COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

INTRODUCTION

VERSION 2.0 – 20 JANUARY 2009

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.

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

The following people have contributed to the Common PKI Specification:

Harald Ahrens, Petra Barzin, Fritz Bauspieß, Andreas Berger, Hans-Joachim Bickenbach, Jobst Biester, Jürgen Brauckmann, Detlef Dienst, Holger Ebel, Arno Fiedler, Dirk Fox, Alfred Giessler, Ernst-Günter Giessmann, Volker Hammer, Tamás Horváth, Karl-Adolf Höwel, Hans-Joachim Knobloch, Ulrike Korte, Rolf Lindemann, Dieter Pfeuffer, Georgios Raptis, Helmut Reimer, Dieter Reul, Olaf Schlüter, Peter Schmidt, Wolfgang Schneider, Josef Peter Winand, Klaus-Dieter Wirth and Eduward van der Zee.

Document History

VERSION DATE	CHANGES
1.0 30.09.2001	First public edition
1.0.1 15.11.2001	A couple of editorial and stylistic changes: • references to SigG-specific issues eliminated from core documents • core documents (Part 1-7) and optional profiles have been separated in different PDF documents.
1.0.2 19.07.2002	Several editorial changes. Part 5 has been added.
1.0.2 11.08.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.03.2004	Several editorial changes, and • inclusion of new Part 8 on XML Signature and Encryption
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Update of document structure and renaming of some parts.

Table of Contents

1 Objectives

1.1 Common PKI Profiles International Standards

IETF standards (RFCs) cover a wide variety of computer and communications applications and provide great flexibility in technical aspects (communication protocols, data formats, procedures etc.). For the realization of interoperable applications, the IETF standards may be "too flexible": at some aspects they offer too many implementation alternatives to choose from, while some other aspects relevant for the specific application area may not be covered by them.

The Common PKI Specification profiles the IETF standards, more closely the RFCs of the PKIX and the SMIME working groups, as well as the technical specifications of W3C and of ETSI to the needs of the intended application area: the secure exchange of emails and documents combined with the use of qualified signatures. This tailoring is achieved by:

- specifying a selection of the numerous technical standards that are relevant for the target application area and that are to be followed by implementers,
- restricting the possible implementation alternatives in order to promote interoperability as well as to reduce the costs of implementation and conformity tests,
- it extends the international standards to cover specific needs or aspects that are not covered by those standards, but that need regulation for the sake of interoperability.

1.2 The Scope of Common PKI

This Common PKI Specification describes data formats and communication protocols to be employed in interoperable PKI-based applications. The specification focuses on security services for authentication (including user identification and data integrity), confidentiality and non-repudiation. The specification concentrates on interoperability aspects, embracing different on-line services of certification service providers (CSPs), such as certification service, directory service and time-stamp service, as well as client applications accessing and relying on those services. As most important target application area, data formats for the secure interchange of emails, XML documents and files via Internet are defined. A typical set-up of PKI components with corresponding Common PKI documents is depicted in Figure 1. (Note that the presented components and respectively their partitioning into sub-modules, such as OCSP server or signature creation module, are only an example. Real-life systems may comprise different types of components and modules.)

The Common PKI specification intends to promote wide interoperability among client applications and CA services, irrespective of the required security level; a characteristic referred to as ***vertical interoperability***. Accordingly, this version of the specification concentrates merely on technical aspects (data structures, protocols, interfaces) and consciously avoids prescribing any specific certificate policy to be applied in conjunction with compliant systems.

Besides issuing this Common PKI Specification, testing facilities have been specified that can be used to assess the conformity of components with the interoperability specification. This *Common PKI Test Specification* describes a set of well-defined tests that provide reproducible results and cover all aspects of the interoperability specification.

1.3 Some History

Lots of efforts have been made in Germany to establish suitable public key infrastructures for the secure interchange of emails and data files. Industrial companies and research institutes have grounded the association TeleTrusT. The MailTrusT Working Group of TeleTrust has developed a series of standards, called MailTrusT (MTT), to achieve interoperability among email and file transfer client software and respectively CA services provided by the member companies. The last version of MTT is MTT v2 [MTTv2], which was mainly used in health care and governmental applications. Refer to www.teletrust.de for more information.

The "German Signature Act" (Signaturgesetz, SigG) defines the general framework for so-called qualified electronic signatures that can be used in legal actions. SigG has been first passed in 1997 and has been modified in 2001 [SigG01] to meet the requirements of the "Directive 1999/93/EC of the European Parliament and of the Council of 13 December on a Community Framework for Electronic Signatures" [ECDIR99]. The signature law and the ordinance on its technical realization (Signaturverordnung, SigV [SigV01]) put very strong security requirements on the entire public key infrastructure providing the means for the signatures, i.e. on signature devices, signature software as well as CA services.

The GISA – German IT Security Agency (Bundesamt für Sicherheit in der Informationstechnik, BSI) has issued a "Signature Interoperability Specification" (SigI), promoting uniform signature and certificate formats for SigG-related applications. Parallel to the efforts of TeleTrusT, companies providing qualified CA services have founded the association "T7" and have issued their own standard, called "Industrial Signature Interoperability Standard" (ISIS, [ISIS99]), which is an enhancement of a subset of SigI. Refer to www.t7-isis.de for more information.

In 2001 TeleTrusT and T7 decided to transfer their technical specification into one common standard, called ISIS-MTT, which is intended to promote wide interoperability among client applications and CA services, irrespective of the required security level. ISIS-MTT should serve as the common industrial standard. In 2008 ISIS-MTT was renamed to Common PKI; the last version to be known under the old name is [ISIS-MTTv1.1].

Both ISIS 1.2 and MailTrusT v2 have been designed to conform to standards of the IETF, especially to those of the PKIX and the SMIME working group. Hence, there are actually only slight differences between the two and they can be made compatible without enormous changes in data formats, equipment and software. The kernel part of Common PKI contains specifications that provide international compatibility in the technical realization. In particular, the Common PKI Specification is a profile to IETF standards as well as to technical specifications of W3C and the European Telecommunications Standards Institute (ETSI). ETSI standards regulate the implementation of qualified signatures and related services, as laid down in the Directive 1999/93/EC. The SigG-Profile, an optional "sub-profile" to Common PKI, implements specific requirements on signatures raised by the German Signature Act and is intended for use only in this specific context.

In earlier versions of the specification (ISIS-MTT 1.0 to 1.1), the fact that a product or service is not mandatorily required to comply with the requirements of the SigG sub-profile was expressed by publishing the latter as an "Optional Profile" document, as opposed to the other "Core Parts". Meanwhile the compliance criteria for different products and services have been defined in a separate Common PKI document. Hence the SigG Profile is maintained as regular part of the specification since Common PKI 2.0. This change in document structure does not imply that all Common PKI compliant products and services must now mandatorily fulfil the requirements specified in the SigG (Sub-)Profile.

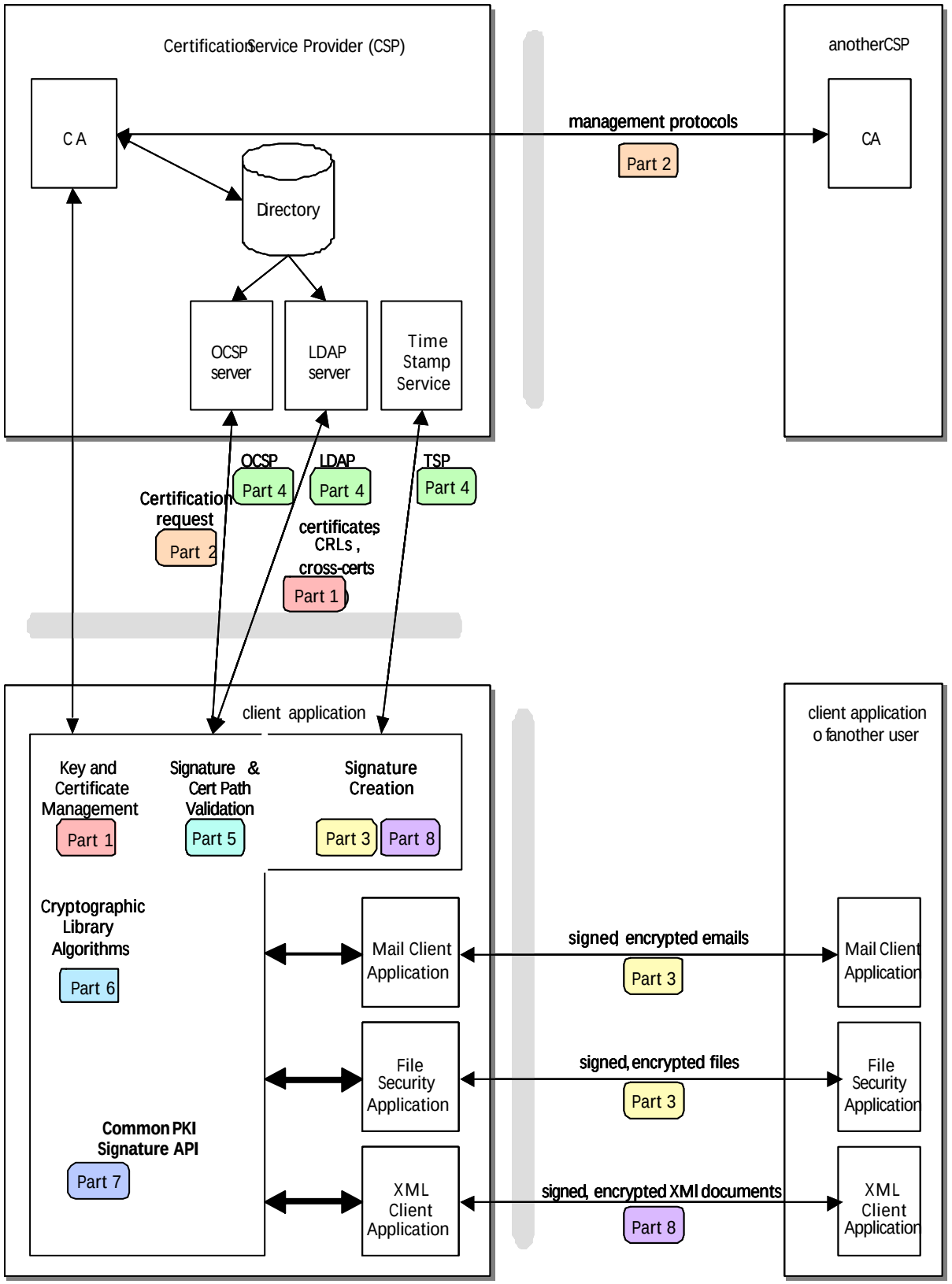


Figure 1: Overview of Common PKI and its relationship to interfaces among PKI components (note that implementations may choose to selectively support only a suitable subset of Common PKI data formats and interfaces)

2 Structure

The current version of the Common PKI Specification comprises the following *Parts*:

- Part 1: Certificate and CRL Profiles
- Part 2: PKI Management
- Part 3: CMS based Message Formats
- Part 4: Operational Protocols
- Part 5: Certificate Path Validation
- Part 6: Cryptographic Algorithms
- Part 7: Signature API
- Part 8: XML based Message Formats
- Part 9: SigG Profile

In addition to the *Specification Parts* of Common PKI, a matching *Common PKI Test Specification* is provided.

Supplemental documentation may also be published as a Common PKI document. An important example are the *Common PKI Compliance Criteria* that define, which of the requirements of the Specification Parts a specific PKI product or service of a certain type (e. g. an OCSP server, a secure e-mail client or a SigG-Profile-compliant certification service).

In-between releases of the Common PKI Specification, *Corrigenda* to the specification and test specification, if necessary, are published as separate documents. These *Corrigenda* become effective immediately with their publication, i.e. the effectual text of the Common PKI Specification will be that of the *Specification Parts* with the changes specified in the *Corrigenda* document applied.

3 Terminology and Notation

3.1 Support Requirements

The Common PKI Specification raises requirements on PKI-components for supporting a variety of objects, such as functions of an API, messages of some communication protocol, and specific fields in some data structure. As a basic approach, the Common PKI Specification consequently distinguishes among requirements of the following two types:

- requirements that have to fulfilled during the *generation* of particular objects, e.g. of an email, a certificate, an OCSP request message, a XML signature, or while calling an API. Such requirements typically enforce constraints on the contents of data and protocol objects as well as restrict the set of applicable API functions or cryptographic mechanisms while generating those objects.
- requirements that have to be fulfilled while *processing* particular objects, e.g. while displaying the content of an email, while decoding and interpreting a certificate, while processing an OCSP request message, while parsing and evaluating a XML data element, or while executing an API function. Such requirements typically enforce the component to accept and properly interpret and evaluate certain contents in data and protocol objects as well as to provide certain API functions or cryptographic mechanisms to properly process those objects.

These two different types of requirements will be denoted by *'GEN'* and respectively by *'PROC'* in shorthand.

Support requirements regarding generation and processing of objects are described by using the key words *MUST, SHALL, SHOULD, RECOMMENDED, MAY, OPTIONAL*, respectively *MUST NOT, SHALL NOT, SHOULD NOT, FORBIDDEN*. These key words will be used in this document using the semantics defined in [RFC2119] and will be typeset in capitals. For clarity, the terminology of [RFC2119] is simplified here to five notions, which are listed in Table 1. The word *SHALL* occurring in RFCs has been translated here to *MUST*. To provide a compact notation for tables we introduce in Table 1 a shorthand notation too.

Table 1: Abbreviations for Key Words to Indicate Support Requirements

	MEANING
++	This sign is equivalent to the key words *MUST, SHALL, MANDATORY.*
+	This sign is equivalent to the key words *SHOULD, RECOMMENDED*.
+-	This sign is equivalent to the key words *MAY, OPTIONAL.*
-	This sign is equivalent to the key words *SHOULD NOT, NOT RECOMMENDED.*
--	This sign is equivalent to the key words *MUST NOT, SHALL NOT, FORBIDDEN*.
n.a.	no information available, not applicable

Support of a specific data field at the *generating* component refers to the requirement whether the component must, should, may, should not or must not *include or fill in* the specified field while generating the object. Support of an API function or cryptographic algorithm at the generating component refers to the requirement whether the component must, should, may, should not or must not *call* a specific API functions or *employ* a specific cryptographic mechanism.

Support of a specific data field at the *processing* component refers to the requirement whether the component must, should, may, should not or must not be able to *interpret or evaluate* the content of the specified field while generating the object. Support of an API function or cryptographic algorithm at the *processing* component refers to the requirement whether the component must, should, may, should not or must not *implement* a specific API function or cryptographic mechanism.

A note on the support of ASN.1 objects: ASN.1 is widely used to specify data and protocol objects. The corresponding encoding rules, such as DER, allow a platform-independent representation of the objects, which is widely used in protocol and data object implementations. We stress that *all Common PKI compliant clients MUST be able to decode or to skip all fields of a DER encoded data or protocol object that are specified in this specification, i.e. even the ones marked as forbidden*. Such fields occur in this specification because they conform to some older and obsolete specification (PKIX, ISIS or MailTrusT) and may thus occur in data objects (certificates, signed documents or CRLs) in current use. Backward compatibility with these objects requires tolerant behaviour of the components processing them. This is just the application of the principle "be strict at what you send and be tolerant at what you receive".

3.2 Common PKI Conformance

A component is called Common PKI compliant, if it satisfies all *requirements* that apply to a specific component and that are specified as *obligatory* ('++') or *forbidden* ('--') in the Common PKI specification. It should be noted that the specification also contains *recommendations* in addition to the requirements that are always explicitly marked ('+' or '-'). Common PKI conformance only refers to requirements and not to recommendations.

3.3 Notation

The Common PKI Specification is intended to be a kind of quick reference. Specifications are provided in tabular form with a reference to corresponding sections of IETF and ETSI documents. Therefore, Common PKI is written in the style of a delta specification that allows to produce a comprehensive specification without reduplicating all information from the referenced standards.

Most tables have the same structure. Each row corresponds to one item, e.g. a field of a data structure. The columns of the tables headed by *#*, *Name*, *Semantics*, *References*, *Support* and *Notes* provide the following information:

#	unique reference number that corresponds to one particular item, e.g. a field of a data structure,
Name	technical name of the field,
Semantics	short description of the meaning of the field in order for ease of reading,
References	reference to clauses in the corresponding IETF, W3C or ETSI standards where the semantics and syntax of the objects are described,
Support	requirements for *generating (GEN)* and *processing (PROC)* components using the shorthand notation of Table 1, and
Notes	further explanatory text that may be given on constraints, permitted value set etc. applying for the described object.

References to Common PKI documents will be given using the following notation:

‘P*x*.T*y*.#*z*’ reference to ‘Part *x*, Table *y*, Row *z*‘, or

‘P*x*.T*y*.[*v*]’ reference to ‘Part *x*, Table *y*, Note *v*’, or

P*x*.S*y.z* reference to section *y.z* of Part *x* in the Common PKI Specification

As readily mentioned in Section 1.2, this Common PKI Specification is a profile to PKIX, W3C and ETSI standards. To allow the reader to quickly locate profiling information, text segments adding new definitions to those profiled documents, replacing requirements or restricting the usage of objects in some way, will be conspicuously indicated by the words ‘**Common PKI Profile’** and the shown fat typesetting.

References

[ECDIR99] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community Framework for Electronic Signatures

[MTTv2] MailTrusT Version 2, March 1999, TeleTrusT Deutschland e.V., www.teletrust.de

[ISIS99] Industrial Signature Interoperability Specification ISIS, Version 1.2, December 1999, T7 i. Gr., www.t7-isis.de

[ISIS-MTTv1.1] Common ISIS-MTT Specifications for Interoperable PKI Applications, Version 1.1, March 2004, T7 i.G. and TeleTrustT e.V.

[RFC2119] Key words for use in RFCs to Indicate Requirement Levels, RFC 2119, March 1997

[SigG01] Law Governing Framework Conditions for Electronic Signatures and Amending Other Regulations (Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften), Bundesgesetzblatt Nr. 22, 2001, S.876, non-official version available at http://www.bundesnetzagentur.de/media/archive/3612.pdf.

[SigV01] Ordinance on Digital Signatures (Verordnung zur digitalen Signatur – SigV), 2001, non-official version available at http://www.bundesnetzagentur.de/media/archive/3613.pdf.

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 1

CERTIFICATE AND CRL PROFILES

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0 30.09.2001	First public edition
1.0.1 15.11.2001	A couple of editorial and stylistic changes: 1) references to SigG-specific issues eliminated from core documents 2) core documents (Part 1-7) and optional profiles have been separated in different PDF documents.
1.0.2 19.07.2002	Several editorial changes and bug-fixes. The most relevant changes affecting technical aspects are: 1) OID { PKIX 9 3 } for pseudonym deleted. (v101.T2.#6) 2) The correct interpretation of badly encoded INTEGERs is no longer required but recommended. (T2.[8]) 3) Encoding Latin-1 characters in UTF8 strings is no longer forbidden, but it is still not recommended. (T6.[2]) 4) Including an *emailAddress* attribute in EE DNames is tolerated by ISIS-MTT for practical compatibility reasons. (T7.#17,[5]) 5) The "dummy" ASN.1 definition of the obsolete *ORAddress* type changed to one that is able to decode the full structure. The definition in v101 could not be compiled. (T8.#13) 6) The support requirements for *AuthorityKeyIdentifier* have been changed to fully comply with RFC2459: *keyIdentifier* is mandatory, *authorityCertIssuer&Serial* optional. (T11.#2..4) The same applies for ACs. (T30.#1) 7) All methods described in RFC2459 are permitted here too to build key identifiers. (T11.[2]) 8) Providing an LDAP-URL in *IssuerAltNames* pointing to the CA certificate is no longer mandatory, but optional. (T16.#2,[3]) 9) The support of *SubjectDirectoryAttributes* in processing components is no longer discouraged (-), but (according to RFC3039) optional. (T17.#1) 10) According to RFC3280, *BasicConstraints* MAY appear in EE-Certs. V1.0.1 advised against this practice. (T18.#1) 11) *NameConstraints* and *PolicyConstraints* MUST be supported by processing components, as these extensions MUST be considered in the validation process, if they are flagged critical. In v1.0.1 this was only recommended. (T19.#1,[1], T20.#1,[1]) 12) *CRLDistributionPoints* is no longer mandatory, but recommended to be supported by processing components. (T21.[1],T30.#2) Applications may use other methods to locate CRLs. 13) As for the generation of PKCs and ACs, *CRLDistributionPoints* is required in case the CA issues indirect CRLs and recommended in "direct" case. (T21.#1,#3,#5, T25.#3) V1.0.1 did not make this distinction. Providing an LDAP-URL is no longer mandatory. 14) *AuthorityInfoAccess* is no longer mandatory, but recommended to be supported by processing components. (T23.[1],T30.#4) Applications may use other methods to obtain status info. 15) The definition of *MonetaryValue* has been extended to the form given by v1.2.1 of [ETSI-QC]. A backward compatibility is automatically given. (T25.#15) 16) Alternative name forms (except *directoryString*), similar to those in the *IssuerAltNames* extension of PKCs, MAY be included in the *issuer* field of ACs. (T28.#4)
1.0.2 11.08.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.03.2004	Several editorial changes. The most relevant changes affecting technical aspects are: 1) *caIssuer* information in *AuthorityInfoAccess* is no longer forbidden but optional. 2) *ExtendedKeyUsage* now follows [RFC3280]. 3) *SubjectAltNames*, *IssuerAltNames* and the *GeneralNames* structure now follow [RFC3280]. 4) *KeyUsage* has been aligned with [ETSI-CPN]. 5) Following [ETSI-CPN], *countryName* is not longer required for end entity subject names. 6) Mandatory use of *UTF8String* encoding for *DirectoryString* elements has been postponed for a transition period

	7) The *gender* attribute is permitted in EE subject names of natural persons. 8) Definitions of ISIS-MTT private attributes for attribute certificates have been moved from the optional SigG Profile to core Part 1. 9) In accordance with RFC 3039bis,.use of *postalAddress* is discouraged. 10) *ReasonFlags* for *CRLDistributionPoints* now follow [RFC3280]. 11) *IssuingDistributionPoints* now follows [RFC3280]. 12) *CRLReason* to RFC 3280 now follows [RFC3280]. 13) *DisplayText* for *CertificatePolicies* now follows [RFC3280].
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Adapted to new versions of the base standards: - ETSI TS 101 861 v1.3.1 - ETSI TS 101 862 v1.3.3 - ETSI TS 102 280 v1.1.1 - RFC 2822 - RFC 2460 - RFC 3490 - RFC 3629 - RFC 3739 - RFC 3986 - RFC 4510 - RFC 4516 - RFC 4519 - RFC 4523 - RFC 5280 - X.509:2005 Various corrections and clarifications.

Table of Contents

1 Preface

This part of the Common PKI specification describes certificate and certificate revocation list (CRL) formats. These formats conform to the most widely accepted international standards, namely to the ITU-T X.509 standard [X.509:2005] and to the PKIX-profile for public key certificates and CRLs [RFC5280]. General information from those referenced documents will not be completely repeated here. Only a short description of the semantics and relevant notes on the usage or value constraints will be given.

Fulfilling the requirements of the special application area of *qualified certificates* is a major goal of this Common PKI Specification. The full compatibility with the PKIX qualified certificate profile [RFC3739] (formerly [RFC3039]) and the ETSI Qualified Certificate Profile [ETSI-QC] Standards of the European Telecommunications Standards Institute (ETSI) will be enforced. As for attribute certificates, the [X.509:1997] format (attribute certificate v1) has been used as basis for this specification.

Besides conformance with international standards, backward compatibility with [ISIS] and [MTTv2] will be provided as far as possible, so that legacy systems and information (e.g. certificates, signed documents) can be used further on. This complex profiling structure is depicted in Figure 1 below. (The figure represents the status as of ISIS-MTT 1.0; several of the base standards have evolved and influenced subsequent versions of ISIS-MTT and Common PKI.)

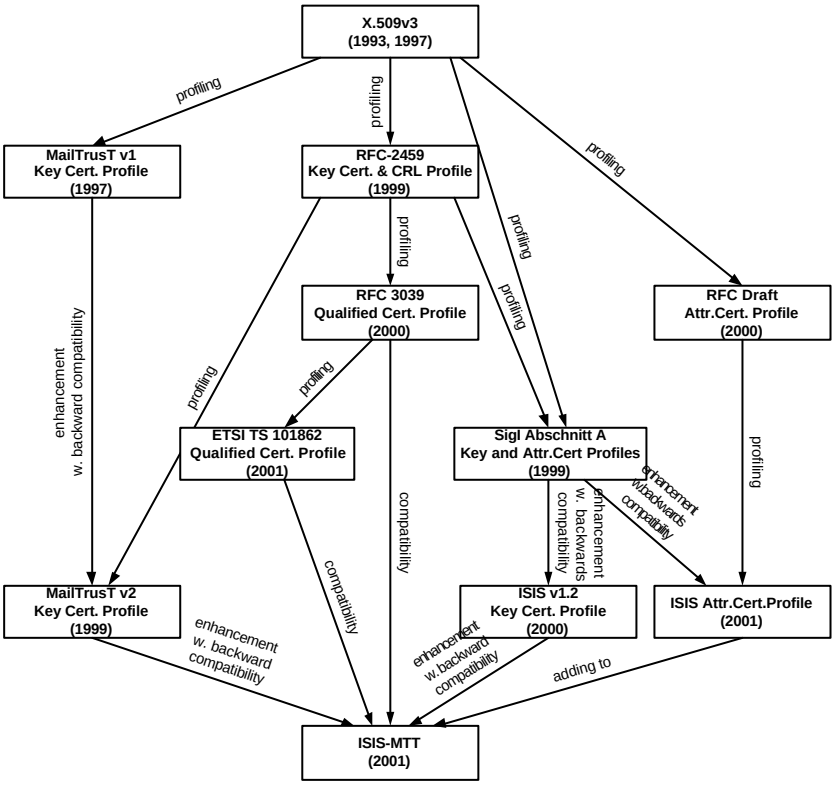


Figure 1: An overview of different standards and profiles on certificate formats

2 Public Key Certificate Format

Table 1: Certificate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**Certificate** ::= SEQUENCE {				4.1.1		
2	tbsCertificate TBSCertificate,	the DER-encoding of this "to be signed" part of the data structure will be signed by the CA			4.1.1.1	T2	
3	signatureAlgorithm AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this certificate			4.1.1.2	T4	
4	signature BIT STRING }	the signature of the CA represented as BIT STRING			4.1.1.3		

Table 2: TBSCertificate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**TBSCertificate** ::= SEQUENCE {				4.1.1.1		
2	version [0] EXPLICIT Version DEFAULT v1,	Version number of the public key certificate format			4.1.2.1	#12	[1]
3	serialNumber CertificateSerialNumber,	Serial number of the certificate			4.1.2.2	#13	[2] [3] [8]
4	signature AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this certificate.			4.1.2.3	T4	[4]
5	issuer Name,	DName of the issuer of this certificate			4.1.2.4	T15	[5]
6	validity Validity,	Validity period of the certificate			4.1.2.5	T3	
7	subject Name,	DName of the certificate holder			4.1.2.6	T5	[6]
8	subjectPublicKeyInfo SubjectPublicKeyInfo	Public key of the certificate holder and the corresponding algorithm			4.1.2.7	#14	[10]
9	issuerUniqueID [1] IMPLICIT UniqueIdentifier OPTIONAL,	a unique identifier for the issuer, if issuer DName is reused over time	—	+	4.1.2.8	#17	[7]
10	subjectUniqueID [2] IMPLICIT UniqueIdentifier OPTIONAL,	a unique identifier for the subject, if subject DName is reused over time	—	+	4.1.2.8	#17	[7]
11	extensions [3] EXPLICIT Extensions OPTIONAL}	Extensions	++	++	4.2	T9	
12	**Version** ::= INTEGER { v1(0), v2(1), v3(2) }	Version number of the certificate format			4.1.2.1		
13	**CertificateSerialNumber** ::= INTEGER	Serial number of the certificate			4.1.2.2		[8]

14	**SubjectPublicKeyInfo** ::= SEQUENCE {	Public key structure			4.1.2.7		
15	algorithm AlgorithmIdentifier	Cryptographic algorithm to be used with the key			4.1.2.7	T4	[9]
16	subjectPublicKey BIT STRING }	Public Key in DER-encoded form			4.1.2.7		[8]
17	**UniqueIdentifier** ::= BIT STRING		--	+	4.1.2.8		[7]
[1]	[RFC5280] Value v3(2) must be used, if any extension is used as expected in this profile. If no extension, but #9 or #10 is present, use v2(1). Otherwise the value is v1(0).						
[2]	[RFC5280]: the serial number MUST be a positive integer, not longer than 20 octets ($1 \leq SN < 2^{159}$, MSB=0 indicates the positive sign!). Processing components MUST be able to interpret such long numbers. **Common PKI Profile:** the above requirements on length apply.						
[3]	[RFC5280]: The *issuer* name and the *serialNumber* of public key certificates (PKCs) MUST identify a unique certificate. **Common PKI Profile:** the uniqueness requirement is extended to all kind of certificates, i.e. for PKCs as well as attribute certificates (ACs). The reason for that is to allow the same CA to issue PKCs as well as ACs (which is the case in current systems) and furthermore to allow the same CRL to contain entries to PKCs as well as to ACs. Note, that [RFC3281] forbids issuing PKCs and ACs at the same time, which is not the case in Common PKI.						
[4]	[RFC5280]: The content must be the same as that of *signatureAlgorithm* in T1.#3.						
[5]	[RFC5280]: The *issuer* name MUST be a non-empty DName. Processing components MUST be prepared to receive the following attributes: *countryName*, *organizationName*, *organizationalUnitName*, *distinguishedNameQualifier*, *stateOrProvinceName*, *commonName*, *serialNumber*, and *domainComponent*. Processing components SHOULD be prepared for attributes: *localityName*, *title*, *surname*, *givenName*, *initials*, *pseudonym*, *and generationQualifier*. [RFC3739]: the issuer DName MUST contain an appropriate subset of the following attributes: *domainComponent*, *countryName*, *stateOrProvinceName*, *organizationName*, *localityName* and *serialNumber*. Additional attributes may be present, but SHOULD NOT be necessary to identify the CA. [ETSI-QC]: the *issuer* name MUST contain the *countryName* attribute. The specified country MUST be the country where the issuer CA is established. [ETSI-CPN]: the *issuer* name MUST contain the *countryName* and the *organizationName* attributes. **Common PKI Profile:** the *issuer* DName MUST be identical to the *subject* DName in the issuer's certificate to allow chain building. The *issuer* DName (i.e. the DName of each CA) MUST contain at least the attributes *countryName* and *organizationName*. *OrganizationName* SHOULD contain the name of the organization that operates the CA.						

[6]	[RFC5280]: the *subject* name MUST be unique for a subject entity (certificate holder) among all certificates issued by the CA and for the whole lifecycle of the CA. The same requirements apply as to the *issuer* field [5]. Instead of including an *emailAddress* DName attribute, the *rfc822Name* alternative of the *subjectAltNames* extension SHOULD be used. [RFC3739]: the *subject* DName MUST contain an appropriate subset of the following attributes: *countryName, commonName, surname, givenName, pseudonym, serialNumber, title, organizationName, organizationalUnitName, stateOrProvincename and localityName*. Additional attributes may be present, but SHOULD NOT be necessary to distinguish the subject name from other subject names within the issuer domain. If a *pseudonym* is given, *surname* and *givenName* MUST NOT be present in the DName. [ETSI-CPN]: The subject field of EE certificates for natural persons SHALL include at least the *commonName* or the *givenNamen* and *surname* attribute. [ETSI-TSP]: The subject name of TSP certificates SHALL contain an appropriate subset of the following attributes: *countryName*, *stateOrProvinceName*, *organizationName* and *commonName*. The *organizationName* and *commonName* SHALL be present. **Common PKI Profile:** the subject name of an end entity MUST at least contain the attribute *commonName*. In an Common PKI-conforming QC, the *commonName* attribute MUST either specify the legal name of the certificate holder or a pseudonym, where the pseudonym MUST be marked with the suffix ":PN". To conform with [RFC3739], certificates MAY contain the same name (including suffix!) additionally in the *pseudonym* attribute too. If a *pseudonym* attribute is present, it MUST contain the same name (including suffix) as the *commonName* attribute. Including a *gender* attribute in EE subject names of natural persons is permitted by Common PKI. Including an *emailAddress* attribute in EE DName is tolerated by Common PKI for practical compatibility reasons (Netscape).
[7]	[RFC5280]: CAs SHOULD generate certificates with unique *subject* and *issuer* DNames, and SHOULD NOT make use of *uniqueIdentifers*. Processing components SHOULD be able to interpret *uniqueIdentifiers*. **Common PKI Profile:** CAs MUST generate certificates with unique *subject* and *issuer* DNames over the entire life cycle of the CA, and MUST NOT make use of *uniqueIdentifers*. Processing components that cannot properly handle *uniqueIdentifiers*, MUST refuse those certificates.
[8]	A note on implementation: the value of the DER-encoding of *INTEGER* types contains the 2's complement form of the number in big endian form (most significant octet first). This is a signed representation, i.e. the most significant bit (MSB) indicates the sign, and must thus be a '0' for natural numbers. It is a common mistake to encode natural numbers, like *CertificateSerialNumber* or the *modulus* and *exponent* of *RSAPublicKey*, in unsigned form. Implementers MUST make sure that a zero octet (00h) is inserted in front of the unsigned form if the MSB of the unsigned value is a '1', e.g. 255 must be encoded as (00h,ffh). As for receiving and processing badly encoded *INTEGERs*, processing components SHOULD be able to retrieve the correct number, if it can be assumed, as in the above mentioned cases, that the represented number is a natural number, e.g. (0xff) must be interpreted as 255 and not as –1.
[9]	[ETSI-CPN]: ETSI strongly recommends to use rsaEncryption.
[10]	**Common PKI Profile**: Whether more than one public key certificate for a particular public key may be issued is a matter of policy that lies beyond the scope of this specification. Note, however, that if several public key certificates exist pertaining to the same public key, any operation done with the corresponding key pair cannot be uniquely attributed to a particular certificate. Therefore it is good practice to avoid issuing a second certificate for a public key for reasons of security and usability. If a second certificate is issued nevertheless, it should only be for the same certificate holder and consistent with the policy (as manifested in particular in the *KeyUsage*, *ExtendedKeyUsage*, *QCStatements* and *CertificatePolicies* extensions) of the original certificate.

Table 3: Validity, Time

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**Validity** ::= SEQUENCE {				4.1.2.5		
2	notBefore Time,				4.1.2.5		[1]
3	notAfter Time }				4.1.2.5		[1]
4	**Time** ::= CHOICE {				4.1.2.5		
5	utcTime UTCTime,		++	++	4.1.2.5.1		
6	generalizedTime GeneralizedTime }		++	++	4.1.2.5.2		
[1]	[RFC5280]: Validity dates before and through 2049 MUST be encoded by CAs as *UTCTime*, dates in 2050 and later as *GeneralizedTime*. Date values MUST be given in the format YYMMDDhhmmssZ resp. YYYYMMDDhhmmssZ, i.e. always including seconds and expressed as Zulu time (Universal Coordinated Time) **Common PKI Profile:** Processing components MUST be able to interpret all date formats, i.e. *GeneralizedTime* too.						

Table 4: AlgorithmIdentifier

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**AlgorithmIdentifier** ::= SEQUENCE {				4.1.1.2		
2	algorithm OBJECT IDENTIFIER,				[RFC 3279]	P6	[1]
3	parameters ANY DEFINED BY algorithm OPTIONAL }				[RFC 3279]	P6	
[1]	For permitted algorithm identifiers and parameters refer to Part 6 (Cryptographic Algorithms) of this Common PKI Specification.						

2.1 Distinguished Names

Table 5: Name

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**Name** ::= CHOICE { RDNSequence }				4.1.2.4	#2	
2	**RDNSequence** ::= SEQUENCE OF RelativeDistinguishedName				4.1.2.4	#3	
3	**RelativeDistinguishedName** ::= SET OF AttributeTypeAndValue				4.1.2.4	#4	
4	**AttributeTypeAndValue** ::= SEQUENCE {				4.1.2.4		
5	type AttributeType,					#7	
6	value AttributeValue }					#8	
7	**AttributeType** ::= OBJECT IDENTIFIER				4.1.2.4		
8	**AttributeValue** ::= ANY DEFINED BY AttributeType				4.1.2.4		

Table 6: DirectoryString

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**DirectoryString** ::= CHOICE {				4.1.2.4		[1]
2	printableString PrintableString (SIZE (1..maxSize)),		+-	++			
3	teletexString TeletexString (SIZE (1..maxSize)),		--	++			[3]
4	utf8String UTF8String (SIZE (1.. maxSize)),		+	++			[2]
5	bmpString BMPString (SIZE (1..maxSize)),		--	++			
6	universalString UniversalString (SIZE (1..maxSize)) }		--	++			

[1]	[RFC5280]: CAs MUST use either the *PrintableString* or *UTF8String* encoding of *DirectoryString*, with two exceptions. When CAs have previously issued certificates with issuer fields with attributes encoded using *TeletexString*, *BMPString*, or *UniversalString*, then the CA MAY continue to use these encodings of the *DirectoryString* to preserve backward compatibility. Also, new CAs that are added to a domain where existing CAs issue certificates with issuer fields with attributes encoded using *TeletexString*, *BMPString*, or *UniversalString* MAY encode attributes that they share with the existing CAs using the same encodings as the existing CAs use. **Common PKI Profile:** Strings MAY be encoded as *PrintableString* in order to ensure a better interoperability with legacy applications. If a string cannot be represented in the *PrintableString* character set, *UTF8String* encoding MUST be used. If permitted by the applicable certificate policy, characters that are not in the *PrintableString* character set MAY be transcribed in *PrintableString* characters according to local conventions for the transcription of national character sets in DNS domain names or E-Mail addresses (e.g. German umlaut “ä” to “ae”).
[2]	**Common PKI Profile:** Following [MTTv2], Common PKI RECOMMENDS using a subset of the UTF8 character set, including only the ANSI/ISO 8859-1 characters (Unicode Latin-1 page). 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. Hence, generating components SHOULD NOT include characters of code pages other than Latin-1. 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]	Note that there are two practices to encode *TeletexString*: some implementations use the T.61 encoding rules using *floating diacritics* (roughly said, “á” will be encoded on two bytes as “´a”). Unfortunately, there are even different code tables in use, but the one from IBM is probably the most widely used. Most Internet applications simply use the ANSI/ISO 8859-1 code table (used by Windows and UNIX systems) to encode strings and tag them as *TeletexString*. Applications SHOULD assume this case, when processing and SHOULD encode in this way, when generating data.

Table 7: Supported X.501 attribute types and their maximal lengths

#	ATTRIBUTE NAME	ATTRIBUTE OID	ASN.1 STRING TYPE	MAXIMAL STRING LENGTH, VALUE CONSTRAINTS (PKIX, IF DIFFERENT)	SUPPORT RFC 5280	SUPPORT RFC 3739	SUPPORT COMMON PKI				NOTES
							CA DNAME		EE DNAME		
					PROC	PROC	GEN	PROC	GEN	PROC	
1	commonName	{id-at 3}	DirectoryString	64	++	++	+-	++	++	++	[1]
2	surName	{id-at 4}	DirectoryString	64 (32768)	+	++	+-	++	+-	++	[2]
3	givenName	{id-at 42}	DirectoryString	64 (32768)	+	++	+-	++	+-	++	[2]
4	serialNumber	{id-at 5}	PrintableString	64	n.a.	++	+-	++	+-	++	
5	title	{id-at 12}	DirectoryString	64	+	++	+-	++	+-	++	
6	organizationName	{id-at 10}	DirectoryString	64	++	++	++	++	+-	++	
7	organizationalUnitName	{id-at 11}	DirectoryString	64	++	++	+-	++	+-	++	
8	businessCategory	{id-at 15}	DirectoryString	128	n.a.	n.a.	-	+	-	+	[3]
9	streetAddress	{id-at 9}	DirectoryString	128	n.a.	n.a.	-	+	-	+	[4]
10	postalCode	{id-at 17}	DirectoryString	40	n.a.	n.a.	-	+	-	+	[4]
11	localityName	{id-at-7}	DirectoryString	128	+	++	+-	++	+-	++	
12	stateOrProvinceName	{id-at 8}	DirectoryString	128	++	++	+-	++	+-	++	
13	countryName	{id-at 6}	PrintableString (SIZE(2))	2 the ISO 3166 code	++	++	++	++	+-	++	
14	distinguishedNameQualifier	{id-at 46}	PrintableString	64 (n.a.)	++	n.a.	+-	++	+-	++	[2]
15	initials	{id-at 43}	DirectoryString	64 (32768)	+	n.a.	+-	+	+-	+	[2]
16	generationQualifier	{id-at 44}	DirectoryString	64 (32768)	+	n.a.	+-	+	+-	+	[2]
17	emailAddress	{pkcs-9 1}	IA5String	128	+ GEN--	n.a.	-	+	-	+	[5]
18	domainComponent	{0 9 2342 19200300 100 1 25}	IA5String	usage described in [RFC4519]	++	++	+-	++	+-	++	[8]
19	postalAddress	{id-at 16}	SEQUENCE SIZE (1..6) OF DirectoryString	6x30, usage described in [RFC3039]	n.a.	n.a.	-	+	-	+	[4]
20	pseudonym	{id-at 65}	DirectoryString	64 (n.a.)	n.a.	++	+-	++	+-	++	[1]
21	dateOfBirth	{id-pda 1}	GeneralizedTime	YYYYMMDD000000Z	n.a.	++	+-	++	+-	++	[6]
22	placeOfBirth	{id-pda 2}	DirectoryString	128 (n.a.)	n.a.	++	+-	++	+-	++	[6]
23	gender	{id-pda 3}	PrintableString (SIZE(1))	„M“ or „F“	n.a.	++	+-	++	+-	++	[6], [7]
24	countryOfCitizenship	{id-pda 4}	PrintableString (SIZE(2))	2 the ISO 3166 code	n.a.	++	+-	++	+-	++	[6]

25	countryOfResidence	{id-pda 5}	PrintableString (SIZE(2))	2 the ISO 3166 code	n.a.	++	+-	++	+-	++	[6]
26	nameAtBirth	{id-commonpki-at 14}	DirectoryString	64	n.a.	n.a.	+-	++	+-	++	[6]

[1]	**Common PKI Profile:** Following the common practice, the pseudonym MUST be put in the *commonName* attribute and marked with suffix ":PN". To conform with [RFC3739], the same name (including suffix) MAY be included in the dedicated *pseudonym* attribute too. If a *pseudonym* attribute is present, it MUST contain the same name (including suffix) as the *commonName* attribute.
[2]	**Common PKI Profile:** This Common PKI specification enforces the length limits published in PKIX documents. If no (practical) limit is set by some PKIX document, an appropriate maximal length is specified here. CAs MUST keep strings in new certificates at most as long as specified here. Clients MUST be able to display strings at least as long as specified here. For the sake of wider interoperability, clients SHOULD be able to parse arbitrarily long strings.
[3]	**Common PKI Profile:** *businessCategory* is not listed in any PKIX documents among the mandatory attributes. Hence, this Common PKI specification discourages from its use. For backward compatibility, processing components SHOULD still be able to interpret the attribute.
[4]	**Common PKI Profile:** *streetAddress* and *postalCode* are not listed in any PKIX documents among the mandatory attributes. Hence, this Common PKI specification discourages from its use. However, since current systems use them to store subjects' or their organizations' postal addresses, processing components SHOULD still be able to interpret these attributes. If *postalAddress* is used, elements of the string list provided in this attribute SHOULD contain all components of the address (including country, postal code, state, locality, street address), listed in the order and form, which is usual in the respective country and which is suitable for multi-lined printing in a regular document. An example for an address in Germany: 1st string element: Turmstraße 123 2nd string element: 10123 Berlin 3rd string element: Germany
[5]	**Common PKI Profile:** Including an *emailAddress* attribute in DNames is tolerated by Common PKI for practical compatibility reasons (Netscape).
[6]	[RFC3739]: The PKIX working group has recognized the demand that personal identification data can be in a separate attribute certificate (e.g. if the PKC should not make this info public). RFC3739 defines a couple of new DName attributes for this purpose (*dateOfBirth*, *placeOfBirth*, *gender*, *countryOfCitizenship*, *countryOfResidence*). According to RFC3739, these attributes are to be stored in the *SubjectDirectoryAttributes* extension of the public key certificate. RFC3739 explicitly states that new attribute types MAY be included according to local definitions. **Common PKI Profile:** In most European countries, the name of a person at his/her birth is a relevant identification attribute. Hence the new attribute *NameAtBirth* is introduced here. The *SubjectDirectoryAttributes* extension MAY be included ONLY in EE certificates of natural persons.
[7]	**Common PKI Profile:** Including a *gender* attribute in EE subject names of natural persons is permitted.
[8]	[RFC5280]: To represent an internationalized domain name, the issuing CA MUST perform the *ToASCII* label conversion specified in Section 4.1 of [RFC3490]. The label SHALL be considered a "stored string". That is, the *AllowUnassigned* flag SHALL NOT be set. **Common PKI Profile:** Processing operations MAY handle domain name labels in *domainComponent* attributes as mere IA5Strings, irrespectively whether they are traditional or converted internationalized domain names.

