candidate Version 1.1 – 22 Mar 2005

Open Mobile Alliance

OMA-WAP-ECMACR-V1_1-20050322-C

Continues the Technical Activities
Originated in the WAP Forum

Contents

Figures

Tables

1. Scope

Open Mobile Alliance (OMA) specifications are the result of continuous work to define industry-wide interoperable mechanisms for developing applications and services that are deployed over wireless communication networks.

The OMA wireless markup scripting language known as ECMAScript – Mobile Profile [ESMP] is strongly based upon ECMAScript Release 3 [ECMA262].

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, 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).

[Base64] *"Privacy Enhancement for Internet Electronic Mail: Part I -- Message Encipherment and Authentication Procedures"* J. Linn, February 1993. http://www.ietf.org/rfc/rfc1421.txt

[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. 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.

[RFC2045] "*Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*", N. Borenstein, N. Freed, November 1996. http://www.ietf.org/rfc/rfc2045.txt

[RFC1738] *"Uniform Resource Locators (URL)"*, T. Berners-Lee, et al., December 1994. http://www.ietf.org/rfc/rfc1738.txt

[RFC2119] *"Key words for use in RFCs to Indicate Requirement Levels"*, S. Bradner, March 1997. http://www.ietf.org/rfc/rfc2119.txt

[RFC2253] "Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names", M. Wahl, et al., December 1997. http://www.ietf.org/rfc/fc2253.txt

[RFC2585] "Internet X.509 Public Key Infrastructure, Operational Protocols: FTP and HTTP", R. Housley, et al., May 1999. http://www.ietf.org/rfc/rfc2585.txt

[RFC3852] "Cryptographic Message Syntax", R. Housley, July 2004. http://www.ietf.org/rfc/rfc3852.txt

[RFC2634] "Enhanced Security Services for S/MIME", RFC 2634, Hoffman, P., Editor, June 1999 http://www.ietf.org/rfc/rfc2634.txt

[TLS-EXT] "Transport Layer Security (TLS) Extensions", S. Blake-Wilson et al., June 2003. http://www.ietf.org/rfc/rfc3546.txt

[CertProf] "Certificate and CRL Profiles", OMA-Security-CertProf-v1_1, Open Mobile Alliance™, http://www.openmobilealliance.org

[WAPWIM] "Wireless Identity Module Part: Security", OMA-WAP-WIM-v1_2, Open Mobile Alliance. http://www.openmobilealliance.org

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

[PKCS#10] PKCS#10: Certification Request Syntax Standard, RSA Laboratories, May 26, 2000, http://www.rsasecurity.com/rsalabs/pkcs/pkcs-10/index.html

[HMAC] Keyed-Hashing for Message Authentication, H. Krawczyk, M. Bellare, R. Canetti, RFC2104, http://www.ietf.org/rfc/rfc2104.txt

[3DES-CBC] 3DES encryption using CBC mode, NIST SP 800-38A 2001 ED

[M2] Definition of padding before encryption, ISO 9797

2.2 Informative References

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

[RFC2246] "The TLS Protocol, Version 1.0", RFC2246, T. Dierks, C. Allen, January 1999. http://www.ietf.org/rfc/rfc2246.txt

[TLSProfile] "TLS Profile and Tunneling Specification", WAP-219-TLS, Open Mobile Alliance http://www.openmobilealliance.org

[WAPWPKI] "Wireless Public Key Infrastructure Specification", WAP-217-WPKI, Open Mobile Alliance http://www.openmobilealliance.org

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

[WMLScriptCrypto] "WMLScript Crypto Library Specification", WAP-161-WMLScriptCrypto, Open Mobile Alliance™, http://www.openmobilealliance.org

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

[ESMP] "ECMAScript Mobile Profile", OMA-WAP-ESMP-v1_0, Open Mobile Alliance, http://www.openmobilealliance.org

[X509] "Information Technology – Open Systems Interconnection – The Directory: Authentication Framework.", ITU-T Recommendation X.509 (1997) | ISO/IEC 9594-8:1998,

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 [RFC2119].

All sections and appendixes, except “Scope” and “Introduction”, 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 Mobile Profile	A scripting language used to program the mobile device. ECMAScript – Mobile Profile 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	Auser 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.
WMLScript	A scripting language based on ECMAScript [ECMA262] that has been modified to better support low bandwidth communication and thin clients.
Key Pair	A set of mathematically related keys, a public key and a private key, that are used for asymmetric cryptography and are generated in a way that makes it computationally infeasible to derive the private key from knowledge of the public key.
Key Slot	A place holder for a key pair that may or may not contain a key
X.509	An ITU-T Recommendation [X509] that defines a framework to provide and support data origin authentication and peer entity authentication services, including formats for X.509 public-key certificates, X.509 attribute certificates, and X.509 CRLs.
X.509v3	An abbreviation for an version 3 X.509 certificate

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
WTLS	Wireless Transport Layer Security
XHTML-MP	XHTML Mobile Profile
TLV	Tag Length Value
AODF	Authetication Object Directory File
PrKDF	Private Key Directory File
PuKDF	Public Key Directory File
APDU	Application Protocol Data Unit

3.4 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]
- OMA 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.

3.5 Acknowledgement

The signText method is based on [JavaScriptSign].

