Common PKI Specifications for Interoperable Applications

from T7

Corrigenda

to

Common PKI Specification 2.0

as of 20 January 2009

Version 1.2.1 – 10 June 2014

Contact Information

The up-to-date version of the Common PKI specification can be downloaded from www.common-pki.org or from www.common-pki.de

Please send comments and questions to common-pki@common-pki.org.

Document History

VERSION DATE	CHANGES
1.0 30 November 2010	Initial Corrigenda to Common PKI Specification v2.0 The following issues are addressed: • Profile the *ESSCertIDv2* field within the *signingCertificateV2* CMS Attribute as introduced by RFC 5035 in accordance with the established Common PKI profile of the predecessor *signingCertificate* and *ESSCertID*. • Complete the update from *signingCertificate* / *ESSCertID to signingCertificateV2* / *ESSCertIDv2* in requirements on file signatures. • Mark the possibility of providing an explicit reference point in time as input to the certificate validation algorithm as Common PKI profiling, as the RFC 5280 validation algorithm always checks for its time of execution. • Correct the W3C URI for SHA-384. • Reflect the Common PKI 2.0 hash algorithm requirements according to Part 6 in the corresponding requirements of Part 8. Correct Algorithm URIs and schema references.
1.1 5 November 2012	The following issues are addressed: • Part 9: Added a clarification about the preference for OCSP vs. CRLs in the context of qualified CA certificates. • Part 5: Added a note about caching of OCSP responses for responder certificates.
1.2 28 April 2014	The following issues are addressed: • Part 1: OAEP Padding for encryption added • Part 1: UTF-8 character subset extended to ISO 8859-15 • Part 6: OAEP algorithm identifiers • Part 8: OAEP usage with more hash algorithms added • References for P8.5.2 added • References in corrigenda to P6.T1#2a corrected
1.2.1 10.06.2014	Some typos corrected

Table of Contents

1 Preface

This document contains a list of corrigenda to correct and clarify the Common PKI Specification v2.0.

The corrigenda become immediately effective with the publication of this document, i.e. the effectual text of the Common PKI specification will be that of the Common PKI Specification v2.0 as of January 20th, 2009 *with the changes specified in this document applied*.

Changes and additions are highlighted by background colour, deletions by ~~background colour and crossed out~~.

2 Corrigenda to Part 1: Certificate and CRL Profiles

1) in P1.T10 add:

	RFC5751 PRIVATE EXTENSIONS							**RFC5751**		
17	`smimeCapabilities / RSAES-OAEP`	`{1 2 840 113549 1 9 15}`	In case a SSCD solely supports RSAES-OAEP as padding mechanism and cannot be addressed using PKCS#1-v1.5 padding this information will be stored in the smimeCapabilities extension. Usage of the smimeCapability in certificates is defined in [RFC4262]	-	++	++	+	2.5.2	T26a	

2) in P1 change in T6:

[2]	**Common PKI Profile:** Following [MTTv2], Common PKI RECOMMENDS using a subset of the UTF8 character set, including only the set union of ANSI/ISO 8859-1 characters (Unicode Latin-1 page) and ANSI/ISO 8859-15. Since Windows and UNIX systems use the ISO 8859-1 codes for displaying characters, this restriction makes software implementation easier: strings can be displayed on those platforms irrespective of locale settings. On the other hand many applications need to be able to correctly display and store characters beyond Latin-1, e.g. in names of eastern European origin, which has lead to additional incorporation of ANSI/ISO 8859-15. Hence, generating components SHOULD NOT include characters ~~of code pages other than Latin-1~~ other than those included in ANSI/ISO 8859-1 or ANSI/ISO 8859-15. Processing components MUST be able to correctly display Latin-1 characters and MAY be able to display other UTF8 characters too. Processing components MUST tolerate (i.e. MUST be able to decode) all UTF8 characters, even if they are unable to display them correctly. In this latter case, non-Latin-1 characters SHOULD be replaced by some well-defined dummy character on the display, e.g. '□'

