Wireless Application Protocol Public Key Infrastructure Definition

Approved Version 1.1 – 01 Mar 2011

Open Mobile Alliance

OMA-WAP-WPKI-V1_1-20110301-A

Continues the Technical Activities
Originated in the WAP Forum

Contents

Figures

Tables

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, and reaching new customers and services. To enable operators and manufacturers to meet the challenges in advanced services, differentiation and fast/flexible service creation WAP Forum defines a set of protocols in transport, security, transaction, session and application layers. For additional information on the WAP architecture, please refer to "*Wireless Application Protocol Architecture Specification"* [WAPARCH].

WAP security functionality includes the Wireless Transport Layer Security [WTLS] and application level security, accessible using the Wireless Markup Language Script [WMLScript]. The security provided in WAP can be of various levels. In the simplest case anonymous key exchange is used for creation of an encrypted channel between server and client; in the next level a server provides a certificate mapping back to an entity trusted by the client; and finally the client itself may possess a private key and public key certificate enabling it to identify itself to other entities in the network. This document is concerned with the infrastructure and procedures required to enable the trust relationships needed for authentication of servers and clients. The term server used here is not limited to WAP gateways and origin servers but may include third parties and content/service providers using the WAP protocols.

The general model is adaptable to many certificate types including X509v3, X9.68 (currently in draft) and the certificate format defined in WTLS. The WTLS certificate has the advantage of being very compact, easily implemented in code, and easily parsed which may be important for initial implementations of WAP clients. In addition, to the extent possible, the WAP PKI will work interchangeably with existing X.509v3 certificates in existing Internet applications, in order to leverage the existing Internet PKIs. Any new format that requires major changes to the installed base of certificate-processing products and CA infrastructure is unlikely to be easily adopted in a short timeframe.

For this reason the general model for this version is that server certificates will use the WTLS certificate format whereas client certificates will use X.509 format, but as far as possible will not be sent over the air nor stored on the client.

2. References

2.1 Normative References

[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
[CERTPROF]	"*Certificate and CRL Profiles*", Open Mobile Alliance™, OMA-Security-CertProf-V1_1, http://www.openmobilealliance.org/
[E2E]	*"WAP Transport Layer End-to-end Security Specification"* , WAP-187-TransportE2ESec, WAP Forum™, http://www.openmobilealliance.org
[ESMPCRYPTO]	*"ECMA Script Crypto Library"*, Open Mobile Alliance™ , OMA-WAP-ECMACR-V1_1, http://www.openmobilealliance.org/
[IOPPROC]	*"OMA Interoperability Policy and Process",* Version 1.1, Open Mobile Alliance™, OMA-IOP-Process-V1_1, http://www.openmobilealliance.org/
[LDAPSRCH]	*"The String Representation of LDAP Search Filters,"* IETF RFC 2254, T. Howes, December 1997. http://www.ietf.org/rfc/rfc2254.txt.
[LDAPURL]	*"The LDAP URL Format,"* IETF RFC 2255, T. Howes. M. Smith, December 1997, http://www.ietf.org/rfc/rfc2255.txt.
[MIME]	"*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
[MIMETYPE]	*"Multipurpose Internet Mail Extensions (MIME) Part 2: Media Types*", N. Freed, N. Borenstein, November 1996. http://www.ietf.org/rfc/rfc2046.txt
[PKCS10]	*"PKCS #10: Certification Request Syntax Version 1.7"*, B. Kaliski, March 1998. http://www.ietf.org/rfc/rfc2986.txt.
[RFC2119]	*"Key words for use in RFCs to Indicate Requirement Levels"*, S. Bradner, March 1997, http://www.ietf.org/rfc/rfc2119.txt
[RFC3280]	*"Internet X.509 Public Key Infrastructure Certificate and CRL Profile,"* R. Housley, et al, April 2002. http://www.ietf.org/rfc/rfc3280.txt.
[RFC2560]	*"Internet X.509 Public Key Infrastructure – Online Certificate Status Protocol – OCSP*," IETF RFC 2560, M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, June 1999. http://www.ietf.org/rfc/rfc2560.txt.
[WAPARCH]	*"WAP Architecture Specification"* , WAP-210-WAPArch, WAP Forum™, http://www.openmobilealliance.org
[PROVARCH]	*"OMA Provisioning Architecture"* , OMA-WAP-ProvArch-V1_1, Open Mobile Alliance™, http://www.openmobilealliance.org
[WAPWTLS]	*"Wireless Transport Layer Security Specification,"* WAP-261-WTLS, WAP Forum™, http://www.openmobilealliance.org
[WIM]	*"Wireless Identity Module Part: Security"*, Open Mobile Alliance™, OMA-WAP-WIM-V1_2, http://www.openmobilealliance.org/
[WMLScript]	*"WMLScript Language Specification"* , WAP-193-WMLScript, WAP Forum™, http://www.openmobilealliance.org
[WMLScriptCrypto]	*"WML Script Crypto Library Specification",* WAP-161-WMLScriptCrypto*,* WAP Forum™, http://www.openmobilealliance.org
[WSP]	*"Wireless Session Protocol"*, Open Mobile Alliance™, WAP-230-WSP, http://www.openmobilealliance.org

2.2 Informative References

[ESMP]	"*ECMAScript Mobile Profile*", OMA-WAP-ESMP-v1_0, Open Mobile Alliance™, http://www.openmobilealliance.org
[RFC1321]	*"The MD5 Message Digest Algorithm,"* , IETF RFC 1321, R. Rivest, April 1992. http://www.ietf.org/rfc/rfc1321.txt
[RFC2510]	*"Internet X.509 Public Key Infrastructure Certificate Management Protocols"*, IETF RFC 2510, C. Adams and S. Farrell, March 1999. http://www.ietf.org/rfc/rfc2510.txt.
[RFC2511]	*"Certificate Request Message Format,"* IETF RFC 2511, M. Myers, C. Adams, D. Solo and D. Kemp, March 1999. http://www.ietf.org/rfc/rfc2511txt.
[RFC2797]	*"Certificate Management Messages over CMS*", IETF RFC 2797, M. Myers, et.al., April 2000. http://www.ietf.org/rfc/rfc2797.txt.
[RFC3647]	*"Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework,"* S. Chokhani and W. Ford , November 2003. http://www.ietf.org/rfc/rfc3647.txt.
[WAPPUSH]	*"WAP Push OTA Protocol"*, WAP-235-PushOTA, WAP Forum™, http://www.openmobilealliance.org
[WMLScript]	*"WMLScript Language Specification"* , WAP-193-WMLScript, WAP Forum™, http://www.openmobilealliance.org

3. Terminology and Conventions

3.1 Conventions

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in [RFC2119].

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

3.2 Definitions

CA Information Service	A service that provides the trusted CA information, which includes the CA root certificate and information necessary to validate the CA root certificate.
Certification Authority	A certification authority is an entity that issues/updates/revokes public key bearing certificates in response to authenticated requests from legitimate registration authorities. The CA holds a private key used to sign domain member key bearing certificates.
PKI Portal	This is the entity that performs the RA and/or CA functions defined in this specification. It is both WAP and PKI aware.
PKIX	The IETF’s Public Key Infrastructure Working Group.
Proof of possession	In order to avoid certain substitution attacks a PKI portal may require that clients (in the PKI sense) demonstrate that they possess the private key corresponding to the public key they wish to be certified. This can be done in many ways, e.g. by having the client sign a challenge.
Registration Authority	A registration authority is an entity authorised to make requests to issue/revoke/update certificates to a CA. The registration authority can be considered similar to an account manager in function and is responsible for member enrollment and/or attribute assignments.
Subscriber Identity Module	A tamper resistant device in a wireless system holding subscriber identity and authentication information. The SIM card can also be used to run applications needing a secure environment.
Trusted CA Information	This refers to the information which is stored by a PKI entity and which indicates that a given certification authority is trusted as a root CA by that entity. This must include a public key and typically also includes a name and a validity period, often stored in the form of a self-signed certificate. In this specification, the term is also used when discussing a transfer format for this information, in particular for use when downloading trusted CA information over the air.
Wireless Identity Module	The WAP Identity Module (WIM) is used in performing WTLS and application level security functions, and especially, to store and process information needed for user identification and authentication. The WPKI may use the WIM for secure storage of certificates and keys.

3.3 Abbreviations

AKID	Authority Key IDentitfier
CA	Certification Authority
CMC	Certificate Management over CMS
CMP	Certificate Management Protocol
CPS	Certification Practice Statement
CRL	Certificate Revocation List
DER	Distinguished Encoding Rules (ASN.1)
EC	Elliptic Curve
GSM ID	Global System for Mobile Communication Identifier
ICC	Integrated Circuit Card

ME	Mobile Equipment
OCSP	On-line Certificate Status Protocol
OTA	Over-the-air
PIN	Personal Identification Number
PKI	Public Key Infrastructure
POP	Proof Of Possession
RA	Registration Authority
RSA	RSA (Rivest, Shamir, Adleman) public key algorithm
SHA-1	Secure Hash Algorithm 1
SIM	Subscriber Identity Module
WIM	Wireless Identity Module

4. Introduction

The goal of the WAP PKI is to reuse existing PKI standards where available, and only develop new standards where necessary to support the specific requirements of WAP.

In addition to those security documents already produced [WAPWTLS], [WMLScriptCrypto], [WIM] the WAP PKI consists of the following documents:

- WAP PKI Definition (this document)
- WAP Certificate and CRL Profiles Specification [CERTPROF]

The purpose of this document is to define the framework of the WAP PKI. It is divided into a number of sections as follows:

- WAP PKI Model, providing a framework for the implementation of the WAP PKI and the relationship between the wireless components of the PKI defined in WAP, and the existing Internet components defined elsewhere. (e.g. IETF PKIX)
- PKI Operations, describing the operations supported by the model such as RootCA key installation and client registration processes
- Static conformance tables, specifying which feature combinations must or may be implemented

5. WAP PKI Model

This section gives an overview of the PKI model being used in the WAP environment. It includes a partial walkthrough of the generic processes involved when a WAP client registers in a, (for it), new PKI.

The current WAP PKI model defines the functionality needed to manage the security functionality defined in WAP1.2. This can be summarised as follows: -

- CA Public Key Certificates used for WTLS Class 2
- Client Public Key Certificates used for WTLS Class 3
- Client Public Key Certificates used in conjunction with [WMLScriptCrypto] **signText**.

The general model adopted in the current version of the WPKI is as follows: -

- WTLS Server and Root CA certificates stored in the device will be according to WTLSCertificate defined in [WAPWTLS]
- Client certificates (WTLS & application) and CA Roots stored in servers will be according to X.509 as profiled in [RFC3280]
- Client certificates (WTLS & application) and CA Roots which are to be sent OTA and/or stored in WAP client devices will be according to X.509 as profiled in [CERTPROF]
- Storage of the certificate URL in the device, rather than the full client certificate, is the preferred model, when X.509 format certificates would otherwise be expected to be transferred OTA.
- Storage of X.509 client certificates in the device is expected to be the exception, unless they are provisioned on the device, through the [WIM] for example.

Certificates can be stored in several locations on the client, on a WIM (either on the same ICC as the SIM or not in a client that supports a SIM) or on the client itself.

5.1 WAP PKI Functions

We now present some illustrative examples to show "typical" WPKI configurations and how these relate to WTLS and **signText**. Note that other configurations are possible, and will be used, so as to match the PKI requirements for particular environments.

5.1.1 WTLS Class 2

The Security layer protocol in the WAP architecture is called the Wireless Transport Layer Security, WTLS. The WTLS layer operates above the transport protocol layer.

The primary goal of the WTLS layer is to provide privacy, data integrity and authentication between two communicating applications. WTLS provides functionality similar to TLS 1.0 and incorporates new features such as datagram support, optimised handshake and dynamic key refreshing. The WTLS protocol is optimised for low-bandwidth bearer networks with relatively long latency.

WTLS Class 2 provides the capability for the client to authenticate the identity of the gateway it is communicating with. The following diagram gives a generic overview of the steps necessary to enable WTLS Class 2.

Currently WTLS operates between a WAP client and a gateway (Case 1 below). Future versions of WAP will allow a WTLS session to terminate beyond the gateway at an application or origin server (Case 2 below).

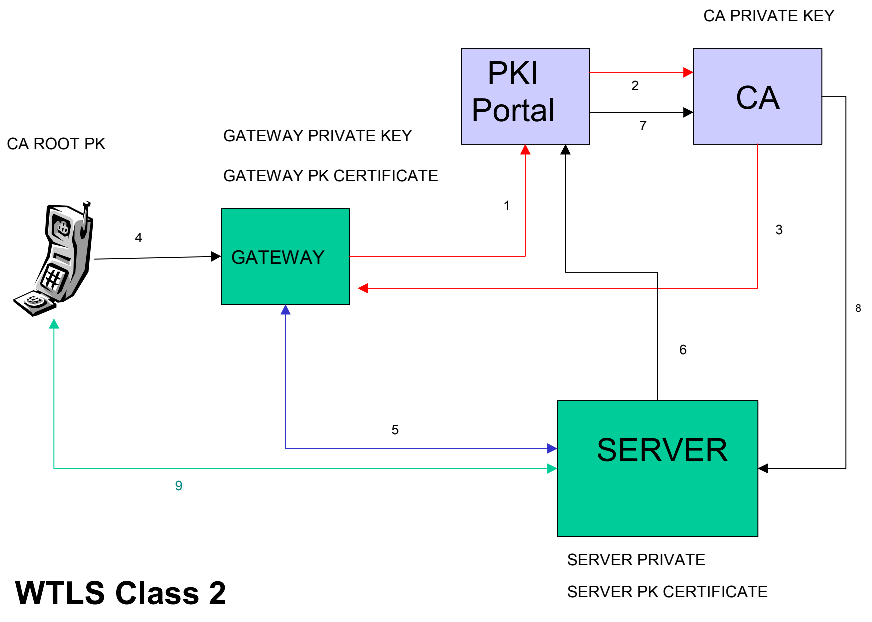

Figure 1 - WTLS Class 2 WPKI

In the diagram above the device is provisioned with (or loads OTA) some CA Root Public Key information. The WAP Gateway Generates a key pair - public key & private key and:

(1) Gateway sends certificate request to PKI Portal

(2) Portal Confirms ID and forwards request to CA

(3) CA send Gateway Public Certificate to Gateway (may be via Portal)

Case 1 - Two Phase Security:

(4) WTLS Session established between Phone and Gateway

(5) SSL/TLS session established between Gateway and Server

Case 2 - “End to End Security model”:

(6) Server sends certificate request to PKI Portal

(7) Portal Confirms ID and forwards request to CA

(8) CA sends Server Public Certificate to Server

(9) WTLS Session Established from Phone to Server (routing is via Gateway, but communication is opaque to Gateway).