2.2 GeneralNames

Table 8: GeneralNames

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**GeneralNames** ::= SEQUENCE SIZE (1..MAX) OF GeneralName				4.2.1.7	#2	
2	**GeneralName** ::= CHOICE {				4.2.1.7		
3	otherName [0] IMPLICIT OtherName,	for identification data of some special syntax not listed below	- (RFC +-)	+-		#12	[1]
4	rfc822Name [1] IMPLICIT IA5String,	Email address in the Internet as described in [RFC2822]	+-	+			[4]
5	dNSName [2] IMPLICIT IA5String,	Internet domain name as in [RFC1034]	+-	+			[4]
6	x400Address [3] IMPLICIT ORAddress,	X400 address as in ITU-T X.411	-	+-		#13	[1]
7	directoryName [4] EXPLICIT Name,	X500 address	+-	+		T5	[2]
8	ediPartyName [5] IMPLICIT EDIPartyName,	name in an Electronic Data Exchange system	-	+-		#14	[1]
9	uniformResourceIdentifier [6] IMPLICIT IA5String,	URI as defined in [RFC1630], allowing uniform resource names (URNs) as well as URLs. Permitted URL forms are specified in [RFC1738], [RFC3986] and [RFC4516].	+-	+			[4]
10	iPAddress [7] IMPLICIT OCTET STRING,	IP address in IPv4 [RFC791] or in IPv6 [RFC2460] format	+-	+			
11	registeredID [8] IMPLICIT OBJECT IDENTIFIER }	a registered OBJECT IDENTIFIER (e.g. of a company or organization)	-	+-			[1]
12	**OtherName** ::= SEQUENCE { type-id OBJECT IDENTIFIER value [0] EXPLICIT ANY DEFINED BY type-id }				4.2.1.7		[1]
13	**ORAddress** ::= SEQUENCE { built-in-standard-attributes SEQUENCE OF ANY, built-in-domain-defined-attributes SEQUENCE OF ANY OPTIONAL, extension-attributes SET OF ANY OPTIONAL }				n.a.		[3]
14	**EDIPartyName** ::= SEQUENCE { nameAssigner [0] EXPLICIT DirectoryString OPTIONAL partyName [1] EXPLICIT DirectoryString }				4.2.1.7	T6	[2]

[1]	[RFC3281]: Conforming implementations MUST be able to support the *dNSName, directoryName, uniformResourceIdentifier*, and *iPAddress* options. This is compatible with the *GeneralName* requirements in [RFC5280]. Conforming implementations MUST NOT use *the x400Address, ediPartyName* or *registeredID* options. Conforming implementations MAY use the *otherName* option to convey name forms defined in Internet Standards. For example, Kerberos [KRB] format names can be encoded into the *otherName*, using a Kerberos 5 principal name OID and a SEQUENCE of the *Realm* and the *PrincipalName*. **Common PKI Profile:** The name forms *x400Address, ediPartyName* or *registeredID* options are considered to be obsolete and are no longer recommended for use.
[2]	*CHOICE* objects are always EXPLICITly tagged, independent of the default tagging modus.
[3]	[RFC5280] defines type *ORAddress* in appendix A.1 following [X.509:2005]. **Common PKI Profile:** As *ORAddress* is considered to be obsolete. Making use of the *ANY* type, the rather elaborate definition in [RFC5280] is replaced in this specification by a shallow "dummy" definition that allows receiving any *ORAddress* values, without actually recognizing the internal data content of the *ORAddress* structure.
[4]	[RFC5280] To represent an internationalized domain name in *GeneralName*, the issuing CA MUST perform the conversion operation specified in Section 4 of RFC 3490, with the following clarifications: in step 1, the domain name SHALL be considered a "stored string". That is, the *AllowUnassigned* flag SHALL NOT be set; in step 3, set the flag called *UseSTD3ASCIIRules*; in step 4, process each label with the *ToASCII* operation; and in step 5, change all label separators to U+002E (full stop). **Common PKI Profile:** Processing operations MAY handle domain names in *GeneralNames* structures as mere IA5Strings, irrespectively whether they are traditional or converted internationalized domain names.

2.3 Public Key Certificate Extensions

Table 9: Extensions

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	`Extensions ::= SEQUENCE SIZE (1..MAX) OF Extension`	a non-empty list of extensions			4.1	#2	
2	`Extension ::= SEQUENCE {`				4.1		
3	`  extnID  OBJECT IDENTIFIER,`	an OID specifying the type of the extension					
4	`  critical  BOOLEAN DEFAULT FALSE,`	critical flag					
5	`  extnValue  OCTET STRING }`	DER-encoding of the extension value					

The order of discussing individual extensions matches the order in [RFC5280].

Table 10: An overview of public key certificate extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT			REFERENCES		NOTES
					GEN CA CERT.	GEN EE CERT.	PROC	RFC	CO. PKI	
	X.509 BASIC EXTENSIONS							**RFC 5280**		
1	AuthorityKeyIdentifier	{2 5 29 35}	An ID identifying the public key (thus possibly several certificates) of the issuing CA.	--	++	++	+ (RFC n.a.)	4.2.1.1	T11	
2	SubjectKeyIdentifier	{2 5 29 14}	An ID identifying user certificates that contain a specific public key.	--	++	+	+ (RFC n.a.)	4.2.1.2	T11	
3	KeyUsage	{2 5 29 15}	Defines the purpose of the key pair (public and private key) corresponding to the public key contained in the certificate	++ (RFC +)	++	++ (RFC +-)	++ (RFC n.a.)	4.2.1.3	T12	
5	CertificatePolicies	{2 5 29 32}	Indicates the policy under which the certificate has been issued and the purposes for which it is to be used.	+-	+-	+-	++	4.2.1.4	T14	[1]
6	PolicyMappings	{2 5 29 33 }	Indicates in a CA certificate that the issuing CA considers its policy to be equivalent to the subject CA's policy.	+	+-	--	+	4.2.1.5	T15	
7	SubjectAltNames	{2 5 29 17}	Alternative technical names of the subject: OtherName, e-mail, DNS name, IP address, URI or other	- (RFC +-)	+-	+-	+	4.2.1.6	T16.#1	
8	IssuerAltNames	{2 5 29 18}	Alternative technical names of the issuing CA: OtherName, e-mail, DNS name, IP address, URI or other	-	+-	+-	+	4.2.1.7	T16.#2	
9	SubjectDirectoryAttributes	{2 5 29 9}	This extension may contain further X.500 attributes of the subject. Qualified certificates MAY store legal identification data (e.g. of a personal identification card, passport or similar) in this extension.	--	- (RFC 3739 n.a.)	+-	+ (RFC 3739 ++)	4.2.1.8	T17	
10	BasicConstraints	{2 5 29 19}	Indicates a CA certificate and defines how deep a certificate may exists below that CA.	++	++	+-	++	4.2.1.9	T18	
10a	NameConstraints	{2 5 29 30}	Indicates a name space in a CA certificate, in which all subject names (or subject alternative names) in subsequent certificates of the path shall be located.	++	+-	--	++	4.2.1.10	T19	
10b	PolicyConstraints	{2 5 29 36}	May be used in CA certificates to constrain path validation.	++	+-	--	++	4.2.1.11	T20	

11	ExtendedKeyUsage	{2 5 29 37}	Indicates purposes for which the key pair can be used, additional to or in place of those in the *KeyUsage* extension.	+-	+-	+-	++ (RFC n.a.)	4.2.1.12	T21	
12	CRLDistributionPoints	{2 5 29 31}	Identifies how CRL information to this certificate can be obtained.	-	+	+/++ dir/ind. CRL	+	4.2.1.13	T22	
12a	InhibitAnyPolicy	{2 5 29 54}	Indicates that the special *anyPolicy* OID is not considered an explicit match for other certificate policies except when it appears in an intermediate self-issued CA certificate.	++	+-	--	+	4.2.1.14	T22a	
12b	FreshestCRL	{2 5 29 46}	This extension (a.k.a. *DeltaCRLDistributionPoint*) identifies how delta CRL information is obtained.	--	+-	+-	+	4.2.1.15	T22b	
	RFC 5280 PRIVATE EXTENSIONS							**RFC 5280**		
13	AuthorityInfoAccess	{id-pe 1}	Access to online validation service and/or policy information of the CA issuing this certificate.	--	+-	+-	+ (RFC n.a.)	4.2.2.1	T23	
13a	SubjectInfoAccess	{id-pe 11}	Indicates how to access information and services for the subject of the certificate.	--	+-	+-	+ (RFC n.a.)	4.2.2.2	T23a	
	RFC 3739 QC PRIVATE EXTENSIONS							**RFC 3739**		
14	BiometricInfo	{id-pe 2}	Stores biometric information for authentication purposes.	--	+-	+-	+	3.2.5	T24	
15	QCStatements	{id-pe 3}	A statement to indicate the fact that the certificate is a Qualified Certificate in accordance with a particular legal system.	- (RFC 3739 +-)	+-	+-	+	3.2.6	T25	[1]
	RFC 2560 PRIVATE EXTENSIONS							**RFC 2560**		
16	OCSPNoCheck	{id-pkix-ocsp 5}	A CA specifies by including this extension in the certificate of an OCSP responder that the requester can trust the certificate and need not obtain revocation information.	-	+-	+-	+	4.2.2.2.1	T26	

[1] Notes on criticality:

Common PKI Profile: For the sake of *vertical interoperability*, these extension SHOULD NOT be marked critical, in spite of the fact that their contents restrict the usability of the certificate in some way. This is definitively a deviation from the criticality principle followed by PKIX documents. The main goal of this *recommendation* is to allow successful verification of signed documents and certificates outside the Common PKI application group. An EE who receives a document carrying a qualified electronic signature, is supposed to be interested primarily in reading the document and being assured that the signature is valid. The intention of this Common PKI Specification is therefore that the EE is able to verify the signature and the corresponding certificates without error messages or warnings, regardless whether he/she/it uses Common PKI-compliant software or not. It is put in the responsibility of the receiving party to employ appropriate software in critical applications. If the legal validity and all legal circumstances and limitation of the signature are to be proven, that receiving party is required to use Common PKI-compliant software.

This flagging and verification policy contributes to achieving interoperability among different security levels, one of the major objectives of this Common PKI Specification.

2.3.1 Standard Certificate Extensions

Table 11: AuthorityKeyIdentifier and SubjectKeyIdentifier

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**AuthorityKeyIdentifier** ::= SEQUENCE {	An ID identifying the public key (thus possibly several certificates) of the issuing CA.	++/++	+ (RFC n.a.)	4.2.1.1		[1]
2	keyIdentifier [0] IMPLICIT KeyIdentifier OPTIONAL,		++	+		#6	
3	authorityCertIssuer [1] IMPLICIT GeneralNames OPTIONAL,		+-	+		T8	
4	authorityCertSerialNumber [2] IMPLICIT CertificateSerialNumber OPTIONAL }		+-	+		T2.#13	
5	**SubjectKeyIdentifier** ::= KeyIdentifier	An ID identifying (possibly multiple) user certificates that contain a specific public key.	++/+	+ (RFC n.a.)	4.2.1.2	#6	[2]
6	**KeyIdentifier** ::= OCTET STRING						

[1] [RFC5280]: *AuthorityKeyIdentifier* MUST be included in all CA and end entity certificates to facilitate chain building. (The only exception is a self-signed CA certificate where *authorityKeyIdentifier.keyIdentifier = subjectKeyIdentifier*).
There are two methods to identify the public key:
a) by putting the *subjectKeyIdentifier* of the issuing CA in the *keyIdentifier* field (*keyIdentifier* MUST contain same ID as the *subjectKeyIdentifier* of the CA certificate)
b) by putting the DName of the issuing CA (as present in the *issuer* field of the of the corresponding CA certificate) and the serial number of the corresponding CA certificate in the fields *authorityCertIssuer* and *authorityCertSerialNumber*.
Note that the information provided by method b) uniquely identifies the certificate rather than the public key.
Both identification methods MAY be used in the same certificate.
Common PKI Profile: We stress that the *keyIdentifier* field MUST contain exactly the same ID as the *subjectKeyIdentifier* of the CA certificate (see [2] below).
If *authorityCertIssuer* is present, it MUST contain exactly one *directoryName* element filled with the *subject* DName of the issuing CA certificate.

[2]	[RFC5280]: To facilitate chain building, this extension MUST be included in all CA certificates and SHOULD be formed using one of the following methods: (a) composed of the SHA-1 hash of the value of the BIT STRING *subjectPublicKey* (excluding tag, length and unused bits!) (b) composed of the bits '0100' followed by the least significant 60 bits of the SHA-1 hash of the value of the BIT STRING *subjectPublicKey* (as above) (c) by a method that generates unique values, e.g. from a monotonically increasing integer sequence *SubjectKeyIdentifier* SHOULD be included in all end user certificates and SHOULD be derived from the public key using method a, or b, **Common PKI Profile:** Similarly to CA certificates, CRL issuers' certificates MUST contain *SubjectKeyIdentifier*. Legacy systems may have built the SHA-1 hash value even in another way, by hashing the BIT STRING excluding tag and length, but including the unused bits. Hence, we stress that processing applications SHOULD NOT assume that the key identifier has been formed using one or the other specific method.

Table 12: KeyUsage

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**KeyUsage** ::= BIT STRING {	Defines the purpose of the key pair (public and private key) corresponding to the public key contained in the certificate	++ (RFC +-)	++ (RFC n.a.)	4.2.1.3		[1]
2	digitalSignature (0),	signature verification for purpose other than (1), (5) and (6) (e.g. authentication, integrity check)	+-	++			
3	contentCommitment (1),	signature expressing the signer's commitment to the semantic content of the signed data	+-	++			[3]
4	keyEncipherment (2),	encryption for the purpose of key transport	+-	++			[2]
5	dataEncipherment (3),	data encryption	+-	++			[2]
6	keyAgreement (4),	public key used in a key agreement protocol (e.g. Diffie-Hellmann)	+-	++			
7	keyCertSign (5),	verification of a signature over a certificate (may be set only in CA certificates)	+-	++			
8	crlSign (6),	verification of a signature over a CRL	+-	++			
9	encipherOnly (7),	if the keyAgreement bit is set, the public key may only be used to encrypt data	+-	++			
10	decipherOnly (8) }	if the keyAgreement bit is set, the public key may only be used to decrypt data	+-	++			

[1]	[RFC5280]: This extension MAY be included in certificates and, when present, SHOULD be marked critical. There are no further constraints regarding the usage of individual flags. [ETSI-CPN]: Key usage in end entity certificates for natural persons is restricted to one of the following settings: contentCommitment bit set (A), contentCommitment and digitalSignature bits set (B), digitalSignature bit set (C), digitalSignature and keyEncipherment bits set (D), and keyEncipherment bit set (E). Qualified end entity certificates are limited to types A, B or C. For certificates to be used to validate commitment to signed content, such as electronic signatures on agreements and/or transactions, ETSI RECOMMENDS type A settings only. **Common PKI Profile:** This extension MUST always be included in CA and end entity certificates and MUST be marked critical. The following restrictions apply for: • CA certificates: the *keyCertSign* bit MUST be set. Additionally, the *crlSign* bit MAY be set too, if the CA uses the corresponding key to sign CRLs too. Other bits MUST NOT be set. • CRL signer certificate: Only the *crlSign* bit MUST be set in the certificate of an instance signing (so-called *indirect*) CRLs of certificates which are issued by another CA instance. • OCSP responder certificates: The *crlSign* bit and only this bit MUST be set, if the CA uses the corresponding key to sign CRLs. OCSP responders are issued end-entity certificates with only the *contentCommitment* bit set and including the *ExtendedKeyUsage* extension with only the *id-kp-OCSPSigning* option (see Table 21). • TSP certificates: Time stamping authorities are issued end-entity certificates with only the *contentCommitment* bit set and including the *ExtendedKeyUsage* extension with only the *id-kp-timeStamping* option (see Table 21). • End entity (EE) user certificates (non-qualified): All permitted purposes MUST be stated in end entity certificates, so that client components are able to find the certificate intended for a specific action. In particular, it is RECOMMENDED that CAs issue separate certificates for the purposes of expressing commitment to the signed content (only *contentCommitment* set), authentication (only *digitalSignature* and optional, if required for technical reasons of the intended applications, *keyEncipherment* set) and encryption (only *dataEncipherment* and *keyEncipherment* set). • End entity (EE) qualified certificates (only defined for purposes of electronic signatures): The *contentCommitment* bit and only this bit MUST be set, if these certificates are to be used to validate commitment to signed content, such as electronic signatures on agreements and/or transactions. These certificates MUST NOT be used for other purposes, like authentication or encryption. Note however that the sole indicator whether a certificate is intended to be qualified is not the *KeyUsage* extension but an appropriate *QCStatement* (see Table 25). Compliant CAs MUST issue certificates that are assigned to exactly one of these types (from CA to EE qualified certificates). In this way, relying software is always able to assign the certificate the intended key purpose from the above list. As for the DER-encoding of the BIT STRING value: for the sake of a unique encoding form, the DER-encoding SHOULD be trimmed to the minimal number of octets, i.e. if the *decipherOnly* bit is not set, the BIT STRING value SHOULD be represented on one single octet. Processing components MUST accept any number of value octets.
[2]	Note on implementation: some legacy systems mark encryption certificates of end entities by setting exclusively the *dataEncipherment* bit, other by setting exclusively the *keyEncipherment* bit. Hence, client components SHOULD use the condition *dataEncipherment* OR *keyEncipherment* to recognize encryption certificates.

[3]	In April 2004 the ITU-T working group on X.509 renamed – without affecting its semantics – bit 1 of the *KeyUsage* extension to *contentCommitment* and declared the previous identifier *nonRepudiation* as being deprecated. The semantics of signature related key usage bits was clarified by ITU-T X.509 as follows: • *digitalSignature*: for verifying digital signatures that are used with an entity authentication service, a data origin authentication service or/and an integrity service. • *contentCommitment*: for verifying digital signatures which are intended to signal that the signer is committing to the content being signed. The type of commitment the certificate can be used to support may be further constrained by the CA, e.g. through a certificate policy. The precise type of commitment of the signer e.g. "reviewed and approved" or "with the intent to be bound", may be signalled by the content being signed, e.g. the signed document itself or some additional signed information. Since a content commitment signing is considered to be a digitally signed transaction, the *digitalSignature* bit need not be set in the certificate. If it is set, it does not affect the level of commitment the signer has endowed in the signed content. • *keyCertSign*: for verifying a CA's signature on certificates. Since certificate signing is considered to be a commitment to the content of the certificate by the CA, neither the *digitalSignature* bit nor the *contentCommitment* bit need be set in the certificate. If either (or both) is set, it does not affect the level of commitment the signer has endowed in the signed certificate. **Common PKI Profile:** Both identifiers MAY be treated as synonyms, but in contrast to RFC 5280 the newer name *contentCommitment* SHOULD be used. In order to alleviate end users' burden to differentiate between a declaration of intent on one hand and user or data origin authentication and integrity purposes of a digital signature operation on the other hand, it is RECOMMENDED to include at most one of the *contentCommitment* (for declaration of intent) and *digitalSignature* (for all other purposes) bits in a certificate. If nevertheless both bits are set, the resulting level of commitment MUST be assessed with regard to the *contentCommitment* bit. Note that according to [RFC5246]chapters 7.4.2 and 7.4.6 certificates for TLS authentication may, depending on the specific key and algorithm type used for the applicable TLS cipher suites, may require the *keyEncipherment* or *keyAgreement* key usage bit set in addition to or even instead of the *digitalSignature* bit.

Table 13: (obsolete)

Table 14: CertificatePolicies

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**CertificatePolicies ::=** SEQUENCE SIZE (1..MAX) OF PolicyInformation	a non-empty list of policy terms	+-	++	4.2.1.4	#2	[1]
2	**PolicyInformation** ::= SEQUENCE {						[2]
3	policyIdentifier CertPolicyId,	an OID representing the policy				#5	
4	policyQualifiers SEQUENCE SIZE(1..MAX) OF PolicyQualifierInfo OPTIONAL }	a non-empty list of policy qualifiers	+-	++		#6	
5	**CertPolicyId** ::= OBJECT IDENTIFIER						
6	**PolicyQualifierInfo** ::= SEQUENCE {						
7	policyQualifierId PolicyQualifierId,					#12	
8	qualifier ANY DEFINED BY policyQualifierId }						
9	**id-qt** OBJECT IDENTIFIER ::= {id-pkix 2}						
10	**id-qt-cps** OBJECT IDENTIFIER ::= {id-qt 1}	The OID referring to qualifier type *CPSUri*					
11	**id-qt-unotice** OBJECT IDENTIFIER ::= {id-qt 2}	The OID referring to qualifier type *UserNotice*					
12	**PolicyQualifierId** ::= OBJECT IDENTIFIER {id-qt-cps | id-qt-unotice }						
13	**CPSUri** ::= IA5String	An URL pointing to a CPS (Certification Practice Statement)					
14	**UserNotice** ::= SEQUENCE {	This user notice is intended to be displayed for a relying party whenever using this certificate.					
15	noticeRef NoticeReference OPTIONAL,	A reference to a textual statement				#17	
16	explicitText DisplayText OPTIONAL }	A textual statement explicitly written in the certificate				#20	[3]
17	**NoticeReference** ::= SEQUENCE {						
18	organization DisplayText,	Name of an organization				#20	
19	noticeNumber SEQUENCE OF INTEGER }	a number identifying a particular textual statement prepared by the organization					
20	**DisplayText** ::= CHOICE {						[3]
20a	ia5String IA5String (SIZE (1..200)),		+-	++			
21	visibleString VisibleString (SIZE (1..200)),		--	+-			
22	bmpString BMPString (SIZE (1..200)),		--	+-			
23	utf8String UTF8String (SIZE (1..200)) }		+	++			

[1]	[RFC5280]: In an end entity certificate, these policy information terms indicate the policy under which the certificate has been issued and the purposes for which the certificate may be used. In a CA certificate, these policy information terms limit the set of policies for certification paths which include this certificate. When a CA does not wish to limit the set of policies for certification paths which include this certificate, it MAY assert the special policy *anyPolicy*, with a value of { 2 5 29 32 0 }. If this extension is critical, the path validation software MUST be able to interpret this extension (including the optional qualifier), or MUST reject the certificate. The number of policy terms in the list is not limited. **Common PKI Profile:** For the sake of *vertical interoperability*, especially for the successful verification of signed documents and certificates outside the Common PKI application group, the extension SHOULD NOT be marked critical. As Common PKI compliant systems are supposed to employ rather strict security policies, receivers of such documents might assume an "appropriately high" level of security, without recognizing the particular policy. It is the responsibility of the receiving person to employ appropriate software in critical applications that checks the certification policy. A further reason for marking this extension non-critical is that qualified certificates may alternatively be marked in the *QCStatements* extension (see Table 25). Non-Common PKI-compliant client software may recognize those indicators and ignore this extension, without loosing information on the applying policy.
[2]	[RFC5280]: *PolicyInformation* SHOULD only contain an OID. Where an OID alone is insufficient, [RFC5280] strongly recommends using the identifiers defined above.
[3]	[RFC5280]: Conforming CAs SHOULD use the *UTF8String* encoding for *explicitText*, but MAY use *IA5String*. Conforming CAs MUST NOT encode *explicitText* as *VisibleString* or *BMPString*. The *explicitText* string SHOULD NOT include any control characters (e.g., U+0000 to U+001F and U+007F to U+009F). When the *UTF8String* encoding is used, then all character sequences SHOULD benormalized according to Unicode normalization form C (NFC) [NFC].

Table 15: PolicyMappings

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**PolicyMappings ::=** SEQUENCE SIZE (1..MAX) OF SEQUENCE {	A non-empty list of equivalent policies. The issuing CA considers its *issuerDomainPolicy* to be equivalent to the subject CA's *subjectDomainPolicy*.	+-/--	+-	4.2.1.5		
2	issuerDomainPolicy CertPolicyId,				4.2.1.5	T14.#5	
3	subjectDomainPolicy CertPolicyId }				4.2.1.5	T14.#5	

Table 16: SubjectAltNames and IssuerAltNames

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**SubjectAltNames** ::= GeneralNames	Alternative technical names of the subject	+-	+	4.2.1.6	T8	[1] [2]
2	**IssuerAltNames** ::= GeneralNames	Alternative technical names of the issuing CA	+-	+	4.2.1.7	T8	[1] [3]
[1]	[RFC5280]: If the extension present, the *GeneralNames* structure MUST be non-empty. Because the alternative name is bound to the public key, all parts of the alternative name MUST be verified by the issuing CA. Multiple name forms and multiple instances of each name form MAY be included.						
[2]	[RFC5280]: if the alternative name serves as a means for identification of the subject (especially if the *subject* field is empty), the extension MUST be marked as critical. **Common PKI Profile:** Since the *subject* field uniquely identifies the subject, the *SubjectAltNames* extension SHOULD NOT be marked critical by compliant CAs. Compliant CAs MUST publish end entity and CA certificates. It is RECOMMENDED that certificates are downloadable from an LDAP server. The corresponding LDAP-URL, including the DName as described in [RFC4516], MAY then be included in the *SubjectAltNames* extension of the PKCs. FTP- and/or HTTP-URLs pointing to the certificate MAY also be included, if it is accessible via FTP or HTTP, as described in Part 4. This information may be useful to locate other certificates of the EE or CA.						
[3]	**Common PKI Profile:** Compliant CAs MUST publish end entity and CA certificates. It is RECOMMENDED that certificates are downloadable from an LDAP server. The corresponding LDAP-URL, including the DName as described in [RFC4516], MAY then be included in the in the *IssuerAltNames* extension of the issued PKCs. FTP- and/or HTTP-URLs pointing to the certificate MAY also be included, if it is accessible via FTP or HTTP, as described in Part 4.						

Table 17: SubjectDirectoryAttributes

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**SubjectDirectoryAttributes ::=** Attributes	This extension may contain further X.500 attributes of the subject	-/+-	+ (RFC 3739 ++)	4.2.1.8	#2	[1] [2]
2	**Attributes ::=** SEQUENCE SIZE (1..MAX) OF Attribute					#3	
3	**Attribute** ::= SEQUENCE {						
4	type AttributeType					#6	
5	values SET OF AttributeValue }					#7	[2]
6	**AttributeType** ::= OBJECT IDENTIFIER						
7	**AttributeValue** ::= ANY						

[1] [RFC3739]: The PKIX working group has recognized the demand that personal identification data can be stored in a qualified public key certificate or in a separate attribute certificate (e.g. if the PKC should not make this info public). RFC3739 defines a couple of new DName attributes for this purpose (*dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence*). According to RFC3739, these attributes are to be stored in the *SubjectDirectoryAttributes* extension of the public key certificate. RFC3739 explicitly states that new attribute types MAY be included according to local definitions.
[RFC3281] does not mention, where to record data of this kind.
Common PKI Profile: Qualified PKCs MAY include legal identification data of the subject in the *SubjectDirectoryAttributes* extension. The same kind of information MAY be included in attribute certificates as separate attribute (i.e. in the 'attributes' field instead of an extension) but using the same *SubjectDirectoryAttributes* syntax.
The following attributes MAY be inserted by compliant CAs:
Standard attributes: *commonName, surname, givenName, title, postalAddress* (with the address of permanent residence)
RFC3739 attributes: *dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence*,
Common PKI attribute: *nameAtBirth*
Processing components SHOULD be able to recognize this extension/attribute. In addition to the attributes, listed above, they SHOULD be prepared too to receive other attribute types of Table 7 in this extension.

[2] Type of the value is defined by the *type* field. (The '88 syntax of ASN.1 does not allow to indicate this fact.) At least one value is required to be contained in the *SET*.

Table 18: BasicConstraints

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**BasicConstraints** ::= SEQUENCE {	Indicates a CA certificate and defines how deep a certificate may exists below that CA.	++/+-	++	4.2.1.9		[1]
2	ca BOOLEAN DEFAULT FALSE,	ca=TRUE indicates a CA certificate ca=FALSE or a missing ca element indicates an end entity. Note that in the DER encoding the DEFAULT value of a SET or SEQUENCE component SHALL NOT be encoded.					
3	pathLenConstraint INTEGER (0..MAX) OPTIONAL }	only meaningful if ca=TRUE, indicates how many CA certificates may be included in the certification path below this CA. That is, pathLenConstraint=0 indicates that only end entity certificates may follow in the path. If this field does no appear, there is no limit to the path length.					
[1]	[RFC5280] This extension MUST appear as a critical extension in all CA certificates that contain public keys used to validate digital signatures on certificates. This extension MAY appear as a critical or non-critical extension in CA certificates that contain public keys used exclusively for purposes other than validating digital signatures on certificates. Such CA certificates include ones that contain public keys used exclusively for validating digital signatures on CRLs and ones that contain key management public keys used with certificate enrollment protocols. This extension MAY appear as a critical or non-critical extension in end entity certificates. **Common PKI Profile:** This extension MAY appear in end entity certificates and MUST appear in CA certificates. It MUST be marked critical.						

Table 19: NameConstraints

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**NameConstraints** ::= SEQUENCE {	Indicates a name space in a CA certificate, in which all subject names (or subject alternative names) in subsequent certificates of the path shall be located.	+-/--	++	4.2.1.10		[1]
2	permittedSubtrees [0] IMPLICIT GeneralSubtrees OPTIONAL,		+-	++		#4	[1]
3	excludedSubtrees [1] IMPLICIT GeneralSubtrees OPTIONAL }		+-	++		#4	[1]
4	**GeneralSubtrees** ::= SEQUENCE SIZE (1..MAX) OF GeneralSubtree					#5	
5	**GeneralSubtree** ::= SEQUENCE {						
6	base GeneralName,					T8	[2]
7	minimum [0] IMPLICIT BaseDistance DEFAULT 0,		--	-		#9	[2]
8	maximum [1] IMPLICIT BaseDistance OPTIONAL }		--	-		#9	[2]
9	**BaseDistance** ::= INTEGER (0..MAX)						

[1] Inserting this extension in a CA certificates, a CA is able to enforce subordinate CAs to choose names from a special subspace of the directory or of a domain when issuing further certificates.
[RFC5280]: This extension MUST be included only in CA certificates.
Note that RFC5280-compliant client software MUST check naming constraints as described in RFC5280, if this (always critical) extension is present. This requires the capability of matching DNames, email addresses, domain names, URI and IP addresses in client software, while other name forms MAY be ignored by the verification procedure.

[2] [RFC5280]: Syntax and semantics are defined for *GeneralName* forms email address, DNS name, URI, IP address and *directoryName*, where *directoryName* constrains the *subject* field whereas the other ones the *subjectAltNames* field of subordinate certificates. The meaning and format of other forms *otherName*, *ediPartyName*, *registeredID* are not defined in [RFC5280] and MAY be ignored by the path validation procedure (Part 5). Within this profile, the *minimum* and *maximum* fields are not used with any name forms, thus minimum is always zero, and maximum is always absent.

Table 20: PolicyConstraints

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**PolicyConstraints** ::= SEQUENCE {	May be used in CA certificates to constrain path validation in two ways: it can be used to prohibit policy mapping or require that each certificate in a path contain an acceptable policy identifier.	+-/--	++	4.2.1.11		[1]
2	requireExplicitPolicy [0] IMPLICIT SkipCerts OPTIONAL,	Indicates the maximal number of additional certificates that may appear in the path before an explicit policy is required.				#4	
3	inhibitPolicyMapping [1] IMPLICIT SkipCerts OPTIONAL }	Indicates the maximal number of additional certificates that may appear in the path before *policyMapping* is no longer permitted.				#4	
4	**SkipCerts** ::= INTEGER (0..MAX)						
[1]	[RFC5280]: If the extension is present, at least one optional field MUST be given. Note that RFC5280-compliant client software MUST check *PolicyConstraints* as described in RFC5280, if this extension is present and is marked critical.						

Table 21: ExtendedKeyUsage

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**ExtendedKeyUsage** ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId	Indicates purposes for which the key pair (public and private key) corresponding to the public key contained in the certificate can be used, additional to or in place of those in the *KeyUsage* extension.	+-	++ (RFC n.a.)	4.2.1.12	#2	[1]
2	**KeyPurposeId** ::= OBJECT IDENTIFIER	OID designating a single key purpose.					[2]
2a	**anyExtendedKeyUsage** OBJECT IDENTIFIER ::= {2 5 29 37 0}	Any required extended key usage.	+-	+-			[2a]
3	**id-kp** OBJECT IDENTIFIER ::= {id-pkix 3}	Branch for key purposes OIDs.					
4	**id-kp-serverAuth** OBJECT IDENTIFIER ::= {id-kp 1}	TLS Web server authentication Consistent only with *KeyUsage* bits (*digitalSignature* and/or *keyEncipherment*) or *keyAgreement*, depending on the key and algorithm type according to the relevant TLS cipher suites, see [RFC5246] chapter 7.4.2.	+-	+-			
5	**id-kp-clientAuth** OBJECT IDENTIFIER ::= {id-kp 2}	TLS Web client authentication Consistent only with *KeyUsage* bits *digitalSignature* or *keyAgreement*, depending on the key and algorithm type according to the relevant TLS cipher suites, see [RFC5246] chapter 7.4.6.	+-	+-			
6	**id-kp-codeSigning** OBJECT IDENTIFIER ::= {id-kp 3}	Signing downloadable code Consistent only with *KeyUsage* bit: *digitalSignature*	+-	+-			
7	**id-kp-emailProtection** OBJECT IDENTIFIER ::= {id-kp 4}	E-mail protection Consistent only with *KeyUsage* bits: *digitalSignature*, *contentCommitment* and/or (*keyEncipherment* or *keyAgreement*), see [RFC3850] chapter 4.4.2.	+-	+-			
8	**id-kp-timeStamping** OBJECT IDENTIFIER ::= {id-kp 8}	Time stamping Consistent only with *KeyUsage* bits: *contentCommitment*	+-	++			[2b]
9	**id-kp-OCSPSigning** OBJECT IDENTIFIER ::= {id-kp 9}	Signing OCSP responses	+-	++			[3]

[1]	[RFC5280]: If the extension is present, then the certificate MUST only be used for one of the purposes indicated. If multiple purposes are indicated the application need not recognize all purposes indicated, as long as the intended purpose is present. If a certificate contains both a critical key usage field and an extended key usage field, then both fields MUST be processed independently and the certificate MUST only be used for a purpose consistent with both fields. If there is no purpose consistent with both fields, then the certificate MUST NOT be used for any purpose.
[2]	[RFC5280]: Key purposes may be defined by any organization with a need. Object identifiers used to identify key purposes MUST be assigned in accordance with IANA or ITU-T Recommendation X.660. **Common PKI Profile:** Other key purposes than those listed in this table MAY be included in the *ExtendedKeyUsage* extension.
[2a]	[RFC5280]: Certificate using applications MAY require that a particular purpose be indicated in order for the certificate to be acceptable to that application. If a CA includes extended key usages to satisfy such applications, but does not wish to restrict usages of the key, the CA can include the special *keyPurposeID anyExtendedKeyUsage*. If the *anyExtendedKeyUsage* key purpose is present, the extension SHOULD NOT be critical.
[2b]	[RFC3161]: A TSP certificate MUST include the *id-kp-timeStamping* OID and MUST NOT include any other key purposes (see Table 12). This extension MUST be critical.
[3]	[RFC2560]: If an OCSP signer is not identical to the issuer of the certificates whose status is asked for, the certificate signer MUST designate this authority to an *authorized responder* by issuing a certificate for that entity. The responder's certificate MUST include the *id-kp-OCSPSigning* OID in *ExtKeyUsage*. **Common PKI Profile:** An OCSP responder certificate MUST NOT include any other key purposes than *id-kp-OCSPSigning* (see Table 12). The responder's certificate MAY be issued by any trusted authority. Client software MUST NOT rely on the authorization rules, i.e. they MUST accept responder certificates issued by any trusted authorities.

Table 22: CRLDistributionPoints

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN "DIRECT"/ INDIR.CRL	PROC	RFC 5280	CO. PKI	
1	**CRLDistPointSyntax** ::= SEQUENCE SIZE (1..MAX) OF CRLDistributionPoint	Identifies how CRL information to this certificate can be obtained.	+/++ (RFC+)	+	4.2.1.13	#2	[1] [2]
2	**CRLDistributionPoint** ::= SEQUENCE {						
3	distributionPoint [0] EXPLICIT DistributionPointName OPTIONAL,		++/+-	+		#6	[2] [3]
4	reasons [1] IMPLICIT ReasonFlags OPTIONAL,		+-	+		#9	[4]
5	cRLIssuer [2] IMPLICIT GeneralNames OPTIONAL }		--/++	+		T8	[2]
6	**DistributionPointName** ::= CHOICE {						
7	fullName [0] IMPLICIT GeneralNames,	a full DName, URL or similar	+-	+		T8	[5]
8	nameRelativeToCRLIssuer [1] IMPLICIT RelativeDistinguishedName}	a DName relative to *crlIssuer*	+-	+		T5	
9	**ReasonFlags** ::= BIT STRING {						
10	unused (0),						
11	keyCompromise (1),						
12	cACompromise (2),						
13	affiliationChanged (3),						
14	Superseded (4),						
15	cessationOfOperation (5),						
16	certificateHold (6),						
17	privilegeWithdrawn (7),						
18	aACompromise (8) }						

[1] Notes on criticality:
Common PKI Profile: If the directory providing validity information about the certificate may be accessed via OCSP, this extension MUST NOT be marked critical. In other cases, it SHOULD NOT be marked critical, as stated in [RFC5280].

[2] Notes on support:
[RFC5280]: it is RECOMMENDED always to include this extension in certificates.
If no *cRLIssuer* is specified, the CRL MUST be issued by the issuer of the revoked certificates in the CRL. (Otherwise we speak about an *indirect* CRL.)
If the certificate issuer is also the CRL issuer, then the *cRLIssuer* field MUST be omitted and the *distributionPoint* field MUST be present.
Common PKI Profile: Compliant CAs MUST issue CRLs and publish them via an LDAP-server. In addition to the LDAP service, the CA MAY publish CRLs via HTTP for cases, where some targeted clients cannot access the LDAP service (e.g. because of a local firewall policy).
The CDP extension MAY contain more than one CDP. These have to be interpreted as alternatives. If access to a specific CDP fails, clients MAY try to access other alternatives. Delta-CRLs, if present in a CDP, MUST be present at the same location as the complete CRL. In the case of segmented CRLs, all segments MUST be present

	at the CDP. Basically, there are two different types of CRLs: 1) "*direct*" CRL: the CA that issued the certificate issues the corresponding CRLs too. In this case, if the *CRLDistributionPoints* is not included, the CRL MUST be located at the same LDAP node (in the *certificateRevocationLists* attribute) as the CA certificate. If it is located at another LDAP node or in another attribute, the corresponding DName (relative to the CA-node or absolute in the same directory) or LDAP-URL MUST be supplied in the *distributionPoint* field. Following [RFC5280], the *cRLIssuer* field MUST NOT be present in this "direct" case. 2) *indirect* CRLs are issued, i.e. the CRLs are signed with a key different from the key of the CA. In this case, the *CRLDistributionPoints* extension MUST be present and MUST include the *cRLIssuer* field containing the *subject* DName of the CRL-issuer and resp. of its signing certificate. The *distributionPoint* field MAY be present, pointing to the CRL (via a DName relative to the node of the CRL-issuer or absolute in the same directory; or via an URL). If the *distributionPoint* field is absent, the CRL MUST be located at the node of the CRL-issuer (in the *certificateRevocationLists* attribute). For the sake of vertical interoperability, it is RECOMMENDED that conforming applications process indirect CRLs in order to validate the revocation status of certificates. Indirect CRLs are frequently encountered in the domain of qualified certificates, where, however, the preferred mechanism of revocation checking is OCSP instead of CRL checking. Therefore support for indirect CRLs is not REQUIRED for applications adhering to the Common PKI core standard (see the Common PKI SigG profile for requirements on SigG-conforming applications).
[3]	*CHOICE* objects are always *EXPLICIT*ly tagged, independent of the default tagging modus.
[4]	[RFC5280]: If no reasons are specified or only one CRL appears in this extension, the CRL MUST include revocations for all reasons
[5]	[RFC5280]: If this field contains an URL, it MUST be a pointer to the current CRL. Accepted URL formats are described in [RFC5280] Section 4.2.1.7. **Common PKI Profile:** If URL forms are present, the *fullName* field MUST at least contain the LDAP-URL of the LDAP server, including the DName of the node holding the CRL, as specified in [RFC4516]. Optionally, the *fullName* field MAY contain an FTP-URL and/or a HTTP-URL, if the CRL is available via FTP or HTTP. Directory access methods are described in Part 4 (Operational Protocols) of this specification.

Table 22a: InhibitAnyPolicy

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**InhibitAnyPolicy** ::= SkipCerts	Indicates that the special *anyPolicy* OID is not considered an explicit match for other certificate policies except when it appears in an intermediate self-issued CA certificate.	+-/--	+	4.2.1.14		[1]
2	SkipCerts ::= INTEGER (0..MAX)	The value indicates the number of additional non-self-issued certificates that may appear in the path before *anyPolicy* is no longer permitted. For example, a value of one indicates that *anyPolicy* may be processed in certificates issued by the subject of this certificate, but not in additional certificates in the path.					
[1]	[RFC5280]: Conforming CAs MUST mark this extension as critical.						

Table 22b: FreshestCRL

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC 5280	CO. PKI	
1	**FreshestCRL** ::= SEQUENCE SIZE (1..MAX) OF CRLDistributionPoint	This extension (a.k.a. *DeltaCRLDistributionPoint*) identifies how delta CRL information is obtained.	+-/+-	+	4.2.1.15	T22#2	[1]
[1]	[RFC5280]: The same syntax is used for this extension and the *cRLDistributionPoints* extension. The same conventions apply to both extensions. Each distribution point name provides the location at which a delta CRL for the complete CRL pertaining to this certificate can be found.						

2.3.2 PKIX Private Certificate Extensions

Table 23: AuthorityInfoAccess

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**id-pkix** OBJECT IDENTIFIER ::= {1 3 6 1 5 5 7}	PKIX OID			4.2.2		
2	**id-pe** OBJECT IDENTIFIER ::= {id-pkix 1}	OID for PKIX private extensions			4.2.2		
3	**AuthorityInfoAccessSyntax** ::= SEQUENCE SIZE (1..MAX) OF AccessDescription	Contains access information to online validation service and/or to policy information of the CA issuing this certificate.	+-	+ (RFC n.a.)	4.2.2.1	#4	
4	**AccessDescription** ::= SEQUENCE {						
5	accessMethod OBJECT IDENTIFIER,	Indicates the type and format of the access info					
6	accessLocation GeneralName }	Location of the info, usually in form of an URL				T8	
7	**id-ad** OBJECT IDENTIFIER ::= {id-pkix 48}				4.2.2.1		
8	**id-ad-ocsp** OBJECT IDENTIFIER ::= {id-ad 1}	an OID for the case, when accessLocation points to an OCSP service of the issuing CA	+-	++	4.2.2.1		[1]
9	**id-ad-caIssuers** OBEJCT IDENTIFIER ::= {id-ad 2}	an OID for the case, when the referenced information lists CAs that have issued certificates for the issuer of this certificate.	+-	+-	4.2.2.1		[2]
[1]	**Common PKI Profile:** If the CA issuing the certificate offers OCSP service, its URL MUST be contained in this extension. The OCSP server MUST be accessed using HTTP. See also Part4 (Operational Protocols) of this specification.						
[2]	**Common PKI Profile:** Common PKI enforces that the certification path can always be unambiguously determined using information available in a signed document respectively certificate. Hence, there is no need to list issuers of certificates of the CA. It is however allowed to be included, since some software uses the *caIssuer* information to access and retrieve CA certificates.						