3) in P1 add T26a

Table 26a: smimeCapabilities / RSAES-OAEP

#	ASN.1 Definition	Semantics	Support		References		Notes
			Gen	Proc	RFC5751 /3560	Co. PKI	
1	`smimeCapabilities OBJECT IDENTIFIER ::= {1 2 840 113549 1 9 15}`	smimeCapabilities extension	++	+	RFC5751		[1]
2	`SMIMECapability ::= SEQUENCE {`	Storing place for the S/MIME capability	++	+	RFC5751		
3	`id-RSAES-OAEP OBJECT IDENTIFIER ::= { RSAES-OAEP }`	OID for RSAES-OAEP	++	+	RFC3560	P6.T5#3	

4	`RSAES-OAEP-params ::= SEQUENCE {`
	`hashFunc [0] AlgorithmIdentifier`
	`DEFAULT sha1Identifier,`
	`maskGenFunc [1] AlgorithmIdentifier`
	`DEFAULT mgf1SHA1Identifier,`
	`pSourceFunc [2] AlgorithmIdentifier`
	`DEFAULT pSpecifiedEmptyIdentifier`
		Indicates that RSAES-OAEP MUST be used with this certificate on the basis of the parameter set provided	++	+	3.	P6.T10 #6-9	[2]

5	`SMIMECapabilities ::= SEQUENCE OF SMIMECapability`	In this case filled in by #3-4	++	+	RFC5751		
[1]	[RFC5751]: There are cases where a SSCD is used for encryption/decryption and can only be addressed using RSAES-OAEP padding. When PKCS#1-v1.5 padding is used the SSCD may be destroyed when too many errors are produced in a row. Note that smimeCapabilities although being defined for S/MIME objects can also be used as certificate extensions based on RFC4262. **Common PKI Profile:** In this case a SMIMECapability extension MUST be inserted into the certificate with one of the values defined in RFC3560. Processing entities SHOULD be able to process this extension. Processing and generating entities MAY use other smimeCapabilities. Processing of this extension SHOULD be performed also in cases where no CMS or S/MIME objects are in place, e.g. when using XML encryption.						
[2]	[RFC3560]: **Common PKI Profile:** Usage of RSAES-OAEP MUST be indicated by generating entities by an entry of id-RSAES-OAEP plus parameters. Processing entities SHOULD be able to process this information.						

Add to References Part 1:

[RFC5751] Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, January 2010

[RFC3560] Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS), July 2003

3 Corrigenda to Part 2: PKI Management

Currently no corrigenda.

4 Corrigenda to Part 3: CMS based Message Formats

1) In P3.T5#9 add

1	*signingCertificateV2 id-aa-signingCertificateV2* {1 2 840 113549 1 9 16 2 47}	Sequence of certificate identifiers starting with the certificate of the signer	[RFC 5035]	3	+-		+-	+-	The *issuerSerial* field of the *ESSCertIDv2* within *signingCertificateV2* MUST not be empty.	[5]

2) Change P3.4.1 to

[~~RFC~~ RFC3852] allows including attribute certificates in the certificate list. For all attribute certificates, which are intended by the signer to be used for the signature, a reference MUST be included in the *signedAttributes* of the corresponding *SignerInfo* using the *SigningCertificateV2* attribute. The *issuerSerial* field of the *ESSCertIDv2* within *SigningCertificateV2* MUST not be empty. This information~~s~~ is intended for the recipient, so that all certificates required for the verification of the file signature can easily be obtained. Note that certificates provided in the ‘certificates’ field are not part of the signed content and are thus not protected against substitution attacks.

5 Corrigenda to Part 4: Operational Protocols

Currently no corrigenda.