5.1.2 WTLS Class 3

WTLS Class 3 authentication is (from the PKI viewpoint) almost the same as the configuration shown below for **signText** - the difference being that the client's private key is used to sign a "challenge" from the WTLS server. Both direct to gateway and end-to-end (to server) modes are possible.

5.1.3 signText

signText provides a mechanism for a client device to create a digital signature of text sent to it using WMLScript.

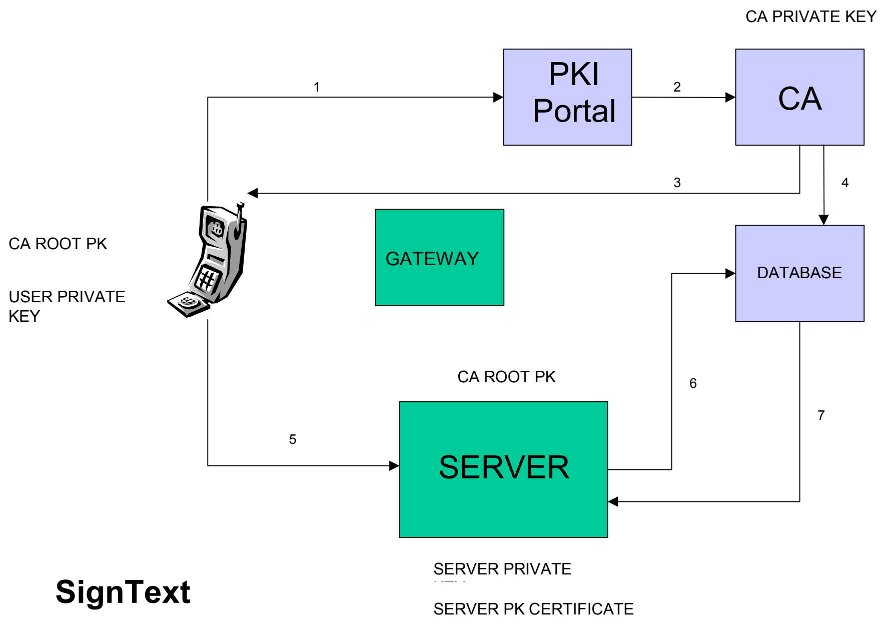

Figure 2 - signText and WPKI

In this case root CA Public Keys must be provisioned (or loaded OTA) in both the device and the server.

(1) Phone requests Certificate from PKI portal (via gateway).

(2) Portal confirms ID and passes request to CA

(3) CA generates User Certificate and sends Certificate URL to client. (Alternatively the CA can send the entire client certificate to the device [to be stored on the WIM for example])

(4) CA populates Database with User Public Key Certificate (if necessary)

(5) User signs transaction at client, and sends transaction, signature & CertificateURL (or certificate) to server (logically via gateway)

(6) Server uses CertificateURL to retrieve User Certificate from database (if not already in possession of certificate)

(7) CA Database sends user certificate to database (if necessary).

5.2 Private Key Capability

WAP clients are assumed to be able to have (at least) two different signing keys - one for WTLS client authentication and one for application layer signing (**signText**). It is envisaged that almost all clients will fit into one of the following classes:

- No private keys
- One private key (either for authentication or signing)
- Two private keys (one for authentication and one for signing)

That is, cases where clients have more than two private keys are not considered further (though may be supported).

In order for a WAP client's signature (whether for authentication or signing) to be trusted by servers, it will sometimes be necessary for the client to be registered in a new PKI, which is trusted by the server. This demonstrates the need for an application layer PKI registration functionality. Note that a client who registers in such a PKI does not necessarily need to trust that PKI, so that installation of trusted CA information may not be required. (Of course, the client may need to be able to trust the PKI in order to authenticate a WTLS server.) The client will however have to be able to identify itself in the server's trusted PKI.

We use the term PKI portal for any PKI entity (typically an RA, CA or OCSP [RFC2560] responder) with which the WAP client communicates during PKI operations. There is no assumption that the client communicates directly with the PKI portal, though any security mechanisms applied (e.g. signature) must be end-to-end between the client and PKI portal. The PKI portal need not be co-located with either the service provider or operator.

In many cases, registration of WAP clients in a PKI will have occurred as part of the provisioning of a WAP device, however this specification also provides mechanisms that allow this registration to be carried out, over the air, after the device has already been provisioned. A "typical" PKI registration therefore may involve the following types of interchanges:

- Client contacts a service provider (e.g. a content provider which supplies some banking application) attempting to use a service that requires a client signature.
- Service provider requires client to be registered in its chosen PKI; Service provider indicates to client that it should contact a PKI portal (maybe also providing some PKI information - e.g. a CA name).
- Client contacts PKI portal and submits certification request; PKI portal acknowledges receipt of request. The acknowledgement message gives some guidance to the client as to what will happen next.
- Normal certification processes occur in the PKI; this may result in near instant certification or may involve a significant time lapse. (The details of the internal PKI operations are out-of-scope here, but are covered by e.g. PKIX.)
- Some time later the client reconnects to the service provider. When producing a signature the client also includes information to identify its certificate.
- The service provider may use this to retrieve the client's certificate from a repository and can then verify the client's signature.

The diagram below shows this style of interaction.

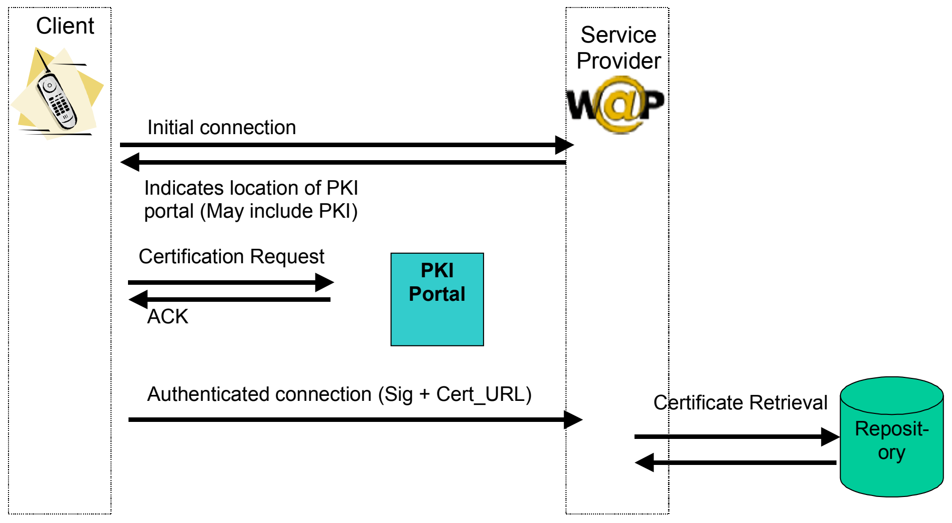

Figure 3 - PKI Portal Interactions

6. PKI Operations

This section describes the PKI operations that are standardized in the WAP context. Note that supporting just these operations does not provide an implementation with a "full" PKI, other standards (e.g. PKIX, [RFC3280]) specify protocols and data formats that can be used in building and deploying a PKI. The approach taken is to minimize the amount of new specification, in order to enable WAP PKI to be deployed quickly.

We first discuss handling of trusted CA information, then WTLS server certification and finally client registration and certificate distribution and certificate URLs.

6.1 Trusted CA Information Handling

Trusted CA information means the information necessary in order to verify a public key certificate issued by that CA, or by CA subordinate to the trusted CA. This information necessarily includes a public key and a name but MAY include other information. In order to provide integrity, trusted CA information is downloaded in self-signed format - note that this does not provide authentication of the data, only integrity. This section specifies two mechanisms for providing the authentication of the trusted CA information, one based on hashing and one based on signing. It is important to note that nothing here provides the "trusted" part of the trusted CA information - trust is provided by the deployed system (including e.g. user acceptance of a CA).

The CA information SHOULD be distributed (i.e. downloaded) to the clients through the appropriate WAP defined protocols such as Provisioning [PROVARCH] or WSP. For Provisioning, the CA Information Service SHOULD be trusted. For WSP, the CA information is pulled when a URL is presented to a user, and only if the URL is not presented in a Push message. The basic server initiated delivery methods such as Push [WAPPUSH] MUST NOT be used.

The client SHOULD accept the CA information only if the CA information is verified through one of the verification methods defined in this specification and the CA information is delivered through the appropriate WAP defined protocols, and the CA Information Service is trusted. If the CA Information Service is not trusted, sufficient user interface functions must be provided to assist the user to make the informed decision on whether or not the downloaded CA information can be trusted before acceptance.

Even if the CA Information Service is trusted, the user MAY be prompted to confirm the acceptance of a new trusted CA information content. If the CA Information Service is not trusted, the user MUST be prompted to confirm the acceptance of a new CA information content and MUST be warned of the security impact of accepting it. If a new CA information content is rejected, it MUST NOT be used in the client.

This specification does not mandate the mechanisms by which a client determines that a CA Information Service is "trusted", or "not trusted". In fact, as "trust" is not a binary relationship, implementers MAY choose to support additional levels of "trust" for CA Information Services.

Clients MUST be able to understand and process the CA information content types, including **application/vnd.wap.hashed-certificate** and **application/vnd.wap.signed-certificate**. Client MAY be able to understand and process the **application/vnd.wap.rollover-certificate** content type.

Clients SHOULD provide mechanisms for deletion of CA information, although this specification does not define such mechanisms. Note that as substantial denial-of-service can result from accidental deletion of CA information, implementers SHOULD take as much care with deletion of CA information, as they take with addition of CA information.

6.1.1 Client Handling of Trusted CA Information

WAP Clients:

- MUST support long-term storage and local administration of trusted CA information provided (e.g. provisioned) in self-signed WTLSCertificate format,
- MAY support long-term storage and local administration of trusted CA information provided (e.g. provisioned) in self-signed X.509 format

The details of the local administration of this data are out of scope of this specification.

6.1.2 Server Handling of Trusted CA Information

WTLS Servers and verifiers of **signText** output:

- MAY support long-term storage and local administration of trusted CA information provided (e.g. provisioned) in self-signed WTLSCertificate format,
- MUST support long-term storage and local administration of trusted CA information provided (e.g. provisioned) in self-signed X.509 format

The details of the local administration of this data are out of scope of this specification.

6.1.3 Out of band hash verification method

Trusted CA information MAY be represented as a self-signed X.509 public key certificate or as a self-signed WTLSCertificate.

The format (see [WAPWTLS] for details of this notation) of the hashed data is:

```
struct {
       CharacterSet character_set;
       opaque                     displayName  <1..2^8-1>;
} CertDisplayName;

enum { sha1 (0), 255 } HashAlgorithm ;

struct {
       uint8                      version;
       CertDisplayName            displayName;
       Certificate                trustedCACert;
       opaque                     cainfo_url <0..2^8-1>;
       HashAlgorithmhash_alg;
} TBHTrustedCAInfo;
```

Item	Description
Version	The version of this data structure. For this specification it MUST be 1.
DisplayName	A name for the CA, suitable for display on the WAP client device.
TrustedCACert	The CA Certificate. WAP clients MUST support the WTLSCertificate format and MAY support the X.509 format. If using X.509 format, this MUST meet the profile specified in [CERTPROF]
cainfo_url	A URL, which MUST be supplied which specifies where the client can get further information about the CA. This URL MUST be potentially accessible from the WAP client. The intention here is to enable the PKI portal to give pointers to the CA's CPS [RFC3647] etc
hash_alg	The algorithm used to hash, this MUST be SHA-1

Table 1 - TBHTrustedCAInfo Structure Definition

The encoding of this is to be the value of an **application/vnd.wap.hashed-certificate** MIME type [MIMETYPE].

The client MUST also ensure that the signature on the self-signed certificate is correct before using the public key contained in the self-signed certificate.

The security of this mechanism consists in downloading the CA information over the air and having the user enter the "display" form of the hash of this information via e.g. the keyboard. The hash value itself is not sent over the air and MUST be sent to the user via an out-of-band channel.

For the purposes of this specification, receiving data out-of-band means simply that the data is not received via a channel that can be attacked at the same time as the in-band (WAE) channel without the attacker expending significant additional resource based on an entirely different attack mechanism. For example, in the context of WAP over GSM/CSD, an unauthenticated SMS channel is considered in-band and should not be used to distribute the hash value.

The device must do whatever possible to ensure that the hash has, in fact, been received via an out-of-band mechanism. The mechanisms used to enforce this are out of scope of this specification.

After the display form of the hash is entered and the CA information has been received (which can occur in either order); the device MUST hash the information received over the air and compare (the leftmost bits of) this against the hash value that was entered.

If they match then the CA information can be accepted. In this case, the CA information SHOULD be displayed to the user and the user SHOULD be allowed to accept or reject the CA information.

If the hash values do not match, then the device SHOULD inform the user, and offer the opportunity to enter the correct hash, either immediately or at a later time. The user SHOULD be presented with the option to cancel the operation, in which case the rejected information MUST be deleted.

In order to allow a long value to be entered by the user, we include in the display form of the hash some checksum bits. The device MUST check that these are correct as the display form of the hash is entered, and MUST only accept the entered value when the checksum validation succeeds. If the user enters an incorrect value, the device MUST inform the user and allow re-entry of the relevant (part of) the display form of the hash.

Note: The checksum allows schemes where the user enters the hash value before downloading the CA information.

If the user enters a sufficient number of bits, then this mechanism is secure.

Users MUST enter exactly 80 "effective" bits of hash, which extracted from the display form and are the leftmost 80 bits of the SHA-1 hash of the CA information.

Let "hash" be the SHA-1 hash of the encoding of (the encoding of) a TBHTrustedCAInfo structure. This consists of 160 bits (h_0,…h_{159}, in network byte order, h_0 being the leftmost bit).

The display form of the hash consists of 30 decimal digits, constructed of 5 blocks of 6 digits. The leftmost 5 digits of each block represent 16 bits of the effective hash, (i.e. can take the value '00000' to '65535' decimal), the sixth digit of each block is a check digit for the block. Block zero consists of h_0 to h_{15} (with h_0 being the most significant bit), followed by the associated check digit; block one consists of h_{16} to h_{31}; followed by the associated check digit etc. That is if h_0 is '1'B and h_1 to h_{15} are all '0'B, then the sixteen bits have the value '8000'H or '32678' decimal, and the check digit is '5' decimal (see below for the check digit calculation method).

The check digit is calculated using the same mechanism as used for credit card numbers, that is, the LUHN formula (mod 10) as described below.

The following steps are required to validate a 6 digit group:

Step 1: Double the value of alternate digits of the number beginning with the second digit from the right (the first right--hand digit is the check digit.)