Table 23a: SubjectInfoAccess

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**SubjectInfoAccessSyntax** ::= SEQUENCE SIZE (1..MAX) OF AccessDescription	Indicates how to access information and services for the subject of the certificate.	+-	+ (RFC n.a.)	4.2.2.2	T23#4	
2	**id-ad-caRepository** OBJECT IDENTIFIER ::= {id-ad 5}	An OID for the case, when the subject is a CA that publishes certificates it issues in a repository.	+-	++	4.2.2.2	T23#7	[1]
3	**id-ad-timeStamping** OBEJCT IDENTIFIER ::= {id-ad 3}	An OID for the case, when the subject offers timestamping services using the Time Stamp Protocol	+-	+-	4.2.2.2	T23#7	[2]
[1]	[RFC5280]: When the *accessLocation* is a *directoryName*, the information is to be obtained by the application from whatever directory server is locally configured. When the extension is used to point to CA certificates, the entry for the *directoryName* contains CA certificates in the *crossCertificatePair* and/or *cACertificate* attributes as specified in [RFC4523]. Where the information is available via LDAP, the *accessLocation* SHOULD be a *uniformResourceIdentifier*. The LDAP URI [RFC4516] MUST include a <dn> field containing the distinguished name of the entry holding the certificates, MUST include an <attributes> field that lists appropriate attribute descriptions for the attributes that hold the DER encoded certificates or cross-certificate pairs [RFC4523], and SHOULD include a <host> (e.g., <ldap://ldap.example.com/cn=CA, dc=example,dc=com?cACertificate;binary,crossCertificatePair;binary>). Where the information is available via HTTP or FTP, *accessLocation* MUST be a *uniformResourceIdentifier* and the URI MUST point to either a single DER encoded certificate as specified in [RFC2585] or a collection of certificates in a BER or DER encoded "certs-only" CMS message. **Common PKI Profile:** The extension MAY include LDAP, HTTP or FTP URLs if the respective service is offered. Other name forms SHOULD NOT be used.						
[2]	[RFC5280]: Where the timestamping services are available via HTTP or FTP, accessLocation MUST be a *uniformResourceIdentifier*. Where the timestamping services are available via electronic mail, accessLocation MUST be an *rfc822Name*. Where timestamping services are available using TCP/IP, the *dNSName* or *iPAddress* name forms may be used. **Common PKI Profile:** According to the TSP profile defined in Common PKI Part 4, a HTTP URL SHOULD be used. Other name forms SHOULD NOT be used.						

Table 24: BiometricData

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 3739	CO. PKI	
1	**BiometricSyntax** ::= SEQUENCE OF BiometricData		+-	+	3.2.5	#2	
2	**BiometricData** ::= SEQUENCE {						
3	typeOfBiometricData TypeOfBiometricData,					#7	
4	hashAlgorithm AlgorithmIdentifier,	ID of the hash algorithm used to hash the biometric image data				T4	
5	biometricDataHash OCTET STRING,	Instead of storing the entire biometric image in the certificate, only a hash of that image occurs here.					
6	sourceDataUri IA5String OPTIONAL }	An URL to the entire biometric image may be stored here.					
7	**TypeOfBiometricData** ::= CHOICE {						
8	predefinedBiometricType PredefinedBiometricType,					#10	
9	biometricDataId OBJECT IDENTIIFER }						
10	**PredefinedBiometricType** ::= INTEGER { picture(0), handwritten-signature(1) }						[1]
[1]	[RFC3739]: It is RECOMMENDED that biometric data in this extension is limited to information types suitable for human verification, i.e. where the decision of whether the information is an accurate representation of the subject is naturally performed by a person.						

Table 25: Qualified Certificate Statement

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 3739	CO. PKI	
0	**id-qcs** OBJECT IDENTIFIER ::= { id-pkix 11 }				A.1	T23#1	
1	**QCStatements** ::= SEQUENCE OF QSStatement	A non-empty list of statements	+-	+	3.2.6	#2	[1]
2	**QSStatement ::=** SEQUENCE {						[1]
3	**statementId** OBJECT IDENTIFIER,						
4	**statementInfo** ANY DEFINED BY statementId OPTIONAL }						
5	**id-qcs-pkixQCSyntax-v1** OBJECT IDENTIFIER ::= {id-qcs 1}	an OID to be used as *statementId* and indicating conformance with the syntax and semantics defined in [RFC3039]. Refers to type *SemanticsInformation* below.	-	+			
5a	**id-qcs-pkixQCSyntax-v2** OBJECT IDENTIFIER ::= {id-qcs 2}	an OID to be used as *statementId* and indicating conformance with the syntax and semantics defined in [RFC3739]. Refers to type *SemanticsInformation* below.	+-	+			
6	**SemanticsInformation ::=** SEQUENCE {	Data type to be used in conjunction with *id-qcs-pkixQCSyntax-v1*.					[2]
7	semanticsIdentifier OBJECT IDENTIFIER OPTIONAL,	SHALL contain an OID defining semantics for attributes and names in certificate fields.					
8	nameRegistrationAuthorities NameRegistrationAuthorities OPTIONAL }	Registration authority responsible for registration of attributes and names associated with the subject.				#9	
9	**NameRegistrationAuthorities** ::= SEQUENCE SIZE(1..MAX) OF GeneralName	some *registeredID* of the semantics or of a certificate policy may occur here				T8	
10	**id-etsi-qcs** OBJECT IDENTIFIER ::= { 0 4 0 1862 1 }	ETSI ID for qualified statements					
11	**id-etsi-qcs-QcCompliance** OBJECT IDENTIFIER ::= {id-etsi-qcs 1}	an OID to be used as *statementId* and indicating that the certificate has been issued in accordance with the EU-directive [ECDIR] as implemented in the country under which law the issuer CA operates. When inserting this OID, the *statementInfo* field is to be omitted.	+-	+	[ETSI-QC] 5.2.1	#10	
12	**id-etsi-qcs-QcLimitValue** OBJECT IDENTIFIER ::= {id-etsi-qcs 2}	an OID to be used as *statementId* in conjunction with the *QcEuLimitValue* statement below	+-	+	[ETSI-QC] 5.2.2	#10	
13	**QcEuLimitValue** ::= MonetaryValue	This statement limits the value of transactions, for which the certificate can be used.	+-	+	[ETSI-QC] 5.2.2	#14	
14	**MonetaryValue** ::= SEQUENCE {						
15	currency Iso4217CurrencyCode,	ISO 4217 code of the currency					
16	amount INTEGER,	limit value = amount * $10^{exponent}$					

17	`  exponent  INTEGER }`						
18	**`Iso4217CurrencyCode`** `::= CHOICE {`						
19	`  alphabetic PrintableString,`		+	++			
20	`  numeric    INTEGER(1..999) }`		-	++			
21	**`id-etsi-qcs-QcRetentionPeriod`** `OBJECT IDENTIIFER ::= {id-etsi-qcs 3}`	an OID to be used as *statementId* in conjunction with the *QcRetentionPeriod* statement below	+-	+	[ETSI-QC] 5.2.3	#10	
22	**`QcRetentionPeriod`** `::= INTEGER`	CAs or a relevant name registration authority retains external information about the owner of qualified certificates. This information allows identifying the physical person in case of dispute. This statement indicates how many years after the expiry date of the certificate such information will be retained.	+-	+			
23	**`id-etsi-qcs-QcSSCD`** `OBJECT IDENTIIFER ::= {id-etsi-qcs 4}`	an OID to be used as *statementId* and indicating that the CA vouches that the private key associated with the public key in the certificate is stored in an SSCD (Secure Signature Creation Device) according to Annex III of [ECDIR]. When inserting this OID, the *statementInfo* field is to be omitted.	+-	+	[ETSI-QC] 5.2.4	#10	
[1]	**Common PKI Profile:** Based on the argumentation presented for *CertificatePolicies* (Table 14.[1]), the extension SHOULD NOT be marked critical. It is the responsibility of the receiving person, to check the conditions in critical applications.						
[2]	[RFC3739]: At least one of *semanticsIdentifier* and *nameRegistrationAuthorities* must be present.						

Table 26: OCSPNoCheck

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	CO. PKI	
1	**id-ad** OBJECT IDENTIFIER ::= { id-pkix 48 }	Arc for access descriptors			RFC5280		
2	**id-ad-ocsp** OBJECT IDENTIFIER ::= { id-ad 1 }				RFC5280	#1	
3	**id-pkix-ocsp** OBJECT IDENTIFIER ::= { id-ad-ocsp }				4.2.1	#2	
4	**id-pkix-ocsp-nocheck** OBJECT IDENTIIFER **::=** {id-pkix-ocsp 5}				4.2.2.2.1	#3	
5	**OCSPNoCheck ::= NULL**		+-	+	4.2.2.2.1		[1]
[1]	[RFC2560]: OCSP clients need to know how to check that an authorized OCSP responder's certificate has not been revoked. A CA MAY specify that an OCSP client can trust a responder for the lifetime of the responder's certificate, i.e. the client need no CRL information. The CA does so by including the extension *OCSPNoCheck*. This SHOULD be a non-critical extension. The value of the extension should be NULL. CAs issuing such a certificate should realized that a compromise of the responder's key, is as serious as the compromise of a CA key used to sign CRLs, at least for the validity period of this certificate. CA's may choose to issue this type of certificate with a very short lifetime and renew it frequently. **Common PKI Profile:** Compliant OCSP responders SHOULD not use this option, status information on the responder's certificate SHOULD always be available.						

3 Attribute Certificate Format

The format for attribute certificates presented here is compatible to the attribute certificate format v1 as specified in the 1997 X.509 standard [X.509:1997]. The PKIX attribute certificate profile [RFC3281], based on attribute certificate format v2 of X.509 [X.509:2005], has also been considered here. The attributes and extensions defined in [RFC3281] are not yet subject of this version of Common PKI.

An attribute certificate may be issued as a separate document or in conjunction with a particular signature key certificate (the base certificate). In the latter case, the validity of the attribute certificate expires at the end of the validity period of the base certificate at the latest. An attribute certificate can be issued and revoked independently of the corresponding base certificate.

Table 27: AttributeCertificate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 3281	CO. PKI	
1	**AttributeCertificate** ::= SEQUENCE {				4.1		
2	acinfo AttributeCertificateInfo,	the DER-encoding of this “to be signed” part of the data structure will be signed by the CA				T28	
3	signatureAlgorithm AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this certificate				T4	
4	signatureValue BIT STRING }	the signature of the CA represented as BIT STRING					

Table 28: AttributeCertificateInfo

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 3281	CO. PKI	
1	**AttributeCertificateInfo** ::= SEQUENCE {				4.1		
2	version AttCertVersion DEFAULT v1,	Version number of the attribute certificate format				#13	[1]
3	subject CHOICE {	Information identifying the subject of this certificate:					[2] [3]
4	baseCertificateID [0] EXPLICIT IssuerSerial,	- either as a reference to his/her base certificate	+-	++		#14	
5	subjectName [1] EXPLICIT GeneralNames },	- or his/her name	+-	++		T8	
6	issuer GeneralNames,	Name of the issuer of this certificate				T8	[4] [5]
7	signature AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this certificate.				T4	[6]
8	serialNumber CertificateSerialNumber,	Serial number of the certificate				T2.#13	[7]
9	attrCertValidityPeriod AttCertValidityPeriod,	Validity period of the certificate				#18	[8]
10	attributes SEQUENCE OF Attribute,	a list of certificate attributes that the actual "useful" content of the attribute certificate				T17	[9]
11	issuerUniqueID UniqueIdentifier OPTIONAL,	a unique identifier for the issuer, if issuer DName is reused over time	--	+		T2.#17	[10]
12	extensions Extensions OPTIONAL }	Extensions	++	++		T9	
13	**AttCertVersion** ::= INTEGER { v1(0) }	Version number of the attribute certificate format			4.1		[1]
14	**IssuerSerial** ::= SEQUENCE {	A reference to a certificate			4.1		[2]
15	issuer GeneralNames,	Name of the issuer of the certificate				T8	
16	serial CertificateSerialNumber,	Serial number of the certificate				T2.#13	
17	issuerUID UniqueIdentifier OPTIONAL }	Unique ID of the certificate	--	+		T2.#17	[10]
18	**AttCertValidityPeriod** ::= SEQUENCE { notBeforeTime GeneralizedTime, notAfterTime GeneralizedTime }				4.1		[8]

[1] [RFC3281] enforces *v2(1)*
Common PKI Profile: *version = v1(0)* in this profile because of incompatibilities of the data structure in *v1* and resp. *v2* (see [3] and [5]).
Hence, *v2* certificates cannot be processed by client software compliant with previous versions of Common PKI (ISIS-MTT) and therefore only with *v1*.

[2] [RFC3281]: In a general context, the *baseCertificateID* option SHOULD be used. The *baseCertificateId.issuer* field MUST contain exactly one *directoryName* that is identical to the *issuer* DName of the base certificate.
The *baseCertificateId.issuerUniqueID* field MUST be filled exactly then, when the *issuerUniqueID* field of the base certificate is present. In this case unique ID of the base certificate MUST be assigned to *baseCertificateId.issuerUniqueID*.
When the *subjectName* option is used, it SHOULD contain only one name. If a base certificate exist, the *subject* name or, if not present, one *subjectAltName* of the base certificate SHOULD be inserted.

[3]	ATTENTION! Attribute certificate formats v1 and v2 differ at this point: v2 contains a 'holder' field, the syntax of which is not compatible with that of 'subject' in v1.
[4]	[RFC3281]: the *issuer* field MUST contain exactly one *directoryName* with the DName of the issuer. **Common PKI Profile:** Besides containing exactly one *directoryName* element, as required above, *issuer* MAY include (as the *IssuerAltNames* extension is not supported in ACs) further alternative name forms as follows. Compliant CAs MUST publish end entity and CA certificates. It is RECOMMENDED that certificates are downloadable from an LDAP server. The corresponding LDAP-URL, including the DName as described in [RFC4516], MAY then be included in the in the *issuer* field of the issued ACs. FTP- and/or HTTP-URLs pointing to the certificate MAY also be included, if it is accessible via FTP or HTTP, as described in Part 4.
[5]	ATTENTION! Attribute certificate formats v1 andv2 differ at this point: [RFC3281]contains a CHOICE object at this position, the first option of which is compatible with 'issuer'.
[6]	Content must be the same as *signatureAlgorithm* in Table 27.3
[7]	[RFC3281]: The same applies as to the *serialNumber* field of key certificates: the serial number must be a positive integer, not longer than 20 octets ($1 \leq SN < 2^{159}$, MSB=0 indicates the positive sign!). Processing components must be able to interpret such long numbers. The issuer name and the serial number MUST identify a unique certificate. **Common PKI Profile:** The uniqueness requirement is extended to all kind of certificates (i.e. for PKCs as well as ACs). The reason for that is to allow the same CA to issue PKCs as well as ACs (which is the case in current systems) and furthermore to allow the same CRL to contain entries to PKCs as well as to ACs. Note, that [RFC3281] forbids CAs to issue PKCs and ACs at the same time.
[8]	**Common PKI Profile:** Both *GeneralizedTime* fields must be encoded according to the format YYYYMMDDHHMMSSZ.
[9]	**Common PKI Profile:** The attributes field gives information about the certificate holder. The syntax allows attributes to contain a SET OF values, i.e. be multi-valued. In the attributes SEQUENCE, each *attributeType* OID may occur only once. Processing components MUST be able to handle multiple values for all attribute types. The attributes SEQUENCE MUST contain at least one attribute.
[10]	**Common PKI Profile:** *issuerUniqueID* is supposed to contain *subjectUniqueID* of the CA's certificate. Since Common PKI-compliant CA certificates must not use *uniqueIDs*, attribute certificates MUST NOT include *issuerUniqueID* either.
[11]	[RFC3281]: The extensions field generally gives information about the attribute certificate as opposed to information about the certificate holder. **Common PKI Profile:** the same guidelines have been applied while developing this specification.

3.1 Attribute Certificate Attributes

Table 29: An overview of attribute certificate attributes

#	EXTENSION	OID	SEMANTICS	MULTI-VALUED	SUPPORT		REFERENCES		NOTES
					GEN	PROC	RFC	CO. PKI	
	RFC3281 ATTRIBUTES (NOT YET PART OF COMMON PKI)						**RFC 3281**		
1	SvceAuthInfo	{id-aca 1}	This service authentication info identifies the AC holder by a name to a server or service.	Y	--	+-	4.4.1	n.a.	[1]
2	AccessIdentity	{id-aca 2}	Identifies the AC holder to a server or service.	Y	--	+-	4.4.2	n.a.	[1]
3	ChargingIdentity	{id-aca 3}	Identifies the AC holder for charging purposes.	N	--	+-	4.4.3	n.a.	[1]
4	Group	{id-aca 4}	Group membership of the AC holder	N	--	+-	4.4.4	n.a.	[1]
5	Role	{id-at 72}	Role allocation of the AC holder	Y	--	+-	4.4.5	n.a.	[1]
6	Clearance	{2 5 1 5 55}	Clearance information about the AC holder	Y	--	+-	4.4.6	n.a.	[1]
	COMMON PKI PRIVATE ATTRIBUTES								[2]
7	Procuration	{id-commonpki-at 2}	Procuration information	Y	+-	+-	n.a.	T29a	
8	Admission	{id-commonpki-at 3}	Professional information	N	+-	+-	n.a.	T29b	
9	MonetaryLimit	{id-commonpki-at 4}	Monetary limit for transactions. The *QcEuMonetaryLimit* QC statement MUST be used in new certificates in place of the extension/attribute *MonetaryLimit* since January 1, 2004. For the sake of backward compatibility with certificates already in use, components SHOULD support *MonetaryLimit* (as well as *QcEuLimitValue*).	N	--	+-	n.a.	T29c	[3] [4]
10	DeclarationOfMajority	{id-commonpki-at 5}	A declaration of majority	N	+-	+-	n.a.	T29d	
11	Restriction	{id-commonpki-at 8}	Some other restriction regarding the usage of this certificate.	Y	+-	+-	n.a.	T29e	[3]
12	AdditionalInformation	{id-commonpki-at 15}	Some other information of non-restrictive nature regarding the usage of this certificate.	Y	+-	+-	n.a.	T29f	
13	SubjectDirectoryAttributes	{2 5 29 9}	Personal identification data. The *SubjectDirectoryAttributes* syntax is used for this purpose.	N	+-	+-	n.a.	T17	[5]
14	QcEuLimitValue id-etsi-qcs-QcLimitValue	{id-etsi-qcs 2}	Instead of including it in a *QCStatements* extension, a monetary limit MAY be specified in an attribute (not an extension) using this QC statement syntax.	N	+-	+-	n.a.	T25#13	[3] [4]

[1]	These extensions are part of [RFC3281]. **Common PKI Profile:** These extensions are NOT YET PART of the current version of Common PKI.
[2]	These attributes were originally defined in the optional SigG-Profile of Common PKI. Applications conforming to the Common PKI core specification MAY include them in attribute certificates.
[3]	**Common PKI Profile:** Attribute certificates with restrictive content MUST always be included in the signed document.
[4]	**Common PKI Profile:** In new certificates, *MonetaryLimit* MUST be replaced by *QcEuLimitValue*, defined in [ETSI-QC]. Instead of inserting a *QCStatements* extension, CAs MAY choose to specify a monetary limit as an attribute using the *QcEuLimitValue* syntax and the *id-etsi-qcs-QcLimitValue* OID. Note that *QcEuLimitValue* is backward compatible with *MonetaryLimit*. Hence, it sufficient for processing components to implement the *QcEuLimitValue* structure and use it to process any attributes with the *id-etsi-qcs-QcLimitValue* and the *id-commonpki-at-monetaryLimit* OIDs. If both *QcEuLimitValue* and *MonetaryLimit* occur in the same certificate (as extension or attribute), they MUST assert the same value and currency. A certificate SHOULD use only one form.
[5]	**Common PKI Profile:** If an AC should contain personal identification data, they MUST be included in an AC as an attribute (not as an extension).

Table 29a: Procuration

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-procuration** OBJECT IDENTIFIER ::= {id-commonpki-at 2}	OID for extension/attribute *Procuration*			n.a.	T43	
2	**ProcurationSyntax ::=** SEQUENCE {	Attribute to indicate that the certificate holder may sign in the name of a third person	+-	+	n.a.		[1]
3	country [1] EXPLICIT PrintableString(SIZE(2)) OPTIONAL,	indicates the country whose laws apply	+-	++			
4	typeOfSubstitution [2] EXPLICIT DirectoryString (SIZE(1..128)) OPTIONAL,	type of procuration (e.g. manager, procuration, custody)	+-	++		T6	
5	signingFor [3] EXPLICIT SigningFor }					#6	
6	**SigningFor** ::= CHOICE {	Identification of the represented (substituted) person via:					
7	thirdPerson GeneralName,	either his/her name				T8 T7	[2]
8	certRef IssuerSerial }	or a reference to his/her base certificate. The base certificate MUST be a qualified PKC.				T28#14	
[1]	**COMMON PKI PROFILE:** The corresponding *ProcurationSyntax* contains either the name of the person who is represented (subcomponent *thirdPerson*) or a reference to his/her base certificate (in the component *signingFor*, subcomponent *certRef*), furthermore the optional components *country* and *typeSubstitution* to indicate the country whose laws apply, and respectively the type of procuration (e.g. manager, procuration, custody).						
[2]	**COMMON PKI PROFILE:** The *GeneralName* MUST be of type *directoryName* and MAY only contain: - RFC3739 attributes, except *pseudonym* (*countryName, commonName, surname, givenName, serialNumber, organizationName, organizationalUnitName, stateOrProvincename, localityName, postalAddress*) and - *SubjectDirectoryName* attributes (*title, dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence and NameAtBirth*).						

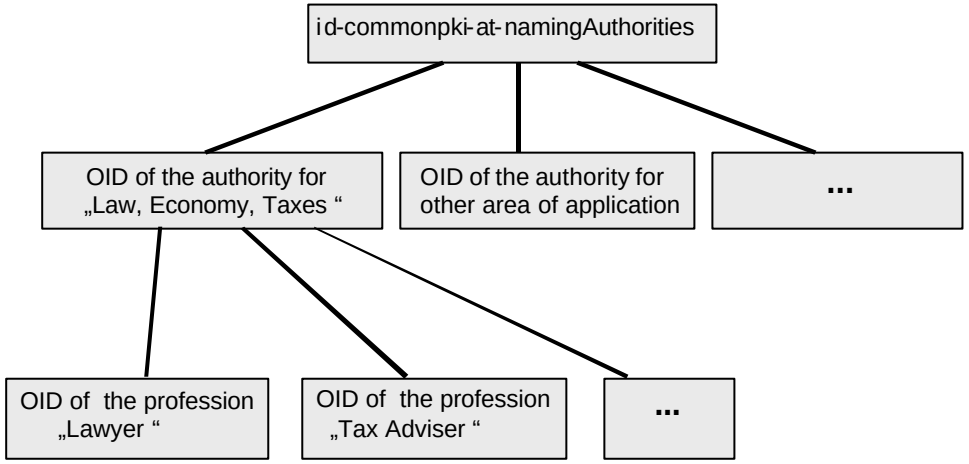
Table 29b: Admission

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-admission** OBJECT IDENTIFIER ::= {id-commonpki-at 3}	OID for extension/attribute *Admission*			n.a.	T43	
2	**id-commonpki-at-namingAuthorities** OBJECT IDENTIFIER ::= {id-commonpki-at 11}				n.a.	T43	
3	**AdmissionSyntax ::=** SEQUENCE {	Attribute to indicate admissions to certain professions	+-	+	n.a.		
4	admissionAuthority GeneralName OPTIONAL,						
5	contentsOfAdmissions SEQUENCE OF Admissions }						[1]
6	**Admissions** ::= SEQUENCE {						
7	admissionAuthority [0] EXPLICIT GeneralName OPTIONAL,					T8	
8	namingAuthority [1] EXPLICIT NamingAuthority OPTIONAL,					#10	
9	professionInfos SEQUENCE OF ProfessionInfo }					#14	
10	**NamingAuthority** ::= SEQUENCE {						
11	namingAuthorityId OBJECT IDENTIFIER OPTIONAL,						
12	namingAuthorityUrl IA5String OPTIONAL,						
13	namingAuthorityText DirectoryString(SIZE(1..128)) OPTIONAL}					T6	
14	**ProfessionInfo** ::= SEQUENCE {						
15	namingAuthority [0] EXPLICIT NamingAuthority OPTIONAL,					#10	
16	professionItems SEQUENCE OF DirectoryString (SIZE(1..128)),					T6	
17	professionOIDs SEQUENCE OF OBJECT IDENTIFIER OPTIONAL,						
18	registrationNumber PrintableString(SIZE(1..128)) OPTIONAL,						
19	addProfessionInfo OCTET STRING OPTIONAL }						
[1]	**COMMON PKI PROFILE:** The relatively complex structure of *AdmissionSyntax* supports the following concepts and requirements: • External institutions (e.g. professional associations, chambers, unions, administrative bodies, companies, etc.), which are responsible for granting and verifying professional admissions, are indicated by means of the data field *admissionAuthority*. An admission authority is indicated by a *GeneralName* object. Here an X.501 directory name (*distinguished name*) can be indicated in the field *directoryName*, a URL address can be indicated in the field *uniformResourceIdentifier*, and an object identifier can be indicated in the field *registeredId*. • The names of authorities which are responsible for the administration of title registers are indicated in the data field *namingAuthority*. The name of the authority can be identified by an object identifier in the field *namingAuthorityId*, by means of a text string in the field *namingAuthorityText*, by means of a URL address in the field *namingAuthorityUrl*, or by a combination of them. For example, the text string can contain the name of the authority, the country and the name of the title register. The URL-option refers to a web page which contains lists with „officially" registered professions (text and possibly OID) as well as further information on these professions. Object identifiers for the component *namingAuthorityId* MAY be grouped under the OID-branch *id-commonpki-at-namingAuthorities* and MAY be applied for by interested authorities.						

See http://www.teletrust.de/fileadmin/files/oid/oid_Antrag.pdf for an application form and http://www.teletrust.de/index.php?id=524 for an overview of registered naming authorities.
However a naming authority is NOT REQUIRED to register under the OID *id-commonpki-at-namingAuthorities* in order to define profession OIDs.

- By means of the data type *ProfessionInfo* certain professions, specializations, disciplines, fields of activity, etc. are identified. A profession is represented by one or more text strings, resp. profession OIDs in the fields *professionItems* and *professionOIDs* and by a registration number in the field *registrationNumber*. An indication in text form MUST always be present, whereas the other indications are optional. The component *addProfessionInfo* MAY contain additional application-specific information in DER-encoded form.

By means of different *namingAuthority*-OIDs or profession OIDs hierarchies of professions, specializations, disciplines, fields of activity, etc. can be expressed as illustrated as a possible example in the figure below. The issuing admission authority SHOULD always be indicated (field *admissionAuthority*), whenever a registration number is presented. Still, information on admissions MAY be given without indicating an admission or a naming authority by the exclusive use of the component *professionItems*. In this case the certification authority is responsible for the verification of the admission information.



This attribute is *single-valued*. Still, several admissions can be captured in the sequence structure of the component *contentsOfAdmissions* of *AdmissionSyntax* or in the component *professionInfos* of *Admissions*.
The component *admissionAuthority* of *AdmissionSyntax* serves as default value for the component *admissionAuthority* of *Admissions*. Within the latter component the default value can be overwritten, in case that another authority is responsible.
The component *namingAuthority* of *Admissions* serves as a default value for the component *namingAuthority* of *ProfessionInfo*. Within the latter component the default value can be overwritten, in case that another naming authority needs to be recorded.

The length of the string objects is limited to 128 characters. It is RECOMMENDED to indicate a *namingAuthorityURL* in all issued attribute certificates. If a *namingAuthorityURL* is indicated, the field *professionItems* of *ProfessionInfo* SHOULD contain only registered titles. If the field *professionOIDs* exists, it has to contain the OIDs of the professions listed in *professionItems* in the same order. In general, the field *professionInfos* SHOULD contain only one entry, unless the

	admissions that are to be listed are logically connected (e.g. they have been issued under the same admission number).

Table 29c: MonetaryLimit

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-monetaryLimit** OBJECT IDENTIFIER ::= {id-commonpki-at 4}	OID for extension/attribute *MonetaryLimit*			n.a.	T43	
2	**MonetaryLimitSyntax ::=** SEQUENCE {	Indicates a monetary limit within which the certificate holder is authorized to act. (This value DOES NOT express a limit on the liability of the certification authority).	+-	++	n.a.		
3	currency PrintableString (SIZE(3)),	ISO code					
4	amount INTEGER,	value = amount•$10^{exponent}$					
5	exponent INTEGER }						

Table 29d: DeclarationOfMajority

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-declarationOfMajority** OBJECT IDENTIFIER ::= {id-commonpki-at 5}	OID for extension/attrib. *DeclarationOfMajority*			n.a.	T43	
2	**DeclarationOfMajoritySyntax** ::= CHOICE {		+-	++	n.a.		
3	notYoungerThen [0] IMPLICIT INTEGER,	indicates a minimum age					[1]
4	fullAgeAtCountry [1] IMPLICIT SEQUENCE {	indicates the majority of the owner with respect to the laws of a specific country					
5	fullAge BOOLEAN DEAULT TRUE,	majority age reached in that country					
6	country PrintableString (SIZE(2)) }	ISO code of that country					
7	dateOfBirth [2] IMPLICIT GeneralizedTime }	date of birth of the certificate owner					[1]
[1]	**COMMON PKI PROFILE:** In the field *notYoungerThan* any age can be specified. In the coding of *dateOfBirth* the format YYYYMMDD000000Z has to be applied.						

Table 29e: Restriction

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-restriction** OBJECT IDENTIFIER ::= {id-commonpki-at 8}	OID for extension/attrib. *Restriction*			n.a.	T43	
2	**RestrictionSyntax** ::= DirectoryString (SIZE(1..1024))	Text indicating some other restriction regarding the usage of this certificate.	+-	++	n.a.	P1.T6	

Table 29f: AdditionalInformation

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-additionalInformation** OBJECT IDENTIFIER ::= {id-commonpki-at 15}	OID for extension/attrib. *AdditionalInformation*			n.a.	T43	
2	**AdditionalInformationSyntax** ::= DirectoryString (SIZE(1..2048))	Text indicating some other information (of non-restrictive nature) regarding the usage of this certificate.	+-	++	n.a.	P1.T6a	

3.2 Attribute Certificate Extensions

Table 30: An overview of attribute certificate extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT		REFERENCES		NOTES
					GEN	PROC	RFC 3281	CO. PKI	
	X.509 BASIC EXTENSIONS								
1	AuthorityKeyIdentifier	{2 5 29 35}	An ID identifying the public key (thus possibly several certificates) corresponding to the signing private key of the issuing CA.	-- (RFC n.a.)	++ (RFC +)	+	4.3.3	T11	[1]
2	CertificatePolicies	{2 5 29 32}	Indicates the policy under which the certificate has been issued and the purposes for which it is to be used.	+-	+-	++	n.a.	T14	[1]
3	CRLDistributionPoints	{2 5 29 31}	Identifies how CRL information to this certificate can be obtained.	- (RFC n.a.)	+/++ dir/ind. CRL (RFC +-)	+	4.3.5	T22	[1]
	RFC5280 PRIVATE EXTENSIONS								
4	AuthorityInfoAccess	{id-pe 1}	Access to online validation service and/or policy information of the CA issuing this certificate.	--	+-	+	4.3.4	T23	[1]
	RFC3739 QC PRIVATE EXTENSIONS								
5	QCStatements	{id-pe 3}	A statement to indicate that the certificate is a Qualified Certificate in accordance with a particular legal system.	-	+-	+	n.a.	T25	[1]
	RFC3281 AC PRIVATE EXTENSIONS								
6	AuditIdentity	{id-pe 4}	A server/service administrator uses this ID to track the behavior of an AC holder, without getting his identity.	(RFC ++)	--	-	4.3.1	n.a.	[2]
7	Targets	{2 5 29 55}	Name of a servers/services, the AC is intended for.	(RFC n.a.)	--	-	4.3.2	n.a.	[2]
8	NoRevAvail	{2 5 29 56}	Indicates that no revocation information will be available for the AC	(RFC --)	--	-	4.3.6	n.a.	[2]
[1]	[RFC3281]: Not all of these extensions are part of [RFC3281]. *AuthorityKeyIdentifier*, *CRLDistributionPoints* and *AuthorityInfoAccess* are supported in order "to assist the AC verifier in checking the signature of the AC." **Common PKI Profile:** Besides *AuthorityKeyIdentifier*, *CRLDistributionPoints* and *AuthorityInfoAccess*, the extensions *CertificatePolicies* and *QCStatements* are supported in this profile. These extensions allow the path validation procedure (see Part 5) to handle ACs in the same way as PKCs. The same criticality and support requirements as well as comments apply for these extensions as in PKCs. Refer to the corresponding tables !								
[2]	**Common PKI Profile:** At the moment, these RFC3281extensions are not yet part of this specification.								

4 CRL Format

Table 31: CertificateList (CRL)

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**CertificateList** ::= SEQUENCE {				5.1.1		
2	tbsCertList TBSCertList,	the DER-encoding of this “to be signed” part of the data structure will be signed by the CA			5.1.1.1	T32	
3	signatureAlgorithm AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this CRL			5.1.1.2	T4	
4	signatureValue BIT STRING }	the signature of the CA represented as BIT STRING			5.1.1.3		

Table 32: TBSCertList

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**TBSCertList** ::= SEQUENCE {						
2	version Version OPTIONAL,	Version number of the CRL format	++	++	5.1.2.1	T2.#12	[1] [8]
3	signature AlgorithmIdentifier,	an identifier of the signature algorithm used by the CA to sign this CRL.			5.1.2.2	T4	[2]
4	issuer Name,	DName of the issuer of this CRL			5.1.2.3	T5	[3]
5	thisUpdate Time,	Date and time when this CRL was issued			5.1.2.4	T3	[4]
6	nextUpdate Time OPTIONAL,	Date and time when the next CRL will be issued	++	++	5.1.2.5	T3	[4] [5] [8]
7	revokedCertificates SEQUENCE OF SEQUENCE {	List of revoked certificates, the "useful" content of the CRL	+-	++	5.1.2.6		[6]
8	userCertificate CertificateSerialNumber,	Serial number of the revoked certificate			5.1.2.6		
9	revocationDate Time,	Date and time at which the certificate was revoked			5.1.2.6	T3	[4]
10	crlEntryExtensions Extensions OPTIONAL	A non-empty list of extensions describing the revoked cert.			5.3	T37	[7]
11	} OPTIONAL,						
12	crlExtensions [0] EXPLICIT Extensions OPTIONAL }	A non-empty list of CRL extensions	++	++	5.2	T33	

[1]	[RFC5280]: *version* MUST be v2(1), if any extensions present in *crlEntryExtensions* or in *crlExtensions*. Since RFC5280 enforces the presence of extension *CRLNumber*, this is always the case. **Common PKI Profile:** conforming to RFC5280,only *v2(1)* CRLs MUST be issued.
[2]	Content must be the same as *signatureAlgorithm* in Table 31.3.
[3]	[RFC5280]: The same constraints apply as for the *issuer* field of key certificates. See Table 2.[4]
[4]	[RFC5280]: The same constraints apply as for the *validity* field of key certificates. See Table 3.[1] The revocation date SHOULD NOT precede the issue date of earlier certificates.
[5]	[RFC5280]: The optional field *nextUpdate* MUST be included in all CRLs. It indicates the date by which the next CRL will be issued. For technical reasons (it takes some time to create the CRL), the next CRL MAY be issued before the indicated date, but MUST NOT be issued any later. CAs should issue CRLs with a *nextUpdate* time equal to or later than all previous CRLs.
[6]	This optional field may be omitted, if there are no revoked certificates
[7]	[RFC5280]: If a CRLcontains a critical CRL entry extension that the application cannot process, then the application MUST NOT use that CRL to determine the status of any certificates. However, applications may ignore unrecognized non-critical CRL entry extensions.
[8]	[RFC5280]: When CRLs are issued, the CRLs MUST be version 2 CRLs, include the date by which the next CRL will be issued in the *nextUpdate* field, include the *CRLnumber* extension, and include the *AuthorityKeyIdentifier* extension.

4.1 CRL Extensions

Table 33: An overview of CRL extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT		REFERENCES		NOTES
					GEN "DIRECT"/ INDIR.CRL	PROC	RFC 5280	CO. PKI	
	X.509 BASIC EXTENSIONS								
1	AuthorityKeyIdentifier	{2 5 29 35}	An ID identifying the public key (thus possibly several certs) corresponding to the signing private key of the issuing CA.	-- (RFC n.a.)	++/++	+	5.2.1	T11	[1] [2]
2	IssuerAltNames	{2 5 29 18}	Alternative technical names of the issuing CA: email, DNS name, IP address, URI	-	-/+- (RFC n.a.)	+	5.2.2	T16.#2	[2]
3	CRLNumber	{2 5 29 20}	Number of the CRL	--	++/++	++	5.2.3	T34	
4	DeltaCRLIndicator	{2 5 29 27}	Indicates that the CRL is a delta-CRL, i.e. contains only entries of the current complete CRL that are not present in a previous complete CRL, the base CRL.	++	+-/+-	++	5.2.4	T35	
5	IssuingDistributionPoint	{2 5 29 28}	Indicates whether the CRL covers revocations for end entity certificates only, for CA certificates only or for a limited set of reason codes and whether it is an indirect CRL.	++	+-/++	+	5.2.5	T36	
6	FreshestCRL	{2 5 29 46}	This extension (a.k.a. *DeltaCRLDistributionPoint*) identifies how delta CRL information is obtained.	--	+-/+-	+	5.2.6	T36a	
7	AuthorityInfoAccess	{id-pe 1}	Access to online validation service and/or policy information of the CA issuing this CRL.	--	+-/+-	+	5.2.7	T36b	

[1] **Common PKI Profile:** The *crlSign*-Flag in the *KeyUsage* extension MUST be set in all CA- or end-entity certificates, that correspond to CRL-signing keys. Issuers of indirect CRLs typically posses an end-entity certificate.

[2] **Common PKI Profile:** As readily described in T22.[2], there are two types of CRLs:

1) "*direct*" CRL: the CA that issued the certificate issues the corresponding CRLs too. This situation can be recognized by relying software if the following conditions apply:
   a. if the *CRLDistributionPoints* extension is missing from the certificate or
   b. it is present, but the *cRLIssuer* field is missing.
2) *indirect* CRL: the CRLs are signed with a key different from the key of the CA. This situation can be recognized by relying software if the *CRLDistributionPoints* extension is present in the certificate and the *cRLIssuer* field holds a DName (different from the *subject* of the CA certificate). Additionally, indirect CRLs MUST include an *IssuingDistributionPoint* extension with *indirectCRL* flag set to true.

So that relying software can locate the certificate of the issuer of an indirect CRL, *AuthorityKeyIdentifier* MUST and *IssuerAltNames* MAY be included in indirect CRLs. The *IssuerAltNames* extension MAY contain the LDAP-URL of the node that holds the CRL-signer's certificate.

Table 34: CRLNumber

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**id-ce-cRLNumber** OBJECT IDENTIFIER ::= { id-ce 20 }	OID to be used in conjunction with extension *CRLNumber*			5.2.3		
2	**CRLNumber** ::= INTEGER (0..MAX)	Syntax of extension *CRLNumber*			5.2.3		[1]
[1]	[RFC5280]: CRLs MUST be assigned numbers of a monotonically increasing sequence. This extension allows easily determining whether a particular CRL supersedes another one. **[Common PKI PROFILE]**: [RFC5280] does not constrain the value or the length of this field. Similarly to *CertificateSerialNumber*, a maximal length of 20 octets will be defined here, i.e. $0 \leq CRLNumber < 2^{159}$ (MSB=0 indicates the positive sign!). Processing components MUST be able to work with such long numbers.						

Table 35: DeltaCRLIndicator

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**id-ce-deltaCRLIndicator** OBJECT IDENTIFIER ::= { id-ce 27 }	Indicates that the CRL is a delta-CRL, i.e. contains only entries of the current complete CRL that are not present in a previous complete CRL, the base CRL. Using a complete CRL and all subsequent delta-CRLs, the relying component is able to continuously maintain a local instance of subsequent complete CRLs.			5.2.4		
2	**BaseCRLNumber** ::= CRLNumber	Syntax of extension *DeltaCRLIndicator*			5.2.4	T34.#2	[1]
[1]	[RFC5280]: The CRL issuer MAY also generate delta CRLs. A delta CRL only lists those certificates, within its scope, whose revocation status has changed since the issuance of a referenced complete CRL. The referenced complete CRL is referred to as a base CRL. The scope of a delta CRL MUST be the same as the base CRL that it references. Conforming applications are not required to support processing of delta CRLs .						

Table 36: IssuingDistributionPoint

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**id-ce-issuingDistributionPoint** OBJECT IDENTIFIER ::= { id-ce 28 }				5.2.5		
2	**IssuingDistributionPoint** ::= SEQUENCE {	Syntax of extension *IssuingDistributionPoint*. Indicates whether the CRL covers revocations for end entity certificates only, for CA certificates only or for a limited set of reason codes.			5.2.5		[1]
	distributionPoint [0] EXPLICIT DistributionPointName OPTIONAL,	If the CRL is stored in an X.500 directory, it will be stored under the entry indicated by this field and which may be different from the directory entry of CA signing the CRL.				T22.#2	[2] [3]
	onlyContainsUserCerts [1] IMPLICIT BOOLEAN DEFAULT FALSE,	Set if CRL contains only end entity certificates					
	onlyContainsCACerts [2] IMPLICIT BOOLEAN DEFAULT FALSE,	Set if CRL contains only end CA certificates					
	onlySomeReasons [3] IMPLICIT ReasonFlags OPTIONAL,	CAs may use this flag to partition their CRL according to the reason of revocation, e.g. on the basis of compromise or routine revocation.				T22..#9	
	indirectCRL [4] IMPLICIT BOOLEAN DEFAULT FALSE,	Indicates that the CRL is an indirect one, i.e. the CRL issuer is not the same entity as the issuer of (some of) the certificates listed in the CRL.					
	onlyContainsAttributeCerts [5] IMPLICIT BOOLEAN DEFAULT FALSE }	Indicates that the CRL only contains revoked attribute certificates.					

[1]	[RFC5280]: It is the decision of the CA whether it issues delta-CRLs. When a CA issues a delta-CRL, it MUST also issue a corresponding complete CRL (the current complete CRL). The delta-CRL and the complete CRL MUST have the same *CRLNumber*.
[2]	CHOICE objects are always *EXPLICIT*ly tagged, independent of the default tagging modus.
[3]	[RFC5280]: If an URL is given, it MUST point to the most current CRL issued by this CA. The URL schemes *ftp*, *http* [RFC1738] [RFC3986], *mailto* [RFC2368] and *ldap* [RFC4516] are defined for this purpose. The URI MUST be an absolute, not relative, pathname and MUST specify the host.

Table 36a: FreshestCRL

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**FreshestCRL** ::= SEQUENCE SIZE (1..MAX) OF CRLDistributionPoint	This extension (a.k.a. *DeltaCRLDistributionPoint*) identifies how delta CRL information is obtained.	+-	+	5.2.6	T22#2	[1]
[1]	[RFC5280]: This extension MUST NOT appear in delta CRLs. The same syntax is used for this extension and the *cRLDistributionPoints* extension. The same conventions apply to both extensions. Each distribution point name provides the location at which a delta CRL for this complete CRL can be found.						

Table 36b: AuthorityInfoAccess

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**AuthorityInfoAccessSyntax** ::= SEQUENCE SIZE (1..MAX) OF AccessDescription	Indicates how to access information and services for the subject of the certificate.	+-	+	5.2.7	T23#4	
2	**id-ad-caIssuers** OBEJCT IDENTIFIER ::= {id-ad 2}	An OID for the case, when the referenced information lists CAs that have issued certificates for the issuer of this CRL.	++	+	4.2.2.1	T23#8	[1]
[1]	[RFC5280]: When present in a CRL, this extension MUST include at least one *AccessDescription* specifying *id-ad-caIssuers* as the *accessMethod*. The *id-ad-caIssuers* OID is used when the information available lists certificates that can be used to verify the signature on the CRL Access method types other than *id-ad-caIssuers* MUST NOT be included.						