4. Introduction

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

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

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

5. Crypto Object Definition

The Crypto object provides cryptographic functionality.

5.1 Properties

No properties are defined.

5.2 Methods

5.2.1 signText()

5.2.1.1 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], [TLSProfile], [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.

5.2.1.2 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 5.2.1.3 `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 5.2.1.5.
Return Value Type:	string If the user approves the operation, the signText method returns a base-64-encoded CMS [RFC3852] 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	-

Table 1 - signText Syntax

5.2.1.3 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.

5.2.1.4 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.

5.2.1.5 Format of the Result String

The result string returned by `signText` is a base-64-encoded CMS [RFC3852] `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 [CertProf] 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 [RFC3852], `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.[1]
`SignerInfos.unsignedAttrs`	The certificate URL (section 5.2.1.6) attribute MAY be present.

Table 2 - CMS SignedData Values for signText

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).

5.2.1.6 Certificate URL Attribute

5.2.1.6.1 Introduction

The CMS SignedData structure [RFC3852] 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 [RFC3852] 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.509v3 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.

5.2.1.6.2 OID

The OID for the attribute is as follows

```
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}
```

5.2.1.6.3 Usage in CMS

This attribute, if present, is included as an unsigned attribute in the CMS message [RFC3852].

5.2.1.6.4 Attribute ASN.1 Definition

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

```
certificateURL ATTRIBUTE ::= {
     WITH SYNTAX     URLs
```