Step 2: Add the individual digits comprising the products obtained in Step 1 to each of the unaffected digits in the original number.

Step 3: The total obtained in Step 2 must be a number ending in zero (30, 40, 50, etc.) for the number to be validated.

Example 1: to validate the group 398719, corresponding to '9BBF'H as the sixteen bits of hash:

Step 1:

```
 3 9 8 7 1 9
x2  x2  x2
------------
 6  16   2
```

Step 2: (6) + 9 +(1+6) + 7 + (2) + 9

Step 3: Sum is 40 = 0 (mod 10): group is validated

Example 2: to validate the group 326785, corresponding to '8000'H as the sixteen bits of hash:

Step 1:

```
 3 2 6 7 8 5
x2  x2  x2
------------
 6  12  16
```

Step 2: (6) + 2 +(1+2) + 7 + (1+6) + 5

Step 3: Sum is 30 = 0 (mod 10): group is validated

6.1.4 Signature Verification Method

With this method the trusted CA information is presented in a signed format as follows:

```
struct {
       uint8                      version;
       CertDisplayName            displayName;
       Certificate                trustedCACert;
       opaque                     cainfo_url <0..2^8-1>;
       Certificate                signerCert;
       SignatureAlgorithm         sig_alg;
} TBSTrustedCAInfo

struct {
       TBSTrustedCAInfo           tc_info;
       Signature                  signature;
} SignedTrustedCAInfo
```

Item	Description
Version	The version of this data structure. For this specification it MUST be 1.
DisplayName	A name for the CA, suitable for display on the WAP client device.
TrustedCACert	The CA Certificate. WAP clients MUST support the WTLSCertificate format and MAY support the X.509 format. If using X.509 format, this MUST meet the profile specified in [CERTPROF].
cainfo_url	A URL, which MUST be supplied, which specifies where the client can get further information about the CA. This URL MUST be accessible from a WAP client. The intention here is to enable the PKI portal to give pointers to the CA's CPS [RFC3647] etc

signerCert	The public key certificate of the signer.
sig_alg	The signing algorithm. This MUST be one of the algorithms specified in [WTLS].
Signature	The signature value

Table 2 - SignedTrustedCAInfo Structure Definition

The encoding of this is to be the value of an **application/vnd.wap.signed-certificate** MIME type [MIMETYPE].

The client MUST be able to trust the signer of this structure before the contained certificate is used. The client MUST verify both the outer and inner signatures (the inner being on the self-certified structure, the outer signature being the signature on the `TBSTrustedCAInfo`).

The client MUST verify the signerCert. The signer may be a CA or an end entity. If the signer is a CA independent of the "new" CA, then there MAY be implications for the certification practices and certification policies of the two CAs (e.g. they might have to be "commensurate" for some definition of commensurate). If the signer is not a CA, then the device MUST ensure that the signer is certified by a currently trusted CA.

Devices MUST provide a mechanism through which some CAs can be marked as trusted for this particular purpose and MUST enforce this privilege when using this mechanism. This means that not all trusted CAs can "introduce" new CAs. Devices SHOULD also provide a mechanism to control whether a CA installed via this mechanism is allowed to "introduce" new CAs. Later versions of this specification MAY include standardized schemes for managing this, and other, PKI related privileges, however this is out of scope for this version.

Note that this means that the first installed CA has the opportunity to fairly easily introduce new CAs, and that users are reasonably likely not to understand the significance of accepting a new CA using this mechanism. For this reason, one configuration which has been examined is where the "operator" provisions its own CA on the device and thereafter limits the ability of other entities to download new CA information. Mechanisms for enforcing this are out of scope of this specification.

6.1.5 Trusted CA Key Roll-over

When it is deemed necessary to update a root certificate, a root level CA operator generates a new root key-pair and a self-signed root certificate. The old root private key is used to sign a message of the form described in section 6.1.4 containing the new root certificate. A CA operator SHOULD store copies of these update messages to distribute to clients who have missed a rollover. Multiple messages of this form may be chained together and distributed in order to update clients who are more than one root certificate away from the current root certificate. A client MAY indicate to the CA operator which root certificate they currently have by sending a hash for the root certificate. Using this information, the CA operator can determine what root certificate rollover information to send.

This mechanism can be used either to rollover a CA key or to extend the lifetime of trusted CA information already present on the device.

To validate a root certificate rollover message consisting of a sequence of structures from 6.1.4 from a CA operator, a client MUST validate each trusted CA information block sequentially. To do this, for each trusted CA information block, a client MUST:

1. Use the root public key that they currently have to verify the signature on the trusted CA information block.
2. Perform all the checks indicated in section 6.1.4 on the trusted CA information block.
3. Accept the root certificate rollover, replacing the current root CA certificate with the new root CA certificate. The user MAY be informed of this.

```
Struct {
   SignedTrustedCAInfo   signed_trusted_CA_information <0..2^16-1>;
}  RootCertificateRolloverBlock
```

This structure MUST be delivered to the device using the **application/vnd.wap.rollover-certificate** MIME type [MIMETYPE].

6.2 Server WTLSCertificate Handling

This section describes how WTLS server certificates (which use the WTLSCertificate format) are handled. First we describe a method to send a certification request to a CA and then we describe a method for retrieving short-lived WTLSCertificates.

The general model here is that the server sends a certification request to a CA. In response the CA may generate and return a long-lived WTLSCertificate or the CA may instead issue a sequence of short-lived WTLSCertificates which can be retrieved by the WTLS server.

6.2.1 Server WTLSCertificate Issuing

Certificate Requests for WTLS Server Certificates SHOULD be PKCS #10 format as specified in [PKCS10]. This value should be Base-64 encoded [BASE64] and demarcated in a human friendly manner as follows:

```
-----BEGIN CERTIFICATE REQUEST-----
<Base-64 encoded RFC 2986 blob here>
-----END CERTIFICATE REQUEST-----
```

This value is typically entered into a web form offered by the CA, or included in an email sent to the CA.

Not all the contents of the request will necessarily appear in the issued WTLS server certificate as CA policies and real world character set constraints must be considered. Further a CA MAY modify values for accuracy or presentation purposes. Where the PKCS#10 [PKCS10] format is used, the naming profile for WTLSCertificates (specified in [WTLS]) SHOULD be used when building the request DN as some or all of the subject name may be presented to the user.

The CA may then return a certificate as a Base-64 encoded [BASE64] WTLSCertificate demarcated in a human friendly manner as follows:

```
-----BEGIN WTLS CERTIFICATE-----
<Base-64 encoded WTLSCertificate blob here>
-----END WTLS CERTIFICATE-----
```

WTLS servers SHOULD support receipt of such a format (e.g. from a file, or cut & paste into an administrative GUI).

Furthermore, a URL MAY be delivered for the short lived certificate retrieval (6.2.2). A secure channel (e.g., SSL) SHOULD be used when the short lived certificate is retrieved. This specification does not define a mechanism for automating delivery of this URL to the WTLS server.

6.2.2 Retrieval of short-lived WTLSCertificates

Server authentication in the wireless environments differs substantially from that in the wired environments because of performance, bandwidth, roundtrips and code-footprint considerations. One of the issues in the wireless environments is checking for revocation. Traditional means such as downloading CRLs are not feasible and means such as OCSP add roundtrips, validation steps and additional trust points on the client. To overcome these issues, we introduce the concept of short-lived server WTLS certificates that convey both authenticity of the server and obviate the need for a separate revocation check.

WAP applications require a server certificate revocation capability, to ensure that, in the event a server is compromised or decommissioned, users cannot unwittingly continue to execute what appear to be valid, secured transactions with a rogue server. Wireless devices typically do not have the local resources nor the communication bandwidth to implement revocation methods used in the wired world such as certificate revocation lists (CRLs) or the online certificate status protocol (OCSP).

To satisfy revocation requirements, WTLS servers MAY implement the *short-lived certificate* model, as the means of satisfying revocation requirements. With this approach, a server or gateway is authenticated once in a *long-term credentials* period – typically one year – with the expectation that the one server/gateway key pair will be used throughout that period. However, instead of issuing a one-year-validity certificate, the certification authority issues a new short-lived certificate for the public key, with a lifetime of, say, 48 hours, every day throughout that year. The server or gateway picks up its short-lived certificate daily and uses that certificate for client sessions established that day. If the certification authority wishes to revoke the server or gateway (e.g., due to compromise of its private key), it simply ceases issuing further short-lived certificates. Clients (in this case, WTLS servers) will no longer be presented with a currently valid certificate, hence will cease to consider the server authenticated.

Note that there must be an overlap period in which both a new short-lived certificate and the preceding short-lived certificate are both valid. Note also that the method requires that a wireless device have a sufficiently accurate clock and, ideally, knowledge of the time zone since certificate validities are normally expressed relative to UTC time. To avert usability problems relating to such timing requirements, it MAY be appropriate to employ a certificate overlap time of at least 24 hours. In particular, where short lived WTLSCertificates for servers last longer than a day then an overlap of at least 24 hours SHOULD be used.

Implementers should take note that use of short-lived WTLS certificates MAY expose the WTLS server to new denial-of-service attacks (e.g. if the attacker tries to flood the responder). Where possible, implementers SHOULD take care that they include measures to detect and recover from such attacks.

6.2.3 Request/Response Protocol

The request for a short-lived certificate is a simple HTTP GET for the URL which was supplied by the CA.

The response is a MIME [MIME] formatted response. For a returned X.509 certificate this is:

```
Content-Type: application/x-x509-user-cert
<binary x.509 blob here>
```

For a returned WTLS certificate :

```
Content-Type: application/vnd.wap.wtls-user-cert
<binary wtls server cert blob here>
```

Note: This certificate is a "user" certificate in the sense that it is not a CA certificate; in this case, the certificate is for a WTLS server.

In the event of an error, the response is:

```
Content-Type: text/plain
<error text>
```

The error text is intended for human consumption. Implementers SHOULD treat all responses which do not contain "good" WTLS certificates as errors.

WTLS servers using this protocol:

- MUST support the use of a "https" URL, (in order to be able to avoid the simple denial-of-service attack resulting from insertion of a text/plain error message - note: this constraint does not say that they "MUST use https", just that it be possible to do so)
- MUST support receipt, and handling, of an **application/vnd.wap.wtls-user-cert** response content type
- MAY support receipt of an **application/x-x509-user-cert** response content type, and,
- MUST support receipt and handling of the text/plain response content type, indicating an error.

6.3 Client Registration

This section describes client certificate enrollment mechanisms and details. Two enrollment models are defined in Section 6.3.1 and Section 6.3.2. The first model is based on WAP concepts and technology as defined in WAP 2.0 and is included to ensure backwards compatibility with existing implementations. The second model defines the use of standard Internet enrollment mechanisms based on PKCS#10 [PKCS10] message formats and describes how a PKI portal can request the generation of a key in the field.

The last three sections, Sections 6.3.3-6.3.5, describing PKI Portal discovery, enrollment authentication and enrollment response, contain text that is common to both models.

Clients SHOULD support the WAP enrollment model as described in section 6.3.1 to ensure backwards compatibility and MUST support the Internet enrollment model as described in Section 6.3.2.

6.3.1 WAP Enrollment Model

The WAP enrollment model assumes that one or more key pairs and X.509 certificates have been provisioned on the device before being shipped to the field. These certificates don't necessarily reflect a user identity but identify the device itself. Typically two keys are provisioned on the device: an authentication key and a non-repudiation key. The authentication key is used during WTLS Class 3 or TLS client authentication. The non-repudiation key is used for application layer signatures as provided for by the [WMLScriptCrypto] **signText** function or the ECMA Script Crypto **signText** function

PKI portals MUST be able to accept any relevant format (specified in WTLS or **signText**) that includes the public key. PKI portals MAY use any additional information provided (e.g. a subject name, or the fact that the key being registered has already been certified by some party) in their further processing. This allows clients with a variety of initial configurations to register with a PKI portal.

WSP is used to transfer necessary naming information and passwords. Content of the naming information is up to the authority, but would typically contain information specified in [RFC2511]. Similarly, passwords are also up to the authority. This information is used to authenticate the user during certification. WTLS or TLS encryption is used to protect the passwords.

6.3.1.1 Proof of possession

The WAP enrollment model dictates that the client connects to the PKI portal and then uses either WTLS Class 3 authentication or the [WMLScriptCrypto] **signText** function so that proof-of-possession can be verified using the corresponding public key. In order to be able to provide this proof-of-possession the client MUST be able to produce a form of the relevant public key that conforms to the WTLS and **signText** specifications. The keys MAY be self-signed or signed by a third party or may even contain a zero length signature. A Client MAY generate such a self-signed format itself.

Note that even though WTLS or **signText** is used, the PKI portal only need verify the signature to ensure proof-of-possession which differs from the normal case of Class 3 WTLS authentication or **signText** verification. Normally a WTLS or **signText** verification implementation will verify the certificate of the client in order to provide authentication. In order to avoid potential configuration or security problems with mixing authentication and proof-of-possession in the same deployment it is RECOMMENDED that PKI portals be deployed on "special" WAP gateways, which are accessed using the end-to-end security mechanisms specified in [E2E]. This configuration means that it is not necessary to mix "proof-of-possession" mode and "authentication" mode in the same gateway, which diminishes the likelihood of misconfiguration resulting in security breaches

6.3.1.2 Certifying Authentication Keys via WTLS

WTLS is used:

- to transfer the public key to be certified
- for Proof of Possession (POP) of the private key
- to authenticate registration service (WTLS server authentication).

The public key is transferred in an existing certificate.

The PKI portal validates POP in this case by the validation of a successful WTLS Class 3 handshake.

Implementers should note that, depending on policy, it may be advisable for the PKI portal *not* to indicate a list of trusted CAs (e.g. in a WTLS handshake) so that the client can use whatever form of public key it chooses. If a PKI portal did indicate some set of CAs, then a client who wasn't previously certified by any of them could drop the connection.

6.3.1.3 Certifying Signing Keys via signText

signText is used:

- to transfer the public key to be certified
- for Proof of Possession (POP) of the private key

The communication between User and PKI portal is based on WSP. Information that the PKI portal requires is passed to the client as WAE content. At least part of the information (possibly containing a challenge from the PKI portal) is signed by the client using the **signText** function. The PKI portal validates POP in this case by the validation of the result of the **signText** function.

An original signing certificate (or self-signed certificate) may be passed in the **signText** result. Note that **signText** implicitly includes a timestamp.

For some applications, a signing certificate may not be required.

6.3.1.4 Certifying Two Keys

If both authentication and signing certificates are required, it may be useful to combine both requests, so that the user would have to enter minimal data. In this case the client and PKI portal MAY interact multiple times.