4.2 CRL Entry Extensions

Table 37: An overview of CRL entry extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT		REFERENCES		NOTES
					GEN "DIRECT"/ INDIR.CRL	PROC	RFC 5280	CO. PKI	
	BASIC EXTENSIONS								
1	ReasonCode	{2 5 29 21}	Reason for the certificate revocation	--	+-	+-	5.3.1	T38	[1]
2	HoldInstructionCode	{2 5 29 23}	A registered instruction identifier indicating the action to be taken when the certificate that has been placed on hold.	--	--	-	[RFC 3280] 5.3.2	T39	[3]
3	InvalidityDate	{2 5 29 24}	Indicates the date on which it is known or suspected that the private key became compromised or the certificate otherwise became invalid.	--	+-	+-	5.3.2	T40	[1]
4	CertificateIssuer	{2 5 29 29}	Used in indirect CRLs to indicate the issuer of the revoked certificate, if it is different from the issuer of the CRL.	++	-/++	++	5.3.3	T41	[2]
[1]	[RFC5280]: Conforming CA's SHOULD include these extensions if such information is available.								
[2]	[RFC5280]: Indirect CRLs MUST include the *CertificateIssuer* extension in CRL entries. "Direct" CRLs SHOULD NOT include this extension.								
[3]	The *HoldInstructionCode* extension is no longer supported in [RFC5280].								

Table 38: ReasonCode

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	`id-ce-cRLReasons OBJECT IDENTIFIER ::= { id-ce 21 }`	OID of the *ReasonCode* extension			5.3.1		
2	`CRLReason ::= ENUMERATED {` `  unspecified (0),` `  keyCompromise (1),` `  cACompromise (2),` `  affiliationChanged (3),` `  superseded (4),` `  cessationOfOperation (5),` `  certificateHold (6),` `  removeFromCRL (8),` `  privilegeWithdrawn (9),` `  aACompromise (10) }`	Reason for the certificate revocation			5.3.1		[1]
[1]	[RFC5280]: CAs are strongly encouraged to include meaningful reason codes. However, if no such information is available, the *ReasonCode* extension SHOULD be absent, instead of giving the code *unspecified*(0). **Common PKI Profile:** If during the revocation of a certificate a key compromise cannot be excluded with sufficient probability, the CA SHALL set the reason code to *keyCompromise* (resp.*cACompromise* or *aACompromise*), so that the reason code *unspecified* or an absent reason code can be treated as "unknown, but key compromise can be excluded with sufficient probability".						

Table 39: HoldInstructionCode

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 3280	CO. PKI	
1	**id-ce-holdInstructionCode** OBJECT IDENTIFIER ::= {id-ce 23}	OID of the *HoldInstructionCode* extension	--	-	5.3.2		[2]
2	HoldInstruction ::= OBJECT IDDENTIFIER	Syntax of the *HoldInstructionCode* extension			5.3.2		
3	holdInstruction OBJECT IDENTIFIER ::= {1 2 840 10040 2 }						
4	id-holdInstruction-none OBJECT IDENTIFIER ::= {holdInstruction 1}	No action specified.	--	-			[1]
5	id-holdinstruction-callissuer OBJECT IDENTIFIER ::= {holdInstruction 2}	Conforming applications MUST call the issuer or reject the certificate.	--	-			
6	id-holdinstruction-reject OBJECT IDENTIFIER ::= {holdInstruction 3}	Conforming applications MUST reject the certificate.	--	-			
[1]	[RFC3280]: The extension MUST be absent from the CRL rather than indicating the *id-holdInstruction-none* code, which is semantically the same.						
[2]	The *HoldInstructionCode* extension is no longer specified in [RFC5280].						

Table 40: InvalidityDate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 5280	CO. PKI	
1	**id-ce-invalidityDate** OBJECT IDENTIFIER ::= { id-ce 24 }	OID of the *InvalidityDate* extension			5.3.2		[2]
2	**InvalidityDate** ::= GeneralizedTime	Syntax of the *InvalidityDate* extension			5.3.2		[1]
[1]	[RFC5280]: The same constraints apply as for the validity field of PKCs. See Table 3.[1]						
[2]	[RFC5280]: This extension provides the date on which it is known or suspected that the private key was compromised or that the certificate otherwise became invalid. This date may be earlier than the revocation date in the CRL entry, which is the date at which the CA processed the revocation.						

Table 41: CertificateIssuer

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN "DIRECT"/ INDIR.CRL	PROC	RFC 5280	CO. PKI	
1	**id-ce-certificateIssuer** OBJECT IDENTIFIER ::= { id-ce 29 }	OID of the CertificateIssuer extension			5.3.3		
2	**CertificateIssuer** ::= GeneralNames	Syntax of the CertificateIssuer extension			5.3.3	T8	[1]
[1]	[RFC5280]: If this extension is not present on the first entry of an indirect CRL, the certificate issuer defaults to the CRL issuer. If this extension is not present in a subsequent entry, the certificate issuer defaults to the issuer of the preceding entry. Practically, an indirect CRL SHOULD be sorted according to the issuers of the entries. **Common PKI Profile:** the *GeneralNames* value MUST contain exactly one *directoryName* item with the *subject* DName in the certificate of the issuing CA.						

5 Cross Certificates

A CA may issue a cross certificate for another CA to allow users of certificates subordinate to the other CA to verify certificates subordinate to the issuing CA. Accordingly, the cross certificate will be stored in the directory entry of the other CA. The directory attribute that stores one or more cross certificates is called *crossCertificatePair* and uses the syntax *CertificatePair* specified in Table 42 below. Note that directory attribute *crossCertificatePair* may have several values, e.g. several certificate pairs.

Table 42: Cross Certificate Pair

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	X.509:05	CO. PKI	
1	**CertificatePair ::=** SEQUENCE {		+-	++	Chap. 11.2.3		[1]
2	issuedToThisCA [0] EXPLICIT Certificate OPTIONAL,		++	++		T1	
3	issuedByThisCA [1] EXPLICIT Certificate OPTIONAL }		+-	+		T1	
[1]	[X.509:2005]: The *issuedToThisCA* elements of the *crossCertificatePair* attribute of a CA's directory entry SHALL store all, except self-issued certificates issued to this CA. Optionally, the *issuedByThisCA* elements of the *crossCertificatePair* attribute, of a CA's directory entry MAY contain a subset of certificates issued by this CA to other CAs. When both the *issuedToThisCA* and the *issuedByThisCA* elements are present in a single attribute value, *issuer* name in one certificate shall match the *subject* name in the other and vice versa, and the subject public key in one certificate shall be capable of verifying the digital signature on the other certificate and vice versa. When a *issuedByThisCA* element is present, the *issuedToThisCA* element value and the *issuedByThisCA* element value need not be stored in the same attribute value; in other words, they can be stored in either a single attribute value or two attribute values. The term *forward* was used in previous editions for *issuedToThisCA* and the term *reverse* was used in previous editions for *issuedByThisCA*. In the case of V3 certificates, none of the above CA certificates shall include a *BasicConstraints* extension with the *cA* value set to *FALSE*. **Common PKI Profile:** The *issuer* and respectively *subject* DNames MUST be identical, in order to allow client components to use simple matching rules in chain building (exact match).						

6 Common PKI Object Identifiers

The following table lists ASN.1 object identifiers introduced in the Common PKI Specification. The *id-commonpki* branch of the OID tree was previously known under the name *id-isismtt* and before that under the name *id-sigi*, the name but not the meaning has been changed in this version.

Table 43: Common PKI Object Identifiers

#	ASN.1 DEFINITION		SEMANTICS	SUPPORT		REFERENCES		NOTES
				GEN	PROC	RFC	CO. PKI	
1	**id-commonpki**	OBJECT IDENTIFIER ::= {1 3 36 8 }		+-	+-	n.a.		
2	**id-commonpki-cp**	OBJECT IDENTIFIER ::= {id-commonpki 1}	Branch for certificate policies	+-	+-	n.a.		
3	**id-commonpki-at**	OBJECT IDENTIFIER ::= {id-commonpki 3}	Branch for attributes and extensions	+-	+-	n.a.		
4	**id-commonpki-at-certHash**	OBJECT IDENTIFIER ::= {id-commonpki-at 13}	OID of an OCSP extension	+-	+-	n.a.	P4.T15	
5	**id-commonpki-at-nameAtBirth**	OBJECT IDENTIFIER ::= {id-commonpki-at 14}	OID of a DName attribute	+-	+-	n.a.	P1.T7	
6	**id-commonpki-at-procuration**	OBJECT IDENTIFIER ::= {id-commonpki-at 2}		+-	+	n.a.	P1.T29a	
7	**id-commonpki-at-admission**	OBJECT IDENTIFIER ::= {id-commonpki-at 3}		+-	+	n.a.	P1.T29b	
8	**id-commonpki-at-monetaryLimit**	OBJECT IDENTIFIER ::= {id-commonpki-at 4}		+-	+	n.a.	P1.T29c	
9	**id-commonpki-at-declarationOfMajority**	OBJECT IDENTIFIER ::= {id-commonpki-at 5}		+-	+	n.a.	P1.T29d	
10	**id-commonpki-at-restriction**	OBJECT IDENTIFIER ::= {id-commonpki-at 8}		+-	+	n.a.	P1.T29e	
11	**id-commonpki-at-namingAuthorities**	OBJECT IDENTIFIER ::= {id-commonpki-at 11}	Branch for registering naming authorities of *Admission* attributes	+-	+-	n.a.	P1.T29b	[1] [2]
12	**id-commonpki-at-additionalInformation**	OBJECT IDENTIFIER ::= {id-commonpki-at 15}		+-	+	n.a.	P1.T29f	

[1]	See http://www.teletrust.de/fileadmin/files/oid/oid_Antrag.pdf for an application form and http://www.teletrust.de/index.php?id=524 for an overview of registered naming authorities.
[2]	At the time of this writing, profession OIDs for the German health care system are defined in the OID sub tree under (1 2 276 0 76 4), see http://www.dimdi.de/dynamic/de/ehealth/oid/verzeichnis.html.

References

[ECDIR]	Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community Framework for Electronic Signatures
[ETSI-CPN]	ETSI TS 102 280 v1.1.1 (2004-03) : X.509 V.3 Certificate Profile for Certificates Issued to Natural Persons
[ETSI-QC]	ETSI TS 101 862 v1.3.3 (2006-01): Qualified Certificate profile
[ETSI-TSP]	ETSI TS 101 861 v1.3.1 (2006-01): Time Stamping Profile
[ISIS]	Industrial Signature Interoperability Specification ISIS, Version 1.2, December 1999, T7 i.Gr., www.t7-isis.de
[MTTv2]	MailTrusT Version 2, March 1999, TeleTrust Deutschland e.V., www.teletrust.de
[NFC]	Davis, M. and M. Duerst, "Unicode Standard Annex #15: Unicode Normalization Forms", October 2006, http://www.unicode.org/reports/tr15/
[RFC1034]	Domain Names – Concepts and facilities, November 1987
[RFC1630]	Universal Resource Identifiers in WWW, June 1994
[RFC1738]	Uniform Resource Locators (URL), December 1994
[RFC2247]	Using Domains in LDAP/X.500 Distinguished Names, January 1998
[RFC2368]	The mailto URL scheme, July 1998
[RFC2460]	Internet Protocol, Version 6 (IPv6) Specification, December 1998
[RFC2560]	X.509 Internet Public Key Infrastructure Online Certificate Status Protocol -OCSP, June 1999
[RFC2822]	Internet Message Format, April 2001
[RFC3039]	Internet X.509 Public Key Infrastructure Qualified Certificates Profile, January 2001
[RFC3161]	Internet X.509 Public Key Infrastructure - Time Stamp Protocol (TSP), August 2001
[RFC3279]	Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002
[RFC3280]	Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile, April 2002
[RFC3281]	An Internet Attribute Certificate Profile for Authorization, April 2002
[RFC3490]	Internationalizing Domain Names in Applications (IDNA), March 2003
[RFC3629]	UTF-8, a transformation format of ISO 10646, November 2003
[RFC3739]	Internet X.509 Public Key Infrastructure: Qualified Certificates Profile, March 2004
[RFC3850]	Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 – Certificate Handling, July 2004
[RFC3986]	Uniform Resource Identifier (URI): Generic Syntax, January 2005

[RFC4510] Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, June 2006

[RFC4516] Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator, June 2006

[RFC4519] Lightweight Directory Access Protocol (LDAP): Schema for User Applications, June 2006

[RFC4523] Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates, June 2006

[RFC5246] The Transport Layer Security (TLS) Protocol Version 1.2, August 2008

[RFC5280] Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile, May 2008

[RFC791] Internet Protocol – DARPA Internet Program Protocol Specification (v4), September 1981

[X.509:1997] ITU-T X.509: Information Technology - Open Systems Interconnection – The Directory: Authentication Framework, 1997

[X.509:2005] ITU-T X.509: Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, 2005

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 2

PKI MANAGEMENT

VERSION 2.0 – 20 JANUARY 2009

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.

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.1 June 26th 2002	Originally, it was planned to use CMP for PKI management. However, the Board has taken the decision to withdraw the prepared draft version for PKI management based on CMP and to follow an alternative approach based on CMC. First published version based on CMC
1.0.2 July 19th 2002	Editorial and stylistic changes, and removal of bugs
1.0.2 August 11th 2003	Incorporated all changes from Corrigenda version 1.2
1.1 March 16th 2004	Several editorial changes.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Adapted to new versions of the base standards: - RFC 2986 - RFC 3851 - RFC 3852 - RFC 4211 - RFC 5272 - RFC 5273 - RFC 5274 Various corrections and clarifications.

Table of Contents

1 Preface

This part of the Common PKI specification addresses online communication between PKI components. It defines a profile for Common PKI components that is mainly based on the Internet document "Certificate Management Messages over CMS (CMC)" [RFC5272], [RFC5273] and [RFC5274], and on the following standards:

- "Cryptographic Message Syntax" [RFC3852],
- "Internet X.509 Certificate Request Message Format" [RFC 4211],
- "PKCS#10: Certification Request Syntax" [RFC2314][1],
- "PKCS#7: Cryptographic Message Syntax" [RFC2315], and
- "S/MIME Version 3.1 Message Specification" [RFC3851].

CMC defines two variants of PKI management protocols. These are called:

- simple enrollment protocol, and
- full enrollment protocol.

The current version of this part of the Common PKI specification does only consider conformance requirements for the simple enrollment protocol that MUST be supported by compliant Common PKI end entities (EEs) and certification authorities (CAs).

Items of the referenced standards that are not explicitly mentioned in this specification SHALL be treated in the same way as specified in the referenced base standards.

Conformance requirements that Common PKI compliant components MUST satisfy, are specified in the following chapter.

[1] Although [RFC2314] was obsoleted by [RFC2986], CMC [RFC5272] still references the older [RFC2314].

2 Simple Enrollment Protocol

The simple enrollment protocol is composed of a simple enrollment request sent from the EE to the CA, and a simple enrollment response returned from the CA to the EE.

The related data objects that are exchanged are a PKCS#10 [RFC2314] certification request data object, and a PKCS#7 [RFC2315] certification response (degenerated *signedData* CMS [RFC3852]) data object.

2.1 Protocol Elements

2.1.1 PKCS#10 Certification Request Data Object

The type for the PKCS#10 certification request is defined by the ASN.1 type *CertificationRequest, which is a sequence of the fields,* listed in Table 1.

2.1.2 PKCS#7 Certification Response Data Object

The PKCS#7 certification response is a CMS data object, whose general syntax is defined by the ASN.1 type *ContentInfo* with the content type *signed-data,* and whose *encapContent* and *signerInfos* fields MUST be absent. The field *certificates* SHALL contain all certificates of the certification path.

The type for *signed-data* is defined by the ASN.1 type *SignedData,* which is a sequence of the fields listed in Table 2.

Table 1: Fields of *CertificationRequest*

FIELDS			REFERENCES				COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	TABLE	GEN	PROC	VALUES	
1	*certificationRequestInfo*	DER encoded certification request information to be signed	RFC 2314 RFC 5272	6.2 3.3.1	++		++EE	++CA		
1.1	version	Version number	RFC 2314	6.1	++		++EE	++CA	v1(0)	
1.2	*subject*	DName of EE	RFC 2314	6.1	++		++EE	++CA	.	[1]
1.3	*subjectPublicKeyInfo*	Information about the public key being certified	RFC 2314	6.1	++		++EE	++CA		[2]
1.4	*attributes*	Set of attributes providing additional information about the subject of the certificate	RFC 2314 RFC 5272	6.1 3.3.1	+-		+-EE	++CA		[5]
1.4.1	*ExtensionReq*	Attribute that allows to incorporate one or more standard X.509v3 extensions within the PKCS#10 request	RFC 5272	3.3.1	+-		+-EE	++CA	OID: 1.2.840.113549.1.9.14	[3]
2	*signatureAlgorithm*	Identifier of the signature algorithm used by the EE to sign this request	RFC 2314	6.2	++		++EE	++CA		[4]
3	*signature*	Signature of the EE calculated over *certificationRequestInfo, and* represented as BIT STRING	RFC 2314	6.2	++		++EE	++CA		

[1]	For permitted distinguished names in *subject* refer to P1.T2.#7 (Certificate and CRL Profiles) of this Common PKI specification. [RFC5272]: This field MAY contain the value NULL, but MUST be present. **Common PKI Profile:** This field MUST be present with a valid NON-NULL value. CAs that receive a *CertificationRequest* with a NULL *subject* name SHALL reject the request, and no response MAY be returned.
[2]	For further requirements concerning *subjectPublicKeyInfo* refer to P1.T2.#14 (Certificate and CRL Profiles) of this Common PKI specification.
[3]	The OID *id-ExtensionReq* identifies this attribute: For permitted extension in the *ExtensionReq* attribute refer to P1.T10 (Certificate and CRL Profiles) of this Common PKI specification.
[4]	For permitted algorithm identifiers refer to Part 6 (Cryptographic Algorithms) of this Common PKI specification.
[5]	According to the syntax defined in [RFC2314] and [RFC5272], the generating application MUST encode an empty SET element, if no attributes are included in the request. **Common PKI Profile**: The processing application SHOULD be prepared that the whole *attributes* element might be omitted by faulty generating applications if no attributes are included in the request.

Table 2: Fields of *ContentInfo* for Certification Responses

FIELDS			REFERENCES				COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	TABLE	GEN	PROC	VALUES	
1	*contentType*	Indication of the type of content	RFC 3852 RFC 2315	3 7	++		++CA	++EE	OID: 1.2.840.113549.1.7.2	[1]
2	*content*	Content of signed-data	RFC 3852 RFC 2315	5.1 9.1	++		++CA	++EE		
2.1	*version*	Version number of CMS syntax	RFC 3852 RFC 2315	5.1 9.1	++		++CA	++EE	1 3	[2]
2.2	*digestAlgorithms*	Collection (including zero) of message digest algorithm identifiers	RFC 3852 RFC 2315	5.1 9.1	++		++CA	++EE		[3]
2.3	*encapContentInfo* *contentInfo*	Data to be protected	RFC 3852 RFC 2315 RFC 5272	5.2 9.1 4.3	 --		 --CA	 ---EE		[4]
2.4	*certificates*	Collection of certificates	RFC 3852 RFC 2315	5.1 9.1	-+		+-CA	+-EE		[5]
2.5	*crls*	Collection of CRLs	RFC 3852 RFC 2315	5.1 9.1	-+		-+CA	-+EE		
2.6	*signerInfos*	Collection of per-signer information	RFC 3852 RFC 2315 RFC 5272	5.1 9.1 4.3	 --		 --CA	 --EE		[4]

[1]	The OID *id-signedData* identifies signed-data.
[2]	Compliant components SHALL always use the value 1, since non-interpreted binary data shall be protected.
[3]	For permitted hash algorithm identifiers refer to P6.S2.1 (Cryptographic Algorithms) of this Common PKI specification.
[4]	This field MUST be absent.
[5]	Compliant components SHOULD include all certificates of the certification path(s) of the signer(s) required by the recipient.

2.2 PKI Messages

2.2.1 PKCS#10 Messages

Compliant EE components MUST support the generation of plain PKCS#10 messages, to be sent to the related CAs.

Compliant CA components MUST support the processing of plain PKCS#10 messages received from EEs.

2.2.2 PKCS#7 Messages

Compliant CA components MUST support the generation of PKCS#7 messages, to be sent to the related EEs.

Compliant EE components MUST support the processing of PKCS#7 messages received from CAs.

2.3 Transport

2.3.1 Transport Mechanisms

Compliant components MAY implement any of the transport mechanisms defined in [RFC5273].

2.3.2 Simple Enrollment Requests

Compliant EE components MUST support the MIME message type *application/pkcs10* for transporting the PKCS#10 certification request objects to the related CAs. The parameter *filename* with the file extension ".p10" MUST be included either in the *Content-Type*, or in the *Content-Disposition* MIME header line.

Compliant CA components MUST support the processing of MIME messages of the type *application/pkcs10*, received from EEs.

2.3.3 Simple Enrollment Responses

Compliant CA components MUST support the message type *application/pkcs7-mime* together with the *smime-type* parameter set to the value *certs-only* for transporting certificates in certification responses.

The related CMS object to be inserted into the resulting *application/pkcs7-mime* MIME entity MUST be of the CMS content type *signed-data* (see Table 2) whose *encapContent* and *signerInfos* fields MUST be absent. The field *certificates* MUST contain all certificates of the certification path. The parameter *filename* with the file extension ".p7c" SHALL be included either in the *Content-Type*, or in the *Content-Disposition* MIME header line.

Compliant EE components MUST support the processing of *certs-only* MIME messages, received from EEs.

Annexes

Annex A: ASN.1 Definitions

This annex contains a list of ASN.1 definitions in alphabetic order that have been used in this part of the Common PKI specification.

```
Attribute ::=                       SEQUENCE {
     attrType                       OBJECT IDENTIFIER,
     attrValues                     SET OF AttributeValue }

Attributes ::=                      SET OF Attribute

AttributeValue ::                   =ANY

CertificateChoices ::=              CHOICE {
     certificate                    Certificate,
     extendedCertificate            [0] IMPLICIT ExtendedCertificate,
     v1AttrCert                     [1] IMPLICIT AttributeCertificateV1,
     v2AttrCert                     [2] IMPLICIT AttributeCertificateV2,
     other                          [3] IMPLICIT OtherCertificateFormat }

CertificateRevocationLists ::=      SET OF CertificateList

CertificateSet ::=                  SET OF CertificateChoices

CertificationRequest ::=            SEQUENCE {
     certificationRequestInfo       CertificationRequestInfo,
     signatureAlgorithm             SignatureAlgorithmIdentifier,
     signature                      Signature}

CertificationRequestInfo ::=        SEQUENCE {
     version                        Version,
     subject                        Name,
     subjectPublicKeyInfo           SubjectPublicKeyInfo,
     attributes                     [0] IMPLICIT Attributes}

CMSVersion ::=                    INTEGER {v0(0), v1(1), v2(2), v3(3), v4(4)}

ContentInfo ::=                     SEQUENCE {
     contentType                    ContentType,
     content                        [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::=                     OBJECT IDENTIFIER

DigestAlgorithmIdentifier ::=  SET OF AlgorithmIdentifier

DigestAlgorithmIdentifiers::=  SET OF DigestAlgorithmIdentifier

EncapsulatedContentInfo ::= SEQUENCE {
     eContentType                   ContentType,
     eContent                       [0] EXPLICIT OCTET STRING OPTIONAL }

id-ExtensionReq OBJECT IDENTIFIER ::=
     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9) 14}

id-signedData OBJECT IDENTIFIER ::=
     {iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }

Signature ::=                       BIT STRING

SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
```

```
SignatureValue ::=              OCTET STRING
SignedData ::=          SEQUENCE {
    version             CMSVersion,
    digestAlgorithms    DigestAlgorithmIdentifiers,
    encapContentInfo    EncapsulatedContentInfo,
    certificates        [0] IMPLICIT CertificateSet OPTIONAL,
    crls                [1] IMPLICIT CertificateRevocationLists OPTIONAL,
    signerInfos         SignerInfos }
Version ::=                     INTEGER
```

Annex B: Abbreviations

ASN.1	abstract syntax notation one
CA	certification authority
CMC	certificate management messages over CMS
CMS	cryptographic message syntax
CRL	certificate revocation list
DER	distinguished encoding rules
EE	end entity
ISIS	industrial signature interoperability specification
MIME	multipurpose internet mail extension
MTT	MailTrusT
PKI	public key infrastructure
S/MIME	Secure MIME

References

[RFC2314] B. Kaliski: PKCS#10: Certification Request Syntax; October 1997

[RFC2315] B. Kaliski: PKCS#7: Cryptographic Message Syntax; October 1997

[RFC2986] PKCS #10: Certification Request Syntax Specification Version 1.7, November 2000

[RFC3851] Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, July 2004

[RFC3852] Cryptographic Message Syntax (CMS), July 2004

[RFC4211] Internet X.509 Public Key Infrastructure Certificate Request Message Format (CRMF), September 2005

[RFC5272] Certificate Management over CMS (CMC), June 2008

[RFC5273] Certificate Management over CMS (CMC): Transport Protocols, June 2008

[RFC5274] Certificate Management over CMS (CMC): Compliance Requirements, June 2008

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 3

CMS BASED MESSAGE FORMATS

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.1 November 15th 2001	First published version
1.0.2 July 19th 2002	Editorial and stylistic changes, and removal of bugs
1.0.2 August 11th 2003	Incorporated all changes from Corrigenda version 1.2
1.1 March 16th 2004	Several editorial changes. The most relevant changes affecting technical aspects are: 1) OPTIONAL use of the older (experimental) MIME message types is permitted. 2) *SigningTime* MUST be encoded as UTCTime until 2049. 3) Unsigned attributes have been added. 4) RFC 3369 has been taken into account.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Renamed to "CMS based Message Formats"- Adapted to new versions of the base standards: - RFC 3850 - RFC 3851 - RFC 3852 - RFC 5035 - ETSI TS 101 733 v1.7.4 Various corrections and clarifications.

Table of Contents

1 Preface

This part of the Common PKI specification addresses message formats to be used during the exchange of data between PKI components. It defines a profile for Common PKI message formats that is mainly based on the Internet documents for S/MIME [RFC 3851], MIME [RFC 2045, RFC 2046], and CMS [RFC 3851].

Items of the referenced standards that are not explicitly mentioned in this specification shall be treated in the same way as specified in the referenced base standards.

This document contains the following chapters:

- Chapter 2 contains requirements for message formats based on S/MIME.
- Chapter 3 lists data structures to be used in S/MIME messages.
- Chapter 4 specifies requirements for file formats for signature and encryption.
- Annex A provides the CMS ASN.1 definitions in alphabetic order.
- References chapter lists the standards on which this part of Common PKI is based.

2 Message Types Based on S/MIME

S/MIME messages allow combining MIME bodies and protected message parts, the latter being constructed accordingly to CMS. Several different MIME types and CMS objects MAY be used in an S/MIME message.

S/MIME supports a variety of message types. Arbitrary MIME messages or parts of a MIME message can be secured by means of digital signatures and encryption. This process can be iterated allowing any level of nesting.

Compliant components SHALL support the Common PKI profile for S/MIME, which is specified in the following sections.

2.1 S/MIME Message Types

Message types are identified by a MIME header field. The type of each MIME message is defined by its *Content-Type* field and an optional set of parameters. The *Content-Type* consists of a media type and a subtype that specify the particular format. [RFC 2045, RFC 2046]. So far the media types

- *text* for textual messages,
- *image* for audio data,
- *video* for video data,
- *application* for all other kinds of data, as for example non-interpreted binary data,
- *multipart* for multiple different data types, and
- *message* for encapsulated messages have been defined by MIME. The last two being designed for composed messages.

CMS objects consist of a content type and the content, which contains the data. Compliant components SHALL support the content types *signed-data* and *enveloped-data* that indicate that the message is protected either by digital signature or by encryption.

S/MIME specifies several message types for encrypted and signed messages. The minimum requirement for compliant components is the support of the following two S/MIME message types:

- *application/pkcs7-mime* for encrypted and signed messages, and
- *multipart/signed* together with *application/pkcs7-signature* for signed messages with separate data and control information in two body parts.

For the sake of interoperability with existing S/MIME products, compliant components MAY alternatively use the older (experimental) message type *application/x-pkcs7-mime* and *application/x-pkcs7-signature* in place of *application/pkcs7-mime* respectively *application/pkcs7-signature* and SHOULD accept these older message types.

Common PKI Profile: For interoperability backward compatibility with older S/MIME applications, header protection through the use of the *message/rfc822* MIME type as described in [RFC 3851] chapter 3.1 SHOULD NOT be used by sending applications. Since

however applying header protection increases security, it is not entirely forbidden.

2.1.1 Message Type for Enveloped Data

Compliant components SHALL support the message type *application/pkcs7-mime* together with the *smime-type* parameter set to the value *enveloped-data* for protecting the confidentiality of any kind of MIME messages.

Compliant components SHALL support the transformations including preparation of MIME entity for encryption, canonicalization, encryption, encoding and composition as specified in S/MIME [RFC 3851, chapter 3].

The canonicalization transformation step can be omitted, if the data are already in a format that can be uniquely interpreted by the recipient. Compliant components SHALL perform the canonicalization step for those content types for which a unique presentation independent of the platform or the environment does not exist. This is for example required for *text* data.

The transfer encoding step can be omitted, if an 8-bit-transparent transportation medium is used, or if S/MIME is used for purposes other than Internet-Mail. Compliant components SHALL perform the transfer encoding step if the message shall always be transported via Internet-Mail.

The related CMS object to be inserted into the resulting application/pkcs7-mime MIME entity SHALL be of the CMS content type *enveloped-data* (see 3.3).

2.1.2 Message Type for Signed Data

Compliant components SHALL support the message type *application/pkcs7-mime* together with the *smime-type* parameter set to the value *signed-data* for protecting the authentication and integrity of arbitrary non clear-signing data. The protected object can be any MIME message.

Compliant components SHALL support the transformations including preparation of MIME entity for signing, canonicalization, signature creation, encoding and composition as specified in [RFC 3851, chapter 3].

The canonicalization transformation step can be omitted, if the data are already in a format that can be uniquely interpreted by the recipient. Compliant components SHALL perform the canonicalization step for those content types for which a unique presentation independent of the platform or the environment does not exist. This is for example required for text data.

The transfer encoding step can be omitted, if an 8-bit-transparent transportation medium is used or if S/MIME is used for purposes other then Internet-Mail. Compliant components SHALL perform the transfer encoding step if the message shall always be transported via Internet-Mail. Transfer encoding, if used, has to comprise the complete message, including the header fields.

The related CMS object to be inserted into the resulting application/pkcs7-mime MIME entity SHALL be of the CMS content type *signed-data* (see 3.2).

2.1.3 Message Type for Certificates-Only Messages

Compliant components SHALL support the message type *application/pkcs7-mime* together with the *smime-type* parameter set to the value *certs-only* for transporting certificates in certification responses.

The related CMS object to be inserted into the resulting application/pkcs7-mime MIME entity SHALL be of the CMS content type *signed-data* (see 3.2) whose *encapContent* and *signerInfos* fields must be absent. The field *certificates* (see 3.2) SHALL at least contain the signer's certificate, and MAY contain all certificates of the certification path.

NOTE

Compliant components SHALL support the MIME message type *application/pkcs10* for transporting the corresponding PKCS#10 objects in certification requests.

2.1.4 Message Type for Signed Data With Multipart Encoding

Compliant components SHALL support the message type *multipart/signed* for protecting the authentication and integrity of arbitrary clear-signing data when multipart encoding applies. The protected object can be any MIME message.

Compliant components SHALL support the transformations including preparation of MIME entity for signing, canonicalization, signature creation, encoding and composition as specified in [RFC 3851, chapter 3].

The canonicalization transformation step can be omitted, if the data are already in a format that can be uniquely interpreted by the recipient. Compliant components SHALL perform the canonicalization step for those content types for which a unique presentation independent of the platform or the environment does not exist. This is for example required for *text* data.

The transfer encoding step can be omitted, if an 8-bit-transparent transportation medium is used or if S/MIME is used for purposes other than Internet-Mail. Compliant components SHALL perform the transfer encoding step if the message shall always be transported via Internet-Mail. Transfer encoding, if used, has to comprise the complete message, including the header fields.

The MIME entity to be signed has to be inserted into the first part of the multipart/signed message. The second part of the multipart/signed message SHALL contain a MIME entity of type *application/pkcs7-signature* which in turn is a CMS object of type *SignedData* (see 3.2) with absent *encapContentInfo.eContent* field.

2.1.5 Message Type for Compressed-Only Messages

Common PKI Profile: Compressed-only S/MIME messages are not considered by the Common PKI specification.

2.2 S/MIME Message Transformations

Compliant components SHALL support the MIME transformations defined in [RFC 3851, chapter 3] that are required to create an S/MIME message with the following exception during the preparation of MIME objects.

Common PKI does not recommend to perform the transfer encoding independent of the transportation medium in order to avoid any unnecessary expansion of data, and to reduce the number of decoding steps required to determine the message type of a received message with multiple encoding. Instead, it is recommended to omit the encoding step, if it is not required.

Compliant components that perform transfer encoding SHALL indicate the used transfer encoding variant (identity, "quoted-printable", or "base64") in the MIME header *Content-Transfer-Encoding*.

Compliant components SHALL use the following MIME header lines for the transformation *composition*, during which CMS objects are inserted into the MIME message:

MIME HEADER LINES FOR ENCRYPTED OR SIGNED OBJECTS

- *Content-Type* including the parameter *name*,
- *Content-Transfer-Encoding*, if applicable, and
- *Content-Disposition* including the parameter *filename* with the file extension ".p7m" for enveloped-data and signed-data CMS objects. The extension ".p7c". SHALL be used for certs-only messages (and ".p10" for PKCS#10 objects).

MIME HEADER LINES FOR MULTIPART SIGNED OBJECTS

- *Content-Type* including the parameters *protocol*, *micalg (sha1, sha256, sha384, sha512, md5* or *unknown)*, and *boundary*,
- *Content-Transfer-Encoding*, if applicable, and
- *Content-Disposition* including the parameter `filename` with the file extension ".p7s" for signed-data CMS objects with absent *encapContentInfo.eContent* field.

3 Data Structures in S/MIME Messages

3.1 Summary of Conformance Requirements

Compliant components SHALL support the data structures *signed-data* and *enveloped-data* as defined in CMS [RFC3852] including all related substructures.

DATA STRUCTURE SIGNED-DATA

Within the data structure *signed-data* compliant components SHALL support the attributes mandated by CMS, and the attribute *signing-time* also defined by CMS. The *signing-time* attribute can be contained either in the *signedAttrs* or *unsignedAttrs* fields.

The *signing-time* attribute SHALL be used with the alternative *GeneralizedTime.*

The support of further attributes is recommended.

DATA STRUCTURE ENVELOPED-DATA

Within the data structure *enveloped-data* compliant components SHALL use the *version* field with the value 0.

Compliant components SHALL NOT use the optional *originatorInfo* field.

Compliant components SHALL NOT use the alternative structure *KeyTransRecipientInfo* for asymmetric key management in the *recipientInfos* field.

Compliant components SHALL use the *version* field within *KeyTransRecipientInfo* with the value 0.

Compliant components SHALL use the alternative *IssuerAndSerialNumber* for the *rid* field within *KeyTransRecipientInfo.*

3.1 General CMS Syntax

The general syntax of cryptographic messages is defined by the ASN.1 type *ContentInfo*, which is a sequence of the fields listed in the following table

Table 1: Fields of *ContentInfo*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*contentType*	Object identifier for the type of the associated and protected object	[RFC 3852]	3	++		++	++	OID: 1.2.840.113549.1.7.1 OID: 1.2.840.113549.1.7.2 OID: 1.2.840.113549.1.7.3	[1] [2] [3, 4]
2	*content*	associated and protected object	[RFC 3852]	3	++	Table 2 Table 6	++ ++	++ ++	*SignedData* *EnvelopedData*	

[1] This OID identifies the *id-data* content type

[2] This OID identifies CMS objects of the type *SignedData.*

[3] This OID identifies CMS objects of the type *EnvelopedData.*

[4] CMS defines further content types for CMS objects that are not considered in Common PKI.

3.2 Signed-data Content Type

The type for *signed-data* is defined by the ASN.1 type *SignedData* is a sequence of the fields listed in the following table.

Table 2: Fields of *SignedData*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of CMS syntax	[RFC 3852]	5.1	++		++	++	1 2 3 4	[1] [5] [2] [5]
2	*digestAlgorithms*	Collection (including zero) of message digest algorithm identifiers	[RFC 3852]	5.1	++		++	++		[3]
3	*encap-ContentInfo*	Data to be protected	[RFC 3852]	5.1	++	Table 3	++	++		
4	*certificates*	Collection of certificates	[RFC 3852]	5.1	+-		+-	+-		[4]
5	*crls*	Collection of CRLs or other revocation status information	[RFC 3852]	5.1	+-		+-	+-		[6]
6	*signerInfos*	Collection of per-signer information	[RFC 3852]	5.1	++	Table 4	++	++		

[1] Compliant components SHALL always use the value 1, if non-interpreted binary data shall be protected.

[2] Compliant components SHALL always use the value 3, if data with assigned format identifiers shall be protected.

[3] For permitted hash algorithm identifiers refer to P6.T1 (One-Way Hash Functions) of this Common PKI specification.

[4]	Compliant components SHALL at least contain the signer's certificate, and, should include all certificates of the certification path(s) of the signer(s) required by the recipient. **Common PKI Profile:** Only public key certificates and attribute certificates of version v1 according to Common PKI Part 1 SHALL be included.
[5]	These versions are currently not considered in Common PKI.
[6]	**Common PKI Profile:** Only CRLs according to Common PKI Part 1 SHOULD be included. Optionally, OCSP responses according to Common PKI Part 4 MAY be included. Other types of revocation status information MAY not be included.

The type for the *encapContentInfo* field is defined by the ASN.1 type *EncapsulatedContentInfo, which is a sequence of the fields,* listed in the following table.

Table 3: Fields of *EncapsulatedContentInfo*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*eContentType*	Object identifier for the type of the associated and protected content	[RFC 3852]	5.2	++		++	++	OID: 1.2.840.113549.1.7.1	[1]
2	*eContent*	Associated and protected content	[RFC 3852]	5.2	+-		-- ++	++ ++		[2] [3]
[1]	Compliant components SHALL support the value for *id-data,* which indicates that the signature is related to non-interpreted binary data. The support for other values is optional.									
[2]	Compliant components SHALL omit the *eContent* field if external signatures have to be constructed for S/MIME message types *multipart/signed*.									
[3]	Compliant components SHALL use the *eContent* field if signatures have to be constructed for S/MIME message types with *smime-type=signed-data.*									

The type for the *signerInfos* set is defined by the ASN.1 type *SignerInfo*, which is a sequence of the fields, listed in the following table.

Table 4: Fields of *SignerInfo*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	[RFC 3852]	5.3	++		++	++	1	[1]
2	*sid*	Identification of the signers certificate	[RFC 3852]	5.3	++		++	++		[2]
3	*digestAlgorithm*	Identification of the signers hash algorithm	[RFC 3852]	5.3	++		++	++		[3]
4	*signedAttrs*	Collection of signed attributes	[RFC 3852]	5.3	+-	Table 5	+- ++	++ ++		[4] [5]
5	*signatureAlgorithm*	Identification of the signers signature algorithm	[RFC 3852]	5.3	++		++	++		[6]
6	*signature*	Digital signature of the signer	[RFC 3852]	5.3	++		++	++		
7	*unsignedAttrs*	Collection of unsigned attributes	[RFC 3852]	5.3	+-	Table 5	+-	++		[7]

[1]	Compliant components SHALL use the value 1, since the *issuerAndSerialNumber* alternative shall be used for the *sid* field.
[2]	Compliant components SHALL always use the *issuerAndSerialNumber* alternative.
[3]	The value provided in this field SHALL be contained in the *SignedData.digestAlgorithms* field (see T2.#2). For permitted hash algorithm identifiers refer to P6.T1 (One-Way Hash Functions) of this Common PKI specification.
[4]	Compliant components MAY include signed attributes in the *signedAttrs* field if the *eContent* field is *id-data.*
[5]	Compliant components SHALL include signed attributes in the *signedAttrs* field if the *eContent* field is not *id-data* or if attributes as for example signing-time shall be linked to the signature.
[6]	Compliant components SHALL support the signature algorithms as specified in part 6 of the Common PKI specification.
[7]	Compliant components MAY include unsigned attributes.

Signed and unsigned attributes are of the ASN.1 type *SET OF Attribute.* The type *Attribute* itself is a sequence of the *attrType* and *attrValues* fields that identify an attribute and respectively contain the set of attribute values. The minimum set of signed attributes that compliant components SHALL support is listed in the following table. This table also provides a list of unsigned attributes that compliant components MAY support.

Table 5: Signed and Unsigned Attributes

ATTRIBUTES			REFERENCES				COMMON PKI			
#	NAME OID	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*content-type* *id-contentType* {1 2 840 113549 1 9 3}	OID for the type of the *ContentInfo* value being signed in *signed-data*	[RFC 3852]	11.1	++		++	++	OID that identifies the type of the data to be signed	[1]
2	*message-digest* *id-messageDigest* {1 2 840 113549 1 9 4}	Hash value of the *encapContentInfo.eContent* value being signed in *signed-data*	[RFC 3852]	11.2	++		++	++	Hash value OCTET STRING	[1]
3	*signing-time* *id-signingTime* {1 2 840 113549 1 9 5}	Time at which the signer claims to have performed the signing process	[RFC 3852]	11.3	+-		+-	++	Signing time	[2], [3]
4	*otherSigCert* *id-aa-ets-otherSigCert* {1 2 840 113549 1 9 16 2 19}	Sequence of certificate identifiers starting with the certificate of the signer	[CAdES]	5.7.3.3	-		-	+-		[2], [5]
5	*certificateRefs* *id-aa-ets-certificateRefs* {1 2 840 113549 1 9 16 2 21}	References to the full set of CA certificates that have been used to validate an electronic signature.	[CAdES]	6.2.1	+-		+-	+-		[4]

6	*revocationRefs* *id-aa-ets-revocationRefs* {1 2 840 113549 1 9 16 2 22}	References to the full set of CRL or OCSP responses that have been used in the validation of the signer and CA certificates in an electronic signature.	[CAdES]	6.2.2	+-		+-	+-		[4]
7	*escTimeStamp* *id-aa-ets-escTimeStamp* {1 2 840 113549 1 9 16 2 25}	Timestamp of the hash of the electronic signature and the complete validation data	[CAdES]	6.3.5	+-		+-	+-		[4]
8	*signingCertificate* *id-aa-signingCertificate* {1 2 840 113549 1 9 16 2.12}	Sequence of certificate identifiers starting with the certificate of the signer	[RFC 2634]	5.4	+-		-	+-	The *issuerSerial* field of the *ESSCertID* within *SigningCertificate* MUST not be empty.	[2], [5]
9	*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	+-		+-	+-		[5]

[1]	Compliant components SHALL support this signed attribute if the optional *signedAttrs* field is used.
[2]	If present, this optional attribute MUST be a signed attribute.
[3]	[RFC 2630]: Dates between 1 January 1950 and 31 December 2049 (inclusive) MUST be encoded as *UTCTime.* Any dates with year values before 1950 or after 2049 MUST be encoded as *GeneralizedTime.* **Common PKI Profile**: Compliant components SHOULD also accept dates between 1 January 1950 and 31 December 2049 encoded as *GeneralizedTime* for backwards compatibility with MailTrusT v2.
[4]	**Common PKI Profile**: Compliant components MAY include this unsigned attribute. For the purpose of providing complete validation data, it is RECOMMENDED that compliant components use this unsigned attribute.