[1] Note that the server can report the fact that it does not support ECDSA in the XHTML page generated in response to the processing of the XHTML data sent by the client.

```
      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.509v3 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.509v3 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 server is unable to retrieve the required certificates via the use of the URL attribute and thus not able to validate the signature on the result string, it will indicate this error to the user in the XHTML page generated in response to the processing of the XHTML data sent by the client.

If the protocol used to retrieve certificates or certificate chains returns a MIME [RFC2045] 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.

5.2.1.7 Implementation Using the WIM

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

In accordance with the recommendation in section 5.2.1.1, 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].

5.2.2 KeyGen method

5.2.2.1 Introduction

WIMs can either be manufactured with key pairs or, if equipped with the OBKG feature, can generate their own key pairs. The following function applies to MEs supporting the second type of WIMs. The OBKG feature is acknowledged to be useful due to the sensitivity of the non-repudiation key pairs, and for legal issues in some countries that impose it as a mandatory feature for non-repudiation keys.

Since a WIM may be issued with non-initialized key pairs we call it a key slot in this specification. A key slot is a place holder for a key pair and may be uninitialized (not ready to be used). A key slot space is allocated in WIM personalisation process during manufacturing.

Some of the parameters that are sent with the KeyGen Script are used for searching for the relevant key pair to generate. Other parameters set new values for some attributes of the key pair such as key label, PIN label for the PIN that protects operation with the key pair, etc.

An authentication MAY be required to gain the right to generate the key pair. The authentication key SHOULD be different for each WIM and and MAY be different for different key pairs in the same WIM. The use of an authentication key to control the right to generate a key is at the discretion of the WIM issuer and is personalised during manufacturing.

This specification describes the ECMA Script that is used in order to trigger an on-board key generation in the WIM. In a response to the Script invocation the "WIM serial number" and a challenge may be returned to the server. This will indicate to the server that an authentication is needed. The remote server has to be able to generate a response to the challenge in order to create a valid authorisation code. The remote server may use the serial number to derive the correct HMAC (or asymmetric) key and then calculate the HMAC (or digital signature) on the challenge value. The HMAC (or digital signature) of the challenge value then becomes the authorization code for the next key generation request for this WIM. The WIM serial number that is returned is the hexadecimal representation of the serial number retrieved from the WIM.

The following example describes how key generation is triggered by the ME when an authentication code is needed, and the Security Element is a WIM:

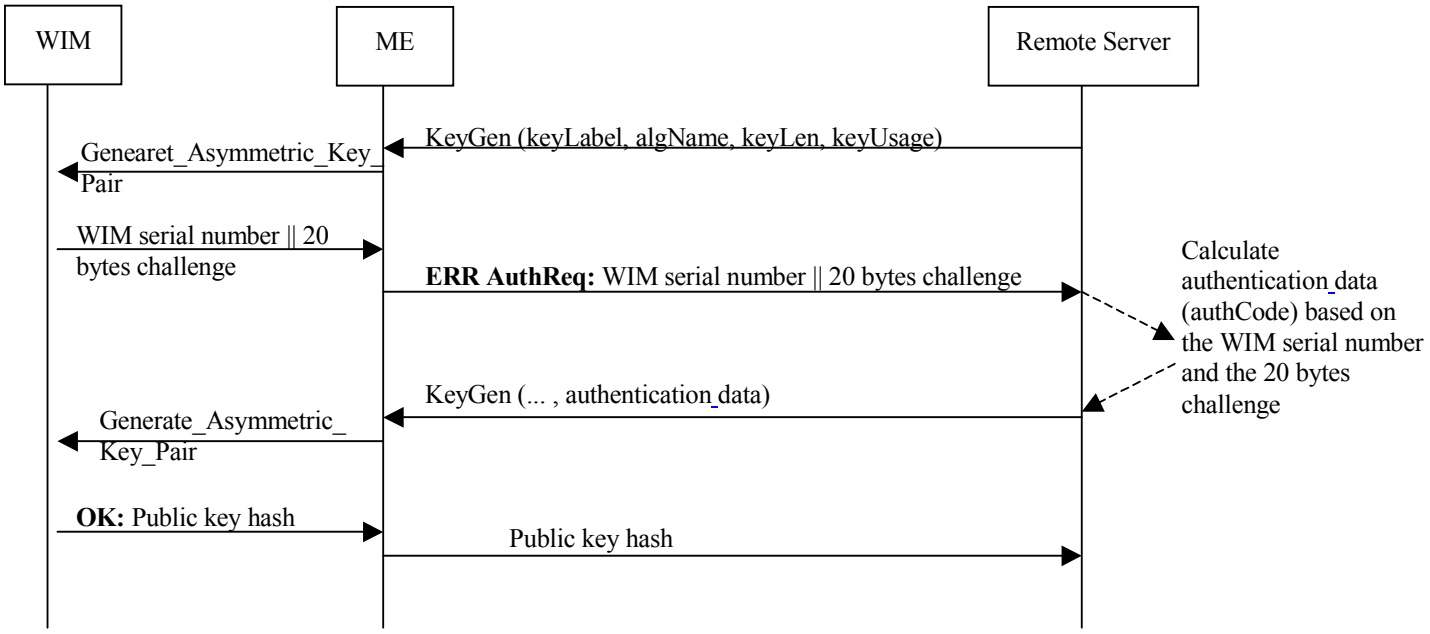

Figure 1 - Key Generation Data Flow

5.2.2.2 Syntax

Syntax: **`resultString = [window.]crypto.keyGen ( keyLabel, keyType, keyLen, keyUsage, EncryptedAttributes, authCode, other )`**

Argument List: `keyLabel` = String

In the case of a WIM this corresponds to the *commonObjectAttributes.label* in [PKCS15] PrKDF entry for the key to generate. If the value is "" then KeyLabel is unspecified.

`keyType`= Integer

0x1000 – Unspecified
0x0000 – RSA
0x0003 – ECC

`keyLen` = Integer

An integer indicating the bit length of the key. For RSA this is the length of the modulus. If the value is 0x0000 then keyLen is unspecified.

`keyUsage` = Integer

Contains the key usage of the key (as in [PKCS15])

0x0000 – Unspecified
0x0004 – Authentication key. In the case of a WIM the [PKCS15] PrKDF entry for the key has "keyUsageFlags" that include the "Authentication Key" flag
0x0100 - Non-repudiation Key

`EncryptedAttributes` = string

A string that contains some encrypted values to set in the WIM as described in the "WIM Based Implementation" section below. The string may be empty.

`authCode` = String

Hexadecimal representation of authentication code that is calculated by the remote server. The authentication code may be generated using an HMAC or asymmetric technique. If the value is "" then authCode is not provided.

An authentication code MAY be required to gain the right to generate the key pair. In the case of a WIM the authentication key SHOULD be different for each WIM and MAY be different for different key pairs in the same WIM. The use of an authentication code to control the right to generate a key is at the discretion of the WIM issuer and is personalised during manufacturing. The function MUST convert the hexadecimal representation of the string to a binary array (octet string) before sending it to the WIM.

In the case of a WIM the authCode is calculated over the Encrypted Attributes parameter followed by a WIM challenge as described in the"WIM Based Implementation" section below.

other = String

RFU - this parameter may be used to convey additional parameters for other public key schemes. It should be set to "" (empty string) in this specification version.

Description:	Provided authorization is granted, this function triggers the generation of a key pair in a key slot matching one or more of the following attributes: key label, key usage flag, algorithm and key length. The function looks for a key slot that matches these attributes. At least one of these parameters must be specified to form the search criteria for a key slot. If more than one key slot match the search criteria it is up to the browser to select a key. It may then display the list of key labels to the user and optionally additional attributes like algorithm, key length etc. If no key slot can be found with the provided criteria, the error code of the first improper parameter shall be returned. The parameter Encrypted Attributes MUST NOT be sent if authCode is not sent. The function SHOULD NOT delete certificates in the WIM if the generated key replaces a key that already had certificates. KeyUsage flag MUST be specified if the key usage is not implicitly identified by other parameters (e.g. keyLabel). In the case of a WIM the function triggers the key generation in the WIM by using the WIM GENERATE_ASYMMETRIC_KEY_PAIR command.