6 Corrigenda to Part 5: Certificate Path Validation

1) Change P5.T2#1 to

1	`bool ValidateCertificate( CertInfo in tbvCert, CertInfoList in tbvCerts, KeyPurpose in intendedKeyUsage, Time in refTime, PolicyConstraints in initialPolicySet, CertInfoList inout trustedCerts, CrlInfoList inout trustedCrls ) {`	This is the main entry point of the certificate path validation algorithm. The ‘to be verified’ target certificate or attribute certificate is passed in *tbvCert*. *tbvCerts* may contain zero or more certificates – other than the ‘to be verified’ certificate – of a path to some root certificate. Most commonly, *tbvCerts* contains certificates trusted by the signing/decrypting party, but not necessarily trusted by the relying party. The required usage of the certified key is indicated in *intendedKeyUsage*. In case of an attribute certificate, this parameter is ignored by the procedure. ~~The point in time, to which status information should be obtained, is passed in *refTime*. It may be the current time (typical for mail authentication, encryption) or some point in the past (typical for non-repudiation service).~~ *pathConstraints* conveys input parameters from the relying application to the basic path validation algorithm (BPVA). These parameters contain policy constraints or naming constraints that have to be verified during path validation. *trustedCerts* MUST contain at least one trusted self-signed root certificate and may contain further CA and EE certificates, all of which having a path to one of those trusted root certificates. These certificates are typically stored on the local system to accelerate the validation procedure. *trustedCerts* may further contain cross-certificates (issued by a trusted CA to some other CA), each having a valid path to one of those root certificates. *trustedCrls* may contain complete CRLs that have previously been downloaded, successfully verified and stored in the local database. This storage allows a reuse of complete CRLs in later validations without needing to access the directory service. *trustedCrls* may furthermore contain complete CRLs that are locally maintained, e.g. by regularly downloading delta-CRLs from an LDAP-Server or by obtaining the list by some out-of-band mechanism (e.g. unsigned CRLs of root certificates). This function returns *true* if the certificate has been successfully verified, including mathematical verification, constraint and status checking; respectively *false* if mathematical check failed, some constraint is not met, a relevant certificate cannot be obtained or has been revoked, status information cannot been obtained or no certification path could have been built to any of the trusted root certificates. *trustedCerts* will be updated with the certificates of a successfully validated path to		

		allow local storage and reuse of validated certificates and corresponding status information. *trustedCrls* will be similarly updated with verified CRLs. **Common PKI Profile:** The point in time, to which status information should be obtained, is passed in *refTime*. It may e. g. be the current time (typical for mail authentication, encryption) or some point in the past (typical for non-repudiation service).		

2) In P5.T7#3 add

3	`CertInfo respCert = response.GetResponderCert();`
	`if( VerifySignature( response.GetToBeSignedData(),`
	`                     responderCert.GetPublicKeyInfo())==false)`
	`  return false;`
	`if( ValidateCertificate( respCert,`
	`                         response.RetrieveCerts(),`
	`                         response.GetProducedAtTime();`
	`                         ocspSigning,`
	`                         pathConstraints,`
	`                         trustedCerts,`
	`                         trustedCrls )==false )`
	`  return false;`
		In case of a definitive response (*responseStatus*=’*successful*’-), the responder certificate is retrieved from the response and the signature over the response is verified. Finally, the responder’s certificate is validated by means of a recursive call to the certificate path validation function.
		Common PKI Profile: Note that Common PKI conforming responses always contain the responder’s signing certificate (P4.T8.[3]). The signing certificate can be identified among the other certificates returned in *certs* (P4.T8.#7) using the information in the *responderID* field (P4.T8.#10).
		Note that if there are several distinct OCSP responder certificates in use by a single OCSP service, this may lead to multiple recursive OCSP requests. In that case it may be advisable for an OCSP client to cache previous OCSP responses on the revocation status of responder certificates for a suitable time interval in order to speed up successive validation processes and reduce network and server load.	