[5]	The *otherSigCert* attribute provides the same functionality as the *signingCertificate* attribute defined by [RFC 2634, 5.4] with the exception that *otherSigCert* can be used with hashing algorithms other than SHA-1. The new *signingCertificateV2* attribute introduced in [RFC5035] does also address the issues of hash functions other than SHA-1 and is intended to replace both the old [RFC2634] *signingCertificate* attribute and the original [CAdES] *otherSigCert* attribute.

3.3 Enveloped-data Content Type

The type for *enveloped-data* is defined by the ASN.1 type *EnvelopedData* is a sequence of the fields listed in the following table.

Table 6: Fields of *EnvelopedData*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	[RFC 3852]	6.1	++		++	++	0	[1]
2	*originatorInfo*	Signer information including certificates and CRLs	[RFC 3852]	6.1	+-		--	--		[1]
3	*recipientInfos*	Collection of per-recipient information	[RFC 3852]	6.1	++	Table 7	++	++		
4	*encryptedContentInfo*	Encrypted data	[RFC 3852]	6.1	++	Table 9	++	++		
5	*unprotectedAttrs*	Collection of non-encrypted attributes	[RFC 3852]	6.1	+-		--	--		[1]

[1]	Compliant components SHALL always use the value 0, which implies that the fields *originatorInfo* and *unprotectedAttrs* MUST be absent, and that all of the *RecipientInfo* structures are of version 0.

The type for the *recipientInfos* set is defined by the ASN.1 type *RecipientInfo*, which is a choice of the alternatives, listed in the following table. These alternatives are used to support three different key management techniques.

Table 7: Alternatives of *RecipientInfo*

ALTERNATIVES			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*ktri*	per-recipient information using key transport	[RFC 3852]	6.2.1	++	Table 8	++	++		[1]
2	*kari*	recipient information using key agreement	[RFC 3852]	6.2.2	++		--	--		[1]
3	*kekri*	recipient information using previously distributed symmetric key-encryption keys	[RFC 3852]	6.2.3	++		--	--		[1]
4	*pwri*	recipient information using a password or shared secret value	[RFC 3852]	6.2.4	++		--	--		[1]
5	*ori*	recipient information for additional key management techniques	[RFC 3852]	6.2.5	++		--	--		[1]

[1] Compliant components shall support the key transport alternative. The other mechanisms are currently not considered in Common PKI.

The type for *ktri* is defined by the ASN.1 type *KeyTransRecipientInfo,* which is a sequence of the fields, listed in the following table. This structure shall also be used for the originator as recipient, if the originator himself wants to be able to decrypt the message.

Table 8: Fields of *KeyTransRecipientInfo*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	[RFC 3852]	6.2.1	++		++	++	0	[1]
2	*rid*	Identification of the recipients certificate	[RFC 3852]	6.2.1	++		++	++		[2]
3	*keyEncryptionAlgorithm*	Identification of the key-encryption algorithm	[RFC 3852]	6.2.1	++		++	++		
4	*encryptedKey*	Encrypted content-encryption key	[RFC 3852]	6.2.1	++		++	++		

[1] Compliant components SHALL always use the value 0, which implies that the fields *originatorInfo* and *unprotectedAttrs* MUST be absent, and that all of the *RecipientInfo* structures are of version 0.

[2] Compliant components SHALL always use the *issuerAndSerialNumber* alternative, which uniquely identifies the certificate of the recipient. This certificate SHALL contain the key usage extension with the *keyEncipherment* bit 2 set. The reason is that only public key encryption keys shall be used for the encryption of the content-encryption key.

The type for *encryptedContentInfo* is defined by the ASN.1 type *EncryptedContentInfo, which is a sequence of the fields*, listed in the following table.

Table 9: Fields of *EncryptedContentInfo*

FIELDS			REFERENCES				COMMON PKI			
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	TABLE	SUPPORT			NOTES
							GEN	PRO	VALUES	
1	*contentType*	Object identifier for the type of the associated and protected content	[RFC 3852]	6. 1	++		++	++	OID: 1.2.840.113549.1.7.1	[1]
2	*contentEncryption Algorithm*	Identification of the content-encryption algorithm	[RFC 3852]	6. 1	++		++	++		
3	*encryptedContent*	Encrypted content-encryption key	[RFC 3852]	6. 1	+-		++	++		

[1] Compliant components SHALL support the value for *id-data,* if non-interpreted binary data have been encrypted. The support for other values is OPTIONAL.

4 File Signature and Encryption

Files stored in an archive or transferred via Internet (using FTP or HTTP) can be encrypted and/or signed. The format of the encrypted/signed file is based on CMS [RFC 3852].

The following CMS container types MUST be supported by Common PKI-compliant components:

- for encrypted data: *enveloped-data*
- for signed data: *signed-data*
- for signed and then encrypted data: *enveloped-data* with *signed-data* as content

Other content types MAY, but need not by supported by compliant components. Other content types SHOULD NOT be created by components, if the file is intended for another user, as it cannot be assumed that the receiver is able to handle those types.

4.1 File Signature

Signed files will be represented by the *SignedData* content type. The *certificates* field of *SignedData* MUST contain the public key certificate of the signer. A reference to this certificate MUST be included in the *signedAttributes* of the corresponding *SignerInfo*. It SHOULD be included using the *SigningCertificateV2* attribute, which is defined in [RFC 5035]. The older *SigningCertificate* attribute form of [RFC 2634] is permitted for backward compatibility, but SHOULD NOT be used. Additionally, the *certs* field SHOULD contain all certificates in the certificate path up to the certificate of root or top-level CA.

[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 *SigningCertificate* attribute. The *issuerSerial* field of the *ESSCertID* within *SigningCertificate* MUST not be empty. These informations are 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.

The signed-data format allows parallel signatures of the file content. This option MUST be supported by Common PKI-compliant components. In essence, additional signatures on the content are appended to a list of signatures in the readily available container. All certificates of the signers are to be collected in the 'certificates' field of *SignedData*. The order of certificates in the list is irrelevant.

The *signing-time* attribute, specifying the time at which the signer (purportedly) performed the signing process, MUST always be present in signed-data, so that the reference time for signature validation can be retrieved from the signed document. Signing-time MUST be a signed attribute.

The *countersignature* attribute type specifies one or more signatures on the contents octets of the DER encoding of the *signatureValue* field of a *SignerInfo* value in signed-data. Thus, the *counterSignature* attribute type countersigns (signs in serial) another signature. For the

simplicity of implementations, counter signatures are not necessary to be supported by compliant components. Hence, the attribute *counterSignature* SHOULD NOT be inserted by components, if the file is intended for another user, as it cannot be assumed that the receiver of the countersigned document is able to verify the counter signature. Nevertheless, components MUST be able to parse the *counterSignature* attribute.

4.2 File Encryption

Three key management techniques are described in CMS to provide for a symmetric content-encryption key: key transport, key agreement, and previously distributed keys. Common PKI-compliant components MUST only support the key transport mechanism, as it is appropriate for the most common PKI-based "store-and-forward" type of communication. Other mechanisms MAY be supported, but should not be used, if the recipient's component is not known to support the used option.

In the key transport mechanism, the symmetric content-encryption key is encrypted using the recipient's public key. Users, encrypting files on their local computer, can use their own public key for this purpose. As recipient's information, including the encrypted symmetric key, MUST always be present in the encrypted file, the use of the enveloped-data container type is indicated (Encrypted-data cannot store such information.).

Annex A: ASN.1 Definitions

This chapter contains a list of ASN.1 definitions that are used in this part of the Common PKI specification in alphabetical order.

```
Attribute ::= SEQUENCE {
  attrType OBJECT IDENTIFIER,
  attrValues SET OF AttributeValue }

AttributeValue ::= ANY

CertificateChoices ::= CHOICE {
  certificate Certificate,
  extendedCertificate [0] IMPLICIT ExtendedCertificate,
  v1AttrCert [1] IMPLICIT AttributeCertificateV1,
  v2AttrCert [2] IMPLICIT AttributeCertificateV2,
  other [3] IMPLICIT OtherCertificateFormat }

CertificateRevocationLists ::= SET OF CertificateList

CertificateSet ::= SET OF CertificateChoices

CMSVersion ::= INTEGER  { v0(0), v1(1), v2(2), v3(3), v4(4) }

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

ContentInfo ::= SEQUENCE {
  contentType ContentType,
  content [0] EXPLICIT ANY DEFINED BY contentType }

ContentType ::= OBJECT IDENTIFIER

DigestAlgorithmIdentifier ::= SET OF AlgorithmIdentifier

DigestAlgorithmIdentifiers::= SET OF DigestAlgorithmIdentifier

EncapsulatedContentInfo ::= SEQUENCE {
  eContentType ContentType,
  eContent [0] EXPLICIT OCTET STRING OPTIONAL }
```

```
EncryptedContent ::= OCTET STRING
```

```
EncryptedContentInfo ::= SEQUENCE {
  contentType ContentType,
  contentEncryptionAlgorithm ContentEncryptionAlgorithmIdentifier,
  encryptedContent [0] IMPLICIT EncryptedContent OPTIONAL }
```

```
EncryptedKey ::= OCTET STRING
```

```
EnvelopedData ::= SEQUENCE {
  version CMSVersion,
  originatorInfo [0] IMPLICIT OriginatorInfoOPTIONAL,
  recipientInfos RecipientInfos,
  encryptedContentInfo EncryptedContentInfo,
  unprotectedAttrs [1] IMPLICIT UnprotectedAttributes OPTIONAL
}
```

```
id-contentType OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 3 }
```

```
id-data OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 1 }
```

```
id-envelopedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 3 }
```

```
id-messageDigest OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 4 }
```

```
id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }
```

```
id-signingTime OBJECT IDENTIFIER ::= { iso(1) member-body(2)
    us(840) rsadsi(113549) pkcs(1) pkcs9(9) 5 }
```

```
IssuerAndSerialNumber ::= SEQUENCE {
  issuer Name,
  serialNumber CertificateSerialNumber }
```

```
KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier
```

```
KeyTransRecipientInfo ::= SEQUENCE {
  version CMSVersion,
  rid RecipientIdentifier,
  keyEncryptionAlgorithm KeyEncryptionAlgorithmIdentifier,
  encryptedKey EncryptedKey}
```

```
MessageDigest ::= OCTET STRING
```

```
OriginatorInfo ::= SEQUENCE {
  certs [0] IMPLICIT CertificateSet OPTIONAL,
  crls [1] IMPLICIT RevocationInfoChoicesOPTIONAL }
```

```
OtherCertificateFormat ::= SEQUENCE {
  otherCertFormat OBJECT IDENTIFIER,
  otherCert ANY DEFINED BY otherCertFormat }
```

```
OtherRevocationInfoFormat ::= SEQUENCE {
  otherRevInfoFormat OBJECT IDENTIFIER,
  otherRevInfo ANY DEFINED BY otherRevInfoFormat }
```

```
RecipientIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier}
```

```
RecipientInfo ::= CHOICE {
  ktri KeyTransRecipientInfo,
  kari  [1] KeyAgreeRecipientInfo,
  kekri [2] KEKRecipientInfo,
  pwri  [3] PasswordRecipientInfo,
  ori   [4] OtherRecipientInfo }
```

```
RecipientInfos ::= SET OF RecipientInfo
```

```
RevocationInfoChoices ::= SET OF RevocationInfoChoice
```

```
RevocationInfoChoice ::= CHOICE {
  crl CertificateList,
  other [1] IMPLICIT OtherRevocationInfoFormat }
```

```
SignatureAlgorithmIdentifier ::= AlgorithmIdentifier
```

```
SignatureValue ::= OCTET STRING
```

```
SignedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

```
SignedData ::= SEQUENCE {
  version CMSVersion,
  digestAlgorithms DigestAlgorithmIdentifiers,
  encapContentInfo EncapsulatedContentInfo,
  certificates [0] IMPLICIT CertificateSet OPTIONAL,
  crls [1] IMPLICIT RevocationInfoChoices OPTIONAL,
  signerInfos SignerInfos}
```

```
SignerIdentifier ::= CHOICE {
  issuerAndSerialNumber IssuerAndSerialNumber,
  subjectKeyIdentifier [0] SubjectKeyIdentifier}
```

```
SignerInfo ::= SEQUENCE {
  version CMSVersion,
  sid SignerIdentifier,
  digestAlgorithm DigestAlgorithmIdentifier,
  signedAttrs [0] IMPLICIT SignedAttributesOPTIONAL,
  signatureAlgorithm SignatureAlgorithmIdentifier,
  signature SignatureValue,
  unsignedAttrs [1] IMPLICIT UnsignedAttributesOPTIONAL }
```

```
SignerInfos ::= SET OF SignerInfo
```

```
SigningTime ::= Time
```

```
SubjectKeyIdentifier ::= OCTET STRING
```

```
Time ::= CHOICE {
  utcTime UTCTime,
  generalTime GeneralizedTime }
```

```
UnprotectedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

```
UnsignedAttributes ::= SET SIZE (1..MAX) OF Attribute
```

References

[CAdES] ETSI TS 101 733 v1.7.4: Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES), July 2008

[RFC 2045] N. Freed, N. Borenstein: Multipurpose Internet Mail Extensions (MIME) l: Part One: Format of Internet Mail Bodies; November 1996

[RFC 2046] N. Freed, N. Borenstein: Multipurpose Internet Mail Extensions (MIME) l: Part Two: Media Types; November 1996

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

[RFC 3851] B. Ramsdell: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, July 2004

[RFC 3852] R. Housley: Cryptographic Message Syntax (CMS), July 2004

[RFC 5035] J. Schaad: Enhanced Security Services (ESS) Update: Adding CertID Algorithm Agility, August 2007

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 4

OPERATIONAL PROTOCOLS

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0 30.09.2001	First public edition
1.0.1 15.11.2001	A couple of editorial and stylistic changes: - references to SigG-specific issues eliminated from core documents - core documents (Part 1-7) and optional profiles have been separated in different PDF documents.
1.0.2 19.07.2002	Several editorial changes and bug-fixes. The most relevant changes affecting technical aspects are: 1) *'dc'* (*DomainComponent*) attribute and *dcObject*, a corresponding auxiliary class added to the list supported X.500 attributes. (T1.#25, T2.#10) 2) Requirements eased on the necessary number of certificates delivered in signed OCSP requests and responses. Only the signing EE certificate MUST be supplied, CA certificates are still recommended to be included. (T5.#13, T8.#7) 3) The *certID* in a *SingleResponse* MUST be built using SHA-1. Processing components (typically the responder) MUST support SHA-1 and SHOULD support RIPEMD160 and MD5. (T6.[5]) 4) the *certID* in a *SingleResponse* MUST be identical to that in the corresponding (single) *Request*. (T8.[7]) 5) The ASN.1 definition of the *good* field was erroneous in v101. The correct tagging modus is IMPLICIT. (T8.#25) 6) HTTP MUST be employed for transporting TSP messages. (Section 5)
1.0.2 11.08.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.03.2004	Several editorial changes.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Adapted to new versions of the base standards: - ETSI TS 101 861 v1.3.1 - RFC 2616 - RFC 3739 - RFC 4510 - RFC 4511 - RFC 4512 - RFC 4513 - RFC 4514 - RFC 4516 - RFC 4517 - RFC 4519 - RFC 4522 - RFC 4523 - RFC 5280 - X.509:2005 Various corrections and clarifications. Removed remarks on delta CRLs, that are covered in Part 1.

Table of Contents

1 Preface

Operational protocols are required in a public key infrastructure (PKI) to deliver certificates, CRLs or certificate status information to certificate using systems, such as mail clients or Internet Browsers. It is the intention of this Common PKI Specification to select a “necessary minimum” of possible repository functions and access methods, which shall be supported by all Common PKI-compliant repositories and client systems. In this way, interoperability within the Common PKI community shall be achieved, which allows the automatic verification of signatures and certificate paths, independently of the client implementation and respectively of the directory service provider. This Common PKI standard builds on the most common form of certificate repository, the X.500 directory and on access methods that are specified in PKIX Internet standards, namely LDAP v3 (Light Weigh Directory Access, Version 3) and OCSP v1 (Online Certificate Status Protocol). As for the transport of protocol information between directory and clients, this specification restricts itself to the TCP/IP-based protocols LDAP (for LDAP-access) and HTTP (for OCSP).

PKIX Standards (RFCs) describe methods for the storage and retrieval of public key certificates (PKCs) and certificate revocation lists (CRLs) of PKCs. Common PKI provides a profile for attribute certificates (ACs) too. Since standardization work on attribute certificates (ACs) has just recently begun at IETF, RFCs does not currently concern how to deal with ACs and CRLs of ACs in a directory. Still, there exist a draft paper [DraftSchema] describing how to include ACs and CRLs on ACs in an LDAP directory schema. Considering that the paper is still in the ‘draft’ state, that the syntaxes and attribute types defined there are not yet supported by off-the-shelf directory servers and that there exists no paper yet on how to deal with ACs within OCSP, this Common PKI Specification proposes to handle ACs and CRLs of ACs within the LDAP/OCSP-infrastructure as if they were PKCs and respectively CRLs of PKCs. This is also the approach followed by current system implementations.

A further important service in a PKI is time-stamping. In order to associate a datum (a message or document) with a particular point in time, a Time Stamp Authority (TSA) needs to be used. This Trusted Third Party provides a "proof-of-existence" for this particular datum at an instant in time. This can then be used, for example, to verify that a digital signature was applied to a message before the corresponding certificate was revoked, thus allowing a revoked PKC to be used for verifying signatures created prior to the time of revocation. The TSA can also be used to indicate the time of submission when a deadline is critical, or to indicate the time of transaction for entries in a log. For the sake of interoperability, this document specifies a time stamp protocol (TSP) to acquire and obtain time stamp from a server. This specification relies on the PKIX standard [RFC3161] and, in particular, on the TSP-Profile of ETSI [ETSI-TSP].

As this Common PKI specification is intended to be kept at the necessary minimum, the transport of certificates and CRLs via email is NOT required to be supported (required by [MTTv2]), whereas the support of FTP and HTTP for the transport as defined in [RFC2585] is optional (just as in [MTTv2]). Other novel services, currently being worked out by IETF, such as Repository Locator Service (to find repository servers of different types and locations), Open CRL Distribution Point, Simple Certificate Validation Protocol, Delegated Path Validation (an extension of OCSP) and Data Certification, are similarly not part of this specification.

1.1 Compatibility Aspects

This specification is based on IETF documents (RFCs and drafts) and contains basically profiling information to tailor those standards to the specific needs of the target application area. Where necessary, this Common PKI specification adds new definitions to those in the PKIX documents or restricts the usage of available data components in some way. As usual in the Common PKI Specification, such definitions are always commented and the corresponding note is marked with the words '**Common PKI Profile**'.

Besides conformance with international standards, backward compatibility with [ISIS] and [MTTv2] will be provided, so that available systems and information (e.g. certificates, signed documents) can further be used.

The LDAP protocol (Lightweight Directory Access Protocol) presented here is based on LDAP v3 [RFC4510] et sqq. Nevertheless, only protocol elements specified in LDAP v2 [RFC1777] are and SHOULD be used in Common PKI-compliant PKI. Special attention will be paid to the handling of attribute certificates (ACs) and revocation lists (CRLs) of ACs, as these content types are currently being worked out by IETF and are thus not yet part of standards (RFCs).

The OCSP v1 protocol, which must be supported by all conforming certification authorities, is defined in [RFC2560] and will be profiled in this Common PKI specification.

When offering or accessing time stamp services, Common PKI–compliant systems MUST apply the protocol defined in [RFC3161] and profiled in [ETSI-TSP]. Except for hash algorithm support, no further profiling information is added by this specification to the profile of ETSI.

2 Directory Access via LDAP

The LDAP protocol (Lightweight Directory Access Protocol) presented here is based on LDAP v3 [RFC4510] et sqq. Basically, only protocol elements specified in LDAP v2 [RFC1777] are and MAY be used in Common PKI-compliant PKI. Nevertheless, Common PKI-compliant systems MUST employ LDAP v3. The reason for this decision lies in the usage of binary attribute types and UTF8 strings in requests as described below.

Basically, attribute values are stored and retrieved by an LDAP v2 directory in string representation, described in [RFC1778]. However, the string representation is basically suited to v1 PKCs and v1 CRLs and is not appropriate for v3 PKCs and v2 CRLs, since there has been no string form defined yet for the numerous extension types included in those data structures.

As a reaction to the above encoding problem of some attribute values, LDAP v3 introduces the *binary* syntax, which is consistent with the above mentioned way of encoding. By including the *binary* option in requests, clients can request the LDAP v3 directory to store or retrieve attribute values of any type (!) in *binary* encoded form. According to [RFC4523], this latter option MUST always be used in requests for storage and requests of certificates and CRLs. This means that requests on LDAP v2 and respectively on LDAP v3 servers are different.

This Common PKI specification proposes to handle ACs and CRLs of ACs within the LDAP/OCSP-infrastructure as if they were PKCs and respectively CRLs of PKCs. This means that ACs and CRLs on ACs will be stored in their DER-encoded binary representation in attributes of type *userCertificate* and respectively *certificateRevocationList*, just as PKCs and respectively CRLs of PKCs. Common PKI-compliant clients MUST be prepared to receive a DER-encoded *AttributeCertificate* object in place of a *Certificate* and to properly process it. There is no difference between the CRL-syntax for PKCs and respectively for ACs, the syntax *CertificateList* is employed in both cases.

Common PKI Profile: Note that handling PKCs and ACs in the same way is a different approach than that followed in [X.509:2005]. In that document, ACs are forced to be kept separated from PKCs: ACs and CRLs of ACs are kept in different directory attributes (*attributeDescriptorCertificate, attributeCertificateRevocationList and attributeAuthorityRevocationList*). Furthermore, [X.509:2005] forbids the same CA to issue the same CRL to keep information about PKCs and ACs at same time. In contrast to that, Common PKI allows CAs to issue PKCs and ACs and to publish corresponding revocation information in the same CRL. In order to be able to unambiguously identify PKCs and ACs issued by the same CA, serial numbers MUST be unique among all PKCs and ACs, a further difference compared to the PKIX scheme.

2.1 The Common PKI LDAP Schema

The nature of this section is purely informative. Its purpose is to provide an example of an LDAP-Schema, and it does not specify requirements on the implementation of an LDAP-Schema.

Common PKI conforming directories shall be prepared to store the following data objects:

- root certificates
- cross certificates
- CA certificates
- end entity (or user) certificates, including PKCs as well as ACs
- revocation lists (CRLs), that may include entries for PKCs as well as ACs
- delta revocation lists, corresponding to the above complete CRLs

This section illustrates a directory schema, i.e. object classes, attribute types and a Directory Information Tree (DIT) structure that MAY (but need not) be used to implement a compliant directory. The following design goals have been followed in the design of the schema:

- end entity certificates and CRLs SHOULD be grouped around the entry representing the issuing CA instance
- as far as possible, standard object classes, attribute types and syntaxes SHOULD be used, defined in RFCs
- it MUST be possible, to find a certain certificate using the *issuer* and *subject* DNames and the certificate serial number contained in the certificate.
- it MUST be possible to search for certificates of an end entity with the help of partial information about the end entity, such as name (*surname* or *commonName*), affiliation (*organization, organizational unit*), address (*postalAddress*, e.g. in case of private persons without affiliation).

The exemplary DIT structure is depicted in Figure 1. In the following, we present object classes and attributes types that MAY be used in the directory entries of the proposed schema. The formal definitions are given in ASN.1 syntax.

Common PKI Profile: Note that the only requirement for a directory to be Common PKI-compliant is that the directory delivers adequate responses to a relatively small set of requests that are specified in Section 2.2. This means that conforming schema implementations MAY

slightly differ from the one described here, according to differences in the "built-in" features (attribute types and object classes) of a directory product or to some other design criteria.

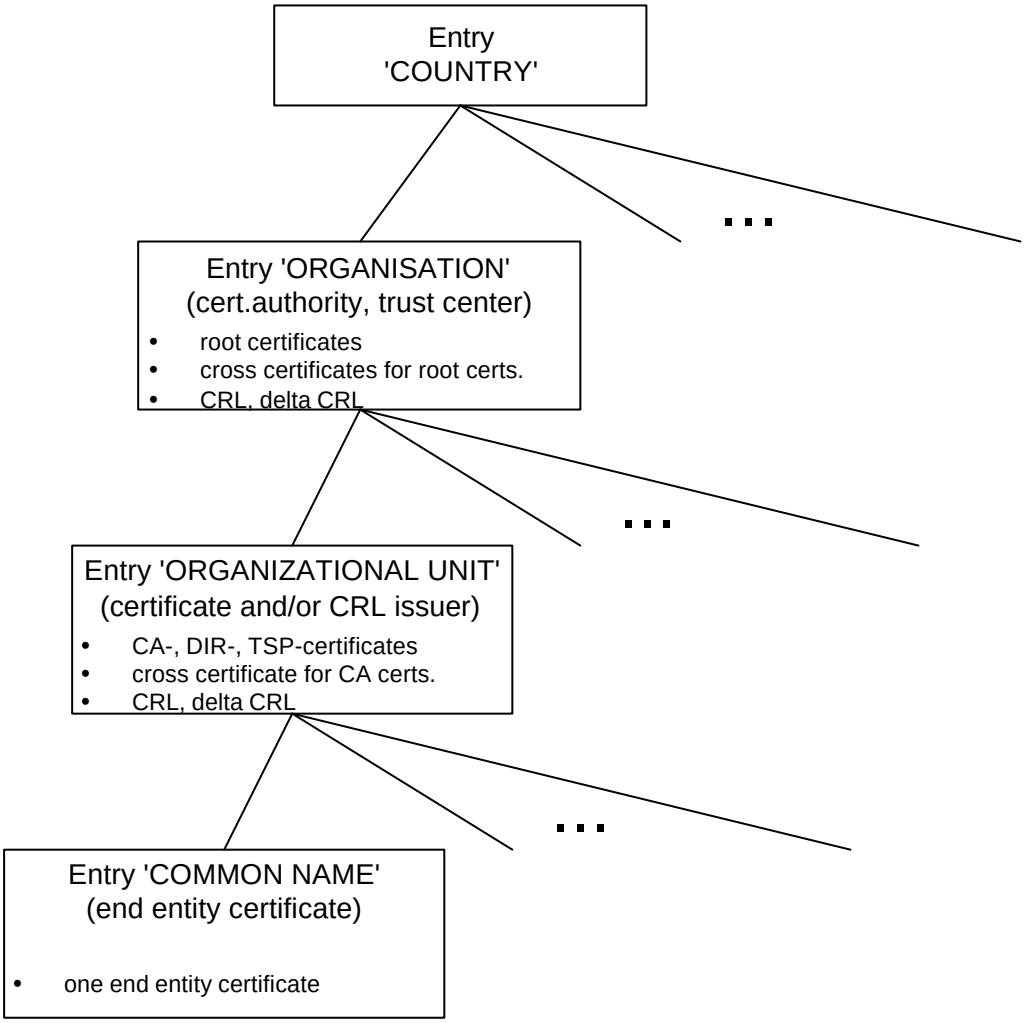


Figure 1: An exemplary DIT structure for Common PKI-compliant directories

Table 1: Attribute Types and Attribute Sets

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT LDAP SERVER	REFERENCES		NOTES
				RFC	TABLE	
	STANDARD X.520 DNAME ATTRIBUTES					
1	(2.5.4.41 NAME 'name' EQUALITY caseIgnoreMatch UBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)	an abstract class used to derive other DName attribute types below 1.3.6.1.4.1.1466.115.121.1.15 refers to the Directory String syntax [RFC4517].	no relevance	RFC4519 2.18		
2	(2.5.4.3 NAME 'cn' SUP name)		++	RFC4519 2.2		
3	(2.5.4.4 NAME 'sn' SUP name)		+	RFC4519 2.32		
4	(2.5.4.42 NAME 'givenName' SUP name)		+	RFC4519 2.12		
5	(2.5.4.5 NAME 'serialNumber' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.44)	1.3.6.1.4.1.1466.115.121.1.44 refers to the Printable String syntax [RFC4517].	++	RFC4519 2.21		[1]
6	(2.5.4.6 NAME 'c' SUP name SYNTAX 1.3.6.1.4.1.1466.115.121.1.11 SINGLE-VALUE)	'countryName' in X.500 1.3.6.1.4.1.1466.115.121.1.11 refers to the Country String syntax [RFC4517].	++	RFC4519 2.2		
7	(2.5.4.7 NAME 'l' SUP name)	'localityName' in X.500	+	RFC4519 2.16		
8	(2.5.4.8 NAME 'st' SUP name)	'stateOrProvinceName' in X.500	+	RFC4519 2.33		
9	2.5.4.9 NAME 'street' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)	'streetAddress' in X.500 1.3.6.1.4.1.1466.115.121.1.15 refers to the Directory String syntax [RFC4517].	+	RFC4519 2.34		
10	(2.5.4.10 NAME 'o' SUP name)	'organizationName' in X.500	++	RFC4519 2.19		
11	(2.5.4.11 NAME 'ou' SUP name)	'organizationalUnitName' in X.500	++	RFC4519 2.20		
12	(2.5.4.12 NAME 'title' SUP name)		+-	RFC4519 2.38		

13	(2.5.4.15 NAME 'businessCategory' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)	occupation of a person 1.3.6.1.4.1.1466.115.121.1.15 refers to the Directory String syntax [RFC4517].	-	RFC4519 2.1		[2]
14	(2.5.4.16 NAME 'postalAddress' EQUALITY caseIgnoreListMatch SUBSTR caseIgnoreListSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.41)	1.3.6.1.4.1.1466.115.121.1.41 refers to the Postal Address syntax [RFC4517].	+	RFC4519 2.23		
15	(2.5.4.17 NAME 'postalCode' EQUALITY caseIgnoreMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.15)	1.3.6.1.4.1.1466.115.121.1.15 refers to the Directory String syntax [RFC4517].	+	RFC4519 2.24		
16	(2.5.4.43 NAME 'initials' SUP name)		+-	RFC4519 2.14		
17	(2.5.4.44 NAME 'generationQualifier' SUP name)		+-	RFC4519 2.11		
18	2.5.4.46 NAME 'dnQualifier' EQUALITY caseIgnoreMatch ORDERING caseIgnoreOrderingMatch SUBSTR caseIgnoreSubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.44)	distinguished name qualifier: disambiguating information to be added to a DName, if for example two DSAs, that are to be merged, contain entries with the same DName 1.3.6.1.4.1.1466.115.121.1.44 refers to the Printable String syntax [RFC4517].	+-	RFC4519 2.		
	PKI-SPECIFIC ATTRIBUTES					
19	(2.5.4.36 NAME 'userCertificate' DESC 'X.509 user certificate' EQUALITY certificateExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.8)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "userCertificate;binary".	++	RFC4523 4.1		
20	(2.5.4.37 NAME 'cACertificate' DESC 'X.509 CA certificate' EQUALITY certificateExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.8)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "cACertificate;binary".	++	RFC4523 4.2		
21	(2.5.4.40 NAME 'crossCertificatePair' DESC 'X.509 cross certificate pair' EQUALITY certificatePairExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.10)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "crossCertificatePair;binary".	++	RFC4523 4.3		

22	(2.5.4.38 NAME 'authorityRevocationList' DESC 'X.509 authority revocation list' EQUALITY certificateListExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.9)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "authorityRevocationList;binary".	++	RFC4523 4.5		
23	(2.5.4.39 NAME 'certificateRevocationList' DESC 'X.509 certificate revocation list' EQUALITY certificateListExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.9)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "certificateRevocationList;binary".	++	RFC4523 4.4		
24	(2.5.4.53 NAME 'deltaRevocationList' DESC 'X.509 delta revocation list' EQUALITY certificateListExactMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.9)	As required by this attribute type's syntax, values of this attribute are requested and transferred using the attribute description "deltaRevocationList;binary".	+-	RFC4523 4.6		
25	(0.9.2342.19200300.100.1.25 NAME 'dc' EQUALITY caseIgnoreIA5Match SUBSTR caseIgnoreIA5SubstringsMatch SYNTAX 1.3.6.1.4.1.1466.115.121.1.26 SINGLE-VALUE)	'domainComponent' in RFC 1274 1.3.6.1.4.1.1466.115.121.1.26 refers to the IA5 String syntax [RFC4517].	+-	RFC4519 2.4		
	ATTRIBUTE SETS USED IN OBJECT CLASS DEFINITIONS					
26	PostalAttributeSet ATTRIBUTE ::= { postalAddress | postalCode | streetAddress }			X.521 5.2		[3]
27	LocaleAttributeSet ATTRIBUTE ::= { localityName | stateOrProvinceName | streetAddress }			X.521 5.3		
28	OrganizationalAttributeSet ATTRIBUTE ::= { PostalAttributeSet | LocaleAttributeSet | businessCategory }			X.521 5.4		[3]

[1]	[X520], [RFC4519]: serial number of a device [RFC3739]: this attribute is used to disambiguate *subject* DNames of qualified certificates, e.g. if a CA would need to issue certificates to different entities, that otherwise have the same DName **Common PKI Profile:** The interpretation of this attribute is as in [RFC3039] and refers to the instance (person or organization) represented by the DName, i.e. to the person, even if the DName indicates an affiliation of the person in form of an organization attribute.
[2]	[X.520]: occupation of some common object, e.g. person or organization [RFC4519] 5.16: This attribute describes the kind of business performed by an organization. **Common PKI Profile:** the interpretation of this attribute is as in [X520], i.e. occupation of a person or organization
[3]	**Common PKI Profile:** These attribute set definitions are not identical with those in X.521. Attributes not listed in this table, being not relevant in this specification, have been left out.

Table 2: Object Classes

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT LDAP SERVER	REFERENCES		NOTES
				RFC	TABLE	
	X.509 OBJECT CLASSES					
1	(2.5.6.0 NAME 'top' ABSTRACT MUST objectClass)	abstract class to derive other classes below	no relevance	RFC4512 2.4.1		
2	(2.5.6.2 NAME 'country' SUP top STRUCTURAL MUST c MAY (searchGuide $ description))	class to define country entries in the DIT	++	RFC4519 3.2		
3	(2.5.6.4 NAME 'organization' SUP top STRUCTURAL MUST o MAY (userPassword $ searchGuide $ seeAlso $ businessCategory $ x121Address $ registeredAddress $ destinationIndicator $ preferredDeliveryMethod $ telexNumber $ teletexTerminalIdentifier $ telephoneNumber $ internationalISDNNumber $ facsimileTelephoneNumber $ street $ postOfficeBox $ postalCode $ postalAddress $ physicalDeliveryOfficeName $ st $ l $ description))		++	RFC4519 3.8		
4	(2.5.6.5 NAME 'organizationalUnit' SUP top STRUCTURAL MUST ou MAY (businessCategory $ description $ destinationIndicator $ facsimileTelephoneNumber $ internationalISDNNumber $ l $ physicalDeliveryOfficeName $ postalAddress $ postalCode $ postOfficeBox $ preferredDeliveryMethod $ registeredAddress $ searchGuide $ seeAlso $ st $ street $ telephoneNumber $ teletexTerminalIdentifier $ telexNumber $ userPassword $ x121Address))		++	RFC4519 3.11		

5	(2.5.6.6 NAME 'person' SUP top STRUCTURAL MUST (sn $ cn) MAY (userPassword $ telephoneNumber $ seeAlso $ description))		++	RFC4519 3.12		
	PKIX-SPECIFIC OBJECT CLASSES					
6	(2.5.6.15 NAME 'strongAuthenticationUser' DESC 'X.521 strong authentication user' SUP top AUXILIARY MUST userCertificate)		6 or 7: ++	RFC4523 5.5		[1]
7	(2.5.6.21 NAME 'pkiUser' DESC 'X.509 PKI User' SUP top AUXILIARY MAY userCertificate)		6 or 7: ++	RFC4523 5.1		
8	(2.5.6.16 NAME 'certificationAuthority' DESC 'X.509 certificate authority' SUP top AUXILIARY MUST (authorityRevocationList $ certificateRevocationList $ cACertificate) MAY crossCertificatePair)		8 or 9: ++	RFC4523 5.7		[2]
9	(2.5.6.22 NAME 'pkiCA' DESC 'X.509 PKI Certificate Authority' SUP top AUXILIARY MAY (cACertificate $ certificateRevocationList $ authorityRevocationList $ crossCertificatePair))		8 or 9: ++	RFC4523 5.2		
10	(1.3.6.1.4.1.1466.344 NAME 'dcObject' SUP top AUXILIARY MUST dc)	An auxiliary class defined in X.500 style to contain a *domainComponent* attribute	+-	RFC4519 3.3		
	COMMON PKI-SPECIFIC OBJECT CLASSES					
11	(2.262.1.10.3.6 NAME 'pkiUserData' DESC 'joint-iso-ccitt(2) bmpt(262) telekom(1) security(10) objectClass(3) pkiUserData(6)' SUP top AUXILIARY MAY (countryName $ serialNumber $ givenName $		++			

	title $ postalAttributeSet $ organizationName $ organizationalUnitName $ organizationalAttributeSet)))					
12	(2.262.1.10.3.7 NAME 'pkiCaData' DESC 'joint-iso-ccitt(2) bmpt(262) telekom(1) security(10) objectClass(3) pkiCaData(7)' SUP top AUXILIARY MUST commonName MAY deltaRevocationList)		++			
[1]	[RFC4523]: This object class is deprecated in favor of *pkiUser*.					
[2]	[RFC4523]: This object class is deprecated in favor of *pkiCA*.					

Table 3: Entries of the Proposed Directory Schema

#	ENTRY NAME	ENTRY STRUCTURE	SEMANTICS	REFERENCES	NOTES
1	COUNTRY	Object class: Country Mandatory attributes: countryName (DName attribute)	This entry is the *root* entry of the DIT in the proposed schema.	T2.#2	
2	ORGANIZATION	Object class: Organization Mandatory attributes: organizationName (DName attribute) Auxiliary object class: pkiCA Optional attributes: caCertificate authorityRevocationList crossCertificatePair certificateRevocationList Auxiliary object class: pkiCAData: Mandatory attributes: commonName Optional attributes: deltaRevocationList	This entry corresponds to a certification authority or a trust center. Each authority MUST be represented by exactly one such entry. The *organizationName* DName-attribute MUST contain the *organizationName* of the authority in the same form as in the issuer field the certificates it issues. If the authority issues certificates for other CAs, then this entry MAY contain: self-signed root-certificates or CA-certificates of the authority, an ARL and/or cross certificates of those certificates and/or a common CRL of CA certificates issued by all signing instances of the authority, If the authority issues certificate for end entities, then the entry MAY contain: a common CRL (and optionally a delta-CRL) of end entity certificates issued by all signing instances of the authority. (Signing instances are represented by subordinate *ORGANIZATION UNIT* entries, see below).	T2.#3,9,11	[1]

3	ORGANI-ZATIONAL UNIT	Object class: OrganizationalUnit Mandatory attributes: organizationalUnitName (DName attribute) Auxiliary object class: pkiCA Optional attributes: caCertificate certificateRevocationList Auxiliary object class: pkiCAData: Mandatory attributes: commonName Optional attributes: deltaRevocationList	This entry corresponds to <u>exactly one</u> signing instance of a certification authority, i.e. to a CA-certificate. Different CA-certificates of a certification authority are stored in different entries of the DIT. The *organizationalUnitName* DName-attribute MUST contain the *commonName* of the signing instance as written in the issuer field of the certificates that have been signed by this instance. This entry MAY optionally contain: - <u>either</u>: exactly one CA-certificate, cross certificates of this CA certificate and/or a CRL (and optionally a delta-CRL) of certificates issued by the CA. - <u>or</u>: a certificate for CRL-signing (DIR-certificate) and corresponding CRLs (and optionally delta-CRLs), if the entry represents a *CRLDistributionPoint* of an indirect CRL. - <u>or</u>: a certificate for OCSP-signing (OCSP-certificate) - <u>or</u>: a certificate for TSP-signing (TSP-certificate) For search facilities, the mandatory *commonName* attribute MUST contain the same *commonName* as the *organizationalUnitName* attribute.	T2.#4,9,11	[2] [3]
4	COMMON NAME	Object class: Person Mandatory attributes: commonName (DName attribute) Optional attributes: surname Auxiliary object class: pkiUser Optional attributes: userCertificate Auxiliary object class: pkiUserData: Optional attributes: countryName | serialNumber | given Name | title | postalAttributeSet | organizationName | organizationalUnitName | organizationalAttributeSet }	This entry corresponds to <u>exactly one</u> end entity certificate. Different certificates of an end entity are stored in different entries of the DIT. The *commonName* DName-attribute MUST be build according to the following pattern: <subject commonName>SER:<cert.serial number> The optional attributes of this entry MAY contain an arbitrary subset of the attributes included in the subject DName of the end entity certificate and serve for search purposes. It is especially RECOMMENDED to include the *serialNumber* attribute, if several users exist with the same *commonName* and *serialNumber* has been used by the CA to distinguish among them, as recommended by [RFC3039]. When used in this context, *businessCategory* refers to the occupation or profession of the user.	T2.#2,7,10	

[1]	**Common PKI Profile:** When using this schema, *organizationName* MUST be unique among all certification authorities of the PKI.
[2]	**Common PKI Profile:** When using this schema, *commonName* MUST be unique among all CA certificates of a certification authority.

2.2 Access Protocol

Basically, only read (reading information at a well-defined entry) and search (searching for an entry with specific attributes) operations will be performed by Common PKI-conforming clients. The following operations MUST be supported by all Common PKI-compliant servers and clients:

Table 4: Access Operations

#	OPERATION	DESCRIPTION	REFERENCES RFC	NOTES
1	bind	An LDAP session will always be opened with a bind operation. Since certificates and CRLs are signed documents, no security measures have to be met when reading or searching the directory. Hence, clients MUST always request the version 3, 'anonymous' session, which is indicated by NULL parameters in the *name* and *authentication* fields. More closely, *name* contains an empty string in this case whereas *authentication* contains the *simple* choice option filled with an empty octet string. Servers MUST allow anonymous read and search requests.	RFC4511 4.2	
2	unbind	Closes or aborts an LDAP session.	RFC4511 4.3	
3	read a particular end entity certificate	End entity certificates can be requested by a client by starting a single-level search at the *COMMON NAME* entry of the end entity. The DName of this entry can be constructed by the client as follows: C=<countryName of issuer>,O=<organizationName of issuer>,OU=<commonName of issuer>, CN=<commonName of subject>,SER=:<cert.serial number>	RFC4511 4.5	
4	read a particular CA	CA certificates can be requested by a client by starting a single-level search at the *ORGANIZATIONAL UNIT* entry of the end entity. The DName of this entry can be constructed by the client as follows: C=<countryName of issuer>,O=<organizationName of issuer >,OU=<commonName of issuer >	RFC4511 4.5	
5	read the certificate of the issuer of a CA certificate	This certificate is stored at an *ORGANIZATION* entry, superior to the entry of the issuing (signing) instance. The certificate can be requested by a client by starting a single-level "search" at that entry. The DName of this entry can be constructed by the client as follows: C=<countryName of issuer>,O=<organizationName of issuer > As this node might contain several certificates, the client must still select the proper one by comparing the issuer of the CA certificate with the subject DName of the returned certificates.	RFC4511 4.5	

6	read the CRL (or delta-CRL) corresponding to an end entity certificate	CRLs are stored either at an *ORGANIZATION* entry for all signing instances of a CA (indirect CRL), at *ORGANIZATIONAL UNIT* entry for a particular signing instance or at a *CRLDistributionPoint*, that is indicated in the certificate that is to be validated. In the former two cases, the CRL can be obtained by starting a subtree-search at the *ORGANIZATION* entry. The DName is as follows: C=<countryName of issuer>,O=<organizationName of issuer > In the latter case, a single level search at the *CRLDistributionPoint* entry (of type *ORGANIZATIONAL UNIT*) will return the CRL.	RFC4511 4.5	
7	read the CRL (or delta-CRL) corresponding to a CA certificate	The CRL can be found by means of a single-level search either at an *ORGANIZATION* entry or in a *CRLDistributionPoint*, indicated in the certificate. DNames are formatted as above.	RFC4511 4.5	
8	search for certificates of an end entity	Using subject DName attributes, a subtree-search can be started either at an *ORGANIZATION* or at an *ORGANIZATIONAL UNIT* entry. The more attribute types are supported by the *PkiUserData* class, the higher the chance to locate exactly the certificate entries of the end entity.	RFC4511 4.5	

3 Directory Access via OCSP

The Online Certificate Status Protocol (OCSP) enables applications to determine the (revocation) state of an identified certificate. OCSP may be used to satisfy some of the operational requirements of providing more timely revocation information than is possible with CRLs and may also be used to obtain additional status information. An OCSP client issues a status request to an OCSP responder and suspends acceptance of the certificate in question until the responder provides a response. This protocol specifies the data that needs to be exchanged between an application checking the status of a certificate and the server providing that status.