Return Value Type:	String • The hexadecimal representation of the public key hash of the newly generated key pair. Public key hash is calculated as defined in WTLS specification [WTLS].
Errors or Exceptions:	Following error codes (as strings) may be returned: • If a key with a specified key Label cannot be found the function returns **`error:noSuchKeyLabel`** • If a key with the specified keyType cannot be found the function returns **`error:noSuchAlg`**. • If a key with the specified key length cannot be found the function returns **`error:noSuchKeyLen`**. • If a key with the specified key usage cannot be found the function returns **`error:noSuchKeyUsage`** • If no parameters are specified to select a key, the function returns `error:noSuchKey` • If an authentication code is required and was not provided, or if the sent authentication code was incorrect, the function returns **`error:AuthReq:cardSerialNumber:Challenge`**. In this response both cardSerialNumber and the Challenge are in hexadecimal format.
Example(s):	`var result = crypto.keyGen("", 0x0000, 1024, 0x0004, "", "", "");`
Reference	

Table 3 - keyGen Syntax

5.2.2.3 WIM based implementation

The function shall use the WIM GENERATE_ASYMMETRIC_KEY_PAIR APDU command to implement key generation.

5.2.2.3.1 Encrypted Attributes parameter

The Encrypted Attributes parameter is the hexadecimal representation of the following data blob:

[C0 L encrypted New PIN value] [C1 L encrypted New PIN label] [C2 L encrypted new key label][2].

Each element in this data blob is encrypted using 3DES in CBC mode as described in section 15.2 of [3DES-CBC]. The initial chaining value for CBC modes shall be zero and padding M2 [M2] shall apply. The encryption key SHOULD be different from the authentication key used for gaining the right to generate the key pair.

Each element in this data blob is explained below:

C0 L encrypted New PIN value	A TLV where tag=C0, L is the length of the following data and "encrypted New PIN value" is a new encrypted PIN value that will be decrypted and set by the WIM for the PIN that protects the newly generated key pair.
C1 L encrypted New PIN label	A TLV where tag=C1, L is the length of the following data and "encrypted New PIN label" is a new encrypted PIN label that will be set by the WIM for the PIN that protects the newly generated key pair. The new PIN label MUST NOT exceed 32 characters.
C2 L encrypted new key label	A TLV where tag=C2, L is the length of the following data and "encrypted new key label" is a new encrypted label that should be set for the newly generated key pair. The new key label MUST NOT exceed 32 characters.

Table 4 - WIM Encrypted Attributes Value

The function MUST convert the hexadecimal representation of the "encryptedAttributes" string to a binary array (octet string) before sending it to the WIM in the GENERATE_ASYMMETRIC_KEY_PAIR APDU command. The function MUST verify the [PKCS15] AODF entry of the PIN that protects the key to be generated. If this AODF entry indicates that the PIN is not initialised (`PKCS15Authentication.TypeAttributes.pinFlags`), and no "encrypted New PIN value" is included in the parameter list, the function MUST ask the user to enter an initial value for that PIN and send it to the WIM with the GENERATE_ASYMMETRIC_KEY_PAIR APDU command. Asking the user to enter an initial value for the PIN MUST be done if and only if the above conditions are fulfilled.

5.2.2.3.2 AuthCode parameter

The AuthCode parameter is the HMAC or digital signature over Encrypted Attributes parameter followed by a WIM challenge. A WIM challenge was previously obtained from the WIM by sending the GENERATE_ASYMMETRIC_KEY_PAIR APDU command without an authentication code or with a wrong authentication code.

The function MUST convert the hexadecimal representation of the "authCode" string to a binary array (octet string) before sending it to the WIM in the GENERATE_ASYMMETRIC_KEY_PAIR APDU command.

5.2.3 genEnrollReq Method

5.2.3.1 Introduction

This function generates a certificate enrolment request for a key identified by the sent parameters. It may require an authorization code before returning an enrolment request as expressed in the sent parameters.

The result is the [Based64] encoding of a [PKCS #10] enrolment request.

[2] The data items between [and] are considered optional.

5.2.3.2 Syntax

Syntax: **`enrollRequest = [window.]crypto.genEnrollReq(nameInfo, keyType, keyLen, KeyUsage, keyIDType, keyID, authCode, Other)`**

Argument List: `nameInfo` = String

This parameter may be used to specify the information included in the Name field in the [PKCS #10] structure. The format of this string MUST conform to encoding rules of RFC 2253 [RFC2253]. The browser SHOULD implement a configuration flag that will allow the user to always view this parameter value in the case that it is specified (not an empty string).

If the parameter is an empty string, the Name information MUST at least contain the card serial number as a serial number attribute (e.g. SN = 12345678) . In the case of a WIM it SHOULD also include the card manufacturer's name as an Organization attribute (e.g. SN = 98765432, O = "Smart Card Manufacturer, Inc"). The SerialNumber is the serial number of the device on which the key resides. In the case of a WIM it is retrieved from the TokenInfo EF.

`keyType` = Integer

This parameter is used to indicate the public key algorithm for which an enrolment request will be generated.

0x1000 – Unspecified

0x0000 – RSA
0x0003 – ECC

`keyLen` = Integer

An integer indicating the bit length of the key parameter. For RSA this is the length of the modulus. If the value is 0x0000 then keyLen is unspecified.

`keyUsage` = Integer