7 Corrigenda to Part 6: Cryptographic Algorithms

1) In P6.T1#2a replace

2a	SHA-384	one-way hash function	[RFC 4055] [XML_ENC] [FIPS 180-2]		n. a.	+	+	OID: 2.16.840.1.101.3.4.2.2 http://www.w3.org/2001/04/~~xmlenc~~#sha384	[4]

by

2a	SHA-384	one-way hash function	[RFC 3560] [XML_DSIG] [FIPS 180-4]		n. a.	+	+	OID: 2.16.840.1.101.3.4.2.2 http://www.w3.org/2001/04/xmldsig-more#sha384	[4]

2) in P6 add T9

Table 9: RSAES-OAEP Parameter Values

Cryptographic Algorithms			**References**			**Common PKI SUPPORT**			**Notes**
#	**Name**	**Semantics**	**Document**	**Chapter**	**Status**	**Gen**	**Proc**	**Values**	
1	id-mgf1	Mask generation function	[RFC3560]	3	+-	++	+	OID: 1.2.840.113549.1.1.8	[1]
2	id-pSpecified	Source function	[RFC3560]	3	+-	++	+	OID: 1.2.840.113549.1.1.9	[1]

[1] The mask generation and source functions are originally defined in [PKCS#1] but is referenced here out of [RFC3560] because this is the reference for P1.T26a where RSAES-OAEP usage is defined within this document.

Table 10: RSAES-OAEP parameters

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3560	CO. PKI	
1	`mgf1SHA1Identifier AlgorithmIdentifier ::= { id-mgf1, sha1Identifier }`	Identifier for SHA1 usage as mask function	++	+	3.	T9#1 T1#1	
2	`mgf1SHA256Identifier AlgorithmIdentifier ::= { id-mgf1, sha256Identifier }`	Identifier for SHA256 usage as mask function	++	+	3.	T9#1 T1#2	
3	`mgf1SHA384Identifier AlgorithmIdentifier ::= { id-mgf1, sha384Identifier }`	Identifier for SHA384 usage as mask function	++	+	3.	T9#1 T1#3	
4	`mgf1SHA512Identifier AlgorithmIdentifier ::= { id-mgf1, sha512Identifier }`	Identifier for SHA512 usage as mask function	++	+	3.	T9#1 T1#4	

5	`pSpecifiedEmptyIdentifier AlgorithmIdentifier ::= { id-pSpecified, nullOctetString }`
	
	`nullOctetString OCTET STRING (SIZE (0)) ::= { ''H }`
		Identifier for default parameter p (empty string)	++	+	3.	T9#2	

6	`rSAES-OAEP-Default-Identifier AlgorithmIdentifier ::= { id-RSAES-OAEP, { sha1Identifier, mgf1SHA1Identifier, pSpecifiedEmptyIdentifier } }`	Identifier for default (based on SHA1)	++	+	3.		
7	`rSAES-OAEP-SHA256-Identifier AlgorithmIdentifier ::= { id-RSAES-OAEP, { sha256Identifier, mgf1SHA256Identifier, pSpecifiedEmptyIdentifier } }`		++	+	3.		
8	`rSAES-OAEP-SHA384-Identifier AlgorithmIdentifier ::= { id-RSAES-OAEP, { sha384Identifier, mgf1SHA384Identifier, pSpecifiedEmptyIdentifier } }`		++	+	3.		
9	`rSAES-OAEP-SHA512-Identifier AlgorithmIdentifier ::= { id-RSAES-OAEP, { sha512Identifier, mgf1SHA512Identifier, pSpecifiedEmptyIdentifier } }`		++	+	3.		

	[RFC3560]: These identifiers for RSAES-OAEP parameter encoding MUST be used for RSAES-OAEP parameter encoding as defined in P1.T26a#4

Add to References Part 6:

[RFC5751] Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification, January 2010

[RFC3560] Use of the RSAES-OAEP Key Transport Algorithm in Cryptographic Message Syntax (CMS), July 2003

8 Corrigenda to Part 7: Signature API

Currently no corrigenda.

9 Corrigenda to Part 8: XML based Message Formats

3) Change P8.T3#4 to

4	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />	++	++	4.3.2	[2]
		<xsd:enumeration value="http://www.w3.org/2001/04/~~xmlenc#ripemd160~~xmldsig-more/rsa-ripemd160" />	-	-+		[2] [3]
		<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />	++	++		[2]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />	+-	+		[2]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmldsig-more#rsa-sha384" />	+-	+		[2]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmldsig-more#rsa-sha512" />	+-	+		[2]

4) Change P8.T3.[2] to

[2]	Delimits the possible algorithms to DSA-SHA1, RSA-SHA1, ~~and~~ RSA-RIPEMD160, RSA-SHA256, RSA-SHA384 and RSA-SHA512.