3.1 Protocol Elements

Table 5 and Table 6 specify the OCSP request message. Due to the flexible syntax, OCSP responses can be of various types (Table 7). There is one basic type of response, *BasicOCSPResponse* (Table 8), that MUST be supported by all PKIX-conforming clients and responders.

Table 5: OCSPRequest

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**OCSPRequest** ::= SEQUENCE {				4.1.1		
2	tbsRequest TBSRequest,	The requestor MAY sign the DER-encoding of this "to be signed" part of the data structure.				#3	[1]
3	optionalSignature [0] EXPLICIT Signature OPTIONAL }	The optional signature of the requestor	+-	+-		#10	[1]
4	**TBSRequest** ::= SEQUENCE {				4.1.1		
5	version [0] EXPLICIT OCSPVersion DEFAULT v1,	Version number of the OCSP protocol				#9	
6	requestorName [1] EXPLICIT GeneralName OPTIONAL,	Name of the requestor	+-	+-	RFC5280 4.2.1.7	P1.T8.#2	[1]
7	requestList SEQUENCE OF Request,	List of single status requests				T6	[2]
8	requestExtensions [2] EXPLICIT Extensions OPTIONAL }	OCSPRequest extensions	+-	++	RFC5280 4.1	T9, P1.T9	
9	**OCSPVersion** ::= INTEGER { v1(0) }				4.1.1		
10	**Signature** ::= SEQUENCE {				4.1.1		[1]
11	signatureAlgorithm AlgorithmIdentifier,	An identifier of the signature algorithm used by the requestor to sign the request			RFC5280 4.1.1.2	P1.T4	
12	signature BIT STRING,	The signature of the requestor					
13	certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }	Certificates that are relevant for the verification of the signature	+-	++			[1]

[1]	[RFC2560]: The requestor MAY choose to sign the request message, e.g. when the responder requires an authentication of users. In this case, the requestor MUST specify its name in the *requestorName* field (#6) and MAY include certificates in the *certs* field (#13) that help the responder to verify the signature. **Common PKI Profile:** If the requestor chooses to sign the request message, *requestorName* MUST contain a *directoryName* with the *subject* DName of the signer's certificate. Alternative names MAY additionally be inserted. So that the request can be validated, *certs* SHOULD contain all certificates of a certificate path, but MUST at least contain the requestor's signing certificate. Responders may choose not to verify the signature, if the OCSP service is publicly available.
[2]	**Common PKI Profile:** the list MUST contain at least one single request.

Table 6: (Single) Request

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	`Request ::= SEQUENCE {`				4.1.1		
2	`  reqCert  CertID,`	Uniquely identifies the certificate being requested				#4	
3	`  singleRequestExtensions [0] EXPLICIT Extensions OPTIONAL }`	(Single) Request extensions	+-	++	RFC5280 4.1	T9, P1.T9	
4	`CertID ::= SEQUENCE {`	Uniquely identifies the certificate being requested by identifying the public key (not certificate!) of its issuer and its serial number.			4.1.1		
5	`  hashAlgorithm  AlgorithmIdentifier,`	Hash algorithm to build hash values below			RFC5280 4.1.1.2	P1.T4	[1]
6	`  issuerNameHash OCTET STRING,`	Hash of issuer's DER-encoded DName, as it occurs in the certificate being requested					
7	`  issuerKeyHash  OCTET STRING,`	Hash of the DER-encoded public key of the issuer of the certificate being requested. Calculated over the public key (excluding tag, length and unused bits in the BIT STRING representation).					[2]
8	`  serialNumber  CertificateSerialNumber }`	Serial number of the certificate being requested			RFC5280 4.1.2.2	P1.T2	
[1]	**Common PKI Profile:** The hash functions to use for certID are defined in Table 1 of Part 6.						
[2]	RFC2560: The hash of the public key is included here, so that the issuer can be identified even in the case, when DNames of two different CAs are accidentally identical.						

Table 7: OCSPResponse

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**OCSPResponse** ::= SEQUENCE {				4.2.1	#4	
2	responseStatus OCSPResponseStatus,	Processing status of the request					
3	responseBytes [0] EXPLICIT ResponseBytes OPTIONAL }	Response data is returned here, if the request be successfully processed	+-	++		#12	
4	**OCSPResponseStatus** ::= ENUMERATED {				4.2.1		
5	successful (0),	Response has valid confirmation	++	++			
6	malformedRequest (1),	Illegal request format, not conforming to the OCSP syntax	++	++			
7	internalError (2),	The OCSP responder reached an inconsistent internal state. The query should be retried, potentially with another responder.	++	++			
8	tryLater (3),	The OCSP responder is in operational status, but temporarily unable to return a status.	++	++			
9		Value '4' is not used.					
10	sigRequired (5),	The server requires the client to sign the request.	++	++			
11	unauthorized (6) }	The client is not authorized to query the server.	++	++			
12	**ResponseBytes** ::= SEQUENCE {				4.2.1		
13	responseType OBJECT IDENTIFIER,	indicates the type of *response*					[1]
14	response OCTET STRING }	DER-encoding of the response data					[1]
[1]	RFC2560: In this profile, only response type BasicOCSPResponse is defined (Table 8). This response type MUST be supported by all conforming clients and responders.						

Table 8: BasicOCSPResponse

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp** OBJECT IDENTIFIER ::= { id-ad-ocsp }		++	++	4.2.1		
2	**id-pkix-ocsp-basic** OBJECT IDENTIFIER ::= { id-pkix-ocsp 1 }	The OID to be used in conjunction with *BasicOCSPRequest*.	++	++	4.2.1		
3	**BasicOCSPResponse** ::= SEQUENCE {		++	++	4.2.1		
4	tbsResponseData ResponseData	The responder signs the DER-encoding of this "to be signed" part of the data structure.			4.2.1	#8	
5	signatureAlgorithm AlgorithmIdentifier,	An identifier of the signature algorithm used by the responder to sign ResponseData			RFC5280 4.1.1.2	P1.T4	
6	signature BIT STRING,	The signature of the responder represented as BIT STRING					[1] [2]
7	certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL }	Certificates that are relevant for the verification of the signature	+-	+	RFC5280 4.1.1	P1.T1	[3]
8	**ResponseData** ::= SEQUENCE {				4.2.1		
9	version [0] EXPLICIT Version DEFAULT v1,	Version of BasicOCSPResponse			RFC5280 4.1.2.1	P1.T2	
10	responderID ResponderID,	Identifier of the responder				#14	
11	producedAt GeneralizedTime,	Time of signing the response					[4]
12	responses SEQUENCE OF SingleResponse,	List of single responses, for all but NOT necessarily in order of the single requests				#18	
13	responseExtensions [1] EXPLICIT Extensions OPTIONAL }	*BasicOCSPResponse* extensions	+-	++	RFC5280 4.1	T9, P1.T9	
14	**ResponderID** ::= CHOICE {				4.2.1		
15	byName [1] EXPLICIT Name,	DName of the responder	+-	++	RFC5280 4.1.2.4	P1.T5	[5]
16	byKey [2] EXPLICIT KeyHash }	Hash of responders public key (see below)	+-	++		#17	[5]
17	**KeyHash** ::= OCTET STRING	SHA-1 hash of responders public key (excluding tag, length and unused bits in the BIT STRING representation)			4.2.1		[6]
18	**SingleResponse** ::= SEQUENCE {	A single response			4.2.1		
19	certID CertID,	Uniquely identifies the queried certificate			4.1.1	T6.#4	[7]
20	certStatus CertStatus,	Certificate status				#24	
21	thisUpdate GeneralizedTime,	The time at which the status being indicated was known to be correct.					[4] [8]

22	`nextUpdate [0] EXPLICIT GeneralizedTime OPTIONAL,`	The time at or before which more up-to-date information will be available.	+-	++			[4] [8]
23	`singleExtensions [1] EXPLICIT Extensions OPTIONAL }`	*SingleResponse* extensions	+-	++	RFC5280 4.1	T9, P1.T9	
24	**`CertStatus`** `::= CHOICE {`				4.2.1		
25	`good [0] IMPLICIT NULL,`	indicates that the certificate IS NOT revoked.	++	++			[9]
26	`revoked [1] IMPLICIT RevokedInfo,`	indicates that the certificate IS revoked, either permanently or temporarily (on hold).	++	++		#28	
27	`unknown [2] IMPLICIT UnknownInfo }`	indicates that the responder does not know about the certificate being requested	++	++		#31	
28	**`RevokedInfo`** `::= SEQUENCE {`				4.2.1		
29	`revocationTime GeneralizedTime,`	time of revocation					
30	`revocationReason [0] EXPLICIT CRLReason OPTIONAL }`	reason of revocation	+-	+-	RFC5280 5.3.1	P1.T38	
31	**`UnknownInfo`** `::= NULL`				4.2.1		

[1]	RFC2560: All definitive response messages (*responseStatus=successful*) MUST be digitally signed. The key used to sign the response MUST belong to one of the following: (a) the CA who issued the certificate(s) in question (b) a Trusted Responder whose public key is trusted by the responder (and installed directly at the client), affected certificates include the *OCSPNocheck* extension. (c) a CA Designated Responder (Authorized Responder) who holds a specially marked certificate issued directly by the CA, indicating in the *ExtendedKeyUsage* extension that the responder may issue OCSP responses for that CA. [DraftOCSPv2]: The above list is extended with the following option: (d) a key associated with the CA (i.e. a CA's *OCSP-signing* key) **Common PKI Profile:** As described in (d) above, the responder's certificate MAY be issued for the CA by some other trusted authority. This set-up allows clients to obtain reliable status information even if the key of the issuing CA has been compromised. This configuration is RECOMMENDED for all Common PKI-compliant CAs. Clients MUST NOT rely on the authorization rules, i.e. they MUST accept responder certificates issued by any trusted authorities.
[2]	RFC2560: If an OCSP responder knows that a particular CA's private key has been compromised, it MAY return the revoked state for all certificates issued by that CA. **Common PKI Profile:** Reliable status information can be delivered, when using the setup (d) described in [1]. In such a configuration, OCSP responders SHOULD in return the actual status, i.e. SHOULD NOT return the *revoked* state, unless the certificate has been explicitly revoked.
[3]	**Common PKI Profile:** So that the response can be validated, *certs* SHOULD contain all certificates of a certificate path, but MUST at least contain the responder's signing certificate.
[4]	**Common PKI Profile:** Time instances MUST be specified using the format YYYYMMDDhhmmssZ.
[5]	**Common PKI Profile:** As all certificates of the certificate path are included in the response, it is not critical which CHOICE option is used here. If *byName* is given, it MUST contain the same DName as the responders *subject* field.
[6]	Remark: If the responder uses the CA public key, this value is identical to the *keyIdentifier* field of the *AuthorityKeyIdentifier* extension in the certificate being requested, if computed according to method a) in P1.T11.[2].
[7]	**Common PKI Profile:** the *certID* in a *SingleResponse* MUST be identical to that in the corresponding (single) *Request*. (T6.#4)

[8]	RFC2560: The *thisUpdate* and *nextUpdate* fields define a recommended validity interval. This interval corresponds to the {*thisUpdate*, *nextUpdate*} interval in a CRL, e.g. if status information has been obtained from a CRL. Responses whose *thisUpdate* time is later than the local system time SHOULD be considered unreliable. Responses whose *nextUpdate* value is earlier than the local system time value SHOULD be considered unreliable. If *nextUpdate* is absent, the responder indicates that newer information is available all the time.
[9]	RFC2560: ATTENTION! As status information delivered by OCSP may be obtained from CRLs, *good* does not necessarily mean that the certificate was ever issued or that the response time lies within the certificate's validity interval. Additional information regarding the status, such as *positive statement of availability or validity*, may be included in response extensions. **Common PKI Profile:** This Common PKI-specification defines the private single response extension *CertHash* that may deliver a *positive statement about the availability* of a certificate. Refer to Table 15 for more information.

Table 9: An overview of OCSP extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT		REFERENCES		NOTES
					GEN	PROC	RFC 2560	TABLE	
	RFC 2560 EXTENSIONS								
1	Nonce	{id-pkix-ocsp 2}	extension *in OCSPRequest* and *ResponseData*: given by a client in a request and expected in the response, aims to prevent replay attacks.	--	+-	+-	4.4.1	T10	
2	CrlID	{id-pkix-ocsp 3}	extension *in ResponseData*: if the responder obtains status information *revoked* or *onHold* from a CRL, the CRL may be identified here.	--	+-	+-	4.4.2	T11	
3	AcceptableResponses	{id-pkix-ocsp 4}	*OCSPRequest* extension: The client may specify in a request, which kinds of responses it expects	--	+-	+-	4.4.3	T12	
4	ArchiveCutoff	{id-pkix-ocsp 6}	extension *in ResponseData* extension: a responder MAY choose to retain revocation information beyond the certificate's expiry date. In this case, the responder SHOULD include the certificate's *cutoff* date, which is obtained by subtracting the retention period from the *producedAt* time.	--	+	++ (RFC +-)	4.4.4	T13	
5	CRL entry extensions		*SingleResponse* extension: All CRL entry extensions may occur in single responses.	--	+-	+-	4.4.5	P1.T37	
6	ServiceLocator	{id-pkix-ocsp 7}	(Single) *Request* extension: a client may request the responder to forward the request to another responder, which is known to be the authorized responder for the queried certificate.	--	+-	+-	4.4.6	T14	
	COMMON PKI PRIVATE EXTENSIONS								
7	CertHash (Positive Statement)	{1 3 36 8 3 13}	*SingleResponse* extension: the responder may include this extension in a response to send the hash of the requested certificate to the requestor. This hash serves as evidence that the certificate is known to the responder (i.e. it is available in the queried directory) and will be used as means to provide a *positive statement of availability*.	--	+-	++		T15 P1.T43.#4	

3.1.1 Standard OCSP Extensions

Table 10: Nonce

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp-nonce** OBJECT IDENTIFIER ::= {id-pkix-ocsp 2}				4.4.1		
2	**Nonce** ::= ANY		+-	+-			[1]
[1]	RFC2560: No syntax is given for this extension value. **Common PKI Profile:** Use the ASN.1 type ANY on this place, in order for clients to be able to parse any returned object type here. As supporting this extension by Common PKI-compliant responders is optional, clients MUST NOT rely on responders returning the nonce.						

Table 11: CrlID

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp-crl** OBJECT IDENTIFIER ::= {id-pkix-ocsp 3}				4.4.2		
2	**CrlID** ::= SEQUENCE {	Specifies a CRL which has been used by the responder to obtain status information	+-	+-	4.4.2		
3	crlUrl [0] EXPLICIT IA5String OPTIONAL,	URL at which the CRL is available	+-	+-			
4	crlNum [1] EXPLICIT INTEGER OPTIONAL,	CRL number	+-	+-			
5	crlTime [2] EXPLICIT GeneralizedTime OPTIONAL }	time of CRL creation	+-	+-			

Table 12: AcceptableResponses

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp-basic** OBJECT IDENTIFIER ::= {id-pkix-ocsp 1}	OID denoting response type *BasicOCSPResponse*.			4.2.1	Table 8.#2	
2	**id-pkix-ocsp-response** OBJECT IDENTIFIER ::= {id-pkix-ocsp 4}	OID to be used with extension *AcceptableResponses*.			4.4.3		
3	**AcceptableResponses** ::= SEQUENCE OF OBJECT IDENTIFIER		+-	+-	4.4.3		[1]
[1]	RFC2560: Responders and clients MUST be capable of responding/receiving BasicOCSPResponse. **Common PKI Profile:** Clients MAY include this extension in the request. If included, the AcceptableResponses MUST contain id-pkix-ocsp-basic. If included in the request, the responder MUST reply with an BasicOCSPResponse object. The responder MAY reply with an BasicOCSPResponse, even if it does not recognize this extension.						

Table 13: ArchiveCutoff

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp-archive-cutoff** OBJECT IDENTIFIER ::= {id-pkix-ocsp 6}				4.4.4		
2	**ArchiveCutoff** ::= GeneralizedTime		+	++	4.4.4		[1]
[1]	RFC2560: A responder MAY choose to retain revocation information beyond the certificate's expiry date. In this case, the responder SHOULD include the certificate's "cutoff" date, which is obtained as follows: cutoff date = producedAt time - retention period. Applications would use the cutoff date to contribute to a proof that a digital signature was (or was not) reliable on the date it was produced even if the certificate needed to validate the signature has long since expired. Remark: The condition cutoff date > expiry date (which is identical to the condition: producedAt time > expiry date + retention period) indicates the fact, that status information returned by the OCSP responder is not any more reliable, i.e. status information may have been deleted. **Common PKI Profile:** The verification of a certificate at some time beyond its expiry date is desirable for message authentication and especially important for non-repudiation services. There are three approaches to provide for status information beyond the expiry date: (a) status information MAY be retained by the OCSP responder and the ArchiveCutoff extension included in the response, (b) status information MAY be retained by the OCSP responder and a *positive statement* ("certificate is available and has not been revoked") included in the response, (c) a valid OCSP response message MAY be included in the digital signature, as proposed in the ETSI standard ES 201 733, so that clients need not query the responder. Common PKI-compliant CAs MUST provide one of the above mechanisms to provide status information on certificates issued for authentication and non-repudiation purposes. Compliant clients MUST support all these mechanisms.						
[2]	**Common PKI Profile:** ArchiveCutoff MUST have the format YYYYMMDD000000Z.						

Table 14: ServiceLocator

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-pkix-ocsp-service-locator** OBJECT IDENTIFIER ::= {id-pkix-ocsp 7}				4.4.5		
2	**ServiceLocator** ::= SEQUENCE {		+-	+-	4.4.5		[1]
3	issuer Name,				RFC5280 4.1.2.4	P1.T5	
4	locator AuthorityInfoAccess OPTIONAL }				RFC5280 4.2.2.1	P1.T23	
[1]	**Common PKI Profile:** Compliant certificates always contain directory access information. Hence, clients are able to find the authorized responder for that certificate. This extension MAY still be supported and included, e.g. if clients within some community are configured to query a well-known responder and support this option.						

3.1.2 Common PKI Private OCSP Extensions

Table 15: CertHash (Positive Statement)

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	TABLE	
1	**id-commonpki-at-certHash** OBJECT IDENTIFIER ::= {1 3 36 8 3 13}						
2	**CertHash** ::= SEQUENCE {		+-	++			[1]
3	hashAlgorithm AlgorithmIdentifier,	The identifier of the algorithm that has been used the hash value below.			RFC5280 4.1.1.2	P1.T4	
4	certificateHash OCTET STRING }	A hash over the DER-encoding of the entire PKC or AC (i.e. NOT a hash over *tbsCertificate*).					

[1] [RFC2560]: The "good" state indicates a positive response to the status inquiry. At a minimum, this positive response indicates that the certificate is not revoked, but does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval. Response extensions MAY be used to convey additional information on assertions made by the responder regarding the status of the certificate such as positive statement about issuance, validity, etc.

Common PKI Profile: The responder may include this extension in a response to send the hash of the requested certificate to the responder. This hash is cryptographically bound to the certificate and serves as evidence that the certificate is known to the responder (i.e. it has been issued and is present in the directory). Hence, this extension is a means to provide a *positive statement of availability* as described in T8.[8]. As explained in T13.[1], clients may rely on this information to be able to validate signatures after the expiry of the corresponding certificate. Hence, clients MUST support this extension.

If a *positive statement of availability* is to be delivered, this extension syntax and OID MUST be used.

A further note on security: Including the hash of the queried certificate in the response prevents impersonation attacks of the following scenario:

Mallory manages to get the private key of a CA. The corresponding CA certificate is immediately revoked. Using the stolen CA key, Mallory creates a faked certificate with the same serial number as an existing one (the original) and containing a new public key. Using the corresponding private key, Mallory signs a message and sends it, along with the faked certificate, to Alice. Alice succeeds to mathematically verify the signature and wants to check the state of the received certificate by sending its serial number to the OCSP server. The server returns the answer *good*, if the original certificate has not been revoked. Having received the response *good*, Alice thinks that the (actually faked) certificate is O.K. and accepts the signature. She is unable to detect that the response corresponds to another certificate than what she was asking about.

This threat is apparently not handled by PKIX documents. The security gap can be closed by including either the certificate or a fingerprint of it in the response, respectively in the *positive statement* as proposed here. It is crucial that the signature of the responder can be reliably verified. Hence, departing from the practice proposed by RFC2560, the certificate of the responder SHOULD be issued by some independent the CA, i.e. not by the CA the certificates of which the responder provides information about. This configuration is described in T8.[1], item d).

3.2 Certificate Contents

3.2.1 Queried certificates

[RFC2560]: In order to convey to OCSP clients a well-known point of information access, CAs SHALL provide the capability to include the *AuthorityInfoAccess* extension (defined in [RFC5280], section 4.2.2.1) in certificates that can be checked using OCSP. Alternatively, the *accessLocation* for the OCSP provider may be configured locally at the OCSP client. CAs that support an OCSP service, either hosted locally or provided by an Authorized Responder, MUST provide for the inclusion of a value for a *uniformResourceIndicator* (URI) *accessLocation* and the OID value *id-ad-ocsp* for the *accessMethod* in the *AccessDescription SEQUENCE*. The value of the *accessLocation* field in the subject certificate defines the transport (e.g. HTTP) used to access the OCSP responder and may contain other transport dependent information (e.g. a URL).

Common PKI Profile: If status information can be obtained via OCSP for a certificate, the *AuthorityInfoAccess* containing an URL for HTTP transport extension MUST be included.

3.2.2 Responder's certificates

[RFC2560]: a certificate's issuer MUST either sign the OCSP responses itself or it MUST explicitly designate this authority to another entity. OCSP signing delegation SHALL be designated by the inclusion of *id-kp-OCSPSigning* in an *extendedKeyUsage* certificate extension included in the OCSP response signer's certificate. This certificate MUST be issued directly by the CA that issued the certificate in question.

[DraftOCSPv2]: This draft allows another trusted authority to certify a key associated with the CA as the CA's *OCSP-signing* key.

Common PKI Profile: As proposed in [DraftOCSPv2], the responder's certificate MAY be issued for the CA by some other trusted authority. The responders certificate, Regardless of whether issued by the CA itself or issued for the CA by some other authority, the responder's certificate MUST include the *extendedKeyUsage* extension with the *id-kp-OCSPSigning* OID. As described in 4.2.2.2 of RFC2560, clients MUST involve this extension in the verification process, when validating an OCSP response.

[RFC2560]: OCSP clients need to know how to check that an authorized responder's certificate has not been revoked. CAs may choose to deal with this problem in one of three ways:

(a) A CA may specify that an OCSP client can trust a responder for the lifetime of the responder's certificate. The CA does so by including the extension *id-pkix-ocsp-nocheck*.
(b) A CA may specify how the responder's certificate be checked for revocation. This can be done using *CRLDistributionPoints* if the check should be done using CRLs or CRL Distribution Points, or *AuthorityInformationAccess* if the check should be done in some other way. Details for specifying either of these two mechanisms are available in [RFC5280].
(c) A CA may choose not to specify any method of revocation checking for the responder's certificate, in which case, it would be up to the OCSP client's local security policy to decide whether that certificate should be checked for revocation or not.

Common PKI Profile: Responder's certificates MUST always include directory access

information, i.e. use option (b) above.

3.3 Transport over HTTP

There is no specific transport protocol specified in RFCs for OCSP. Similarly, there is no dedicated "well-known" port reserved for OCSP. Common PKI compliant systems MUST employ the Hypertext Transfer Protocol (HTTP) [RFC2616] to transport OCSP messages between clients and a server. If no port number is provided in the corresponding URL, the commonly used port No. 80 MUST be used. Using HTTP has the advantage that software components are easy to implement and that transport over firewalls and proxies usually does not require any special configuration. It is furthermore possible to provide for secure transmission using Transport Layer Security (TLS) or Secure Socket Layer (SSL). Note that since all relevant OCSP messages are signed and carry only public information, it is not indeed necessary to provide for such additional security.

An OCSP request will be sent to the responder by means of the *POST* method. The request message MUST include the following lines:

```
POST <responder URL>
...
Content-Type: application/ocsp-request
Content-Length: ...
<the DER-encoded OCSPRequest object >
...
```

If the POST-request could be processed, the server MUST return response status 200 (OK) and MUST include the DER-encoding of the resulting *OCSPResponse* object in the response message. No transport encoding (e.g. to base-64 encoding) is to be applied, i.e. messages are to be transported in unaltered, pure binary form.

4 Directory Access via FTP and HTTP

The standard access mechanism for Common PKI-compliant directories is LDAP v3, which provides access to certificates and CRLs including search and matching facilities. This Common PKI specification is intended to be kept at the "necessary minimum" needed for interoperability of client and server applications of the PKI. Therefore, the transport of certificates and CRLs via email is NOT any longer required to be supported (required by [MTTv2]), whereas the support of FTP and HTTP for the transport as defined in [RFC2585] is optional (just as in [MTTv2]). This means that Common PKI-compliant directory services MAY, but need not make certificates and CRLs available for download via FTP and/or HTTP and respectively that Common PKI-compliant clients MAY but need not be prepared to obtain them in this way.

If a certificate is made available via FTP or HTTP, the corresponding FTP/HTTP-URI MAY be included in the *SubjectAltNames* extension of the certificate. Certificate file names MAY be built according to one of the following patterns:

[ftp|http]://<CAdomain>/<IssuerCommonName>/<uniqueCommonName>.<CertSerialNumber>.cer

[ftp|http]://<CAdomain>/<IssuerCommonName>/<commonName>.<DNserialNumber>.<CertSerialNumber>.cer

If a CRL is made available via FTP or HTTP, the corresponding FTP/HTTP-URI MAY be included in the *SubjectAltNames* extension of the certificate. CRL file names MAY be built according to one of the following patterns:

[ftp|http]://<CAdomain>/<IssuerCommonName>/all.crl

[ftp|http]://<CAdomain>/<IssuerCommonName>/delta.crl (in case of a delta CRL)

Note that the naming of certificates and CRL files corresponds to their DNames in the Common PKI directory schema.

5 Time Stamp Protocol (TSP)

Common PKI–compliant systems MUST apply the protocol defined in [RFC3161] and further profiled in [ETSI-TSP], when offering or accessing time stamp services. Cryptographic algorithms, in particular hash algorithms, SHALL be supported according to the requirements defined in Common PKI Part 6.

Common PKI compliant applications and TSAs MUST transport TSP messages via HTTP. Using HTTP has the advantage that software components are easy to implement and that transport over firewalls and proxies usually does not require any special configuration. It is furthermore possible to provide for secure transmission using Transport Layer Security (TLS), as proposed in [RFC3161].

A time-stamp request will be sent to the TSA by means of the *POST* method. The request message MUST include the following lines:

```
POST <TSA URL>
...
Content-Type: application/time-stamp-request
Content-Length: ...
<the DER-encoded TimeStampReq object >
...
```

If the POST-request could be processed, the server MUST return response status 200 (OK) and MUST include the DER-encoding of the resulting *TimeStampResp* object in the response message. No transport encoding (e.g. to base-64 encoding) is to be applied, i.e. messages are to be transported in unaltered, pure binary form.

No specific method is specified in this version of Common PKI for requestor authentication. A future version shall consider this issue. RFC3161 proposes TLS and CMS for this purpose.

References

[DraftOCSPv2] Online Certificate Status Protocol, version 2, draft-ietf-pkix-ocspv2-02.txt, March 2001

[ETSI-TSP] ETSI TS 101 861 v1.6.1: Time Stamping Profile, January 2006

[ISIS] Industrial Signature Interoperability Specification ISIS, Version 1.2, December 1999, T7 i.Gr., www.t7-isis.de

[MTTv2] MailTrust Version 2, March 1999, TeleTrust Deutschland e.V., www.teletrust.de

[RFC1777] Lightweight Directory Access Protocol, RFC 1777, March 1995

[RFC2252] Lightweight Directory Access Protocol (v3): Attribute Syntax Definitions, RFC 2252, December 1997

[RFC2560] Internet X.509 Public Key Infrastructure - Online Certificate Status Protocol – OCSP, RFC 2560, June 1999

[RFC2585] Internet X.509 Public Key Infrastructure Operational Protocols: FTP and HTTP, RFC 2585, May 1999

[RFC2616] Hypertext Transfer Protocol -- HTTP/1.1, June 1999

[RFC3161] Internet X.509 Public Key Infrastructure - Time Stamp Protocol (TSP), RFC 3161, August 2001

[RFC3281] An Internet Attribute Certificate Profile for Authorization, <draft-ietf-pkix-ac509prof-09.txt>, 8th June 2001

[RFC3739] Internet X.509 Public Key Infrastructure: Qualified Certificates Profile, March 2004

[RFC4510] Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, June 2006

[RFC4511] Lightweight Directory Access Protocol (LDAP): The Protocol, June 2006

[RFC4512] Lightweight Directory Access Protocol (LDAP): Directory Information Models, June 2006

[RFC4513] Lightweight Directory Access Protocol (LDAP): Authentication Methods and Security Mechanisms, June 2006

[RFC4514] Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names, June 2006

[RFC4516] Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator, June 2006

[RFC4517] Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules, June 2006

[RFC4519] Lightweight Directory Access Protocol (LDAP): Schema for User Applications, June 2006

[RFC4522] Lightweight Directory Access Protocol (LDAP): The Binary Encoding Option, June 2006

[RFC4523] Lightweight Directory Access Protocol (LDAP) Schema Definitions for X.509 Certificates, June 2006

[RFC5280] Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile, May 2008

[SigI-A4] Signature Interoperability Specification – Chapter B3: Zeitstempel, Version 3.0, März 1999, GISA (BSI)

[X.509:2005] ITU-T X.509: Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, 2005

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 5

CERTIFICATE PATH VALIDATION

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.2 19.07.2002	First public edition of Part 5.
1.0.2 11.08.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.08.2004	Several editorial changes.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Adapted to new versions of the base standards: - RFC 4510 - RFC 4516 - RFC 4523 - RFC 5280 - X.509:2005 Various corrections and clarifications. Corrections in handing policy identifiers during the certificate path validation. Editorial changes in the pseudo-code for closer familiarity with RFC 5280.

Table of Contents

1 Preface

The purpose of certificate path validation is verifying the binding between an end entity (a user, an organization or a server) and his/her/its public key. This binding is certified by an authority that issues a public key certificate (PKC) for the end entity (EE), which is called the *subject* of the certificate. The subject is identified in the certificate by a distinguished name (DN). Alternative names of the subject, such as email address, can additionally be contained in the certificate. The certificate is authenticated by the signature of the issuing authority over the certificate's content.

Other users, wanting to use the public key of an entity (for encryption or for signature verification), may obtain his/her/its PKC from a public repository or directory. If fetched from a public directory, the relying party needs to be able to verify whether the public key is indeed authentic, i.e. it belongs to the intended communication partner. This can be done by verifying the signature over the entity's certificate by means of the public key of the issuing authority. The authenticity of this authority key must however be checked by verifying the PKC of the authority. This procedure of recursively verifying certificates of issuers of other certificates can be terminated, when a trusted public key or certificate can be used at a verification step. A trusted key or certificate can be obtained from a trusted authority using some reliable out-of-band procedure or mechanism and must be stored securely on the local system. The trusted public key is called a *security anchor* or a *root key*. The chain of certificates up to the trusted key is called *certificate path*, whereas the procedure is called *certificate path validation*.

The Common PKI Specification is intended for *hierarchical* PKIs, where root keys are issued by top-level trusted authorities that issue certificates for other certification authorities (CAs). Such a trusted public key of an authority is usually published in form of a self-signed certificate, i.e. where the issuer of the certificate is the same identity as the subject and which is signed by the private key that corresponds to the certified public key. For the sake of interoperability, Common PKI-compliant authorities MUST publish their public keys in form of self-signed certificates. In this document, it is always assumed that the certificate path includes a trusted self-signed certificate as last element.

For security reasons, some constraints must be checked while validating the certification path. These constraints are specified in certificate extensions, such as *BasicConstraints*, *CertificationPolicy*, *PolicyConstraints* etc., and must be considered while validating the certificate path. Certificates may get revoked before their expiry date. Hence, it is important to obtain up-to-date information from a trusted server about the revocation status of each certificate of the path. The most common technique for providing certificate status information is issuing certification revocation lists (CRLs). Hence, Common PKI-compliant CAs MUST issue CRLs and publish them in an LDAP directory. Optionally, CAs MAY provide an on-line OCSP-service. Information about how to access these LDAP- and OCSP-services is included in the *CRLDistributionPoints* and respectively in the *AuthorityInfoAccess* extensions of all, except root, certificates.

Reliable status information about root certificates cannot be obtained relying on the same trusted root. Typically, no CRLs are issued for self-signed root certificates, as the CRL should be signed using the corresponding root key itself. Hence, no valid CRL can be issued after the root certificate gets revoked. Therefore, some other reliable out-of-band mechanism, such as a communiqué, shall be used in case of revoking a self-signed root certificate. In the path validation algorithm, presented in this specification, root certificates are assumed to be inherently valid. Clients SHOULD offer the possibility to remove trusted root-certificates from the local system or mark them invalid.

A major goal of the Common PKI Specification is to tailor the usage of different certificate extensions in such a manner that an automatic verification of signatures and certificates – i.e. a verification without the interaction or judgement of the relying person - is always possible. This is also a prerequisite for automatic verification performed by non-human end entities, like servers. This part of the Common PKI Specification describes an algorithm for automating the certificate path validation procedure. Conforming applications are not required to implement exactly this algorithm, but they MUST be functionally equivalent with respect to the external behaviour, i.e. a compliant implementation of the verification procedure MUST yield the same result (*valid* or *invalid*) as the presented algorithm, if entering the same certificate(s) and requesting verification for the same time of reference.

The Common PKI Specification is intended to be used in an environment where several root CAs may exist in a hierarchical certification structure, where the CAs may even follow different policies. Cross-certificates may build links among different certification domains. To provide for wide interoperability among CAs and client software, this document specifies an algorithm for building a certificate path to a trusted root in an environment with multiple root CAs and cross-certification; as well as an algorithm for validating that certificate path.

The validation of a certificate involves obtaining and validating up-to-date status information from a directory service. Special attention has been paid throughout the entire specification to provide client software with information in order to be able to locate directory services and to obtain certificates, CRLs and on-line status information. Furthermore, the validation of CRLs and of OCSP-responses has been addressed too.

The certificate path building and validation algorithm has been extended to process end-entity attribute certificates (AC). So that an automatic verification of such paths is always possible, some specific extensions used by the validation procedure must be present in conforming ACs as well. This raises some requirements on the contents of ACs.

2 Certificate Path Validation Procedure

In the following we present a procedure for building and validating a certificate path. Conforming applications are not required to implement exactly this algorithm, but they MUST be functionally equivalent with respect to the external behaviour, i.e. compliant implementations of the validation procedure MUST be able to build some existing certificate path and yield the same result ("valid", "invalid") for this particular path and the same time of reference.

Certificate path validation is influenced by a number of input policy and naming constraint parameters that are specified by the application according to the validation policy of the relying party (refer to T1.#6 below). The validation algorithm described here is generic in the sense that it supports all policy and naming constraints that are supported by the basic path validation algorithm (BPVA) in [RFC5280]. Applications with a fixed, limited set of policy or naming constraint parameters MAY chose not to implement those parts of the algorithm, which will never be active due to the specific input parameter settings. Still, the implementation is considered compliant, if it delivers the same results for the limited parameter set as the generic version. An example is an implementation that never processes naming constraints or one that always inhibits policy mapping.

Many of the data types used in the presented procedure correspond to ASN.1 types, described in Part 1 (Certificate and CRL Profile). These data types borrow the name of the corresponding ASN.1 data type (e.g. *Certificate, Name*). They are defined here as object classes that offer methods for accessing embedded data fields (e.g. *GetIssuer*()), as usual in object-oriented programming. Some new data types are introduced in Table 1.

Table 1: Common Data Types

#	DATA TYPE	DESCRIPTION	RFC	NOTES
1	`typedef enum {` `  RootCACert;` `  SelfIssuedCACert ;` `  CACert;` `  CrossCACert;` `  EndEntityPKC;` `  EndEntityAC;` `} CertType;`	The *CertType* enumeration type is used to classify certificates. Self-signed certificates are certificates where the digital signature may be verified by the public key bound into the certificate. Self-signed root CA certificates are used to convey a public key for use to begin certification paths. Self-issued certificates are CA certificates in which the issuer and subject are the same entity. Self-issued certificates are generated to support changes in policy or for key roll-over operations. Self-issued certificates are not counted, when evaluation path length, naming and policy constraints during path validation. In other CA certificates the issuer and subject are different entities. Regular CA certificates describe a trust relationship between two CAs within one PKI hierarchy. Cross-certificates are typically issued by a CA of one PKI hierarchy to a CA in another PKI hierarchy to create a trust relation on one direction. End entity public key certificates are issued to subjects that are not authorized to issue certificates. Attribute certificates are issued only for end entities.	3.2	
2	`class CertInfo {`	This data structure can be seen as the basic item of the local certificate repository. It is used to		

	`  CertType  certType;` `  bool  revoked;` `  Time  revocTime;` `  CRLReason  revocReason;` `  Time  statusInfoNextUpdate;` `  Certificate  cert;` `  AttributeCertificate acert;` `};`	store one PKC or AC and corresponding information. The *certType* member makes searching for specific certificate types easier. The *revoked* flag is set if the certificate has been revoked. If the certificate has been revoked, *revocTime* contains the time of the revocation, otherwise the date in *validity.notAfter*. If the certificate has been revoked and the reason for that is known, *revocReason* contains the reason of the revocation, otherwise the value 'unspecified'. *statusInfoNextUpdate* is initialized to the date in the *validity.notBefore* field of the certificate and contains the date of the most recent on-line status check respectively the date when CRL information still can be considered as valid, i.e. the date in the *nextUpdate* field, minus 1 second, of the most recently downloaded CRL. Actual implementations may reduce or extend this information.		
3	`typedef vector<CertInfo> CertInfoList;`	The *CertInfoList* type is an ordered list of *CertInfo* objects. This data structure models the local certificate depository too.		
4	`typedef enum {` `  certSigning,` `  crlSigning,` `  ocspSigning,` `  timeStamping,` `  nonRepudiation,` `  dataOrKeyEncryption,` `  dataAuthentication` `} KeyPurpose;`	The *KeyPurpose* enumeration type identifies the key usage options that are relevant for the Common PKI Specification. The usage of a key pair resp. of the corresponding PKC is constrained as indicated in the *BasicConstrains*, the *KeyUsage* and the *ExtendedKeyUsage* extensions. Note that a PKC may possibly be authorized for more than one of the purposes, e.g. a CA certificate may be used to sign certificates and CRLs as well.		
5	`typedef vector<CertPolicyId> PolicyList;`	The *PolicyList* type contains a list of policy OIDs.		
6	`class PathConstraints {` `  PolicyList  userInitialPolicySet,` `  bool  initialExplicitPolicy,` `  bool  initialAnyPolicyInhibit,` `  bool  initialPolicyMappingInhibit,` `  GeneralNames initialPermittedSubtrees,` `  GeneralNames initialExludedSubtrees` `};`	The *PathConstraints* data structure 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. In particular: *userInitialPolicySet* contains a set of initial policy identifiers naming the policies that are acceptable to the relying party or application. The special policy value *anyPolicy* indicates that the relying party is not concerned about certificate policy and accepts any policy. The set must not be empty. The default value is a set with the single value *anyPolicy*. *initialExplicitPolicy* indicates if the relying party requires the path having a valid policy explicitly declared by CAs in the certificates. The default value is *false*, i.e. the relying does not require having an explicitly declared valid policy. Still, a CA in the hierarchy may enforce explicit policy declaration by including the *PolicyConstraints* extension and properly setting the *requireExplicitPolicy* variable. *initialAnyPolicyInhibit* indicates whether the relying party accepts the policy OID *anyPolicy* if it is included in a certificate. The default value is false, i.e. *anyPolicy* is accepted by the relying party as declared policy. Still, a CA in the hierarchy may inhibit processing *anyPolicy* by including the *InhibitAnyPolicy* extension. *initialPolicyMappingInhibit* indicates whether the relying accepts policy mapping. The default	6.1.1 (c) (f) (g) (e)	

		value is *false*, i.e. the relying party allows policy mapping. Still, a CA in the hierarchy may inhibit policy mapping by including the *PolicyConstraints* extension and properly setting the *inhibitPolicyMapping* variable. *initialPermittedSubtrees* indicates for each name type a set of subtrees within which all *subject* and *subjectAltNames* names in all certificates in the path must fall. The default value is an empty *GeneralNames* object, indicating that the relying party is not concerned about such name constraints. CAs may further restrict the constraints by including the *NameConstraints* extension and properly setting the *permittedSubtrees* variable. (h) *initialExcludedSubtrees* indicates for each name type a set of subtrees within which no *subject* and *subjectAltNames* names in the certificates in the path may fall. The default value is an empty *GeneralNames* object, indicating that the relying party is not concerned about such name constraints. CAs may further restrict the constraints by including the *NameConstraints* extension and properly setting the *excludedSubtrees* variable. (i)		
7	`typedef IA5String LdapUrl;`	An URL for accessing a directory over LDAP. As described in [RFC4516], the URL format does not only contain a server address, but parameters for the LDAP-read or search operation.		
8	`typedef IA5String OcspUrl;`	An URL for accessing the OCSP-service of a directory. The standard transport mechanism for OCSP-messages is HTTP.		
9	`class CrlInfo {` `  CertificateList crl;` `};`	The *CrlInfo* structure contains all information about a CRL. For the simplicity of the algorithm description, CRL segmentation is not considered in this document and *CrlInfo* contains merely a CRL object. We only note here that *CrlInfo* should actually be able to contain different segments of a CRL. Different segments of the same CRLs can be identified by the *IssuingDistributionPoint* CRL extension.		
10	`typedef vector<CrlInfo> CrlInfoList;`	The *CrlInfoList* type is an ordered list of *CrlInfo* objects.		

The validation procedure is divided into several subroutines that cover well-defined sub-tasks to be performed – possibly many times – during the validation. The procedure, respectively its subroutines, is presented as pseudo-program-code, using a C++-like syntax and semantics. The main entry point of the procedure is *ValidateCertificate()* (see Table 2). This function expects the 'to be verified' EE certificate, a list of further certificates (all of, some of or more than those in a path to a root trusted by the signing/decrypting party), a set of policies accepted by the relying party or application, and a reference point in time, at which validity is to be investigated. The function returns *true* in case of success and *false* if path building or validation fails. More distinguishing answers and error messages about the performed verification steps and about the exact reasons of failure should be given by applications. Client applications are especially encouraged to perform as many steps of the procedure as possible and return a list of failed actions. The description of the behaviour on failure is not subject of the current version of the Common PKI Specification.