Contains the key usage of the key for which an enrolment request must be generated.

0x0000 – Unspecified

0x0004 - Authentication Key. In the case of a WIM this means that the [PKCS15]PrKDF entry for the key has "keyUsageFlags" that includes an "Authentication Key" flag.

0x0100 - Non-repudiation Key.

`keyIDType` = Integer

This parameter may be used to specify a specific key

0 - NONE: No key identifier is supplied.

1 - USER_KEY_HASH: A SHA-1 hash of the user public key is supplied in the next parameter. The browser MUST use the signature key that corresponds to the GIVEN public key hash. If the key is not available the browser MUST return error:noSuchKey.

2 - KEY_LABEL: If the key label is known, it may be used to identify the key for which an enrolment request is required. The key label MUST corresponds to the commonObjectAttributes.label in the [PKCS15] entry for the key. If the key is not available the browser MUST return error:noSuchKey.

`keyID` = String

Identify the key based on the previous parameter. If the previous parameter is USER_KEY_HASH the value is the hexadecimal representation of the public key hash. Public key hash is calculated as defined in WTLS specification [WAPWTLS].

`authCode` = String

Hexadecimal representation of authentication code that is calculated by the remote server. If the value is "" then authCode is not provided. The authentication code MAY be required to obtain an enrolment request. The use of an authentication code to control access to the GenEnrollReq function is at the discretion of the WIM issuer. The authentication code may be generated using a symmetric or asymmetric technique. In the case of a WIM the authentication key SHOULD be different for each WIM and MAY be different for different key pairs in the same WIM. The function MUST convert the hexadecimal representation of the string to a binary array (octet string) before sending it to the WIM.

Other = String

RFU - this parameter may be used to convey additional parameters for other public key schemes. It should be set to "" (empty string) in this specification version.

Description:	This function generates an enrolment request for a key identified by the algName, keyLen, keyUsage, keyIDType or keyID parameters. If the provided parameters do not identify a unique key, it is up to the browser to select a key. The browser MAY prompt the user to nominate a specific key. The **GenEnrollReq** function may require an authorization code before returning an enrolment request. If an authorization code is required but not supplied, the **GenEnrollReq** will return the WIM serial number and a challenge. The invoker of the function must then compute the HMAC or digital signature on the challenge for the specified device. The result serves as the authorization code for the next invocation of the **GenEnrollReq** command. In some scenarios authorization is required before using the corresponding private key to sign the certification request. In these scenarios the browser MUST ensure the user has proper authorization. Depending on the security element in use, the user may grant authorization by entering the correct PIN. The result is the [Based64] encoding of a [PKCS #10] enrolment request. The format of the enrolment request is specified in section 3 of this document. The [Based64] encoding of the result is returned as a String.
Return Value Type:	String The [Based64] encoding of the enrolment request.
Errors or Exceptions:	Following error codes (as strings) may be returned: • If the specified algorithm is not available in the WIM implementation, the function returns **`error:noSuchAlg`**. • If a key of the specified key length can not be found, the function returns **`error:noSuchKeyLen`**. • If a key of the specified usage can not be found the function returns **`error:noSuchKeyUsage`** • If a keyIDType was specified but a matching key could not be found for the keyID the function returns **`error:noSuchKey`**. • If an authentication code is required and was not supplied as part of the enrolment request or was incorrectly specified, the function will return **`error:AuthReq:cardSerialNumber:Challenge`**. In this response both the cardSerialNumber and the Challenge is in a hexadecimal format. The PKI portal MUST generate a response to the challenge in order to create a valid authorisation code. The PKI portal may use the serial number to derive the correct HMAC key and then calculate the HMAC on the challenge value. A digital signature may be used instead of an HMAC. The result of the HMAC or signature calculation then becomes the authorization code for the next enrolment request for this WIM.

Example(s):	`var dn = "CN = John Smith, O = Foo Co, C = US";`
	
	`var request = crypto.genEnrollReq (dn, RSA, 1024, 0x0100, 0, "", "", "");`

Reference	

Table 5 - genEnrollReq Syntax

5.2.3.3 Format of Enroll Result

5.2.3.3.1 Introduction

The **GenEnrollReq** command will return an error or a well-formed enrolment request. In this section the Enroll Request message is defined. The Enroll Request will take the form of a PKCS #10 certificate request as defined in [PKCS #10].

In addition, this specification defines mechanisms to indicate to the PKI an assurance as to how the key was generated and stored. This assurance may indicate that the key was generated on trusted hardware (such as a WIM). This assurance is provided through the inclusion of an attribute in the attributes field of the CertificationRequestInfo structure. The assurance information MUST be one of :

- Digital signature using a public key formatted as a CMS [RFC3852] message.
- An HMAC using a symmetric key formatted as a CMS [RFC3852] message.

The data on which the HMAC or digital signature is calculated should include the public key for which an assertion is provided as well as an indication of the type of assertion that is made.

5.2.3.3.2 Enrolment Request Format

The enrolment request will follow the certificate enrolment format as defined in [PKCS#10]. We repeat some of that information in this document for the sake of clarity.

A [PKCS #10] message consists of a top level **`CertificationRequest`**. The ASN.1 definition of the Certification Request is shown below for reference.