6.3.1.5 Sample WML Certification Request Information

In order to improve consistency across different PKIs the following piece of WMLScript illustrates gathering registration information from clients. There is no assumption that the PKI portal will honor this information, e.g. the portal MAY replace naming or other information (except the public key).

The public key to be certified is transferred to the portal using either WTLS (as described in Section 6.3.1.2) or **signText** (as described in Section 6.3.1.3). This sample script allows users to request certification of either authentication or signing keys. In either situation, the session SHOULD be protected by WTLS. This script prompts the user for the name to appear in the certificate, a unique identifier provided by the CA, a password to authenticate the user to the CA and the type of certificate requested. This information is then concatenated with fields separated by a colon (:). If certification of a signing key is requested, a random challenge (nonce) provided by the portal is included and the string is signed to provide proof-of-possession.

```
extern function ProduceRequest() {
       var bbnull = " ";
       Dialogs.alert("Certificate Registration");
       Dialogs.alert("Please leave fields blank if you do not have the
                      requested values.");

      var Name = Dialogs.prompt("Name:", bbnull);
      var ID = Dialogs.prompt("Unique ID:", bbnull);
      var Password = Dialogs.prompt("Password:", bbnull);

      var Type = Dialogs.confirm("Which type of certificate are you
                                  requesting?", "Authentication", "Signing");
      var Request;

      if (Type)
           Request = Name + ID + Password;
      else {
           var nonce = "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";
           /* PKI Portal should replace above value with a random value
              that is unique for each transaction. */
           // Comment this out if you don't have a Crypto implementation

           Request = Crypto. (nonce + ":" + Name + ":" + ID + ":"
                                        +Password, 5, 0);
           // Request = nonce + ":" + Name + ":" + ID + ":" + Password, 5,0;
      }
    return Request;
}
```

6.3.2 Internet Enrollment Model

This section describes enrollment functionality based on PKI concepts currently supported by the majority of Internet based browsers. Specifically this includes the ability to request that a client generate a key pair and the use of PKCS#10 [PKCS10] certificate request messages for certificate enrollment. Unlike the WAP model, which depends on the use of either the [WMLScriptCrypto] **signText** function or modified WTLS server, this model takes advantage of the ECMA Script Crypto functions **keyGen** and **genEnrollReq** over xHTML.

This model differs from the WAP model in the following ways:

- The mechanism to enroll for any key, be it an authentication or non-repudiation key, is the same.
- There is no assumption as to the existence of a key pair or certificate for use during the enrollment.
- A PKI portal can request that the client generate a key.

In addition this model can be used to enroll for keys that have been provisioned in the device as described in the WAP model.

Section 6.3.2.1.describes how a PKI portal can request a new key be generated by a client. Section 6.3.2.2 describes the mechanisms used to enroll any key pair into a particular PKI.

See Appendix E for additional informational text and data flows on the use of the ECMA Script crypto **keyGen** and **genEnrollReq** functions.

6.3.2.1 Key Generation

In some scenarios the PKI Portal requires the ability to indicate that a key pair be generated in the field. This functionality is achieved by the use of the **keyGen** ECMA script function as defined in [ESMPCRYPTO].

Clients MUST have the ability to parse and properly respond to key generation request received via the **keyGen** script.

6.3.2.2 Certificate Enrollment

In this model the enrollment mechanisms are the same no matter what kind of key is being enrolled into a PKI. Keys used for authentication and keys used for non-repudiation purposes use the same mechanism. This functionality is achieved by the use of the **genEnrollReq** ECMA script function as defined in [ESMPCRYPTO].

Clients MUST have the ability to parse and properly respond to the enrollment request received via the genEnrollReq script.

PKI portals MUST have the ability to validate and properly respond to PKCS#10 [PKCS10] message received in response to an **genEnrollReq** function.

PKI portals SHOULD have the ability to validate a keyGenAssertion, as defined in the **genEnrollReq** ECMA script function in [ESMPCRYPTO], attribute that may be contained within the PKCS#10 [PKCS10] message.

Proof-of-Possession is implicit in the PKCS#10 [PKCS10] message as the structure, which contains the public key to be certified, is signed with the corresponding private key. The PKI portal MUST validate the signature of the PKCS#10 message.

6.3.2.3 Examples

The following ECMA Scripts [ECMA] illustrate the use of the ECMAScriptCrypto functionality to communicate with a PKI.

6.3.2.3.1 Key Generation Example

```
<html>
<script type="text/ecmascript">
 function GenerateKey() {

     // Prompt user for a few parameters

     var null = "";
     var keyLabel = "";                          // Use any available key
```

```
        var keyAlgString = confirm("Algorithm :", "RSA", "ECC");
        var keyLength = prompt("Key Length [512 - 1024]: ", "1024");
        var keyUsageString = confirm("Key Usage", "Auth", "Signature");

        var keyType, keyUsage;

        if (keyAlgString) {
            keyType=0x0000;
        }
        else {
            keyType=0x0003;
        }

        if (keyUsageString) {
           keyUsage=0x0004;
        }
        else {
           keyUsage=0x0100;
        }

        var result = crypto.keyGen(keyLabel, keyType, keyLength, keyUsage,
                                   null, null, null);
        return result;
   }
   </script>

   <body>

   <form>
   <input type="button" value="Generate Key Pair" onclick="GenerateKey()"/>
   </form>

   </body>
   </html>
```

6.3.2.3.2 genEnrollReq Example

```
   <html>
   <script type="text/ecmascript">
    function GenerateCertificateRequestMessage() {
        var RSA = 0x0000;
        var AuthKey = 0x0004;
        var name=prompt("Name: ");
        var company=prompt("Company: ");
        var country=prompt("Country: ");

        var dn = "CN = " + name + "O = " + company + "C = " + country;

        var request = crypto.genEnrollReq
                         (dn, RSA, 1024, AuthKey, 0, "", "", "");

        return result;
   }
   </script>

   <body>
   <p>Certificate Enrollment Page</p>
   <form>
```

```
<input type="button" value="Generate P10 Request"
       onclick="GenerateCertificateRequestMessage()"/>

</form>

</body>
</html>
```

6.3.2.3.3 Key Generation and GenEnrollReq Example

```
<html>

<script type="text/ecmascript">

function GenerateCertificateRequestMessage(keyID) {

     var keyType = 0x1000;   // unspecified
     var keyUsage = 0x0000;  // unspecified
     var keyLength = 0x0000; // unspecified
     var name=prompt("Name: ");
     var company=prompt("Company: ");
     var country=prompt("Country: ");

     var dn = "CN = " + name + "O = " + company + "C = " + country;

     var request = crypto.genEnrollReq(dn, keyType, keyLength, keyUsage,
                                       1, keyID, "", "");

     return request;
}

function GenerateKey() {

     // Prompt user for a few parameters

     var null = "";
     var keyLabel = "BankCoAuthKey"; // Use the Bank Co.’s Auth Key

     var keyType = 0x1000;   // unspecified as keyLabel is present
     var keyUsage = 0x0000;  // unspecified as keyLabel is present

     var result = crypto.keyGen(keyLabel, keyType, keyLength, keyUsage,
                                null, null, null);
     return result;
}

function GenAndEnroll() {
  var publicKeyHash;
  var pkcs10;

  publicKeyHash = GenerateKey();

  pkcs10 = GenerateCertificateRequestMessage(publicKeyHash);
  return pkcs10;
}

</script>
```

```
<body>
<p>Key Gen and Certificate Enrollment Page</p>

<form>
<input type="button" value="Gen Key and Enroll" onclick="GenAndEnroll()"/>
</form>

</body>
</html>
```

6.3.3 PKI Portal Discovery

Where clients are required, or wish, to register over-the-air with a PKI, they contact a PKI portal. The client will typically discover the PKI portal either via manual browsing or through a URL contained within a WML or xHTML page. Different URLs can be used to select between different PKI registration options. For example, the certificates resulting from registration starting at "pkip.org.com/pkip/banking" and "pkip.org.com/pkip/stock-trades" may reflect separate certification and certificate policies.

The PKI portal MAY be a combined CA and RA (really just a CA) in which case there will be no need to use PKI messaging in order to create certificates.

6.3.4 Authenticating the Enrollment

There are many ways for PKI portals to authenticate the enrollment of a certificate. A PKI portal that trusts the issuer (e.g. an operator or bank) of any device certificates could choose to use the validity of these certificates to determine authenticity of the enrollment. Also, the PKI portal may pre-authenticate users by sending (out of band) usernames and passwords which are used during enrollment.

The client MAY supply further information, or the PKI portal MAY derive additional information from other sources. The PKI portal MAY re-format the public key and other information into a certificate request to be sent to a CA. This request MAY use CMP [RFC2510] or CMC [RFC2797] formatting as appropriate.

6.3.5 Enrollment Response

The PKI portal MAY then respond to the client in a standard fashion using the response types defined below. The response MAY include a certificate and/or certificate URL resulting from the exchange or an indication that the client SHOULD return later to retrieve this information.

6.3.6 Delivery of Certificates

In some cases it is sufficient that the CA only publish the certificate (in an LDAP directory or other repository) or stores it in a database, and therefore the certificate does not need to be delivered to the handset. So, it is sufficient to acknowledge the user (using WAE application) that the certificate has been successfully issued and published. In this case there is no information about the certificate in the handset, just the original key id, which can be used in WTLS or TLS and in applications using signatures (**signText**).

If the certificate has to be delivered, this can be done using WAE with the WSP content type

```
application/x-x509-user-cert (WSP assigned number 0x1B).
```

Another scenario is to deliver only a certificate “pointer”. An example of such is a certificate URL, used in PKCS#15 and WIM specifications. Finally, certification can require significant elapsed time. The WAE application should be contracted to handle this. E.g. the user is informed that certification is initiated, but actual certificate delivery would take place later. In either of these cases the WSP content type **application/vnd.wap.cert-response** MAY be returned or an application specific response MAY be returned.

Note: If the client passes a certificate URL rather than the certificate itself, it is requesting the server to do work (i.e. retrieve the certificate indicated in the certificate URL) prior to the client authenticating itself. A potential denial of service attack exists where a client deliberately passes an invalid certificate URL. Servers may protect themselves against such attacks by various means, e.g. by only "following" URLs which match some configured criteria.

The “ca_domain” value may be used e.g. by clients to compare it with values given by origin servers in [WMLScriptCrypto] Library’s **signText** TRUSTED_KEY_HASH fields. Thus issuing a CertResponse cert_info with a ca_domain of IdentifierType “null” would make that impossible.

The content corresponding to **application/vnd.wap.cert-response** contains the structure defined below:

```
enum { cert_info (0), cert(1), referral(2), (255) } CertRespType;

struct {
  uint8               version;
  CertRespType        type;
    select (type) {
        case cert_info:
              CertDisplayName        display_name;
              Identifier                           ca_domain;
              Identifier                           subject;
              opaque                 url<0..255>;
       case cert:
              CertDisplayName        display_name;
              Identifier                           ca_domain;
              Identifier                           subject;
              X509Certificate        cert;
        case referral:
              opaque                 url<0..255>;
              uint32                 seconds_to_wait;
    }
} CertResponse;
```

Item	Description
version	The version of this data structure. For this specification it MUST be 1.
cert_info	These fields contain the details of a certificate which has been issued for the client.
	display_name: (max 32 chars, so it can fit in a PKCS#15 label) SHOULD be a human readable name which indicates the services for which the certificate is useful. This field MUST NOT be empty. The character set used here SHOULD be UTF8 (in order to be stored in WIM, it MUST be UTF8).
	ca_domain: SHOULD contain the hash of the CA’s public key in an Identifier.key_hash_sha field. Otherwise the IdentifierType “null” alternative MUST be used to indicate the absence of an explicit CA domain. Note that the ca_domain field might, in some cases, not match the AKID from the client certificate, if e.g. the issuing CA is subordinate CA within a hierarchy and the WTLS servers use the root of the hierarchy as the ca_domain.
	subject: MUST contain the hash of the certified subject public key.
	url: this field, which SHOULD be used contains a URL usable to retrieve the relevant certificate. Section 6.4 specifies the URL scheme. The client SHOULD store this URL along with the ca_domain and use these as required in WTLS or **signText**. Note that this field is expected to be present as in most cases we wish to avoid sending client certificates to clients.
cert	**display_name**: as above

Item	Description
	ca_domain: as above **subject**: as above **cert**: this field contains the client's certificates. Note that, where possible, implementers are encouraged to use the cert_info option in preference to this one. When used, this option normally contains a single X.509 certificate. The client SHOULD store the certificate along with the associated private key. If possible, the client SHOULD verify that the key in the certificate matches its private key.
referral	The client MAY check back at the URL specified (by url) after a delay (specified by seconds_to_wait). A PKI portal, which returns a referral, MUST ensure that clients who do check back again receive a response of the same type (i.e. **application/vnd.wap.cert-response**). Clients MAY use this method to automate retrieval of certificates or certificate URLs.

Table 3 - CertResponse Structure Definition

6.4 Client Certificate URLs

Rather than pass client certificates over-the-air, the approach taken is to prefer to pass certificate URLs over-the-air, which allow the related client certificate to be retrieved by relying parties.

The certificate URL has been used in the PKCS#15 and WIM specifications. It is defined as a standard URL. Functionally it can be anything that enables getting a single certificate path in response to a request to that URL. Possible protocols that could use this include HTTP, LDAP and FTP.

In the WAP PKI the client MAY have multiple certificates for the same key pair, for example if it has registered the same key with more than one PKI. This imposes a way for clients to distinguish the certificate which the relying party should use to verify a client signature (whether for authentication or for signing). We use the certificate URL, which can be stored in the WIM as the mechanism for distinguishing certificates.

The certificate URL SHOULD be used as follows: The client sends the certificate URL to a server which then uses that to get the certificate. The server uses the certificate but never sends that to the handset. This method clearly saves bandwidth. One problem in this approach may be that the server (that would need the user certificate) may not have access rights to a repository containing this certificate (if that is in a private directory).

The party (e.g. WTLS server or signature verification utility) that retrieves the certificate using a URL MUST check that the certificate content matches with the information indicated in the URL (like issuer and serial number).

There are two URL schemes defined: LDAP URLs and HTTP URLs. As devices do not need to parse the URLs, the use of the schemes only impacts the PKI portals and servers.

6.4.1 HTTP Scheme

The format of the HTTP based certificate URL SHOULD be as follows:

```
http://<base URL>?in=<issuer name>&sn=<serial number>[other URL parameters]
```

where

`<base URL>` identifies a server/program;

`<issuer name>` identifies the certificate issuer. It is a Base-64 encoding [BASE64] of the DER encoded **Issuer** field in the X.509 certificate.