Table 2: ValidateCertificate()

#	PSEUDO-CODE	COMMENTS	RFC	NOTES
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.		

2	`if( tbvCert.KeyUsagePresent()==true )` `{` `  if( CheckKeyUsage( tbvCert, intendedKeyUsage )==false )` `  return false;` `}`	It is practical to check at this early stage whether the certificate is authorized for the intended key usage indicated in parameter *intendedKeyUsage*. Permitted key uses are indicated in the *KeyUsage* and the *ExtendedKeyUsage* extensions of *tbvCert*. If the intended usage is not permitted, *ValidateCertificate*() returns *false*. **Common PKI Profile:** Note that the *KeyUsage* extension MUST be present in all PKCs and is always critical (P1.T12.[1]).		
3	`CertInfoList tbvPath, trustedPath;` `tbvPath.Clear();` `if( BuildAndValidateCertPath( tbvCert,` `tbvCerts,` `refTime,` `pathConstraints,` `trustedCerts,` `trustedCrls,` `tbvPath,` `trustedPath )==false )` `  return false;`	Compose and validate a certificate path using function *BuildAndVerifyCertPath* (Table 3). Return *false* if no valid path could be built.		
4	`trustedCerts.UpdateCertList( trustedPath );`	If verification succeeds, *trustedCerts* will be updated with the certificates of a successfully verified path to allow their reuse: certificates of *trustedPath* not yet present in *trustedCerts* will be inserted in the list, status information of certificates readily present in the list will be updated to contain the most recent date of checking the status.		
5	`  return true;` `}`	Validation succeeded, return *true*		

2.1 Building the Certificate Path

A PKI can be illustrated by a directed graph: each vertex represents a key-pair of an entity (i.e. a CAs or an EE) whereas an edge *c(A,B)* from *A* to *B* represents a certificate, signed by A and containing the public key *B*. Self-signed root certificates are represented by an edge c*(A,A)* returning to the same vertex. The certificate path validation algorithm described in this document is intended to be used in hierarchical PKIs with possibly many multiple root CAs and root keys. A *hierarchical* PKI can be depicted as a *tree*: each vertex *A* (including the root *R*) can be reached along a directed path from the root *R*, but the graph normally contains no cycles, except edges *e(R,R)*, which belong to self-signed root certificates. Multiple edges (i.e. certificates) leading from some *A* to some *B* are similarly allowed. If multiple root keys exist in parallel, the graph of the PKI consists of PKI-domains, having no "ordinary" connections. Cross-certificates may build bridges among those islands and enable a relying party to validate a certificate even then, when the certificate holder and the relying party (the verifier) do not share a common most trusted root. See an example of such a PKI in Figure 1. Cross certificate are denoted by *cc(X,Y)*.

The algorithm presented here is constructed to handle cross-certificates and to be able to build a path – possibly via cross-certificates – from the certificate holder entity to any specific root key, if such a path exist in the graph. The presented algorithms can cope with cycles in the graph, which should be avoided in the praxis for performance reasons. For example, not only edges *cc(B2,A)*, *cc(A,C)* and *cc(C,B2)* built a cycle in Figure 1, but readily the edges *cc(B2,A*) and *cc(A,B2)*.

Building a certificate path to a trusted root is not straightforward and implies searching the PKI graph. The presented algorithm follows a "depth-first" searching strategy, i.e. explores a path "in entire depth" before trying alternative paths "in breadth". The *Depth-First Path Building Algorithm (DPBA)* is sketched in concise form below:

1) Start from the "to be verified" certificate $c(A_2,A_1)$, signed by key A_2 of some authority and containing the key A_1 of an end entity and enter the following steps with parameter $i=1$,
2) if $A_{i+1}=A_i$, that is if a root-certificate has been found and:
   a) the root certificate is trusted, terminate the search. The certificates $c(A_{i+1},A_i)$, $i=1…n$ comprise the certificate path.
   b) the root certificate is NOT trusted, track back to the most recently visited "open" vertex A_i, i.e. one with the largest possible i and with further certificates to chose from at step 3).
3) if $A_{i+1}?A_i$, that is the selected certificate is not a root certificate, select some certificate $c(A_{i+1},A_i)$ signed by some authority key A_{i+1} from the set of all available certificates (CA- and cross-certificates) containing A_i and proceeds with parameter $i+1$ to step 2). At this point the algorithm recurs and extends thus the path towards a root.

The decision at step 3), which certificate (i.e. edge) to explore next, is the second relevant characteristics of the searching strategy. In the algorithm presented below, we employ the following selection criterion:

a) First choose certificates present in the local database. According to our assumptions, the local database contains only certificates that are part of some path to a root trusted by the user that have been validated at least once. The "reuse" of readily explored paths may radically reduce the efforts while building a path (or a segment of it) to the same root.

b) Second choose certificates delivered by the certificate user. Besides the EE certificate used for signature or encryption, the certificate user may deliver other certificates of a certificate path he has used to validate the certificate. Typically, these certificates comprise the "official" certificate path of the certificate owner, containing only "regular" CA-certificates (i.e. no cross-certificates).

c) If none of the previously mentioned certificates leads to a trusted root, fetch other certificates of the entity from the directory. Directories must contain all cross-certificates issued for a CA key.

Due to the above search principles, the algorithm typically explores the "official" path to the root trusted by the certificate owner EE. Then, if this root happens not to be trusted by the verifying party, the algorithm explores alternative paths – possibly via cross-certificates – at higher order keys (i.e. keys of authorities higher in the hierarchy) before exploring alternative paths at lower order keys. This matches the common practice that they are typically the higher order, and especially the root, authorities who issue cross-certificates among each other.

Say, user *I* wants to let the algorithm validate certificate *c(F,K)* of user *K*. User *I* trusts only the key *B1*, e.g. this is the key stored on his smartcard. If the local database contains the certificates *c(B1,B1)* and *cc(B1,B2)*, the search algorithm first finds the path (K,G,C,B2,B1), then path (K,G,C,A,B2,B1) and finally (K,G,F,B2,B1).

We emphasize that actual implementations are NOT mandated to implement any specific searching strategy and selection criterion, but MUST be able to find some appropriate path to a trusted root, if such a path exist. Note furthermore that it is not necessary to maintain a local database. Beyond giving the theoretical framework, we describe here just one, supposedly efficient, variant of the numerous possible implementations. The certificate path can however be built and verified, even if the local database is empty (up to one trusted root certificate).

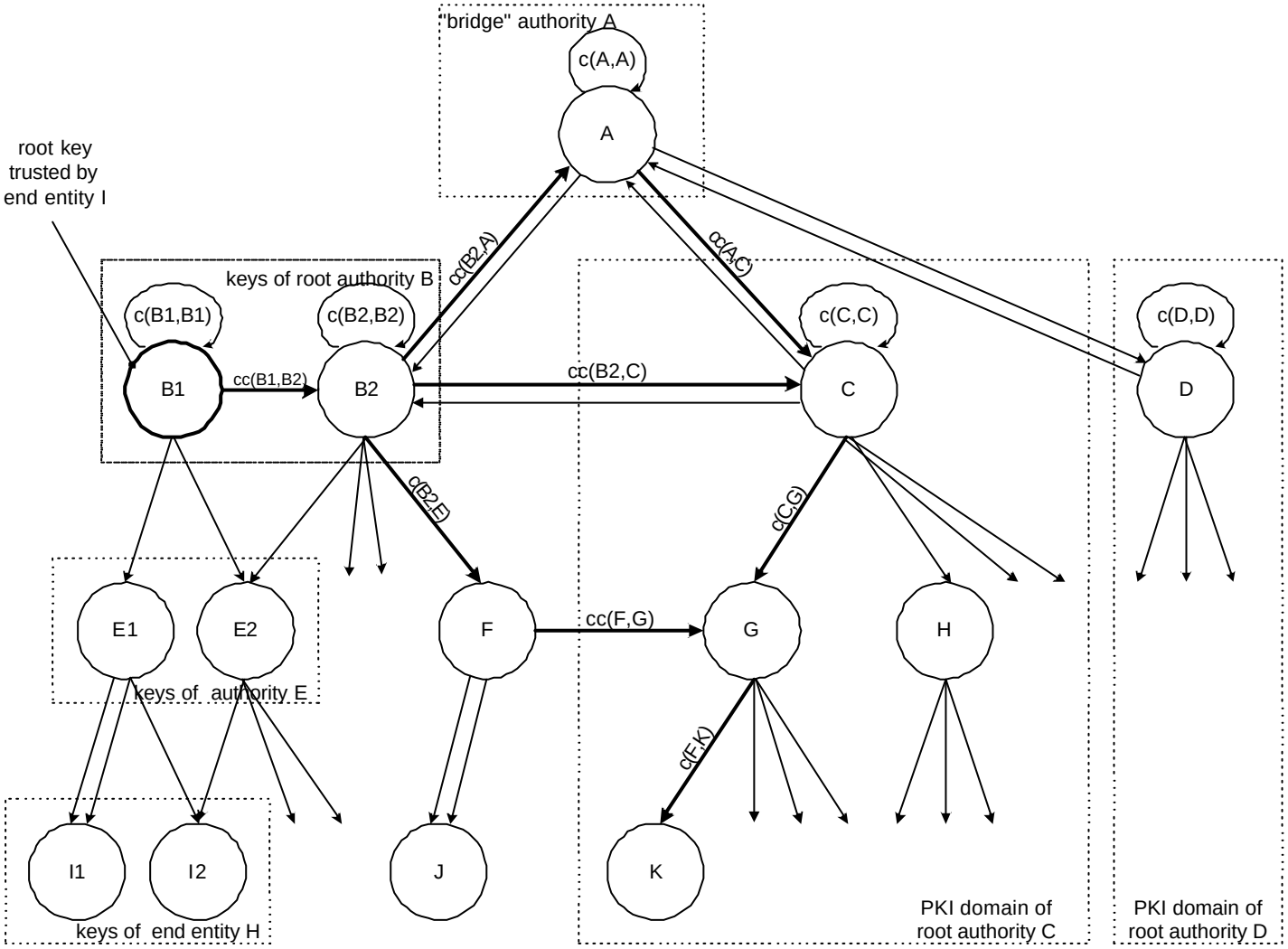


Figure 1: An example of a PKI with multiple root CAs and cross-certification

Table 3: BuildAndValidateCertPath( )

#	PSEUDO-CODE	COMMENTS	REF. TO DPBA	NOTES
1	`bool BuildAndValidateCertPath(` `  CertInfo in tbvCert,` `  CertInfoList in tbvCerts,` `  Time in refTime,` `  PathConstraints in pathConstraints,` `  CertInfoList in trustedCerts,` `  CrlInfoList in trustedCrls,` `  CertInfoList in tbvPath,` `  CertInfoList out trustedPath )` `{`	This function performs "depth-first" search in the PKI graph and builds a certificate path to a root certificate, as described at the beginning of this chapter. This function is recursively called during the search procedure and each time it is called, it performs steps 2) and 3) of the DPBA. The 'to be verified' certificate (PKC or AC) is passed in *tbvCert* to the function. *tbvPath* carries readily built segments of the path through recursive calls. The other parameters have the same semantics as in Table 2. In case of success, the function returns *true* and the constructed and verified path in *trustedPath*. The structure of the path is: • for all i in $\{1,n-1\}$, the subject of certificate i is the issuer of certificate $i+1$, • certificate $i=1$ is a trusted self-signed root certificate, • certificate $i=n$ is the 'to be verified' target certificate If no path could be built or validation failed, the function returns *false*.		
2	`  if( tbvPath.FindCert( tbvCert ) )` `  return false;`	If *tbvCert* is readily present in *tbvPath*, it indicates having run into a cycle in the PKI graph. To avoid infinite looping, backtracking is initiated by returning *false*.		
3	`  tbvPath.InsertAtFront( tbvCert );`	The *tbvCert* is inserted at the front of the path, i.e. as item with index 1.		
4	`  if( tbvCert.GetCertType() == RootCACert )` `  {`	If the certificate just reached is a self-signed root certificate, the search terminates.	2)	
5	`  if( trustedCert.findCert( tbvCert )` `  {` `  if( ValidateCertPath( tbvPath` `  tbvCerts,` `  refTime,` `  pathConstraints,` `  trustedCerts,` `  trustedCrls )==false )` `  return false;` `  trustedCertPath = tbvPath;` `  return true;` `  }`	If the root certificate is trusted by the user, i.e. it occurs in *trustedCerts*, the function calls *ValidateCertPath*() to validate the path (Table 4). If the path cannot be validated for some reason (e.g. some certificate expired, policy constraints cannot be met or directory services were not available), backtracking is initiated by returning *false*. In this way, the algorithm is able to track back to some "open" vertex in the search graph and explore an alternative path. If validation succeeds, the path is copied to the output variable *trustedCertPath* and *true* is returned.	2)a)	
6	`  else` `  return false;` `  }`	If the root certificate cannot be found among those trusted by the user, backtracking is initiated by returning *false*.	2)b)	
7	`  if( tbvCert.AuthKeyIdIsPresent() == false )` `  return false;` `  if( tbvCert.AuthKeyIdContainsKeyId() == false )` `  return false;` `  OCTET_STRING authorityKeyId;`	As the certificate just added to the path is not a root certificate, the algorithm is now about to build the path further. To be able to find all certificates containing the key used to sign *tbvCert*, (i.e. a signer or authority certificate to *tbvCert*), the authority	3)	

	`authorityKeyId = tbvCert.GetKeyIdFromAuthKeyId();`	key identifier will be retrieved from the *keyIdentifier* field of the *AuthorityKeyIdentifier* extension of *tbvCert*. **Common PKI Profile:** Note that the *AuthorityKeyIdentifier* extension MUST always be present. (P1.T11.[1]) The *keyIdentifier* field MUST always be present and MUST be include in the *SubjectKeyIdentifier* of the corresponding CA certificate. The *authorityCertIssuer* and *authorityCertSerialNumber* fields MAY also be present in *AuthorityKeyIdentifier*. (P1.T11.[1]) Note: An application may prefer to follow exactly the "official" path of *tbvCert*, if it is indicated in the *authorityCertIssuer* and *authorityCertSerialNumber* fields. For simplicity, we avoid here describing this option.		
8	`CertInfoList issuerCerts;` `CertInfo issuercert;` `if( trustedCerts.findCertWithSubjectKeyId(authorityKeyId, issuerCerts) )` `{` `for( int i=0; i<issuerCerts,size(); i++ )` `{` `issuerCert = issuerCerts.GetItem(i);` `if( BuildAndValidateCertPath(` `issuerCert,` `tbvCerts, refTime, pathConstraints,` `trustedCerts, trustedCrls,` `tbvPath, trustedPath )==true )` `return true;` `}` `}`	The variable *issuerCerts* collects all certificates (root-CA-, CA- and cross-certificates) of the issuer of *tbvCert*, which contain the public key used to sign *tbvCert*. First, the locally available certificates of *trustedCerts* will be scanned to find appropriate certificates. For each appropriate certificate, it will be attempted by recursively calling *BuildAndValidateCertPath*() to build and validate a path to a trusted root. If a trusted path can be built, the function returns *true*. If not, the algorithm proceeds to the next authority certificate in *issuerCerts* or, if none of them led to success, to step #9. **Common PKI Profile:** The *SubjectKeyIdentifier* MUST always be present in CA certificates and MUST have the same value as the *keyIdentifier* in *AuthorityKeyIdentifier* extension of the issued certificates.	3)a)	
9	`if( tbvCerts.findCertWithSubjectKeyId( authorityKeyId, issuerCerts ) )` `{` `for( int i=0; i<issuerCerts,size(); i++ )` `{` `issuerCert = issuerCerts.GetItem(i);` `if( BuildAndValidateCertPath(` `issuerCert,` `tbvCerts, refTime, pathConstraints,` `trustedCerts, trustedCrls,` `tbvPath, trustedPath )==true )` `return true;` `}` `}`	If step #8 failed, proper authority certificates will be searched in the list *tbvCerts*, trusted by the decrypting/signing party. For each proper certificate, it will be attempted by recursively calling *BuildAndValidateCertPath*() to build and validate a path to a trusted root. If a trusted path can be built, the function returns *true*. If not, the algorithm proceeds to the next authority certificate in *issuerCerts* or respectively to step #10.	3)b)	

10	CertInfoList downloadedCerts; if(<using URLs in alt.names to locate authority certs>) { LdapUrl authCertUrl; if(tbvCert.GetCertType() != EndEntityAC) { if(tbvCert.IssuerAltNamesIsPresent() && tbvCert.IssuerAltNamesContainsLdapUrl()) { authCertUrl = tbvCert.getFirstLdapUrlFromIssuerAltNames(); } } else { authCertUrl = tbvCert.getFirstLdapUrlFromIssuer(); } if(authCertUrl.IsEmpty()) return false; if(RequestCaAndCrossCertsViaLdap(authCertUrl, downloadedCerts)==false) return false; } else { downloadedCerts = <use some alternative method to download certs of the issuing authority > }	If step #9 failed, it will be attempted to download certificates of the issuing authority (Root-CA, CA- and cross-certificates) from the directory to the *downloadedCerts* variable. The application MAY use some alternative method to locate and obtain those authority certificates. **Common PKI Profile:** The LDAP-URL pointing to the CA certificate SHOULD be included in the *IssuerAltNames* extension of PKCs and resp. in the *issuer* field of ACs. Further notes on the storage of cross-certificates: Whereas CA-certificate are usually stored in a *caCertificate* attribute at the directory entry of the authority entity, cross-certificates should be found in a *crossCertificatePair* attribute. According to [X.509:2005], cross-certificates MUST occur in the *issuedToThisCA* fields of c*rossCertificatePair* attributes in the directory entry of the certificate owner, i.e. the subject CA. (P4.T1.[21]) Additionally, the same certificates MAY be published in the *issuedByThisCA* fields of *CrossCertificatePair* attributes in the directory entry of the trusted, issuing CA. Applications may achieve better performance, if collecting all *'issuedByThisCA'* cross-certificates at once from the directory entry of the CA they trust and storing them locally.		
11	if(downloadedCerts.findCertWithSubjectKeyId(authorityKeyId, issuerCerts)) { for(int i=0; i<issuerCerts,size(); i++) { issuerCert = issuerCerts.GetItem(i); if(BuildAndValidateCertPath(issuerCert, , tbvCerts, refTime, pathConstraints, trustedCerts, trustedCrls, tbvPath, trustedPath)==true) return true; } }	Proper certificates of the authority, i.e. those with the right key identifier, will be selected in *issuerCerts*. For each proper authority certificate, it will be attempted by recursively calling *BuildAndValidateCertPath*() to build and validate a path to a trusted root. If a trusted path can be built, the function returns *true*. If not, the algorithm proceeds to the next authority certificate in *issuerCerts* or respectively to step #10.	3)c)	
12	return false; }	No trusted path could be built from any authority certificate, return false;		

2.2 Validating the Certificate Path

The following algorithm is compatible with the *'Basic Path Validation Algorithm'*, briefly *BPVA*, presented in Section 6.1 of RFC 5280. Some minor modifications (corrections and enhancements) have been applied to BPVA, which are conspicuously indicated by the words '**Common PKI Profile'**. The algorithm assumes that certificates do not use subject or unique identifier fields or private critical extensions, as recommended in [RFC5280] and as strictly enforced by Common PKI. However, if these components appear in certificates, they MUST be processed. Finally, policy qualifiers are also neglected for the sake of clarity and simplicity.

Table 4: ValidateCertPath()

#	PSEUDO-CODE	COMMENTS	RFC	NOTES
	BASIC PATH VALIDATION		**6.1.1**	
1	`bool ValidateCertPath( CertInfoList in tbvPath,` `CertInfoList in tbvCerts,` `Time in refTime,` `KeyPurpose in intendedKeyUsage,` `PathConstraints in pathConstraints,` `CertInfoList inout trustedCerts,` `CrlInfoList inout trustedCrls )` `{` `int n = tbvPath.size();`	This function performs basic certificate path validation. *tbPath* is built by *BuildAndValidateCertPath*() and contains the *n* certificates of a path to a trusted root as follows: • for all *i* in {1,*n*-1}, the subject of certificate *i* is the issuer of certificate *i*+1, • certificate *i*=1 is a trusted self-signed root certificate, • certificate *i*=*n* is the 'to be verified' target certificate The other function parameters have the same meaning and constraints as those of *BuildAndValidatePath*() in Table 3. *tbvCerts* may contain further certificates that are only used in validating ternary signed objects, like CRLs. This function returns *true* if the certificate could be successfully validated, including its digital signature verification and checking constraints; respectively *false*, if its digital signature verification failed or some constraints cannot be met.	6.1.1 (a), (b), (d), (c), (h), (i),	

	INITIALIZATION		**6.1.2**	
2	`PolicyList validPolicySet = { anyPolicy };`	A set of certificate policy identifiers comprising the policies recognized by the CAs along the certificate path together with policies deemed equivalent through policy mapping. *validPolicySet* is initialized with a single policy item *anyPolicy*, indicating that no specific policy has been found yet which applies for the path. At the end of basic path validation, the set will either contain a number of valid policy OIDs or will be empty, if no valid policies were found. At the end of path validation, the valid policy set will be matched against the policies accepted by the relying party (i.e. against *userInitialPolicySet*). **Common PKI Profile:** For the sake of simplicity, policy qualifiers – which are only carried along, but not evaluated in BPVA – are ignored. Furthermore, the data structure holding valid policies is a set of policy OIDs rather than a tree of complex data objects. Still, the present algorithm is functionally equivalent with the BPVA in [RFC5280]. The simplifications lead to a more transparent algorithm design as well as to less error-prone implementations.	6.1.2 (a)	
3	`GeneralNames permittedSubtrees = pathConstraints.initialPermittedSubtrees;`	*permittedSubtrees* contains a set of root names defining a set of subtrees within which all *subject* and *subjectAltNames* names in subsequent certificates in the certification path MUST fall. The list is initialized with subtrees accepted by the relying party. Applications conforming to this profile MUST be able to process name constraints that are imposed on the *directoryName* name form and SHOULD be able to process name constraints that are imposed on the *rfc822Name*, *uniformResourceIdentifier*, *dNSName*, and *iPAddress* name forms. For the syntax and semantics of name values refer to Section 4.2.1.10 of [RFC5280].	6.1.2 (b)	
4	`GeneralNames excludedSubtrees = pathConstraints.initialExcludedSubtrees;`	*excludedSubtrees* contains a set of root names defining a set of subtrees within which no *subject* or *subjectAltNames* name in subsequent certificates in the certification path may fall. The list is initialized with subtrees refused by the relying party. Only the name forms listed under #2 need to be supported.	6.1.2 (c)	

5	`int explicitPolicy = pathConstraints.initialExplicitPolicy ? 0 : n+1;`	The counter *explicitPolicy* indicates the number of certificates at the current and lower levels in the path that may have no valid policy explicitly declared by the CAs. If zero, an explicit valid policy is needed in the certificates at this and lower levels. Once set, this variable may be decreased, but may not be increased. (That is, if a CA in the path requires an explicit policy, a later certificate cannot remove this requirement.) If the relying party requires an explicit policy, the initial value is *0*. Otherwise the initial value is *n+1*, which indicates that no explicit policy is required, unless a CA lower in the hierarchy enforces this by means of including the *PolicyConstraints* extension and properly setting the *requireExplicitPolicy* variable.	6.1.2 (d)	
6	`int inhibitAnyPolicy = pathConstraints.initialAnyPolicyInhibit ? 0 : n+1;`	The counter *inhibitAnyPolicy* indicates the number of certificates at the current and lower levels in the path that may have *anyPolicy*. If zero, *anyPolicy* is not allowed in the certificates at this and lower levels. Once set, this variable may be decreased, but may not be increased. (That is, if a CA in the path inhibits *anyPolicy*, a later certificate cannot remove this requirement.) If the relying party inhibits *anyPolicy*, the initial value is *0*. Otherwise the initial value is *n+1*, which indicates accepting *anyPolicy* along the path, as long as one CA lower in the hierarchy does not prohibit it by means of including the *InhibitAnyPolicy* extension.	6.1.2 (e)	
7	`int policyMapping = pathConstraints.initialPolicyMappingInhibit ? 0 : n+1;`	The counter *policyMapping* indicates the number of certificates at the current and lower levels in the path at which policy mapping may be applied. If zero, policy mapping is not allowed in the certificates at this and lower levels. Once set, this variable may be decreased, but may not be increased. (That is, if a certificate in the path prohibits policy mapping, a later certificate cannot remove this requirement.) If the relying party inhibits policy mapping, the initial value is *0*. Otherwise the initial value is *n+1*, which indicates that policy mapping is allowed along the path, as long as one CA lower in the hierarchy does not prohibit it by means of including the *PolicyConstraints* extension and properly setting the *inhibitPolicyMapping* variable.	6.1.2 (f)	
8	`int maxPathLength = n;`	*maxPathLength* integer is initialized to *n*, is decremented for each non-self-issued certificate in the path, and may be reduced to the value in the *pathLenConstraint* field within the *BasicConstraints* extension of a CA certificate.	6.1.2 (k)	
	BASIC CERTIFICATE PROCESSING		**6.1.3**	
9	`for( int i=1; i<=n; i++ ) {`	This for cycle runs through all certificates of the path, starting at the trusted root certificate and ending at the end-entity certificate.		
10	`CertInfo &tbvCert = certPath.GetItem(i);`	*tbvCert* is just a reference (or alias) to the *i*th item of the path, which is the 'to be verified' certificate at this step.		

11	`CertInfo &issCert;` `if( i>1 )` `  issCert = certPath.GetItem(i-1);` `else` `  issCert = certPath.GetItem(i);`	*issCert* is just a reference (or alias) to the certificate of the issuer of *tbvCert*. It can be a CA-, a root-CA- or a cross-certificate. *issCert* contains all parameters of the verifying key in step *i*: public key, public key algorithm, public key parameters, issuer name.	6.1.2 (g...j) 6.1.4 (c...f)	
12	`if( VerifySignature( tbvCert,` `  issCert.GetPublicKeyInfo() )==false )` `  return false;`	Verify signature over *tbvCert* using public key and signature algorithm of the issuing CA, return *false* if fails.	6.1.3 (a)(1)	
13	`if( (refTime < tbvCert.GetValidityNotBefore()) or` `  (refTime > tbvCert.GetValidityNotAfter()) )` `  return false;`	Check whether *refTime* lies within the validity period.	6.1.3 (a)(2)	
14	`if( CheckRevocationStatus( tbvCert,` `  tbvCerts,` `  refTime,` `  pathConstraints,` `  trustedCerts,` `  trustedCrls )==false )` `  return false;`	Check whether the certificate has been revoked before *refTime* and is not currently on hold status that commenced before *refTime*. This may be determined by obtaining a CRL or requesting online status checking. If a sufficiently recent CRL or sufficiently recent status information is locally available, i.e. if the most recent time the status is known to be valid lies at or after *refTime*, the local information may be applied.	6.1.3 (a)(3)	
15	`if( tbvCert.GetIssuer() != issCert.GetSubject() )` `  return false;`	Verify that certificates correctly chain, i.e. the issuer of *tbvCert* is the subject of *issCert*.	6.1.3 (a)(4)	
16	`bool isSelfIssuedIntermediate =` `  (tbvCert.GetCertType()==SelfIssuedCACert) and (i<n);` `if( not(isSelfIssuedIntermediate) )` `{` `  if( permittedSubtrees.containDName(` `  tbvCert.GetSubject())==false )` `  return false;` `  if( permittedSubtrees.containAllGeneralNames,` `  tbvCert.GetSubjectAltNames())==false )` `  return false;`	This and the following steps are skipped, if certificate *i* is a self-issued intermediate certificate: Verify that the subject name and each alternative name in the *subjectAltNames* extension (critical or non-critical) are consistent with the *permittedSubtrees* variable. **Common PKI Profile:** Applications conforming to this profile MUST be able to process name constraints that are imposed on *directoryName*, *rfc822Name*, *uniformResourceIdentifier*, *dNSName*, and *iPAddress* name forms, independently of the criticality of the *subjectAltName* extension. Restrictions apply only when the specified name form is present in *tbvCert*. If a constrained name type is absent the certificate, the certificate is acceptable.	6.1.3 step (b)	
17	`  if( excludedSubtrees.containDName(` `  tbvCert.GetSubject())==true )` `  return false;` `  if( excludedSubtrees.containAnyOfGeneralNames(` `  tbvCert.GetSubjectAltNames())==true )` `  return false;` `}`	Verify that the subject name and each alternative name in the *subjectAltNames* extension (critical or non-critical) are consistent with the *excludedSubtrees* variable. **Common PKI Profile:** The same remarks apply for *excludedSubtrees* as for *permittedSubtrees* above.	6.1.3 step (c)	

18	`PolicyList certPolicySet = tbvCert.GetCertPolicyOIDs();` `if( (certPolicySet.empty()==false ) and` `(validPolicySet.empty()==false ) )` `{` `if( (inhibitAnyPolicy==0) and` `not(isSelfIssuedIntermediate) )` `certPolicies.remove( anyPolicy );` `bool certPolicyAny = certPolicySet.contains(anyPolicy);` `bool validPolicyAny = validPolicySet.contains(anyPolicy);` `if( certPolicyAny==false ) //case 6.1.3(d)(1)` `if( validPolicyAny==false ) //case 6.1.3(d)(1)(i)` `validPolicySet = Intersection( validPolicySet,` `certPolicies);` `else //case 6.1.3(d)(1)(ii)` `validPolicySet = certPolicySet;` `else //case 6.1.3(d)(2)` `{} //validPolicySet remains unchanged` `}`	The *certPolicySet* list holds all policy OIDs that are present in *CertificatePolicies*. If there is no policy information in the certificate, the list remains empty. If *anyPolicy* is not allowed (since *inhibitAnyPolicy* is in effect and the certificate is not a self-issued intermediate certificate), remove it from *certPolicySet.* In the rest of this step the *validPolicySet* will be maintained (if not already empty) according to the information in *tbvCert* (if not empty): • *If anyPolicy is neither present in certPolicySet nor in validPolicySet, then the resulting matching policy set is the intersection of the policy sets.* • *anyPolicy in validPolicySet matches all policies in certPolicySet, so the resulting matching policy is certPolicySet.* • *On the other hand, anyPolicy in certPolicySet matches all policies in validPolicySet, so the resulting matching set is validPolicySet itself, i.e. no change is needed.* **Common PKI Profile:** *GetCertPolicyOIDs*() MUST consider policy OIDs in the *CertificatePolicies* extension as well as in *QualifiedCertificateStatements*.	6.1.3 (d)(1) (d)(2) (d)(3)	
19	`if( certPolicySet.empty()==true )` `validPolicySet = {};`	If the *CertificatePolicies* extension is not present, clear *validPolicySet*. **Common PKI Profile:** *GetCertPolicyOIDs*() MUST consider policy OIDs in the *CertificatePolicies* extension as well as in *QualifiedCertificateStatements*.	6.1.3 (e)	
20	`if( (explicitPolicy==0) and` `validPolicySet.IsEmpty() )` `return false;`	If issuers or the relying party enforce (via *RequireExplicitPolicy* respectively via *initialExplicitPolicy*) the path to have an explicit policy, but there is no valid policy, the validation fails.	6.1.3 (f)	
21	`if( tbvCert.ContainsUnknownCriticalExtensions() )` `return false;` `tbvCert.ProcessOtherExtensions();`	Return *false*, if there are unknown critical extensions in the certificate. Process (at least) the other critical extensions.	6.1.4 (o) 6.1.5 (f)	
22	`if( i==n ) break;`	The last certificate in the path has been processed. Skip the rest of the cycle by quitting the *for* cycle and proceed to the wrap-up procedure.		
	PREPARE FOR CERTIFICATE i+1	The cycle is not in the last loop yet and prepares for the next certificate in the path. The currently processed certificate must be a CA certificate.	**6.1.4**	
23	`if( tbvCert.PolicyMappingIsPresent() and` `tbvCert.PolicyMapping.contains(anyPolicy) )` `return false;`	If the *PolicyMapping* extension is present, it must not contain *anyPolicy* in any field, otherwise the validation fails.	6.1.4 (a)	

24	`if( tbvCert.PolicyMappingIsPresent() )` `{` `  PolicyList issuerDomainPolicySet = {};` `  PolicyList subjectDomainPolicySet = {};` `  for( int j=1; j<PolicyMapping.size(); j++ )` `  {` `  CertPolicyId issPol=PolicyMapping[j].getIssuerPolicy();` `  CertPolicyId subPol=PolicyMapping[j].getSubjectPolicy();` `  if( validPolicySet.contains(issPol) or` `  validPolicySet.contains(anyPolicy) )` `  {` `  issuerDomainPolicySet.add( issPol );` `  subjectDomainPolicySet.add( subPol );` `  }` `  }` `  validPolicySet = Subtract( validPolicySet,` `  issuerDomainPolicySet );` `  if( policyMapping>0 )` `  validPolicySet = Union( validPolicySet,` `  subjectDomainPolicySet );` `}`	First, the mapped polices are collected in *issuerDomainPolicySet* and respectively *subjectDomainPolicySet* . If *policyMapping>0*, policy identifiers may be mapped: If an *issuerDomainPolicy* matches one policy in *validPolicySet* (by exact math or by matching *anyPolicy*), the corresponding *subjectDomainPolicy* replaces *issuerDomainPolicy* in *validPolicySet*. If *policyMapping=0*, policy identifiers must not be mapped and all matching *issuerDomainPolicy* items are removed from *validPolicySet*. **Common PKI Profile:** The policy mapping operations are performed here on a set of policy OIDs instead of a tree of policy data objects. Still, the simplified mapping procedure yields the same results as the BPVA in [RFC5280].	6.1.4 (b) (b)(1) (b)(2)	
25	`if( tbvCert.NameConstraintsIsPresent() )` `{` `  if( tbvCert.ExcludedSubtreesIsPresent() )` `  {` `  permittedSubtrees = Intersection( permittedSubtrees,` `  tbvCert.GetPermittedSubtrees() );` `  }` `  if( tbvCert.ExcludedSubtreesIsPresent() )` `  {` `  excludedSubtrees = Union( excludedSubtrees,` `  tbvCert.GetExcludedSubtrees() );` `  }` `}`	If *permittedSubtrees* is present in the certificate, set the *permittedSubtrees* state variable to the intersection of its previous value and the value indicated in the extension field. If *excludedSubtrees* is present in the certificate, set the *excludedSubtrees* state variable to the union of its previous value and the value indicated in the extension field. Note that the *NameConstraints* extension may only occur in CA certificates.	6.1.4 (g)(1) (g)(2)	
26	`if( tbvCert.GetCertType() != SelfIssuedCACert )` `{` `  if( explicitPolicy>1 ) explicitPolicy--;` `  if( policyMapping>1 ) policyMapping--;` `  if( inhibitAnyPolicy>1 ) inhibitAnyPolicy--;` `}`	If the certificate is not a self-issued certificate, decrement the policy related counters.	6.1.4 (h)(1) (h)(2) (h)(3)	

27	`if( tbvCert.PolicyConstraintsIsPresent() )` `{` `  if( tbvCert.RequireExplicitPolicyIsPresent() )` `  {` `  int r = tbvCert.GetRequireExplicitPolicy();` `  if( r < explicitPolicy )` `  explicitPolicy = r;` `  }` `  if( tbvCert.InhibitPolicyMapping() )` `  {` `  int q = tbvCert.GetInhibitPolicyMapping();` `  if( q <policyMapping )` `  policyMapping = q;` `  }` `}`	If a *PolicyConstraints* extension is included in the certificate, modify the *explicitPolicy* and *policyMapping* state variables as follows: (1) If *requireExplicitPolicy* is present and is less than *explicitPolicy*, then set it to the value in the extension. (2) If *inhibitPolicyMapping* is present and is less than *policyMapping*, then set it to the value in the extension. Note that the *PolicyConstraints* extension may only occur in CA certificates.	6.1.4 (i)(1) (i)(2)	
28	`if( tbvCert.InhibitAnyPolicyIsPresent() )` `{` `  int q = tbvCert.GetInhibitAnyPolicy();` `  if( q < inhibitAnyPolicy )` `  inhibitAnyPolicy = q;` `}`	If *InhibitAnyPolicy* extension is included in the certificate, modify the *inhibitAnyPolicy* state variable as follows: If the value of *InhibitAnyPolicy* extension is less than *inhibitAnyPolicy* state variable, then then set it to the value in the extension. Note that the *InhibitAnyPolicy* extension may only occur in CA certificates.	6.1.4 (j)(3)	
29	`if( tbvCert.IsCaCertificate()==false )` `  return false;`	All certificates of the path, where $i<n$, must be issuer certificates (i.e. CA, root-CA or cross-certificates). Check for these certificates that the *BasicConstraints* extensions is present in the certificate and that the CA-flag is set.	6.1.4 (k)	
30	`if( tbvCert.GetCertType() != SelfIssuedCACert )` `  if( maxPathLength>0 )` `  maxPathLenght--;` `  else` `  return false;`	If the certificate is not a self-issued certifcate, verify that *maxPathLength* is greater than zero and decrement *maxPathLength* by one.	6.1.4 (l)	
31	`if( tbvCert.PathLenConstraintIsPresent() )` `{` `  int len = tbvCert.GetPathLenConstraint();` `  if( len < maxPathLength )` `  maxPathLength = len;` `}`	If *pathLenConstraint* is present in *BasicConstraints* and is less than *maxPathLength*, set *maxPathLength* to the value of *pathLenConstraint*.	6.1.4 (m)	
32	`if( tbvCert.GetKeyCertSignKeyUsageBit() != true )` `  return false;`	If *KeyUsage* extension is present, ensure the *keyCertSign* bit is set. **Common PKI Profile:** Note that the *KeyUsage* extension MUST be present and MUST be marked critical (P1.T12.[1]).	6.1.4 (n)	
33	`}`	End of the *for* cycle on code line #9.		
	WRAP-UP PROCEDURE	All certificates in the path have been processes with success. The wrap-up procedure verifies whether the verified path suffices policy requirements.	**6.1.5**	
34	`if( explicitPolicy>1 ) explicitPolicy--;`	Decrement counter *explicitPolicy*, which will be used below.	6.1.5 (a)	

35	`if( tbvCert.PolicyConstraintsIsPresent() and tbvCert.GetRequireExplicitPolicy()==0 ) explicitPolicy = 0;`	If *PolicyConstraints* is included in the certificate and *requireExplicitPolicy* has value 0, then set *explicitPolicy* to 0.	6.1.5 (b)	
36	`if( explicitPolicy>0 ) return true;`	The condition *explicitPolicy>0* indicates that neither issuers nor the relying party enforce the path to have an explicit valid policy. So the matching of the valid policies against *userInitialPolicySet* in the next step is skipped and the procedure returns *true*.	end of 6.1.5	
37	`if(validPolicySet.IsEmpty()) //6.1.5 (g)(i) return false; PolicySet matchingPolicySet = {}; PolicySet &userPolicySet = //an alias pathConstraints.userInitialPolicySet; bool userPolicyAny = userPolicySet.contains(anyPolicy); bool validPolicyAny= validPolicySet.contains(anyPolicy); if(userPolicyAny==true) //6.1.5 (g)(ii) matchingPolicySet = validPolicySet; else //6.1.5 (g)(iii) if(validPolicyAny==false) //6.1.5 (d)(iii)(1,2) matchingPolicySet = Intersection(validPolicySet, userInitialPolicySet); else //6.1.5 (g)(iii)(3) matchingPolicySet = userInitialPolicySet; if(matchingPolicySet.IsEmpty()) return false;`	If the path has no valid policy, the validation fails. Otherwise the valid policy set is matched against set of policies accepted by the relying party by calculating the matching set of the *validPolicySet* and the *userInitialPolicySet* (alias *userPolicySet*). If the matching set is empty, then the path policy is inconsistent with user's required policy and the algorithm returns *false*.	6.1.5 (g)	
38	`return true; }`	All checks have been successfully passed, return *true.*		

2.3 Checking the Revocation Status

Table 5: CheckRevocationStatus()

#	PSEUDO-CODE	COMMENTS	NOTES
1	`bool CheckRevocationStatus( CertInfo inout tbvCert,` `CertInfoList in tbvCerts,` `Time in refTime,` `PathConstraints in pathConstraints,` `CertInfoList inout trustedCerts,` `CrlInfoList inout trustedCrls )` `{`	The 'to be verified' PKC or AC is passed in *tbvCert*. If a status check to this certificate has ever taken place and has been stored in the local database, this information is assumed to be present in *tbvCert*. (If the status has never been investigated, the *statusInfoNextUpdate* variable contains the *startOfValidity*.) The point in time, to which status information should be obtained, is passed in *refTime*. A list of trusted certificates is passed in *trustedCert*. The function returns *false* if the certificate has been revoked or the directory service cannot be reached. Otherwise the function returns *true*.	
2	`if( refTime <= tbvCert.statusInfoNextUpdate )` `{` `if( !tbvCert.revoked )` `return true;` `else` `return (refTime < tbvCert.revocTime)` `}`	If status information is locally available and it is more recent than *refTime*, then: - return *true* if status was 'good'; - return *true* if status was 'revoked', but *refTime* is earlier than *revocTime;* - return *false* otherwise. If no status information is available or it is older than *refTime*, then obtain up-to-date status information from a server as described in the following, since a certificate: - having status '*good*' at the time indicated in *statusInfoNextUpdate*, may have been revoked since then; - '*revoked*' at the time indicated in *statusInfoNextUpdate* and having been '*on hold*', may have been released since then.	
3	`if( tbvCert.GetCertType() == RootCACert )` `return false;`	In the validation algorithm presented in this document, root certificates are assumed to be inherently valid, as reliable status information to a root certificate about cannot be obtained relying on the same trusted root. Relying software should use some other reliable out-of-band mechanism to maintain locally available status information. For the sake of theoretical correctness, the presented algorithm returns here *false* here, because the status cannot be reliably investigated. Actual implementations may override this step with the user's agreement.	
4	`if( tbvCert.AuthorityAccessInfoPresentAndContainsOcspUrl() )` `return CheckStatusViaOcsp( tbvCert,` `refTime,` `initialPolicySet,` `pathConstraints,` `trustedCerts,` `trustedCrls );`	This step is OPTIONAL. Actual implementations MAY or MAY NOT choose to support OCSP. If so and the certificate contains OCSP access info in the *AuthorityAccessInfo* extension, the revocation status will be checked using OCSP. It may furthermo re be advantageous, to check first for an appropriate, locally available CRL, before using an on-line service.	

	else		
5	return CheckStatusUsingCRL(tbvCert, issCert, tbvCerts, refTime, pathConstraints, trustedCerts, trustedCrls); }	The revocation status will be investigated using CRLs.	

Table 6: CheckStatusUsingCRL()

#	PSEUDO-CODE	COMMENTS	NOTES
1	bool CheckStatusUsingCRL(CertInfo inout tbvCert, CertInfo in issuerCert, CertInfoList in tbvCerts, Time in refTime, PathConstraints in pathConstraints, CertInfoList inout trustedCerts, CrlInfoList inout trustedCrls) {	This function checks revocation status of the 'to be verified' PKC or AC, passed in *tbvCert*, by means of obtaining and checking a corresponding, sufficiently recent CRL. This will be done in the following fundamental steps: (1) using information in *tbvCert*, identify and obtain a proper CRL, i.e. a sufficiently recent CRL, corresponding to *tbvCert* (Steps #2…#5), (2) using information in the CRL, identify and obtain a proper certificate (i.e. one with the signing key and permitted for CRL signing) of the CRL issuer and validate it using the certificate validation algorithm (Steps #6...#11), (3) verify the signature over the CRL (Step #12), (4) check status of *tbvCert* (Steps #13...#15). If a status check to *tbvCert* has ever taken place and has been stored in the local database, this information is assumed to be present in *tbvCert*. (If the status has never been investigated, the *statusInfoNextUpdate* variable contains the *startOfValidity*.) *issuerCert* contains the certificate of the issuer of *tbvCert*. The semantics of the other parameters is identical to that in *ValidateCertificate*() (Table 2). The function returns *false* if the certificate has been revoked or the directory service cannot be reached. Otherwise the function returns *true*.	
2	CRLDistributionPoint cdp = tbvCert.GetFirstCdp();	Typically, the *CRLDistributionPoints* extension contains just one CDP, but the syntax allows giving information to more than one CDP. This is the case when the CA segments the CRL according to different sub-domains or revocation reasons. Segmentation increases client performance, if large CRLs are to be handled. By storing downloaded segments, only segments that run out of validity need to be downloaded again. (Another way of increasing performance is maintaining local copies of a large CRL by means of regularly downloading delta-CRLs.) For simplicity of the description here, it is assumed that merely one CDP is present. The *cdp* variable may remain empty, if the *CRLDistributionPoints* extension is absent. Applications SHOULD be able to handle segmented CRLs. **Common PKI Profile:** Conforming certificates MUST contain the *CRLDistributionPoint* extension in case of indirect CRLs. "Direct" CRLs MUST either be stored at the node of the CA issuing the certificate in question or a *CRLDistributionPoint* extension MUST be included with directory access information.	