```
CertificationRequest ::= SEQUENCE {
   certificationRequestInfo      CertificationRequestInfo,
   signatureAlgorithm
   AlgorithmIdentifier{{SignatureAlgorithms}},
   signature                     BIT STRING
}
```

The **`CertificationRequest`** message contains the **`CertificationRequestInfo`** structure. The **`CertificateRequestInfo`** structure is replicated for reference purposes.

```
CertificationRequestInfo ::= SEQUENCE {
      version                    INTEGER { v1(0) } (v1,...),
      subject                    Name,
      subjectPKInfo              SubjectPublicKeyInfo{{ PKInfoAlgorithms }},
      attributes                 [0] Attributes{{ CRIAttributes }}
}
```

`CertificationRequestInfo` is the structure that is signed by the private key corresponding to the public key that is to be enrolled. The assurance information is included in the **`CertificationRequestInfo`** structure as an **`Attribute`**.

The attribute has the following format:id-keygen-assertion OBJECT IDENTIFIER ::=

```
{joint-iso-itu-t(2), identified-organizations(23) wap(43) attributes(2) 2}
```

-- Assigned OID from WAP OID tree

```
keyGenAssertion ATTRIBUTE ::= {
        WITH SYNTAX Assertion
```

```
        ID id-keygen-assertion
        }


Assertion ::= CHOICE {
        signedData SignedData,
        authenticatedData [0] AuthenticatedData
        ... -- For future expansion
        }
```

The **SignedData** and **AuthenticatedData** types are defined in CMS [RFC3852]. The signature or MAC is computed on the DER encoding of a value of type **AssuranceInfo** (see below). The **ContentInfo** included in the **SignedData** and AuthenticatedData type must be of type **data**. The **ContentInfo** MUST be included in the **SignedData** (or **AuthenticatedData**) structure. The fields of the **signedData** object have the following values:

Component	Value
version	1
digestAlgorithms	sha-1
encapContentInfo.eContentType	id-data
encapContentInfo.eContent	Present (DER encoded AssuranceInfo structure). In the case of a WIM it is returned by the Generate Key Assurance APDU command.
certificates	Assurance signing certificate [CertProf]. Intermediate CAs required to chain up to a trusted CA MAY be included. In the case of a WIM a handle to the certificate (public key hash) is returned by the Generate Key Assurance APDU command. The ME can then fetch the certificate from the WIM.
crls	Not present.
signerInfos.version	1 (or 3, see below)
signerInfos.sid .issuerAndSerialNumber	The issuer and serial number for the assurance certificate used to sign the data. If this information is not available in the User Agent (for instance if the signer is only known through a certificate URL), subjectKeyIdentifier as a form of signer identifier may be used as an alternative. In this case, according to [RFC3852], the version needs to be 3.)
signerInfos.digestAlgorithm	sha-1
signerInfos.signedAttrs	Not 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. In the case of a WIM a handle to the certificate (public key hash) is returned by the Generate Key Assurance APDU command. The ME can then fetch the certificate URL from the WIM.

Table 6 - CMS SignedData Values for genEnrollReq

For the **AuthenticatedData**, the SHA-1 hash algorithm MUST be used. The subject of managing the symmetric keys used by the HMAC algorithm is beyond the scope of this document. The HMAC key MUST be at least 128-bits long. The fields of the **AuthenticatedData** object have the following values:

Component	Value
version	0
Mac algorithm	HMAC-SHA1 OBJECT IDENTIFIER ::= { iso(1) identified-organization(3)dod(6) internet(1) security(5) mechanisms(5) 8 1 2 }
encapContentInfo.eContentType	id-data
encapContentInfo.eContent	Present (DER encoded AssuranceInfo structure). In the case of a WIM it is returned by the Generate Key Assurance APDU command.
AuthenticatedAttr	Not present.
RecipientInfo.KEKRecipientInfo.version	4
RecipientInfo.KEKRecipientInfo.kekid	Set to 1.
RecipientInfo.KEKRecipientInfo.keyEncryptionAlgorithm	des-ede3-cbc OBJECT IDENTIFIER ::= { iso(1) member-body(2) us(840) rsadsi(113549) encryptionAlgorithm(3) 7 } 3DES in CBC mode as described in section 15.2 of [3DES-CBC] . The initial chaining value for CBC mode shall be zero and padding M2 [M2] shall apply.
RecipientInfo.KEKRecipientInfo.encryptedKey	The HMAC key encrypted with the shared key. In the case of a WIM it is returned by the Generate Key Assurance APDU command.
Mac	The calculated HMAC value. In the case of a WIM it is returned by the Generate Key Assurance APDU command.

Table 7 - CMS AuthenticatedData Values for genEnrollReq

5.2.3.3.3 AssuranceInfo

The **AssuranceInfo** is the data that is signed or MACed. It is the data on which the assertion is made. The **AssuranceInfo** has the following structure:

```
AssuranceInfo::= SEQUENCE {
   deviceID                    UTF8String (SIZE (1..64)) OPTIONAL,
   subjectPKInfo               SubjectPublicKeyInfo,
   keyAssertion                KeyAssertion,
   ... -- For future use
}