`<serial number>` identifies the serial number of the certificate. It is a Base-64 encoding [BASE64] of the DER encoded **serialNumber** field in the X.509 certificate.

`[other URL parameters]` are additional, optional, URL parameters.

[MIME] requires that a Base-64 encoding [BASE64] be represented in lines of no more than 76 characters each. In order to prevent URL encoding problems however, the Base-64 encoding of the fields above must not include line feeds and thus must contain only one line. Note that when Base-64 padding characters (“=”) are used, they must be encoded into the URL format as “%3D”.

Example values from a certificate:

Issuer name:

```
C=US, O=Wap HTTP Searches Inc.
```

DER encoding of issuer name

```
0x302E310B300906035504061302555331 1F301D060355040A (cont'd)
  131657617020485454502053656172636865732049 6E632E
```

Base-64 encoding [BASE64] of DER encoded issuer name

```
MC4xCzAJBgNVBAYTAlVTMR8wHQYDVQQKExZXYXAgSFRUUCBTZWFyY2hlcyBJbmMu
```

Certificate serial number

```
2
```

DER encoding of certificate serial number

```
0x020102
```

Base-64 encoding [BASE64] of DER encoding of certificate serial number:

```
AgEC
```

A URL for the certificate might be (line break for readability):

```
http://www.example.org/cert?in=MC4xCzAJBgNVBAYTA
lVTMR8wHQYDVQQKExZXYXAgSFRUUCBTZWFyY2hlcyBJbmMu&sn=AgEC
```

The URL MUST be sent to the identified server using a HTTP GET message. The response is a MIME [MIME] formatted response. For a returned X.509 certificate this is:

```
Content-Type: application/x-x509-user-cert
<binary X.509 blob>
```

6.4.2 LDAP Scheme

The format of the LDAP based certificate URL SHOULD be as specified in [LDAPURL] and [LDAPSRCH]. The matching rule shall be “certificateExactMatch”, matching on issuer name and certificate serial number.

Example:

```
ldap://ldap.wap/cn=Wap%20User,o=Wap%20LDAP%20Searches%20Inc.,c=US?userCertificat
e??(userCertificate:2.5.13.34:=123456$o=Wap%20LDAP%20Searches%20Inc.,c=US)
```

Note – line break is for display purposes only, and is not present in the actual URL.

Note – Many LDAP servers do not support the “certificateExactMatch” matching rule. Thus, LDAP clients may need to reformat the LDAP URL to remove the matching rule before making the request. All of the values in the userCertificate attribute will then be returned to the LDAP client, which will then have to process the matching rules itself.

The response is an LDAP formatted response.

7. Static Conformance Requirements

This static conformance requirement lists a minimum set of functions that can be implemented to help ensure that WAP implementations using WPKI will be able to inter-operate. The “Status” column indicates if the function is mandatory (M) or optional (O). Where a reference to an entire section of the specification is given without further qualification then implementations MUST support all "MUST" statements in the section, and MAY support all "SHOULD" and "MAY" statements.

Where no sub-function is given a later subsection of this section contains details or the "Status" column applies to relevant parts of the entire section (where the relevant parts are clear from the context).

This section in its entirety only applies to WAP implementation that claim conformance to "the WAP PKI". Parts of this section do apply to all WAP implementations that implement either **signText** or WTLS Classes 2 or 3.

7.1 Client Options

Item	Function	Reference	Status	Requirement
WPKI-C-001	Public key capabilities	6	M	
WPKI-C-002	Private key capabilities	6	O	WPKI-C-010 AND WPKI-C-011 AND WPKI-C-012 AND WPKI-C-013 AND WPKI-C-014 AND WPKI-C-016

7.1.1 Client Public Key Capability Options

This chapter presents client options related to public key operations used for verification purposes (that is, WTLS server authentication). This chapter is related to all WPKI-capable clients.

Item	Function	Reference	Status	Requirement
WPKI-C-003	Local trusted CA information handling.	6.1.1	M	
WPKI-C-004	OTA Trusted CA information download. Hashed CA information.	6.1.3	M	OR
WPKI-C-005	OTA Trusted CA information download. OTA Signed CA information.	6.1.4	M	
WPKI-C-006	Trusted CA key roll-over rollover-certificate[1]	6.1.5	O	
WPKI-C-007	Public key algorithms for certificate signature validation: Either RSA or ECC		M	WPKI-C-008 OR WPKI-C-009
WPKI-C-008	RSA Private key algorithms for signature verification.	[WAPWTLS]	O	WTLS-C-060
WPKI-C-009	ECC Private key algorithms for signature verification. (Curves as defined in [WAPWTLS])	[WAPWTLS]	O	WTLS-C-060

[1] The justification for this is that the useful lifetime of a WAP device or server is expected to be significantly shorter than the typical validity of root CA information.

7.1.2 Client Private Key Capability Options

This chapter is only related to clients that support private keys. The client may implement private key support using WIM or otherwise (including software only).

Item	**Function**	**Reference**	**Status**	**Requirement**
WPKI-C-010	Private key capability (note: this doesn't mean all clients have key pairs, just that they be capable of storing and using private keys) Support for the authentication key (WTLS Class 3)	5.2	O	WTLS-C-070
WPKI-C-011	Private key capability (note: this doesn't mean all clients have key pairs, just that they be capable of storing and using private keys) Support for the **signText** key.	5.2	O	WMLSCrypt-C-001
WPKI-C-012	OTA WTLS client authentication (here used for registration)	6.3.1.2	O	WTLS-C-070
WPKI-C-013	WMLScriptCrypto **signText** (here used for registration)	6.3.1.3	O	WMLSCrypt-C-001
WPKI-C-014	Certificate delivery: Handling of cert-response messages.	6.3.6	O	WIM-ICC-016 AND WIM-C-016
WPKI-C-015	Certificate delivery: x509-user-cert	6.3.6	O	WIM-ICC-016 AND WIM-C-016
WPKI-C-016	Private key algorithms for signing (either RSA or ECC)		O	WPKI-C-017 OR WPKI-C-018
WPKI-C-017	RSA Private key algorithms for signing	[WAPWTLS]	O	WTLS-C-070
WPKI-C-018	ECC Private key algorithms for signing. (Curves as defined in [WAPWTLS])	[WAPWTLS]	O	WTLS-C-070
WPKI-C-023	Support for WAP enrollment model	6.3	O	WPKI-C-010 AND WPKI-C-11 AND WPKI-C-012 AND WPKI-C-013
WPKI-C-024	Support for Internet enrollment model	6.3	M	WPKI-C-025 AND WPKI-C-026
WPKI-C-025	Client ability to parse and properly respond to key generation request received via the **keyGen** script	6.3.2.1	M	ECMACR-C-050
WPKI-C-026	Client ability to parse and properly respond to the enrollment request received via the **genEnrollReq** script	6.3.2.2	M	ECMACR-C-070

7.1.3 Client root certificate storage options

This section specifies the options for client access to root certificates.

Item	**Function**	**Reference**	**Status**	**Requirement**
WPKI-C-023	Support for non-public WTA	5	O	WPKI-C-019 OR WPKI-C-020 OR WPKI-C-022

WPKI-C-019	Accessing root certificates: The client shall be able to access certificates stored in a WIM on the same ICC as the SIM. This option applies to clients that support a SIM and a WIM only.	5	O	WIM-ICC-015 AND WIM-ICC-016 AND WIM-C-017 AND WIM-C-018
WPKI-C-020	Accessing root certificates: The client shall be able to access certificates stored in a WIM not on the same ICC as the SIM. This applies to clients that support a WIM only	5	O	WIM-ICC-015 AND WIM-ICC-016 AND WIM-C-017 AND WIM-C-018
WPKI-C-022	Accessing root certificates: The client shall be able to access certificates stored on the client itself.	5	O	

7.2 Server Options

Since there appears to be 2 types of server, namely the PKI Portal and the WTLS Server, I would add the following Item to select the appropriate type:

Item	Function	Reference	Status	Requirement
WPKI-S-100	Server supports WPKI	6	M	WPKI-S-101 OR WPKI-S-102
WPKI-S-101	Server is a PKI Portal	6	O	WPKI-Portal-S-001 AND WPKI-Portal-S-006 AND WPKI-Portal-S-007
WPKI-S-102	Server is a WTLS Server	6	O	WPKI-S-001 AND WPKI-S-007

7.3 PKI Portal Options

This section specifies the options for PKI portal implementers.

WPKI-Portal-S-001 Status is affected by WPKI-S-100 to WPKI-S-102.

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-001	Public key capabilities	6	O	WPKI-Portal-S-003 AND WPKI-Portal-S-004

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-002	Support Client private keys	6	O	WPKI-Portal-S-012 AND WPKI-Portal-S-013 AND WPKI-Portal-S-014 AND WPKI-Portal-S-016

7.3.1 PKI Portal Public Key Capability Options

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-003	OTA Trusted CA certificate download support: Hashed-certificate.	6.1.3	O	
WPKI-Portal-S-004	OTA Trusted CA certificate download support: Signed root.	6.1.4	O	
WPKI-Portal-S-005	Trusted CA key roll-over: Rollover-certificate.	6.1.5	O	

7.3.2 PKI Portal Options to Support WTLS Server Certification

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-006	Handling of PKCS#10 long term server certification requests.	6.2.1	O	
WPKI-Portal-S-007	Responses to PKCS#10 requests:	6.2.1	O	WPKI-Portal-S-008
WPKI-Portal-S-008	Responses to PKCS#10 requests: Direct return of WTLSCertificate.	6.2.1	O	
WPKI-Portal-S-009	Responses to PKCS#10 requests: Direct return of X.509 Certificate.	6.2.1	O	
WPKI-Portal-S-010	Responses to PKCS#10 requests: Return of URL.	6.2.1	O	
WPKI-Portal-S-011	Short-lived certificate retrieval protocol	6.2.3	O	

7.3.3 PKI Portal Options to Support Client Registration

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-012	WTLS client authentication (here used for registration)	6.3.1.2	O	WTLS-S-070
WPKI-Portal-S-013	WMLScriptCrypto **signText** support (here used for registration)	6.3.1.3	O	
WPKI-Portal-S-014	Certificate delivery: cert-response	6.3.6	O	
WPKI-Portal-S-015	Certificate delivery: x509-user-cert	6.3.6	O	
WPKI-Portal-S-016	Support for certificate URL retrieval:	0	O	WPKI-Portal-S-017
WPKI-Portal-S-017	Support for certificate URL retrieval: HTTP scheme	6.4.1	O	

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-018	Support for certificate URL retrieval: LDAP scheme	6.4.2	O	

7.3.4 PKI Portal Options to interact with X.509 PKI

Item	Function	Reference	Status	Requirement
WPKI-Portal-S-019	Support CMP between PKI portal and RA/CA	6.3	O	
WPKI-Portal-S-020	Support CMC between PKI portal and RA/CA	6.3	O	
WPKI-Portal-S-021	Ability to validate and properly respond to PKCS#10 messages received in response to a **genEnrollReq** function.	6.3.2.2	M	
WPKI-Portal-S-022	Ability to validate a keyGenAssertion, as defined in the **genEnrollReq** ECMA script function	6.3.2.2	O	
WPKI-Portal-S-023	Validate the signature of the PKCS#10 request	6.3.2.2	M	

7.4 WTLS Server Options

This section describes PKI options related to WTLS servers.

WPKI-S-001Status and WPKI-S-007 Status are affected by WPKI-S-100 to WPKI-S-102.

Item	Function	Reference	Status	Requirement
WPKI-S-001	Local trusted CA information handling	6.1.2	O	
WPKI-S-002	Production of PKCS#10 requests	6.2.1	O	
WPKI-S-003	Handling responses to PKCS#10 requests: Direct return of WTLSCertificate	6.2.1	O	
WPKI-S-004	Handling responses to PKCS#10 requests: Direct return of X.509 Certificate	6.2.1	O	
WPKI-S-005	Handling responses to PKCS#10 requests: Return of URL	6.2.1	O	
WPKI-S-006	Short-lived cert retrieval protocol	6.2.3	O	
WPKI-S-007	Support for certificate URL retrieval.	0	O	WPKI-S-008
WPKI-S-008	Support for certificate URL retrieval: HTTP scheme.	6.4.1	O	
WPKI-S-009	Support for certificate URL retrieval: LDAP scheme.	6.4.2	O	

7.5 signText Verifier Options

Item	Function	Reference	Status	Requirement
WPKI-Verif-S-100	Support for **signText** verification.	6	O	WPKI-Verif-S-001 AND WPKI-Verif-S-002
WPKI-Verif-S-001	Local trusted CA information handling.	6.1.2	O	
WPKI-Verif-S-002	Support for certificate URL retrieval.	6.4	O	WPKI-Verif-S-003

WPKI-Verif-S-003	Support for certificate URL retrieval: HTTP scheme.	6.4.1	O	
WPKI-Verif-S-004	Support for certificate URL retrieval: LDAP scheme.	6.4.2	O	

Appendix A. Requirements and Recommendations (Informative)

This (informative) annex records the set of requirements and recommendations that were used as a starting point in developing the WAP PKI specification. It is included in order to provide readers with useful background and to indicate the direction(s) in which the WAP PKI may develop.

Note that the use of MUST/SHOULD here is non-normative, that is, there is no implication that conformance to any part of this section is required for conformance to the WAP PKI.

In this section, requirements and recommendations, which the WAP PKI needs to meet, are presented. Each section also contains some recommendations that the WAP PKI should be defined to meet.

Note that some of these requirements and recommendations are not technical requirements on WAP PKI protocols or data formats, but rather requirements on the deployed PKI (as a whole) with which the WAP entities are interacting. For example, the statements relating to legal significance (or otherwise) are not technical requirements, but can affect the implementation and operation of a WAP aware PKI

Headings are simply for convenience and bear no other significance.

A.1 Trusted CA handling

1. Clients must be able to securely add new trusted CA certificates (or similar information), that was not previously trusted by the client, operator or manufacturer.
2. It must be possible to add new trusted CA certificates (or similar information) over-the-air.
3. Clients must be able to delete trusted CA certificates (or similar information).
4. It must be possible to use information distributed out-of-band to help secure the addition of trusted CA certificates (or similar information).
5. Where out-of-band information is used to secure the addition of trusted CA certificates, the out-of-band information may be supplied either before or after the trusted CA certificates (or similar information) have been added. The trusted CA certificates MUST not be used before the out-of-band information has been used.

Note: Out-of-band is harder than for the Internet case since the phone is pretty much always in-band. This should be borne in mind by implementers and those deploying WAP PKIs.

Recommendations

1. It should be possible to leverage an operator, manufacturer, or other previously trusted CA certificates, to assist the client in assuring the validity of new trusted CA certificates (or similar information).
2. Deletion or addition of trusted CA certificates (or similar information) should provide for explicit user consent.
3. Clients should be able to view the set of currently trusted CA certificates.