5) Change P8.T5#3 to

3 4	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#sha1" />	++	++	4.3.3.5	[1]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#ripemd160" />	-	-+		[1] [2]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#sha256" />	+	+		[1]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmldsig-more#sha384" />	+	+		[1]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#sha512" />	+	+		[1]

6) Change P8.T5.[1] to

[1]	Delimits the possible algorithms to SHA-1, ~~and~~ RIPEMD160, SHA-256, SHA-384 and SHA-512.

7) In P8.T5#1 add

```
<xsd:restriction base="xsd:anyURI">
    <xsd:enumeration
        value="http://www.w3.org/2000/09/xmldsig#sha1" />
    <xsd:enumeration
        value="http://www.w3.org/2001/04/xmlenc#sha256" />
    <xsd:enumeration
        value="http://www.w3.org/2001/04/xmldsig-more#sha384" />
    <xsd:enumeration
```

```
                value="http://www.w3.org/2001/04/xmlenc#sha512" />
        <xsd:enumeration
                value="http://www.w3.org/2001/04/xmlenc#ripemd160" />
</xsd:restriction>
```

8) In P8.T5#1 change

```
<xsd:import namespace="http://www.w3.org/2001/04/xmlenc#"
        schemaLocation="osciene.xsd" />
```

to

```
<xsd:import namespace="http://www.w3.org/2001/04/xmlenc#"
        schemaLocation="xenc-schema.xsd" />
```

9) In P8.T5#2 add

```
<xsd:restriction base="xsd:anyURI">
        ...
        <xsd:enumeration
                value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" />
        <xsd:enumeration
                value="http://www.w3.org/2009/xmlenc11#rsa-oaep" />
        <xsd:enumeration
                value=" http://www.w3.org/2009/xmlenc11#mgf1sha224" />
        <xsd:enumeration
                value=" http://www.w3.org/2009/xmlenc11#mgf1sha256" />
        <xsd:enumeration
                value=" http://www.w3.org/2009/xmlenc11#mgf1sha384" />
        <xsd:enumeration
                value=" http://www.w3.org/2009/xmlenc11#mgf1sha512" />
```

10) Add after P8.T11#11

4	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#rsa-oaep" />	+-	+	5.5.2	[2]
		<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#mgf1sha1" />				
		<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#mgf1sha224" />				
		<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#mgf1sha256" />				
		<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#mgf1sha384" />				
		<xsd:enumeration value="http://www.w3.org/2009/xmlenc11#mgf1sha512" />				

10 Corrigenda to Part 9: SigG-Profile

1) In section 6 add

SigG-conforming applications that support revocation checking by CRL as alternative to OCSP MUST be able to process indirect CRLs.

In the context of qualified certificates OCSP SHOULD be used instead of CRLs, as it is not possible to reliably handle all possible states of the revocation status in a certificate chain by using indirect CRLs, and as a consequence, a signature validation might unnecessarily produce the result "undetermined".

Note: There are scenarios (e.g. in medical emergencies) where online connectivity is not in place. In such situations the usage of CRLs is preferable to not performing revocation checks at all.

In the context of SigG the DName of a CRL-issuer registered in the *CRLDistributionPoints* extension of a certificate changes over time. In this case the CRL is signed by a different CRL-issuer than the one registered in the *CRLDistributionPoints* extension at the time of certification. If a client conforming to this profile (and optional a non-SigG client) downloads the CRL from the CDP URI and encounters this situation, it SHOULD check if the (valid, see also P1.T12.[1]) CRL-issuer, which signed the CRL, can be validated to the same root CA as the certificate being checked. If this is true, then the CRL SHOULD be considered as if it were signed by the original CRL-issuer.