KeyAssertion ::= ENUMERATED {
```

```
    injected-into-device (0),
    generated-on-device(1),
    -- Note: '(0)' and '(1)' are not needed but can be here for clarity
    ...
}
```

The value of **deviceID** identifies the device where the key was generated as an UTF8String. If the WIM is implemented on a smartcard or SIM, the **deviceID** MAY be set to the ICCID of the smartcard or SIM. The length of the **deviceID** value MUST NOT exceed 64 characters.

The **`KeyAssertion`** field is used to indicate if the key was generated on the WIM or if the key was injected into the WIM. The **`SubjectPublicKeyInfo`** MUST be the same as that included in the **`CertificationRequestInfo`** structure. The structure MUST be DER encoded.

5.2.3.4 WIM based implementation

5.2.3.4.1 Creation of Assurance Assertion

The WIM must be capable of generating/providing the assurance signature on request. This is achieved through a single "Generate Key Assurance" APDU command. This is the only command that can access or use the assurance key. The ME MUST send the "Generate Key Assurance" APDU command to the WIM if it cannot find the public key in the [PKCS15] PuKDF entry for the selected key. If the ME can find the public key in the [PKCS15] PuKDF entry for the selected key it MAY choose to format a [PKCS #10] without the Key AssuranceInfo. The WIM is responsible for formatting the **`AssuranceInfo`**, selecting the assurance key and generating the assurance. It is not possible to specify the data to be signed (i.e. the **`AssuranceInfo`**) externally.

If the assurance key is an RSA key, the result returns a PKCS#1 signature and the **`AssuranceInfo`**. The ME is responsible for formatting the **`KeyGenAssertion`** as specified in this document. In this case the **`KeyGenAssertion`** contains a CMS [RFC3852] message. The "Generate Key Assurance" APDU command also returns the signer certificate, or certificate URL, that is then used by the ME in formatting the **`KeyGenAssertion.`**

If the assurance key is symmetric, "Generate Key Assurance" APDU command returns an HMAC and the **`AssuranceInfo`**. The ME will then format the **`KeyGenAssertion`** as defined in this document and include it in the **`CertificationRequestInfo`** structure.

It may not be required to place the key assurance certificate or key assurance public key on the WIM. The public key may be obtained from a certificate identified by a certificate URL.

5.2.3.4.2 Formatting of Enrolment Request

The ME is responsible for formatting the enrolment request. The WIM is responsible for performing the cryptographic operations, including signature generation, assurance generation and PIN validations.

In the case of a WIM the ME obtains the assurance information from the WIM. Once it has the assurance information it formats the assurance attribute **`KeyGenAssertion`**.

Once the ME has the assurance attribute, it proceeds to generate the **CertificationRequestInfo**. If no Name information was provided as a function parameter, the ME may obtain information from the WIM (e.g. the serial number in the EF(TokenInfo) file for WIMs) to complete the Name information in the **CertificationRequestInfo**. The ME then proceeds to calculate the SHA-1 hash on the **CertificationRequestInfo**. The hash result is presented to the WIM for signature generation. Before the signature is generated the relevant PIN for the specified key has to be presented to the WIM to confirm usage of the private key (e.g. PIN-NR for NR key and PIN-G for an authentication key if PIN-G was not validated earlier in the session).

After the ME obtained the signature, it proceeds to construct the **`CertificationRequest`**. The DER encoded **`CertificationRequest`** is [Based64] encoded and returned as the function result. The function result may then be used by the PKI to enroll a new user.

Appendix A. Static Conformance Requirements

This appendix is 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	5.2.1	M	
ECMACR-C-002	signText options	5.2.1.3	M	
ECMACR-C-003	signText option "ask"	5.2.1.3	M	
ECMACR-C-004	signText option "auto" processed	5.2.1.3	O	
ECMACR-C-005	signText option "auto" recognized	5.2.1.3	M	
ECMACR-C-006	signText option "nocert"	5.2.1.3	O	
ECMACR-C-010	signText signed attributes	5.2.1.5	M	ECMACR-C-011 OR ECMACR-C-012
ECMACR-C-011	signText signed signing time attribute	5.2.1.5	O	
ECMACR-C-012	signText signed random nonce attribute	5.2.1.5	O	
ECMACR-C-013	signText signed signing certificate attribute	5.2.1.5	O	
ECMACR-C-014	signText signed signing certificate attribute with issuerSerial	5.2.1.5	O	
ECMACR-C-015	signText unsigned certificate URL attribute	5.2.1.6	O	
ECMACR-C-019	signText any other attribute	5.2.1.5	O	
ECMACR-C-020	signText hash algorithm SHA-1	5.2.1.5	M	
ECMACR-C-030	signText signing algorithm	5.2.1	M	ECMACR-C-031 OR ECMACR-C-032