A.2 Registration

1. Clients must be able to register within PKIs over-the-air based on either pre-existing trusted -ca information or on trusted certificates (or similar information) that is added as part of the registration process or without having to install trusted certificates (or similar information) for the PKI into which they are registering.
2. It must be possible to use an existing registration to bootstrap a new registration.
3. In order to register their identity with a public key, clients must be able to prove possession of the related private key.
4. Clients must be able to de-register themselves from a PKI.
5. Clients must be able to register either their transport or application layer keys in new PKIs.
6. Clients that support key generation must be able to register newly generated keys in PKIs.

Recommendations

1. Clients should not have to trust a PKI, into which they are registering, except insofar as they potentially accept liability by doing so.
2. It should be possible to use shared secrets, distributed out-of-band, to authenticate clients to the PKI during or after registration.
3. It should be possible to use a shared secret, distributed out-of-band, to authenticate the PKI to clients during or after registration.
4. Clients should be able to (but need not) inform a PKI portal when they de-register.
5. Clients should be able to view their current set of registrations.

A.3 Assurance levels

Recommendations

1. In order to meet service provider's authentication requirements, the WAP PKI should provide for a range of assurance levels.
2. PKI management operations should be suitable for a range of assurance levels, though the details of the PKI messages do not, themselves, determine a level of assurance.

A.4 Legal signatures

1. It must not be the case that all registrations are such that the client's signatures have legal significance.
2. The user must be made aware if registration in a PKI exposes her to any potential liability due to the fact that signatures verified with the new certificate may be legally significant.

Recommendations

1. It should be possible to register a client in a PKI so that the client's signatures can have legal significance.

A.5 General

1. In addition to PKI portal operation by service provider themselves, it must be possible for PKI portal operation by third parties, trusted by the service providers for the PKI operations required.
2. It must be possible to use an end-to-end WTLS connection from clients to PKI portals. This doesn't mean that all PKI operations must use this, just that it be possible.
3. It must be possible to use server-authenticated e2e WTLS connections between clients and PKI portals.
4. It must be possible to use client-authenticated e2e WTLS connections between clients and PKI portals (in cases where clients have a previous registration usable by the PKI portal).
5. It must be possible for PKI portals to produce standard PKI (e.g. CMC/CMP) certification requests, e.g. [RFC2510][RFC2797] where the portal acts as an RA.
6. Where cross-certificates exist and are usable, then it must be possible to use them (without clients necessarily having to know of their existence).

Recommendations

1. The amount of user input from clients should be minimized.
2. Clients should only have to store (or re-enter) minimal persistent information about each PKI; any such persistent information must be usable to identify client certificates to the relying party.

A.6 Privacy

1. Operators must not have to be a part of the registration process.
2. It must be possible for PKI portals to keep operators unaware of client registrations

Recommendations

1. A client should have overall control over privacy, but may contract with an operator, that the operator is allowed to be involved and, (optionally, given such a contract), that the operator may, (under the client's direction), provide client information to the PKI portal, in order to reduce the client's need to enter data
2. Client's identities (in the various PKIs) should not be easily correlated in order to protect against activity tracking

A.7 Security

1. Registration protocols must protect against replay-attacks.
2. Registration protocols must allow for generation of new key pairs in clients.
3. Clients must be able to check the validity of server and CA certificates.

A.8 Performance/Quality-of-Service

Recommendations

1. Roundtrips should be minimized which may mean that registration options are constrained depending on the bearer.

Appendix B. MIME Types (Informative)

This (non-normative) annex provides a cross reference to the parts of this specification where MIME types [MIMETYPE] are used, and where appropriate, provides the relevant WSP assigned number for each MIME type

MIME Type	Content[2]	WSP assigned number	Reference
application/x-x509-user-cert	X.509: Certificate	0x1B	6.2.3, 6.3.6 6.4.1
application/vnd.wap.wtls-user-cert	WTLSCertificate	0x19	6.2.3
application/vnd.wap.hashed-certificate	HashedTrustedCAInfo	0x38	6.1.3
application/vnd.wap.signed-certificate	SignedTrustedCAInfo	0x39	6.1.4
application/vnd.wap.rollover-certificate	RootCertificateRollover Block	0x3F	6.1.5
application/vnd.wap.cert-response	CertResponse	0x3A	6.3.6
text/plain	Unstructured error string in ASCII text	N/A	6.2.3

Table 4 - MIME Types

[2] Unless otherwise stated, all content is either the binary WTLS encoding of the named structure or the DER encoding of the named ASN.1 type

Appendix C. Device Certificate (Informative)

This informative annex describes a practice that may be used in certificate registration.

C.1 Definition of Device Certificate

A device that has a private key capability (like WIM) may be supplied to the user with initial certificates which are not personalized for the user. In that case, the user needs to obtain a certificate which binds the public key with a user identity, relevant to a Registration Authority (RA).

The RA, in order to accept a public key, may need to be aware that the corresponding private key is contained in a secure device and handled in a secure way in all circumstances. This may be required due to business related security reasons, or due to legislation regarding digital signatures.

Security of a private-public key pair includes

- it is a good quality key pair (randomness, algorithm specific checking done e.g. for RSA)
- no copies of the private key is left outside the device if the key pair was generated outside the device (this applies at least for keys used for digital signatures)
- it unfeasible to obtain the private key afterwards from the device
- PINs protecting usage of the private key, are well managed

Security of the key pair needs to be guaranteed by the manufacturer (or issuer) of the device (e.g. WIM card). If registration is done physically (i.e., the registration officer and the user meet physically, and the officer is able to see the device), it may be possible verify the authenticity of the device visually. This may not be sufficient. Also, it is not possible if the registration key takes place without a physical contact, i.e., using a remote connection.

To make it possible to securely authenticate a manufacturer of device containing the private key, a device certificate may be used. The device manufacturer, when generating a key pair, creates a certificate for the key pair.

The meaning of a device certificate is that the device manufacturer guarantees the quality of the key, the device storing the key and the related procedures. The device manufacturer may formulate a related practice statement. Security evaluation or audit procedures may be used.

Examples of issuers of device certificates are

- Operators issuing SIM-WIM cards. In this case, the issuer may indicate the actual card manufacturer.
- Smart card manufacturers

The main purpose of a device certificate is to assist registration procedure. However, in some case a certificate can be used for actual identification purposes (for some services), and for registration purposes (for other services).

It may or may not be adequate for an issuer of device certificates to run a certificate revocation service.

C.1.1 Content of a Device Certificate

A device certificate should comply to [CERTPROF]. The following tables describe an example of the actual contents.

Field	**Content**
Certificate serial number	Up to the manufacturer. Eg, part or the device serial number (like ICC ID) combined with a key number.
Issuer	Manufacturer identification. May include model etc. information. May indicate the device issuer and the original manufacturer.
Valid not before	Date of creating/storing the key and certificate.

Valid not after	End of expected maximum lifetime of the device.
Subject	E.g. the device serial number (like ICC ID) in the subject serialNumber attribute.
Public key	Public key associated with the private key in the device.
Key usage extension	Indicates operation that the device supports with this key.

Key Usage	**Supported Operation**
nonRepudiation	Digital signature with user confirmation. The device requires user verification (PIN) for every signature operation.
digitalSignature	Digital signature used for authentication (eg, for WTLS RSA handshake).
keyAgreement	Used in WTLS ECDH handshake.
keyEncipherment	Used for unwrapping a key.

C.2 Verification of a Device Certificate

The Registration authority should be able to verify the device certificate. In order to do that, the RA should have access to the manufacturer CA certificate (containing the manufacturer public key). Based on that, the RA may verify the device certificate, and thus become convinced that the key that is being registered has proper security.

In practice, the manufacturer may have a single CA certificate to certify all keys, or it may have a top CA for certification of intermediate CAs that certify actual keys. The manufacturer (top) CA may have been certified by a 3 rd party CA, which makes it easier to securely distribute the manufacturer (top) CA certificates of different manufacturers.

Manufacturer certificates are sent to a Registration Authority in a certificate registration process using normal methods like WTLS handshake or **signText**. Here, the RA may indicate which authorities (signing device certificates) it accepts.

C.3 Creation of a Manufacturer Certificate

There are different cases to create key pairs, and the associated methods to create device certificates, when using the WAP Enrollment model as described in Section 6.3.1

C.3.1 Case 1: Key Generation Outside of the Device

In this case, the key pair is generated outside the device and then saved in the device. In this case the generation procedure and saving needs to be highly secure. The advantage in this method is that the device need not support key generation, which may be demanding for a low-end device while maintaining good quality of the key. The disadvantage is that the generation procedure must be highly secure which may be administratively difficult to achieve.

The procedure of creating the key pair and device certificate is

1. create the key pair
2. save the private key in the device
3. erase all copies of the private key outside of the device
4. create the device certificate data for the public key
5. sign it with the manufacturer key
6. save the device certificate (or certificate URL) in the device

C.3.2 Case 2: Key Generation in the Device during Manufacturing

In this case, the key pair is generated inside the device as a part of the manufacturing process.

The procedure of creating the key pair and device certificate is in this case

- instruct the device to create the key pair
- retrieve the public key
- create the device certificate data
- sign it with the manufacturer key
- save the device certificate (or certiifcate URL) in the device

C.4 Use of Device Certificates in the Internet Enrollment Model

When using the Internet Enrollment model, as defined in Section 6.3.2 a device certificate is only required when a client wishes to add a keyGenAssertion structure using a PKCS#10 certificate request as outlined in [ESMPCRYPTO].

Appendix D. ECMAScript keyGen and genEnrollReq Dataflows

This appendix is informative

D.1.1 Introduction

The definition of the ECMA Script crypto functions **genEnrollReq** and **keyGen** allow for the implementation of a mobile PKI model that is more closely aligned with the model currently in wide spread use in wired clients. The movement away from the WAP based PKI model, to one based on well known and broadly supported Internet standards allow for the use of existing infrastructure and a more flexible PKI.

This informative appendix is intended to outline the high level concepts of the new ECMAScript **genEnrollReq** and **keyGen** functionality.

This document describes the OMA functionality defined to enable the generation and registration of keys into a PKI. Both key generation and key registration (also called enrollment) events may be triggered by the same entity, or they may be triggered by distinct entities; however key registration is always preceded by key generation. A key generation event may be followed by multiple key registration events, allowing for a single key to be enrolled into multiple PKIs. This approach provides a scalable solution and ease of use since a user may only need a single key, and in some cases a single PIN, while having the advantage of enrolling into multiple PKIs and transacting with multiple merchants. To allow for a standard and easily understood mechanism for generating and registering keys, two new ECMAScript functions, **keyGen** and **genEnrollReq**, have been defined to trigger key generation and key registration respectively.

Unlike the WAP model where key pairs were provisioned in a factory and distributed manually, the new **keyGen** function allows for new key pairs to be generated in the field. In some scenarios the ability to include assurances that the key was generated in a secure manner is required. These assurances may specify that a key was generated per a specific policy and on some approved Security Element (SE), such as a WIM.

In order to accommodate this requirement, the ability to include an assurance signature or an assurance message authentication code (MAC) may be included in the registration request via the **genEnrollReq** function. The information that is signed or MACed includes the public key and may include additional information indicating the kind of assertion that is being made. This assurance information is then verified by the PKI to prove that the key was generated in a secure manner. This mechanism currently can indicate not only if the key was generated on-board, but may also be used to indicate if a key was injected into a SE. In addition the registration messages generated by the **genEnrollReq** function are based on industry standard PKCS#10 message formats, ensuring tight integration with the existing CA and PKI infrastructure.

The ability to invoke the key generation and key registration events through ECMAScript commands in conjunction with the ability to obtain an assurance of the fact that the key was generated on-board allows the link between key generation and key registration to be re-established.

To support the requirement where explicit authorization is needed to perform a key generation or key registration, both the **keyGen** and **genEnrollReq** functions allow for authorization data to be included. Authorization to use the **keyGen** function allows the business case where an SE owner (such as an operator) has the ability to charge for the use of empty key slots on its cards to third party vendors. Simillary, authorization to use **genEnrollReq** a particular key into a PKI allows the key owner to contol who can reuse a particular key in a separate PKI.

D.1.2 Participants

When considering on-board key generation and key registration it is possible to identify three participants. These are:

The PKI

The PKI is responsible for issuing certificates and can initiate key generation and request enrollment information via the use of the ECMAScript **genEnrollReq** and **keyGen** commands. Key generation may occur only once, while a key may be enrolled into multiple PKIs.

The separation of key generation and key registration makes it possible to support a model where it is possible to trigger key generation through another (possibly proprietary) mechanism and still have a standard way to enroll the key in a PKI.

Mobile entity

In this model the mobile entity is responsible for interpreting and acting upon the new ECMAScript **genEnrollReq** and key gen commands it will receive from a PKI. Mainly this involves the proper generation, signing and formatting of the response that is returned to the ECMAScript command. In the case where an SE is being used, it also interacts with the SE for cryptographic operations.

In some cases, the ME is also responsible for interacting with the SE for administrative functionality such as the setting and verifying relevant PIN's. When this is the case, the ME also is responsible for interacting with the user when necessary. An example of this would be the selecting of a new PIN at key generation time.

The interpretation and implementation of the ECMAScript commands is part of the browser implementation on the mobile entity.

Security Element

The SE will perform operations triggered by the mobile entity including the generation of new keys in addition to standard cryptographic functionality such as signature generation used for POP. SE's that support the key assurance concept will be responsible for the generation of these assurances values. Other SE functionality would include the management of user PINs and any relevant flags.

The following picture summarizes the three entities and their roles in the on-board key generation and registration process.

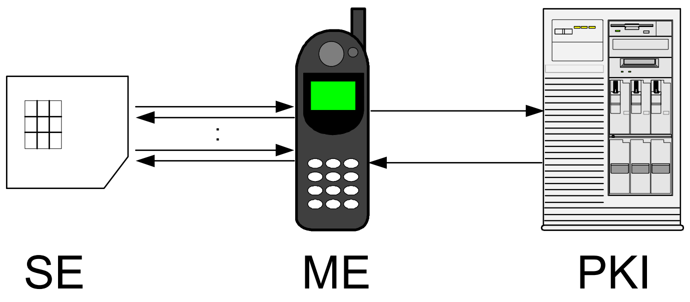

Figure 4 – Dataflow Participants

D.1.3 Example Scenarios

This section is informative.

Scenario 1

This scenario describes the generation of a key via the **keyGen** function. There is no requirement that the key generation command be authenticated. Once generated this key can be enrolled into a PKI using the enrol function at a later time. The device is responsible for sending the appropriate key generation command to the SE.

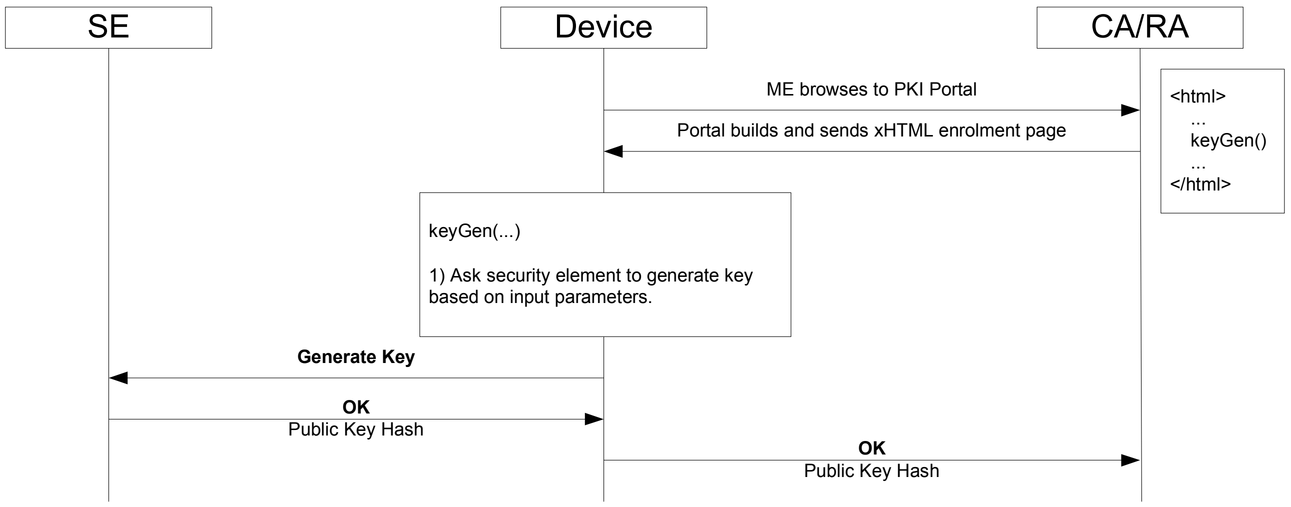

Figure 5 - keyGen Dataflow

Scenario 2

This scenario describes the enrollment of a previously generated key into a PKI. The generation of the key assurance structure is not included for simplicity. (See Scenario 5 for key assurance details) Like Scenario 1, there is no authorization needed to enroll. Once the device finds an appropriate key based on the **genEnrollReq** function input parameters, it retrieves the public key and builds the PKCS#10 CertificationRequestInfo structure. Once built the device then sends the hash to be signed by the appropriate private key managed by the SE. The device then creates the PKCS#10 Certification request structure based on the PKCS#1 signature returned by the SE and returns the Base-64 encoded [BASE64] structure to the function. The PKI is the responsible for verifying the signature, authenticating the identity and issuing the certificate.

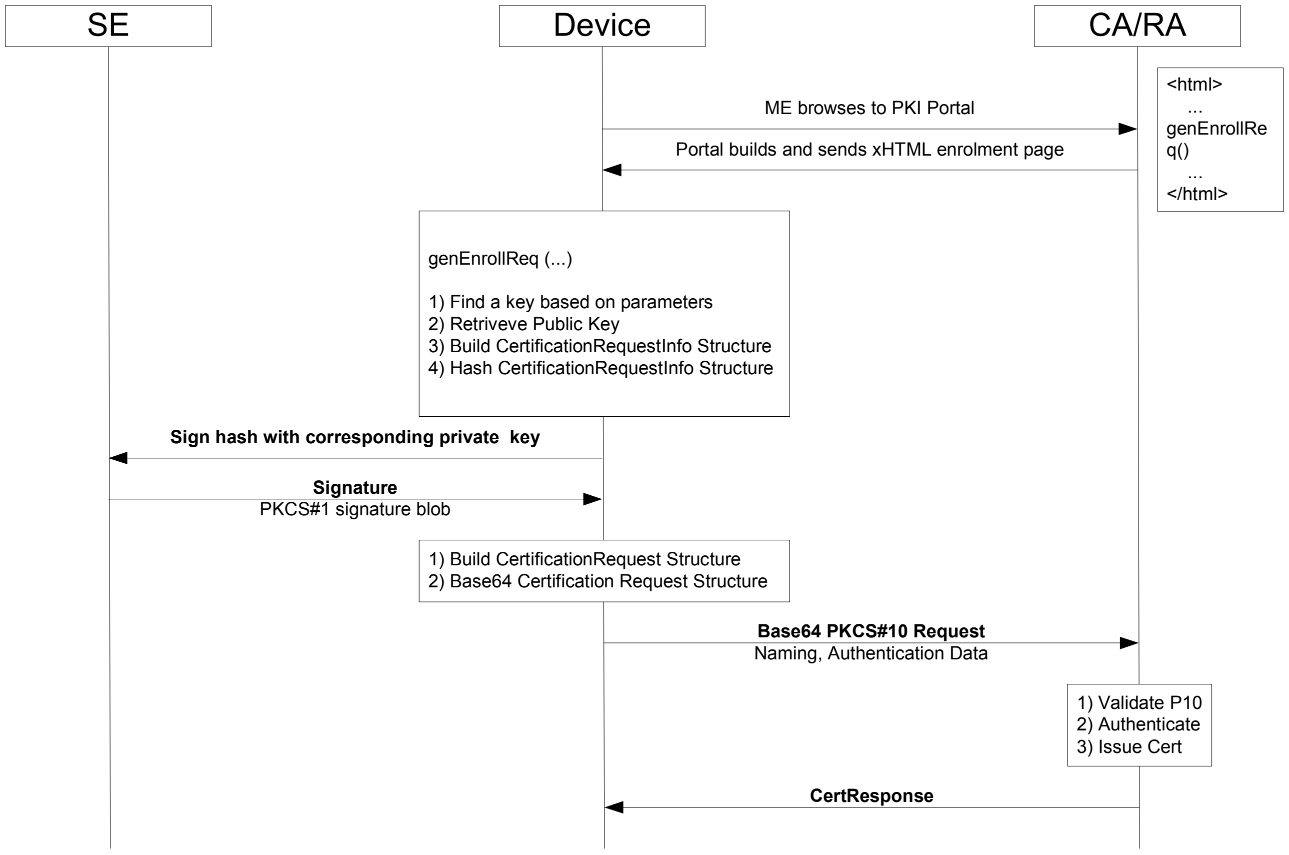
SE
Device
CA/RA
<html> ... genEnrollReq() ... </html>
ME browses to PKI Portal
Portal builds and sends xHTML enrolment page
genEnrollReq (...)
1) Find a key based on parameters
2) Retriveve Public Key
3) Build CertificationRequestInfo Structure
4) Hash CertificationRequestInfo Structure
Sign hash with corresponding private key
Signature
PKCS#1 signature blob
1) Build CertificationRequest Structure
2) Base64 Certification Request Structure
Base64 PKCS#10 Request
Naming, Authentication Data
1) Validate P10
2) Authenticate
3) Issue Cert
CertResponse

Figure 6 - genEnrollReq dataflow

Scenario 3

This scenario depicts a call to both the **keyGen** and **genEnrollReq** functions in a single xHTML page.

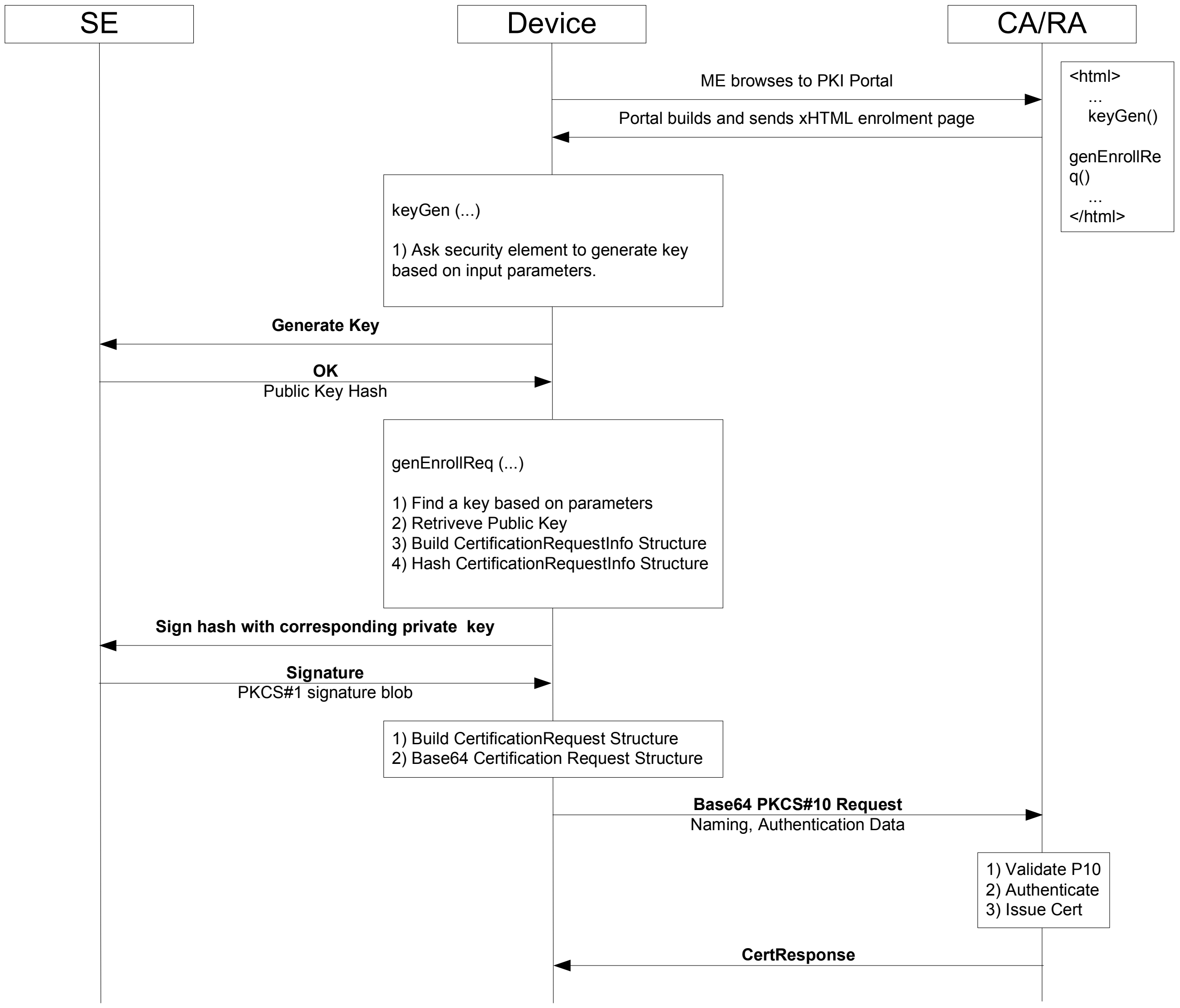
SE
Device
CA/RA
ME browses to PKI Portal
Portal builds and sends xHTML enrolment page
<html> ... keyGen() genEnrollReq() ... </html>
keyGen (...)
1) Ask security element to generate key based on input parameters.
Generate Key
OK
Public Key Hash
genEnrollReq (...)
1) Find a key based on parameters
2) Retriveve Public Key
3) Build CertificationRequestInfo Structure
4) Hash CertificationRequestInfo Structure
Sign hash with corresponding private key
Signature
PKCS#1 signature blob
1) Build CertificationRequest Structure
2) Base64 Certification Request Structure
Base64 PKCS#10 Request
Naming, Authentication Data
1) Validate P10
2) Authenticate
3) Issue Cert
CertResponse

Figure 7 - keyGen and genEnrollReq dataflow

Scenario 4

This scenario depicts a key generation and enrollment data flow as described in Scenario 3, however it assumes that the SE being used is a WIM and thus details the WIM specific commands used by the device. This scenario does not depict the case where explicit authorization is needed for either key generation or enrollment.

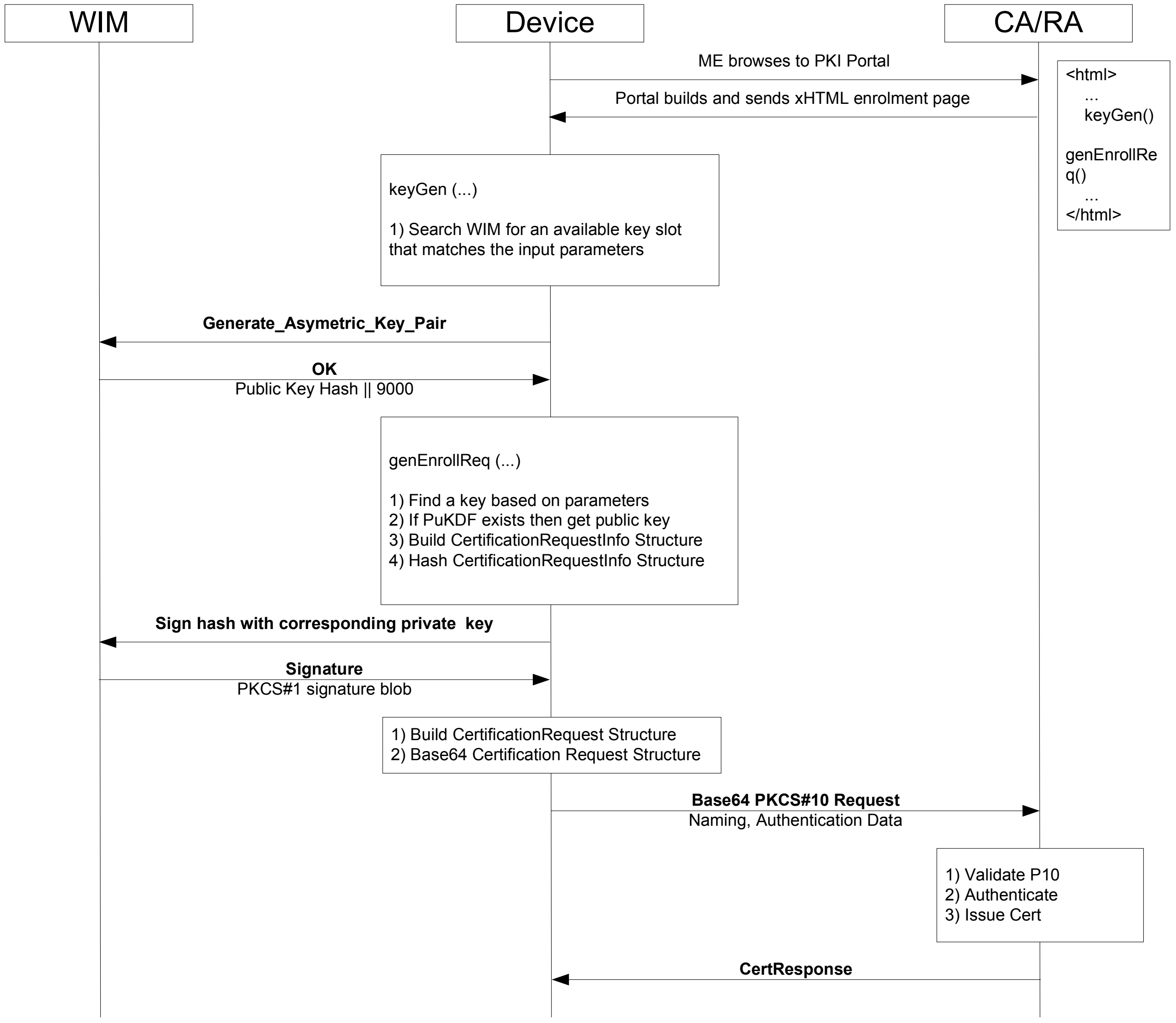
WIM
Device
CA/RA
ME browses to PKI Portal
Portal builds and sends xHTML enrolment page
<html> ... keyGen() genEnrollReq() ... </html>
keyGen (...)
1) Search WIM for an available key slot that matches the input parameters
Generate_Asymetric_Key_Pair
OK
Public Key Hash || 9000
genEnrollReq (...)
1) Find a key based on parameters
2) If PuKDF exists then get public key
3) Build CertificationRequestInfo Structure
4) Hash CertificationRequestInfo Structure
Sign hash with corresponding private key
Signature
PKCS#1 signature blob
1) Build CertificationRequest Structure
2) Base64 Certification Request Structure
Base64 PKCS#10 Request
Naming, Authentication Data
1) Validate P10
2) Authenticate
3) Issue Cert
CertResponse