ECMACR-C-031	signText signing algorithm RSA	5.2.1	O	
ECMACR-C-032	signText signing algorithm ECDSA	5.2.1	O	
ECMACR-C-040	signText use of signature keys that are distinct from authentication keys	5.2.1.1	O	
ECMACR-C-041	signText use of WIM	5.2.1.7	O	
ECMACR-C-050	keyGen	5.1.2.2.2	M	
ECMACR-C-051	KeyGen keyLabel parameter	5.1.2.2.2	O	ECMACR-C-055
ECMACR-C-053	keyGen keyType parameter	5.1.2.2.2	O	
ECMACR-C-054	keyGen keyLen parameter	5.1.2.2.2	O	
ECMACR-C-055	keyGen keyUsage parameter	5.1.2.2.2	O	
ECMACR-C-056	keyGen encryptedAttributes parameter	5.1.2.2.2	O	ECMACR-C-057
ECMACR-C-057	keyGen authCode parameter	5.1.2.2.2	O	
ECMACR-C-070	GenEnrollReq	5.1.2.3.2	M	
ECMACR-C-071	GenEnrollReq nameInfo parameter	5.1.2.3.2	O	
ECMACR-C-072	GenEnrollReq keyType parameter	5.1.2.3.2	O	
ECMACR-C-073	GenEnrollReq keyLen parameter	5.1.2.3.2	O	
ECMACR-C-074	GenEnrollReq keyUsage parameter	5.1.2.3.2	O	
ECMACR-C-075	GenEnrollReq keyIDType parameter	5.1.2.3.2	O	
ECMACR-C-076	GenEnrollReq keyID parameter	5.1.2.3.2	O	
ECMACR-C-077	GenEnrollReq authCode parameter	5.1.2.3.2	O	

A.2 Server Conformance

Item	Function	Reference	Status	Requirement
ECMACR-S-001	SignText	5.2.1	M	
ECMACR-S-010	signText signed attributes	5.2.1.5	M	
ECMACR-S-011	signText signed signing time attribute	5.2.1.5	M	
ECMACR-S-012	signText signed random nonce attribute	5.2.1.5	M	
ECMACR-S-013	signText signed signing certificate attribute	5.2.1.5	M	
ECMACR-S-014	signText signed signing certificate attribute with issuerSerial	5.2.1.5	O	
ECMACR-S-015	signText unsigned certificate URL attribute	5.2.1.6	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	5.2.1.5	M	
ECMACR-S-020	signText hash algorithm SHA-1	5.2.1.5	M	
ECMACR-S-030	signText signing algorithm	5.2.1.5	M	
ECMACR-S-031	signText signing algorithm RSA	5.2.1.5	M	
ECMACR-S-032	signText signing algorithm ECDSA	5.2.1.5	O	
ECMACR-S-040	signText, retrieve client certificate from the URL	5.2.1.6.4	O	
ECMACR-S-041	signText, HTTP scheme for certificate URL	5.2.1.6.4	O	
ECMACR-S-042	signText, application/pkix-cert	5.2.1.6.4	O	
ECMACR-S-043	signText, application/pkix-pkipath	5.2.1.6.4	O	
ECMACR-S-044	signText, check certificate hash	5.2.1.6.4	O	
ECMACR-S-050	KeyGen	5.1.2.2.2	M	
ECMACR-S-051	KeyGen keyLabel parameter	5.1.2.2.2	O	ECMACR-S-055
ECMACR-S-053	keyGen keyType parameter	5.1.2.2.2	O	
ECMACR-S-054	keyGen keyLen parameter	5.1.2.2.2	O	
ECMACR-S-055	keyGen keyUsage parameter	5.1.2.2.2	O	
ECMACR-S-056	keyGen encryptedAttributes parameter	5.1.2.2.2	O	ECMACR-S-057
ECMACR-S-057	keyGen authCode parameter	5.1.2.2.2	O	
ECMACR-S-070	GenEnrollReq	5.1.2.3.2	M	
ECMACR-S-071	GenEnrollReq NameInfo param	5.1.2.3.2	O	
ECMACR-S-072	GenEnrollReq keyType param	5.1.2.3.2	O	
ECMACR-S-073	GenEnrollReq keyLen parameter	5.1.2.3.2	O	
ECMACR-S-074	GenEnrollReq keyUsage parameter	5.1.2.3.2	O	
ECMACR-S-075	GenEnrollReq keyIDType parameter	5.1.2.3.2	O	
ECMACR-S-076	GenEnrollReq keyID parameter	5.1.2.3.2	O	
ECMACR-S-077	GenEnrollReq authCode parameter	5.1.2.3.2	O	

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

This appendix is informative.

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

Appendix C. Differences between WMLScript Crypto Library and ESMP Crypto Object (Informative)

This appendix is informative.

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

- In ESMP 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 ESMP Crypto Object signText, signing options are encoded as a string. In WMLScript signText, they are encoded as binary.
- In ESMP 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 ESMP Crypto Object signText

Appendix D. Differences between ESMP Crypto Object and JavaScript Crypto Methods (Informative)

This appendix is informative.

ESMP 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
OMA-WAP-ECMACR-V1_0	15 June 2004	Approved in the OMA WPKI enabler release

E.2 Draft/Candidate Version 1.1 History

Document Identifier	Date	Sections	Description
Candidate Version OMA-WA-ECMACR-V1_1-20050322-C	22 March 2005		TP ref: OMA-TP-2005-0091-OBKG-V1_0-for-Candidate-approval
Draft Version OMA-WAP-ECMACR-V1_1	02 February 2005	Many	Updates based on consistency review. See OMA-CONRR-OBKG-V1_0-20041202-D
Draft Version OMA-WAP-ECMACR-V1_1	20 August 2004	2.1, 2.2	Editorial corrections in the references, change of doc name
Draft Version OMA-WAP-ECMACR-V1_1	26 March 2004		Candidate version
Draft Version OMA-WAP-ECMACR-v1_1	12 March 2004	5.1.2.2 5.1.2.3	Update the function descriptions to adhere to the description format of the signText function
Draft Version OMA-WAP-ECMACR-v1_1	11 December 2003	6, 7	Added two new functions: KeyGen and GenEnrollReq according to CRs OMA-SEC-2003-0049R02-obkg-ecma-keygen and OMA-SEC-2003-0050R02-obkg-ecma-enroll
Draft Version OMA-WAP-ECMACR-v1_0	25 September 2003		Draft version from WAP. Has passed the consistency review in WAP Forum.