Figure 8 - keyGen and genEnrollReq dataflow using WIM

Scenario 5

This scenario extends Secnario 4 above with the addition of the key assurance attribute in the PKCS#10 request. In this scenario the device must first request the key assurance information from the WIM before it creates the PKCS#10 message and signature.

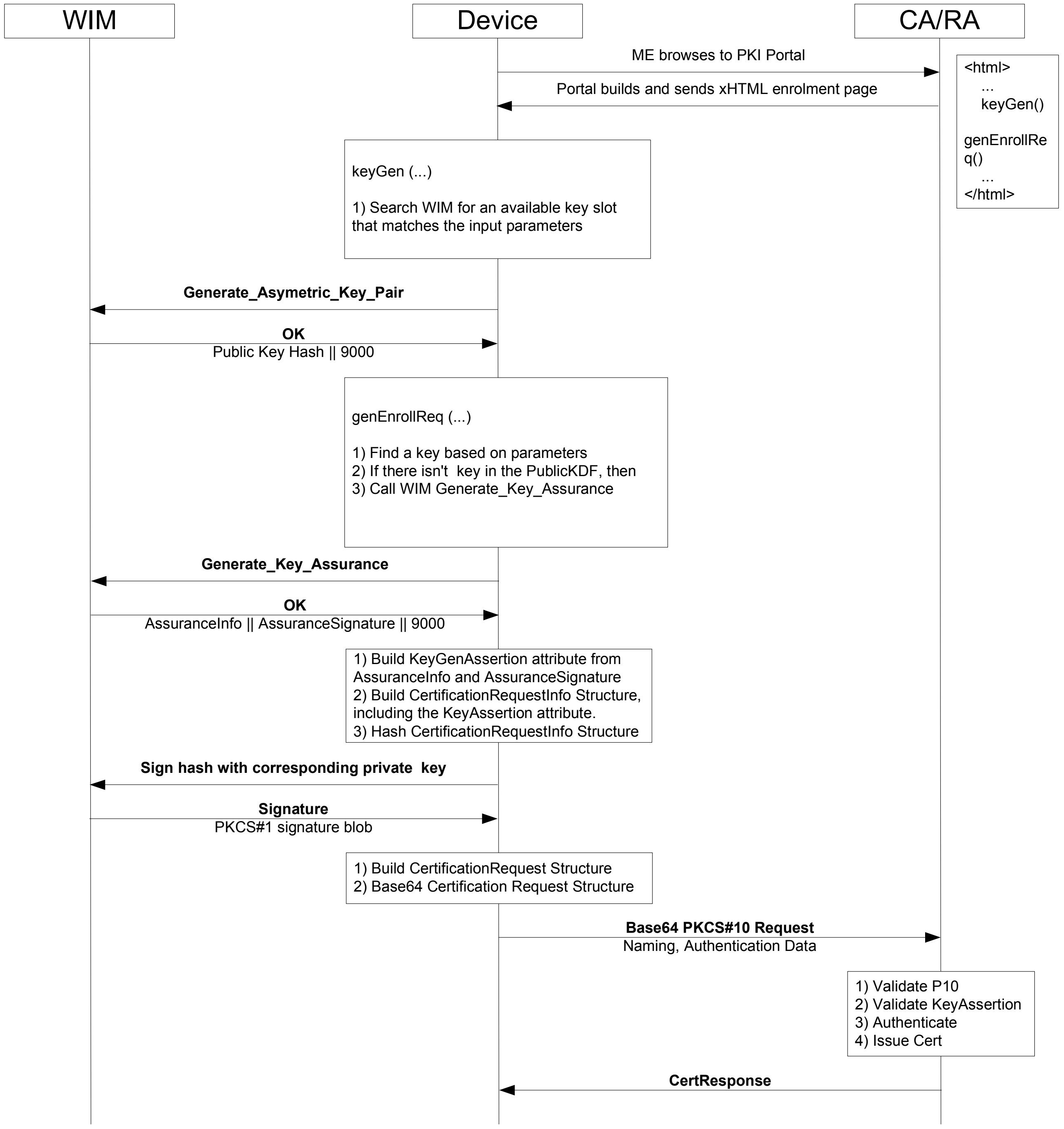
WIM
Device
CA/RA
ME browses to PKI Portal
Portal builds and sends xHTML enrolment page
<html>
...
keyGen()
genEnrollReq()
...
</html>
keyGen (...)
1) Search WIM for an available key slot that matches the input parameters
Generate_Asymetric_Key_Pair
OK
Public Key Hash || 9000
genEnrollReq (...)
1) Find a key based on parameters
2) If there isn't key in the PublicKDF, then
3) Call WIM Generate_Key_Assurance
Generate_Key_Assurance
OK
AssuranceInfo || AssuranceSignature || 9000
1) Build KeyGenAssertion attribute from AssuranceInfo and AssuranceSignature
2) Build CertificationRequestInfo Structure, including the KeyAssertion attribute.
3) Hash CertificationRequestInfo Structure
Sign hash with corresponding private key
Signature
PKCS#1 signature blob
1) Build CertificationRequest Structure
2) Base64 Certification Request Structure
Base64 PKCS#10 Request
Naming, Authentication Data
1) Validate P10
2) Validate KeyAssertion
3) Authenticate
4) Issue Cert
CertResponse

Figure 9 – Example Enrollment dataflow using key assurance

Scenario 6

Scenario 6 is the most complex scenario. It shows the data flow in a case where explicit authorization is necessary to both generate a key and enroll for a certificate. In addition, the key assurance information is included.

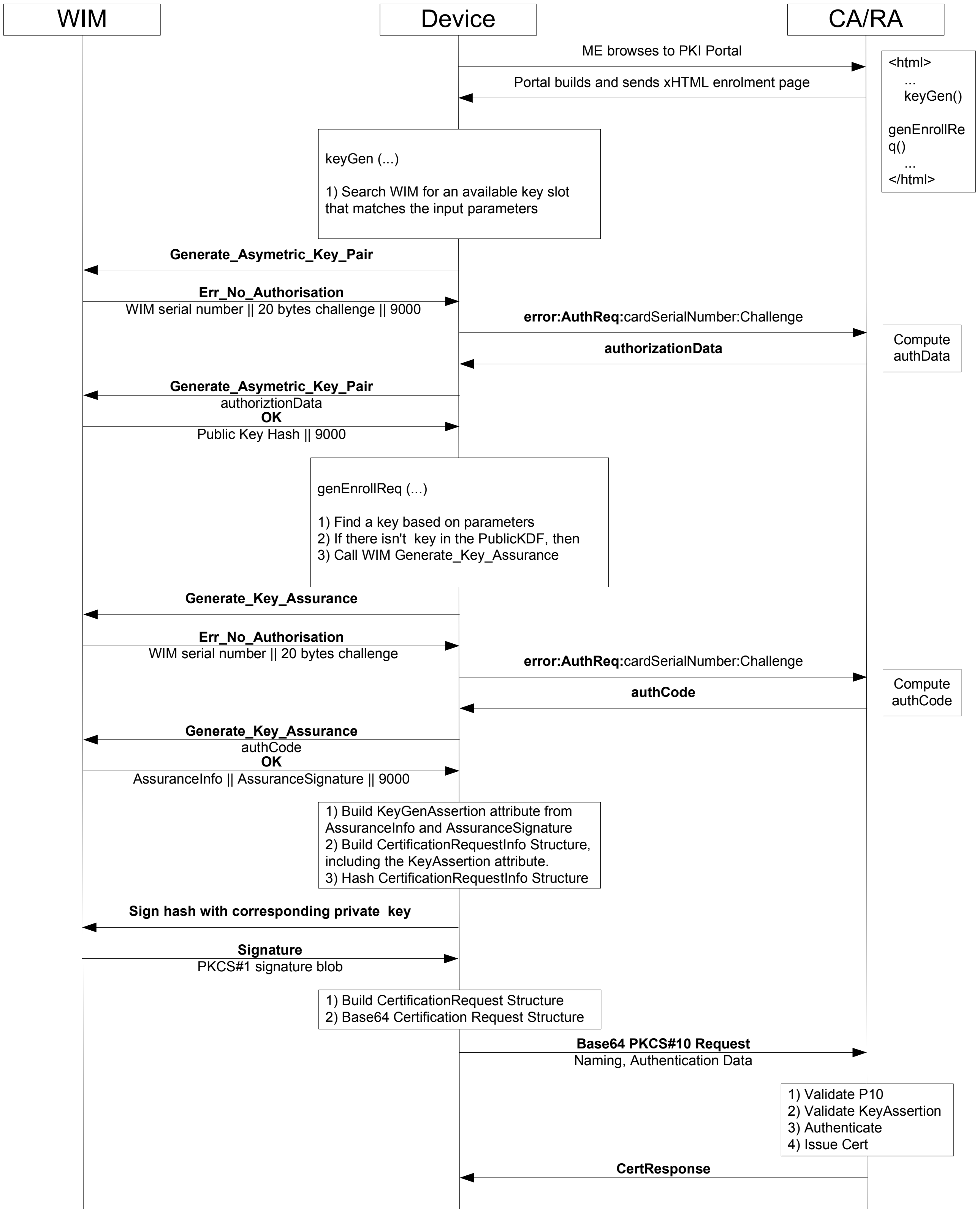
WIM
Device
CA/RA
ME browses to PKI Portal
Portal builds and sends xHTML enrolment page
<html> ... keyGen() genEnrollReq() ... </html>
keyGen (...)
1) Search WIM for an available key slot that matches the input parameters
Generate_Asymetric_Key_Pair
Err_No_Authorisation
WIM serial number || 20 bytes challenge || 9000
error:AuthReq:cardSerialNumber:Challenge
Compute authData
authorizationData
Generate_Asymetric_Key_Pair
authoriztionData
OK
Public Key Hash || 9000
genEnrollReq (...)
1) Find a key based on parameters
2) If there isn't key in the PublicKDF, then
3) Call WIM Generate_Key_Assurance
Generate_Key_Assurance
Err_No_Authorisation
WIM serial number || 20 bytes challenge
error:AuthReq:cardSerialNumber:Challenge
Compute authCode
authCode
Generate_Key_Assurance
authCode
OK
AssuranceInfo || AssuranceSignature || 9000
1) Build KeyGenAssertion attribute from AssuranceInfo and AssuranceSignature
2) Build CertificationRequestInfo Structure, including the KeyAssertion attribute.
3) Hash CertificationRequestInfo Structure
Sign hash with corresponding private key
Signature
PKCS#1 signature blob
1) Build CertificationRequest Structure
2) Base64 Certification Request Structure
Base64 PKCS#10 Request
Naming, Authentication Data
1) Validate P10
2) Validate KeyAssertion
3) Authenticate
4) Issue Cert
CertResponse

Figure 10 - keyGen and enrollment requiring authentication

Appendix E. WAP Specification Change History

The following table represents the change history present in the in the old WAP document. It is included here for completeness and is intended for informational and historical purposes only.

Type of Change	Date	Section	Description
Class 0	03-Mar-2000		The initial version of this document.
Class 2	09-Aug-2000	7.3.5	Delivery of certificates syntax
Class 2	09-Aug-2000	7.3.5, 7.4.2	Problem with return value for LDAP Scheme
Class 2	09-Aug-2000	6, 7.1	Removing WTAI specification from WPKI
Class 3	09-Aug-2000	6, 8.1.3	Finding CA certificates
Class 3	09-Aug-2000	6, 8.1.3	Fix to above CR
Class 3	09-Aug-2000	6.3, 7.3	Architecture Consistency Review Rec#1
Class 3	09-Aug-2000	3.2, 7.3, 8.2.4, A.5	Architecture Consistency Review Rec#2
Class 3	09-Aug-2000	7.1.4	Clarify TrustedCACert in TBSTrustedCAInfo
Class 3	09-Aug-2000	Annex C	Manufacturer certificate
Class 3	26-Oct-2000	8	Clarify TrustedCACert in TBSTrustedCAInfo
Class 3	26-Oct-2000	6, 8.1.3	Manufacturer certificate
Class 3	24-Apr-2001	7.4.1	Clarification of HTTP Certificate URLs

Appendix F. Change History (Informative)

F.1 Approved Version 1.1 History

Reference	Date	Description
WAP-217-WPKI-20000303-d	03 Mar 2000	First WAP WPKI spec
WAP-217-WPKI-20010424-A	24 Apr 2001	Added the following three SIN's WAP-217_100-WPKI-20000809-d WAP-217_101-WPKI-20001026-d WAP-217_103-WPKI-20010424-d
OMA-WAP-WPKI-V1_1-20110301-A	01 Mar 2011	Status changed to Approved by TP: OMA-TP-2011-0068-INP_OBKG_V1_0_ERP_for_Final_Approval