3	`bool crlIsIndirect;` `if( cdp.IsEmpty() )` `  crlIsIndirect = false;` `else if( cdp.ContainsCrlIssuer() )` `  crlIsIndirect = true;` `else` `  crlIsIndirect = false;`	In this step, it will be determined, whether the required CRL is an indirect one. If a *CRLDistributionPoints* extension in the certificate contains CRL access information and any of the CDPs contains the *crlIssuer* field, an indirect CRL is to be used. **Common PKI Profile:** The support of indirect CRLs is RECOMMENDED.	
4	`Name crlIssuerDName;` `if( crlIsIndirect )` `  crlIssuerDName = cdp.crlIssuer.GetDirectoryName();` `else` `  crlIssuerDName = tbvCert.GetIssuerDName();`	The DName of the CRL-issuer is determined. **Common PKI Profile:** Note that the CDP MUST contain the DName of the issuer of each indirect CRL (P1.T22.#5 & [5]).	
5	`CrlInfo tbvCrl;` `if( trustedCrls.findCrlInfo( crlIssuerDName, tbvCrl )==false )` `  tbvCrl.nextUpdate = <minimal date value>;` `if( refTime >= tbvCrl.nextUpdate )` `{` `  if( <using URLs in CDP/alt.names to locate CRL> )` `  {` `  LdapUrl crlUrl;` `  if( crlIsIndirect )` `  crlUrl = cdp.distributionPoint.GetFirstLdapUrl();` `  else` `  crlUrl = tbvCert.GetFirstLdapUrlFromIssuerAltNames();` `  CrlInfoList downloadedCrls;` `  if( RequestCrlsViaLdap( crlUrl, downloadedCrls )==false )` `  return false;` `  if( downloadedCrls.findCrlInfo(crlIssuerDName,tbvCrl)==false )` `  return false;` `  }` `  else` `  {` `  tbvCrl = <use some alternative method to download the CRL>` `  }` `}`	At this step, the proper CRL is either selected from the local database of *trustedCrls*. The proper CRL is identified by means of the DName of the issuer of the CRL, which is contained in the *crlIssuerDName* variable. If it is not sufficiently recent, it will be downloaded from an LDAP server by means of the *RequestCrlsViaLdap()* function.. This *RequestCrlsViaLdap()* function returns *false* immediately, if the service cannot be connected. Note that CRLs are usually stored in a *certificateRevocationList* or an *authorityRevocationList* attribute of the CDP in the directory. (P4.T1.[22] & [23]) Theoretically, there may be several CRLs present at an LDAP node. The proper CRL is identified in the *findCrlInfo*() function by means of the *crlIssuerDName* variable. Application MAY use alternative methods to obtain the proper CRL or MAY choose to check all CRLs present at the given node. [RFC5280]: Note that the X.509 optional field *nextUpdate* MUST be included in all CRLs.	
6	`AuthorityKeyIdentifier crlIssuerKeyId =` `  tbvCrl.GetAuthorityKeyId();`	At this step, the key identifier of the signing certificate of the CRL issuer is determined. **Common PKI Profile:** The *AuthorityKeyIdentifier* extension MUST always be present in a conforming CRL (P1.T33.[1]). Note furthermore that *IssuerAltNames* SHOULD be present in indirect CRLs and SHOULD contain an LDAP-URL of the CRL issuer's signing certificate. (P1.T33.[2]).	

7	`CertInfoList crlIssuerCerts;` `CertInfo crlIssuerCert;` `if( trustedCerts.findCert( crlIssuerKeyId,` `crlIssuerCert ) )==false )` `{` `for( int i=0; i<crlIssuerCerts.size(); i++ )` `{` `crlIssuerCert = crlIssuerCerts.GetItem(i);` `if( ValidateCertificate ( crlIssuerCert,` `tbvCerts, refTime, crlSigning, pathConstraints,` `trustedCerts, trustedCrls )==true )` `goto #12;` `}` `}`	Analogously to Steps #8…#12 in Table 3, Steps #7…#11 of this function try locate a proper certificate of the CRL signer. Proper certificates will be: - first searched in the local database (represented here by *trustedCerts*) (Step #7), - then among the "non-trusted" certificates delivered in *tbvCerts* (Step #8) - and finally in a directory or some other external resource (Steps #9,#10). For each proper certificate found a validation will be attempted by calling *ValidateCertificate*(). This may involve, as usual, building different paths as long as a path to a trusted root certificate is found and validated. Proceed to Step #12 as soon as a valid certificate is found.	
8	`if( validCrlIssuerCertFound==false &&` `tbvCerts.findCertWithSubjectKeyId( crlIssuerKeyId,` `crlIssuerCert ) )` `{` `for( int i=0; i<crlIssuerCerts.size(); i++ )` `{` `crlIssuerCert = crlIssuerCerts.GetItem(i);` `if( ValidateCertificate ( crlIssuerCert,` `tbvCerts, refTime, crlSigning, pathConstraints,` `trustedCerts, trustedCrls )==true )` `goto #12;` `}` `}`	In this step, a proper certificate of the CRL signer will be searched among the certificates delivered in *tbvCerts*. It will be attempted to validate each proper certificate.	
9	`CertInfoList downloadedCerts;` `if( <using URLs in alt.names and CDPs to locate issuer certs> )` `{` `LdapUrl crlIssCertUrl;` `if( tbvCrl.IssuerAltNamesIsPresentAndContainsLdapUrl() )` `crlIssCertUrl = tbvCrl.getFirstLdapUrlFromIssuerAltNames();` `else if( cdp.IsEmpty()==false )` `crlIssCertUrl = cdp.distributionPointName.getFirstLdapUrl();` `if( crlIssCertUrl.IsEmpty() )` `return false;` `if( crlIsIndirect )` `{` `if(RequestCertsViaLdap(crlIssCertUrl,downloadedCerts)==false)` `return false;` `}` `else` `{` `if( RequestCaAndCrossCertsViaLdap(crlIssCertUrl,` `downloadedCerts)==false )` `return false;` `}` `}`	In this step, URLs will be determined for directory access. **Common PKI Profile:** Note that *IssuerAltNames* SHOULD be present in indirect CRLs and SHOULD contain the LDAP-URL of the CRL issuer's signing certificate. (P1.T33.[2]).	

	else { downloadedCerts = <use some alternative method to download certs of the CRL issuer> }		
10	if(downloadedCerts.findCertWithSubjectKeyId(crlIssuerKeyId, crlIssuerCerts)) { for(int i=0; i<crlIssuerCerts.size(); i++) { crlIssuerCert = crlIssuerCerts.GetItem(i); if(ValidateCertificate (crlIssuerCert, tbvCerts, refTime, crlSigning, pathConstraints, trustedCerts, trustedCrls)==true) goto #12; } }	In this step, a proper certificate of the CRL signer will be searched among the *downloadedCerts*. It will be attempted to validate each proper certificate.	
11	return false;	Return false, if no valid certificate of the CRL issuer has been found in Steps #7….#10.	
12	if(VerifySignature(tbvCrl.GetToBeSignedData(), crlIssuerCert.GetPublicKeyInfo())==false) return false; trustedCrls.UpdateCrlList(crl);	The signature over the CRL is verified. If successful, the CRL is added to (respectively updated if readily present) in the list of *trustedCrls* for reuse.	
13	CrlEntry crlEntry; if(tbvCrl.FindEntry(tbvCert.GetIssuerDName() tbvCert.GetSerialNumber(), crlEntry) == false { tbvCert.revoked = false; tbvCert.revocTime = tbvCert.GetValidityNotAfter(); tbvCert.revocReason = 'unspecified'; tbvCert.statusInfoNextUpdate = tbvCrl.NextUpdate; return true; }	As the CRL has been found valid, now we can check the status of *tbvCert*. Retrieve revocation info from the matching CRL entry, if present. The matching entry can be identified by means of the *issuer* and the *serialNumber* of *tbvCert*. *SerialNumber* is part of the entry (P1.T32.#8), whereas the *issuer* is indicated by: - either in the *CertificateIssuer* extension (P1.T47.#4) in the entry in question or in a preceding entry most near to the entry in question. (The *CertificateIssuer* entry extension MUST be used in indirect CRLs.) - or in the *issuer* field of the "direct" CRL (T32.#4), if not indicated by a *CertificateIssuer* extension. If *tbvCert* is not listed in the CRL, it will be considered valid. *statusInfoNextUpdate* will be set to the *nextUpdate* time of the CRL to maintain the local database.	
14	Time thisUpdate = tbvCrl.GetThisUpdate(); if(thisUpdate > GetCurrentTime()) return false; Time nextUpdate; if(tbvCrl.NextUpdateIsPresent()==false) return false; nextUpdate = tbvCrl.GetNextUpdate(); if(nextUpdate < GetCurrentTime()) return false;	*tbvCert* has been found in the CRL. The dates in the fields *thisUpdate* and *nextUpdate* are retrieved and checked at this step for plausibility as RECOMMENDED in P4.T8.[7] for OCSP responses. Note that the *nextUpdate* field MUST always be present. (See P1.T32.[5])	

15	tbvCert.revoked = true; tbvCert.revocTime = crlEntry.GetRevocationDate(); if(crlEntry.ReasonCodeIsPresent()) tbvCert.revocReason = crlEntry.GetReasonCode(); else tbvCert.revocReason = 'unspecified'; tbvCert.statusInfoNextUpdate = nextUpdate; return (refTime < tbvCert.revocTime); }	*tbvCert* turned out to be revoked. Retrieve revocation information from *crlEntry*. The reason of the revocation MAY be given in the *reasonCode* extension.	

Table 7: CheckStatusViaOcsp()

#	PSEUDO-CODE	COMMENTS	NOTES
1	`bool CheckStatusViaOcsp( CertInfo inout tbvCert,` `Time in refTime,` `PathConstraints in pathConstraints,` `CertInfoList inout trustedCerts,` `CrlInfoList inout trustedCrls )` `{`	The 'to be verified' PKC or AC is passed in *tbvCert*. If a status check to this certificate has ever taken place and has been stored in the local database, this information is assumed to be present in *tbvCert*. (If the status has never been investigated, the *statusInfoNextUpdate* variable contains the *startOfValidity*.) The point in time, to which status information should be obtained, is passed in *refTime*. A list of trusted certificates is passed in *trustedCert*. The function returns *false* if the certificate has been revoked or the OCSP service cannot be reached. Otherwise the function returns *true*.	
2	`OcspUrl url = tbvCert.GetFirstHttpUrl();` `OcspRequest request;` `CertID tbvCertID;` `tbvCertID.Set( sha_1,` `SHA1( tbvCert.GetIssuer() ),` `SHA1( tbvCert.GetPKWithoutTagLenUnusedBits()),` `tbvCert.GetSerialNumber() );` `request.FillInOcspRequest( tbvCertID );` `OcspResponse response;` `if( RequestStatusInfoViaOcsp( url, request, response )==false )` `return false;`	The URL of the OCSP service will be extracted and a request will be generated. The function returns *false*, if the service cannot be connected to or it returned an error code in *responseStatus* (P4.T7.#2). **Common PKI Profile:** Note that *certID* (P4.T6.#4) MUST be build using SHA1 and respectively the OID *'sha_1'*.	
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).	
4	`if( response.ArchiveCutoffIsPresent() )` `{` `Time cutoffDate = response.GetArchiveCutoff();` `if( cutoffDate > tbvCert.GetValidityNotAfter() )` `return false;` `}`	The condition cutoff date > expiry date (which is identical to the condition: *producedAt* time > expiry date + retention period) indicates the fact, that status information returned by the OCSP responder is not any more reliable, e.g. if the certificate and corresponding status information have been deleted from the directory. (P4.T13.[1])	
5	`SingleResponse singleResp;` `if( response.FindSingleResponse( tbvCertID, singleResp )==false )` `return false;`	The appropriate single response is read from *response*.	
6	`if( AnyOfUserPoliciesEnforcesPositiveStatement(`	Some policies may demand that the responder delivers evidence that the certificate has	

	pathConstraints, userInitialPolicySet)) { if(singleResp.CertHashIsPresent()==false) return false; CertHash certHash = singleResp.GetCertHash(); if(Hash(tbvCert.DerEncode(), certHash.hashAlgorithm) != certHash.certificateHash) return false; }	been indeed issued by the CA and it is present in the directory. (P4.T15.[1]) If this is required the certificate hash, delivered in the single response will be proven against the hash value built from *tbvCert*.	
7	Time thisUpdate = singleResp.GetThisUpdate(); if(thisUpdate > GetCurrentTime()) return false; Time nextUpdate; if(singleResp.NextUpdateIsPresent()) { nextUpdate = singleResp.GetNextUpdate(); if(nextUpdate < GetCurrentTime()) return false; } else { nextUpdate = thisUpdate + 1 sec; }	The dates in the fields *thisUpdate* and *nextUpdate* are retrieved and checked for plausibility as RECOMMENDED in P4.T8.[7].	
8	if(response.GetStatus()=='good') { tbvCert.revoked = false; tbvCert.revocTime = tbvCert.GetValidityNotAfter(); tbvCert.revocReason = 'unspecified'; tbvCert.statusInfoNextUpdate = nextUpdate; return true; } else if(response.GetStatus()=='revoked') { tbvCert.revoked = true; tbvCert.revocTime = singleResp.GetRevocationTime(); if(singleResp.RevocReasonIsPresent()) tbvCert.revocReason = singleResp.GetRevocationReason(); else tbvCert.revocReason = 'unspecified'; tbvCert.statusInfoNextUpdate = nextUpdate; return (refTime < tbvCert.revocTime); } else return false; }	After successful verification of the signature and the certificate path, the status information is retrieved from the appropriate single response and added to *tbvCert*. **Common PKI Profile:** Note that Common PKI conforming responders may return status 'good' only if they possess definite knowledge about the requested certificate's status.	

References

[RFC2560] Internet X.509 Public Key Infrastructure - Online Certificate Status Protocol – OCSP, RFC 2560, June 1999

[RFC4510] Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map, RFC4510, June 2006

[RFC4516] Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator, RFC 4516, June 2006

[RFC5280] Internet X.509 Public Key Infrastructure – Certificate and Certificate Revocation List (CRL) Profile, May 2008

[X.509:2005] ITU-T X.509: Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, 2005

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 6

CRYPTOGRAPHIC ALGORITHMS

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.1 15.11.2001	First public edition
1.0.2 19.7.2002	Conformance requirements in chapter 2 “Algorithm Support” are presented in tables Editorial and stylistic changes, and removal of bugs 1) AES has been announced for the next version
1.0.2 11.8.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.03.2004	Several editorial changes. The most relevant changes affecting technical aspects are: 1) RSAES-OAEP must no longer be supported, but MAY. 2) All specified CMS signature algorithm OIDs for RSA MUST be accepted. 3) RIPEMD must no longer be supported, but SHOULD be accepted and SHOULD NOT be generated. 4) DES3-CBC must no longer be supported, but SHOULD be accepted and SHOULD NOT be generated. 5) AES is now included as possible content encryption algorithm. 6) Cryptographic algorithms required and/or recommended for XML have been added.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Reflected name change of RegTP to BNetzA. Adapted to new versions of the base standards: - BNetzA Notification 2008 - ANSI X9.42:2003 - ANSI X9.62:2005 - FIPS 46-3 - FIPS 180-2 - PKCS#1 v2.1 - PKCS#5 v2.1 - ISO/IEC 10118-3:2004 - RFC 3279 - RFC 3370 - RFC 385ß - RFC 3851 - RFC 3852 - RFC 3447 - RFC 4231 - RFC 5280 - W3C XML Signature Syntax and Processing (Second Edition) Various corrections and clarifications.

Table of Contents

1 Preface

This part of the Common PKI specification defines a list of approved cryptographic algorithms for digital signatures, encryption and subject public keys to be supported by implementations that comply with the Common PKI specification.

It is mainly based on the PKIX documents [RFC 5280] and [RFC 3279], the W3C documents [XML_SIG] and [XML_ENC], and the OSCI profile [OSCI]. It contains all supplementary specifications, recommendations and restrictions the Common PKI document has defined in addition to the corresponding base documents.

The S/MIME standard version 3.1 documents [RFC 3370], [RFC 3850], [RFC 3851] and [RFC 3852] have been taken into account.

In addition to the requirements, which have to be fulfilled by conforming implementations, recommendations are made for supporting further algorithms.

Items of the referenced standards that are not explicitly mentioned in this specification SHALL be treated in the same way as specified in the referenced base standards.

Conformance requirements that Common PKI compliant components MUST satisfy are specified in the following chapter.

2 Algorithm Support

This chapter identifies a list of cryptographic algorithms required and/ or recommended by the different parts of the Common PKI specification.

Most of the algorithms identified in the following sub-chapters are described in

- the PKIX documents [RFC 5280] and [RFC 3279], and
- for XML based data structures in the W3C documents [XML_DSIG] and [XML_ENC].

For all other algorithms, e.g. all encryption algorithms, references to the corresponding specifications are provided.

The following tables provide information for each algorithm, including

- the short name,
- the respective object identifier,
- or in the case of XML the related W3C link, and
- requirements and recommendations for conforming implementations.

2.1 One-Way Hash Functions

A cryptographic hash function is used to compute the message digest of a document to be signed. A hash function must be collision-resistant which means that it is computationally infeasible to find two different documents yielding the same message digest (which implies that it is also infeasible to find a different document yielding the same message digest as a given document).

Common PKI compliant components SHALL satisfy the conformance requirements for one-way hash function as specified in Table 1.

2.2 Signature Algorithms

A signature algorithm is applied to the message digest (output value of the hash function) of the document to be signed to generate a signature.

Signature algorithms are used for signing certificates, revocation lists, PKI messages and both S/MIME and PEM messages. Algorithm identifiers are used in the corresponding fields of certificates, CRLs and messages to identify the applied signature algorithm. The signature algorithm identifier identifies both the hash function and the signature algorithm, e.g. RSA.

Common PKI compliant components SHALL satisfy the conformance requirements for signature algorithms as specified in Table 2.

2.3 Content Encryption Algorithms

A content encryption algorithm is applied in order to encrypt data, whereas a key encryption algorithm (chapter 2.5) is used for encrypting the associated content encryption key. Encryption algorithms are applied for the encryption of both confidential PKI-messages and S/MIME and PEM messages, as well as for the encryption of XML documents.

Common PKI compliant components SHALL satisfy the conformance requirements for data encryption algorithms as specified in Table 3.

2.4 Symmetric Key Wrap

Common PKI compliant components that support XML SHALL satisfy the conformance requirements for symmetric key wrap algorithms as specified in Table 4.

2.5 Key Encryption Algorithms

Key encryption algorithms are used for the encryption of content encryption keys (chapter 2.3). The used key encryption keys are the public keys of the intended recipients of the encrypted content.

Common PKI compliant components SHALL satisfy the conformance requirements for key encryption algorithms as specified in Table 5. Both are specified in [PKCS#1].

2.6 Key AgreementAlgorithms

Key agreement is only considered in Common PKI for components that support XML.

Common PKI compliant components that support XML SHALL satisfy the conformance requirements for key agreement algorithms as specified in Table 6.

2.7 Subject Public Key Algorithms

Common PKI compliant components SHALL satisfy the conformance requirements for subject public key algorithms whose related OIDs are contained in a certificate as specified in Table 7.

2.8 Message Authentication Algorithms

Message authentication algorithms are applied for the protection of PKI messages, especially for the authentication of initial certification requests and revocation requests.

Common PKI compliant components SHALL satisfy the conformance requirements for symmetric key based MAC (message authentication code) algorithms as specified in Table 8.

Table 1: One-Way Hash Functions

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	SHA-1	one-way hash function	[RFC 3279] [RFC 3370] [XML_DSIG]	2.1.3 2.1	++	++	++	OID: 1.3.14.3.2.26 http://www.w3.org/2000/09/xmldsig#sha1	[1] [2] [3]
2	SHA-256	one-way hash function	[RFC 4055] [XML_ENC] [FIPS 180-2]		n. a.	+	+	OID: 2.16.840.1.101.3.4.2.1 http://www.w3.org/2001/04/xmlenc#sha256	[1] [4]
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]
3	SHA-512	one-way hash function	[RFC 4055] [XML_ENC] [FIPS 180-2]		n. a.	+	+	OID: 2.16.840.1.101.3.4.2.3 http://www.w3.org/2001/04/xmlenc#sha512	[4]
4	RIPEMD-160	one-way hash function	[RIPEMD-160] [ISO/IEC 10118-3] [XML_DSIG]		n. a.	-	+	OID: 1.3.36.3.2.1 http://www.w3.org/2001/04/xmlenc#ripemd160	[5] [3]
5	MD2	one-way hash function	[RFC 3279] [RFC 1319]	2.1.1	-	--	--	OID: 1.2.840.113549.2.2	
6	MD5	one-way hash function	[RFC 3279] [RFC 3370] [RFC1321]	2.1.2 2.2	-	--	+-	OID: 1.2.840.113549.2.5	[6]

[1]	**Common PKI Profile:** SHA-1 is the preferred one-way hash function. This requirement is conformant with the PKIX and the XML_DSIG documents. SHA-1 is defined in [FIPS 180-2] and [ISO/IEC 10118-3]. In cases where SHA-1 will not be used due to security considerations, the preferred one-way hash function is SHA-256.
[2]	S/MIME requires that sending and receiving agents MUST support SHA-1.
[3]	This is only a requirement for compliant components that support XML.
[4]	SHA-256, SHA-384 and SHA-512 are referenced in XML_ENC, but not in XML_DSIG.
[5]	**Common PKI Profile:** The support of the RIPEMD-160 hash function on the processing side is recommended. This algorithm is published in [BNetzA08] as an algorithm appropriate and allowed for signing according to the German law on digital signatures [SigG01]. Neither PKIX nor [RFC 3370] specifies RIPEMD-160. Therefore it SHOULD NOT be used on the generation side for the sake of interoperability with PKIX and/or S/MIME compliant components.
[6]	Receiving agents SHOULD support MD5 for providing backward compatibility with MD5-digested S/MIME v2 SignedData objects.

Table 2: Signature Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	sha1WithRSAEncryption	RSA signature algorithm	[RFC 3279] [RFC 3851] [FIPS 180-2] [ISO/IEC 10118-3] [XML_DSIG]	2.2.1 2.2	+-	++	++	OID: 1.2.840.113549.1.1.5 http://www.w3.org/2000/09/xmldsig#rsa-sha1	[1,3,4,6, 9] [7]
2	sha256WithRSAEncryption	RSA signature algorithm	[RFC 4055] [FIPS 180-2] [RFC 4051]		n. a.	+-	+	OID: 1.2.840.113549.1.1.11 http://www.w3.org/2001/04/xmldsig-more#rsa-sha256	[1,3,4,6] [7]
2a	sha384WithRSAEncryption	RSA signature algorithm	[RFC 4055] [FIPS 180-2] [RFC 4051]		n. a.	+-	+	OID: 1.2.840.113549.1.1.12 http://www.w3.org/2001/04/xmldsig-more#rsa-sha384	[1,3,4,6] [7]
3	sha512WithRSAEncryption	RSA signature algorithm	[RFC 4055] [FIPS 180-2] [RFC 4051]		n. a.	+-	+	OID: 1.2.840.113549.1.1.13 http://www.w3.org/2001/04/xmldsig-more#rsa-sha512	[1,3,4,6] [7]
4	rsaSignatureWithRipemd160	RSA signature algorithm	[RIPEMD-160] [ISO/IEC 10118-3] [RFC 4051]		n. a.	-	+	OID: 1.3.36.3.3.1.2 http://www.w3.org/2001/04/xmldsig-more/rsa-ripemd160	[2,3,6,9] [7]
5	md2-WithRSAEncryption	RSA	[RFC 3279]	2.2.1	+-	--	--	OID: 1.2.840.113549.1.1.2	[1,3,4,6]

		signature algorithm							
6	md5WithRSAEncryption	RSA signature algorithm	[RFC 3279] [RFC 3851]	2.2.1 2.2	+-	--	+-	OID: 1.2.840.113549.1.1.4	[1,3,4,6]
7	dsa-with-sha1	DSA signature algorithm	[RFC 3279] [RFC 3851] [FIPS 186-2] [XML_DSIG]	2.2.2 2.2	 ++	+- ++	++ ++	OID: 1.2.840.10040.4.3 http://www.w3.org/2000/09/xmldsig#dsa-sha1	[5] [7,8]
8	ecdsa-with-SHA1	ECDSA signature algorithm	[RFC 3279] [X9.62]	2.2.3	+-	+-	+-	OID: 1.2.840.10045.4.1	
9	RSASSA-PSS	RSA signature algorithm	[RFC 4055]	3	+-	+-	+-	OID: 1.2.840.113549.1.1.10	[10,11]

[1]	The PKIX documents do not make any recommendation which of the RSA signature algorithms (md2withRSAEncryption, md5withRSAEncryption, sha1WithRSAEncryption) should be preferred **Common PKI Profile:** sha1WithRSAEncryption is the preferred signature algorithm. In cases where sha1WithRSAEncryption will not be used due to security considerations, the preferred signature algorithm is sha256WithRSAEncryption.
[2]	**Common PKI Profile:** The support of the RIPEMD-160 hash function on the processing side is recommended. This algorithm is published in [BNetzA08] as an algorithm appropriate and allowed for signing according to the German law on digital signatures [SigG01]. Neither PKIX nor S/MIME specifies RIPEMD-160. Therefore it SHOULD NOT be used on the generation side for the sake of interoperability with PKIX and/or S/MIME compliant components.
[3]	Conforming implementations SHALL use the PKCS1-v1_5 padding and encoding conventions described in PKSC#1 [RFC 3447]. The parameter component of this algorithm identifier shall be the ASN.1 type NULL.
[4]	S/MIMEv3.1 requires that receiving agents MUST support rsaEncryption with SHA-1 hash for message signature. Receiving agents MUST be capable of verifying signatures on certificates and CRLs made with md5withRSAEncryption and sha-1WithRSAEncryption with key sizes from 512 bits to 2048 bits.
[5]	S/MIMEv3.1 requires that receiving agents MUST support dsa-with-sha1 for message signature. Receiving agents MUST be capable of verifying signatures on certificates and CRLs made with dsa-with-sha1.
[6]	If any of the RSA based signature algorithms is used to sign CMS messages, the hash function OID is explicitly stated in the *digestAlgorithm* field of the *SignerInfo* (P3.T4.#3). In accordance with [RFC 3370] the OID to be inserted in the *signatureAlgorithm* field of the *SignerInfo* (P3.T4.#5) MUST be rsaEncryption (OID: 1.2.840.113549.1.1.1) when generating a signed CMS message, regardless which RSA based signature algorithms is used. When processing a signed CMS message the OIDs for *sha-1WithRSAEncryption* and *rsaSignatureWithripemd160* MUST also be accepted in the *signatureAlgorithm* field of the *SignerInfo*, provided that the respective hash function is present in *digestAlgorithm* field.
[7]	This is only a requirement for compliant components that support XML.
[8]	Note that DSA is the default signature algorithm in [XML_DSIG].
[9]	S/MIMEv3.1 requires that sending agents MUST support either dsa-with-sha1 or rsaEncryption with SHA-1 hash for message signature.
[10]	[PKCS#1] recommends the use of RSASSA-PSS for new applications. **Common PKI Profile:** Although RSASSA-PSS is considered more secure than RSA signature schemes based on PKCS#1_v1.5 padding, its use may lead to interoperability problems due the fact that it is not supported in [RFC 3370] and [RFC 3851]. Therefore it is OPTIONAL.
[11]	Support for RSASSA-PSS in XML digital signatures is currently under discussion at W3C. The algorithm identifier proposed in [XMLDSIG-PSS] is http://www.w3.org/2007/09/xmldsig-pss/#rsa-pss.

Table 3: Content Encryption Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	des-cbc	content encryption algorithm	[RFC 3851] [FIPS 46-3]	2.7	n.a.	++	++	OID: 1.3.14.3.2.7	[1], [6]
2	des-ede3-cbc	content encryption algorithm	[RFC 3851], [X9.52], [FIPS 46-3], [XML_ENC]	2.7	++	++	++	OID: 1.2.840.113549.3.7 http://www.w3.org/2001/04/xmlenc#tripledes-cbc	[2], [6] [7]
3	des3-cbc	content encryption algorithm	[X9.17] [MTTv2]			-	+	OID: 1.3.36.3.1.3.2.1	[3], [6]
4	rc2-cbc	content encryption algorithm	[RFC 3851]	2.7	+	--	+-	OID: 1.2.840.113549.3.2	[4]
5	aes128-cbc	content encryption algorithm	[RFC 3851] [FIPS 197] [RFC 3565] [XML_ENC]	2.7	+	+-	+-	OID: 2.16.840.1.101.3.4.1.2 http://www.w3.org/2001/04/xmlenc#aes128-cbc	[5] [7]
	aes192-cbc							OID: 2.16.840.1.101.3.4.1.22 http://www.w3.org/2001/04/xmlenc#aes192-cbc	[7]
	aes256-cbc							OID: 2.16.840.1.101.3.4.1.42 http://www.w3.org/2001/04/xmlenc#aes256-cbc	[7]

[1]	The DES algorithm is defined in [FIPS 46-3]; the cipher block-chaining mode (CBC) is defined in [FIPS 81]. The padding mechanism to be applied is described in the PEM specification [RFC1423] and in the PKCS#5 specification [PKCS#5].
[2]	S/MIMEv3.1 requires that sending and receiving agents MUST support encryption and decryption with DES-EDE3-CBC in 3-key mode of operation as defined in [X9.52] and [FIPS 46-3]. They SHOULD support encryption and decryption with AES at a key size of 128, 192, and 256 bits.
[3]	Triple-DES was standardized in [X9.17] in 2-key mode of operation. **Common PKI Profile:** des3-cbc is specified in [MTTv2] and SHOULD therefore be accepted for backwards compatibility with MailTrusT v2 compliant components. However S/MIME does not specify this algorithm. Therefore it SHOULD NOT be used on the generation side for the sake of interoperability with S/MIME compliant components.
[4]	S/MIMEv3 requires that receiving agents SHOULD support encryption and decryption using the RC2 [RFC 3370] or a compatible algorithm at a key size of 40 bits.
[5]	Three AES algorithm identifiers are defined for key sizes of 128,192, and 256 bits. The OIDs for AES content encryption algorithms are defined in [RFC 3565].
[6]	At least one of these algorithms MUST be supported during the generation process.
[7]	This is only a requirement for compliant components that support XML.

Table 4: Symmetric Key Wrap Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	3DES	key encryption algorithm	[XML_ENC] [X9.52]			++	++	http://www.w3.org/2001/04/xmlenc#kw-tripledes-cbc	[1]
2	AES128	key encryption algorithm	[XML_ENC] [FIPS 197]			++	++	http://www.w3.org/2001/04/xmlenc#kw-aes128-cbc	[1]
2	AES192	key encryption algorithm	[XML_ENC] [FIPS 197]			-	+	http://www.w3.org/2001/04/xmlenc#kw-aes192-cbc	[1]
4	AES256	key encryption algorithm	[XML_ENC] [FIPS 197]			++	++	http://www.w3.org/2001/04/xmlenc#kw-aes256-cbc	[1]
[1] This is only a requirement for compliant components that support XML.									

Table 5: Key Encryption Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	rsaEncryption	key encryption algorithm	[PKCS#1]	7.2	+	++	++	OID: 1.2.840.113549.1.1.1	[1]
2	RSA PKCS#1 v1.5	key encryption algorithm	[PKCS#1] [XML_ENC]	7.2	+	++	++	http://www.w3.org/2001/04/xmlenc#rsa-1_5	[3]
3	RSAES-OAEP	key encryption algorithm	[PKCS#1] [XML_ENC]	7.1	++	+-	+-	OID: 1.2.840.113549.1.1.7 http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p	[2] [3]

[1] S/MIMEv3 requires that sending and receiving agents SHOULD support Diffie-Hellman defined in [RFC 2631], and MUST support rsaEncryption.

RSAES-PKCS1-v1_5 is included in [PKCS#1] only for compatibility with existing applications, and is not recommended for new applications.

Common PKI Profile: For compatibility reasons, RSAES-PKCS1-v1_5 is the .preferred key encryption algorithm in Common PKI.

[2] [PKCS#1] recommends the use of RSAES-OAEP for new applications, e.g. for wrapping of AES content encryption keys.

Common PKI Profile: Although RSAES-OAEP is considered more secure than rsaEncryption, its use may lead to interoperability problems due the fact that it is not supported in [RFC 3370] and [RFC 3851]. Therefore it is OPTIONAL.

[3] This is only a requirement for compliant components that support XML.

Table 6: Key Agreement Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	Diffie-Hellman	key agreement algorithm	[XML_ENC] [RFC2631]			+-	+-	http://www.w3.org/2001/04/xmlenc#dh	[1]
[1] **Common PKI Profile:** This is only a requirement for compliant components that support XML. Diffie-Hellman key agreement is only considered in Common PKI for components that support XML.									

Table 7: Subject Public Key Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	rsaEncryption	RSA keys	[RFC3279]	2.3.1	+-	++	++	OID: 1.2.840.113549.1.1.1	
2	dhpublicnumber	Diffie-Hellman keys	[RFC3279]	2.3.3	+-	n. a.	n. a.	OID: 1.2.840.10046.2.1	[1]
3	dsa	DSA signature keys	[RFC3279]	2.3.2	+-	+	++	OID: 1.2.840.10040.4.1	[2]
4	keyExchangeAlgorithm	KEA public keys	[RFC3279]	2.3.4	+-	n. a.	n. a.	OID: 2.16.840.1.101.2.1.1.22	[3]
5	ecPublicKey	ECDSA and ECDH keys	[RFC3279] [X9.62]	2.3.5	+-	+-	+	OID: 1.2.840.10045.2.1	[4]

[1] **Common PKI Profile:** Diffie-Hellman key agreement [X9.42] is not considered in Common PKI.

[2] DSA is defined in [FIPS 186-2].

[3] **Common PKI Profile:** KEA key exchange is not considered in Common PKI.

[4] This OID is used in public key certificates for both ECDSA signature keys and ECDH encryption keys.
Common PKI Profile: This OID can only be used in public key certificates for ECDSA signature keys, since Diffie-Hellman key agreement [X9.42] is not considered in Common PKI

Table 8: Message Authentication Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	desMAC	message authentication algorithm	[FIPS 113]		+-	++	++	OID: 1.3.14.3.2.10	[1]
2	hmac-SHA1	message authentication algorithm	[RFC2104] [RFC2202]		+-	+ 	+ 	OID: 1.3.6.1.5.5.8.1.2	[2]
			[XML_DSIG]			-	+-	http://www.w3.org/2000/09/xmldsig#hmac-sha1	[3,4]
3	hmac-SHA256	message authentication algorithm	[RFC4231]		+-	+-	+-	OID: 1.2.840.113549.2.9	[2]
			[RFC4051]	2.2.2		-	+-	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256	[3,4]
4	hmac-SHA384	message authentication algorithm	[RFC4231]		+-	+-	+-	OID: 1.2.840.113549.2.10	[2]
			[RFC4051]	2.2.2		-	+-	http://www.w3.org/2001/04/xmldsig-more#hmac-sha384	[3,4]
5	hmac-SHA512	message authentication algorithm	[RFC4231]		+-	+-	+-	OID: 1.2.840.113549.2.11	[2]
			[RFC4051]	2.2.2		-	+-	http://www.w3.org/2001/04/xmldsig-more#hmac-sha256	[3,4]

[1] The DES-MAC uses DES as defined in [FIPS 46-3] and data authentication as defined in [FIPS 113].

[2] The support of other mechanisms, like DES3-MAC is recommended.

[3] This is only a requirement for compliant components that support XML.

[4] In the case of components that support XML, the usage of HMAC is entirely discouraged for the time being. Conforming XML clients SHOULD NOT make use of HMAC. The reason why we do not exclude the element in this profile is the fact that it is used with good reasons in [XKMS]. It may happen that in the future XKMS will become important for Common PKI and thus HMAC may return. So leaving it here will perhaps then make things a little easier.

References

BNetzA08]	Federal Network Agency for Electricity, Gas, Telecommunications, Post and Railway: Notification in Accordance with the Electronic Signatures Act and the Electronic Signatures Ordinance (Overview of Suitable Algorithms), published in German Federal Gazette (Bundesanzeiger) No 19, pp 376 of 5 February 2008 (in German)
[FIPS 113]	Federal Information Processing Standards (FIPS PUB) 113: Computer Data Authentication; May 1995
[FIPS 186-2]	Federal Information Processing Standards (FIPS PUB) 186: Digital Signature Standard; January 2000
[FIPS 197]	Federal Information Processing Standards (FIPS PUB) 197: Advanced Encryption Standard; November 2001
[FIPS 46-3]	Federal Information Processing Standards (FIPS PUB) 46: Data Encryption Standard (DES); October 1999
[FIPS 81]	National Institute of Standards and Technology (formerly National Bureau of Standards): DES Modes of Operation. December 1980
[ISO/IEC 10118-3]	ISO/IEC 10118-3:2004: IT-Security Techniques Hash Functions Part 3: Dedicated Hash-Functions
[MTTv2]	MailTrusT Version 2, March 1999, TeleTrusT Deutschland e.V., www.teletrust.de
[OSCI]	OSCI Leitstelle: OSCI Transport, Version 1.2, Bremen, 6. Juni 2002
[PKCS#1]	RSA Laboratories, “PKCS #1 v2.1: RSA Cryptography Standard”, June 2002
[PKCS#5]	RSA Laboratories, “PKCS #1 v2.1: Password-Based Encryption Standard,” October 2006
[RFC 1319]	Kaliski, B.: The MD2 Message-Digesting Algorithm, April 1992
[RFC 1321]	Kaliski, B.: The MD5 Message-Digesting Algorithm, April 1992
[RFC 1423]	Balenson D.: Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers, February 1993
[RFC 2104]	H. Krawczyk, M. Bellare, R. Canetti: HMAC: Keyed-Hashing for Message Authentication, February 1997
[RFC 2202]	Cheng, P. and R. Glenn, Test Cases for HMAC-MD5 and HMAC-SHA-1, September 1997
[RFC 2268]	Rivest, R., A Description of the RC2 (r) Encryption Algorithm, January 1998
[RFC 2631]	Rescorla, E., "Diffie-Hellman Key Agreement Method", June 1999
[RFC 3279]	Bassham L., Housley R., Polk W.: Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, April 2002
[RFC 3370]	R. Housley: Cryptographic Message Syntax (CMS) Algorithms, August 2002
[RFC 3447]	J. Jonsson, B. Kaliski: Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1, February 2003

[RFC 3565] J. Schaad: Use of the Advanced Encryption Standard (AES) Encryption Algorithm in Cryptographic Message Syntax (CMS); July 2003

[RFC 3850] B. Ramsdell: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling, July 2004

[RFC 3851] B. Ramsdell: Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification, July 2004

[RFC 3852] R. Housley: Cryptographic Message Syntax (CMS), July 2004[RFC 4051] D. Eastlake 3rd: Additional XML Security Uniform Resource Identifiers (URIs), April 2005

[RFC 4055] J. Schaad, B. Kaliski and R. Housley: Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, June 2005

[RFC 4231] M. Nystrom: Identifiers and Test Vectors for HMAC-SHA-224, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512, December 2005

[RFC 5280] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, W. Polk: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008

[RIPEMD-160] H. Dobbertin, A. Bosselaers und B. Preneel: RIPEMD-160: A strengthened version of RIPEMD, Fast Software Encryption – Cambridge Workshop 1996; LNCS, Band 1039, S71 – 82, Springer-Verlag; April 1996

[SigG01] Law Governing Framework Conditions for Electronic Signatures and Amending Other Regulations (Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften), Bundesgesetzblatt No. 22, 2001, p. 876, http://www.bundesnetzagentur.de/media/archive/3612.pdf

[X9.17] American National Standard X9.17:1985: Financial Institution Key Management (Wholesale)

[X9.42] American National Standard X9.42:2003: Public Key Cryptography for the Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography

[X9.52] American National Standard X9.52:1998: Triple Data Encryption Algorithm Modes of Operation. 1998

[X9.62] American National Standard X9.62:2005: Public Key Cryptography for the Financial Services Industry, The Elliptic Curve Digital Signature Algorithm (ECDSA)

[XML_DSIG] W3C: XML Signature Syntax and Processing (Second Edition), 10 June 2008, http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/

[XMLDSIG-PSS] K. Lanz, D. Bratko and P. Lipp: RSA-PSS in XMLDSig, 25 September 2007, http://www.w3.org/2007/xmlsec/ws/papers/08-lanz-iaik/

[XML_ENC] W3C: „XML Encryption Syntax and Processing“, 10 December 2002, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 7

SIGNATURE API

VERSION 2.0 – 20 JANUARY 2009

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.

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
2.0 20/Jan/2009	Complete rewrite of the previously PKCS#11 based Part 7 of ISIS-MTT 1.1.

Table of Contents

1 Preface

This part of the Common PKI specification defines a profile of the eCard API framework (BSI Technische Richtlinie TR-03112 as defined in the seven documents [TR-03112-1] to [TR03112-7]) that is specified for use in German governmental smart card applications. The eCard API is in turn mainly based on international standards, in particular [SOAP], [OASIS-DSS], [ISO24727-3], [XAdES] and [CAdES].

The eCard API defines Interfaces on three different layers as seen in Figure 1. The API profile specified in this document summarizes the high-level signature, verification, encryption an decryption functions with the necessary management and lower-level functions that a generic smart card application SHOULD call to provide signature and/or encryption functionality based on Common PKI message formats.

In other terms, an implementation of the API framework MUST at least provide these functions to the amount specified in this document. Of course it MAY implement the full extent of the eCard PKI framework.

This document contains the following sections:

- Chapter 2 specifies general mechanisms and data types of the API
- Chapter 3 lists the required API functions
- Annex A provides a header file for a C/C++ binding of the API
- Annex B provides a package definition for a Java binding of the API
- Annex C provides an extended (w.r.t. [TR-03112-4]) schema for Card Information Files
- References to the standards on which this part of Common PKI is based.

2 General Mechanisms and Data Structures

2.1 Architecture

Figure 1 illustrates the general architecture of the eCard API framework as defined in [TR-03112-1]. Note that a PKI application may directly access the API interfaces of any of the three layers below.

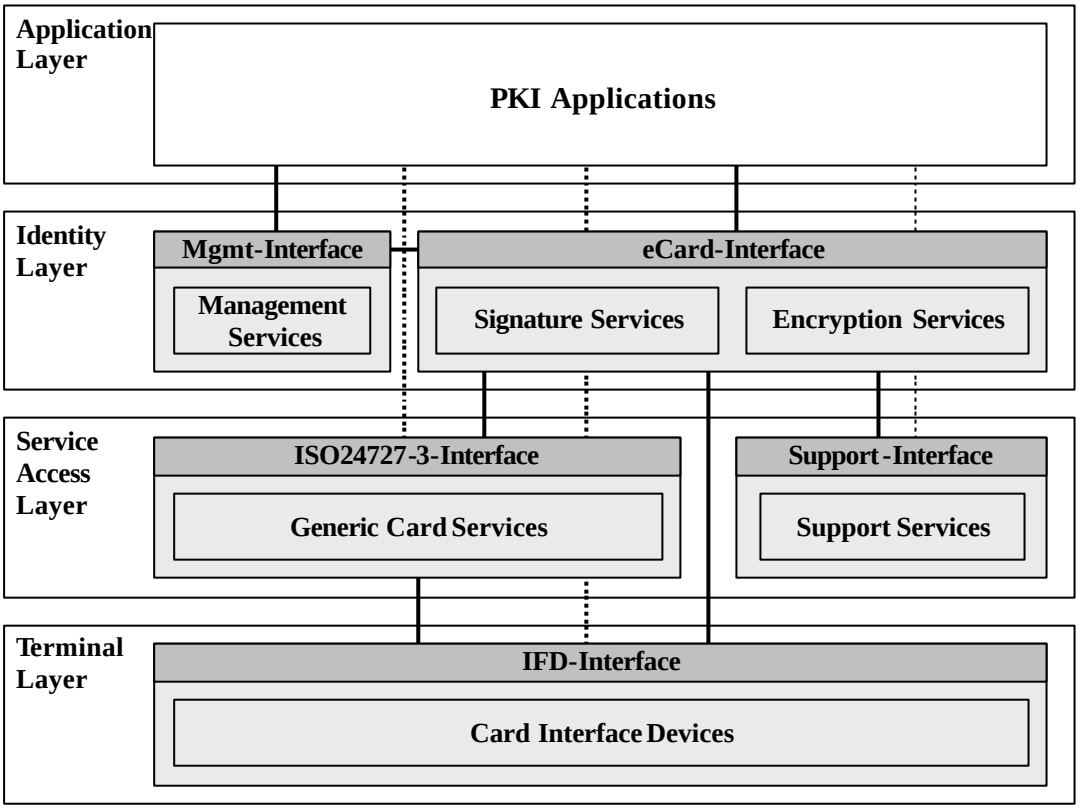


Figure 1: Architecture of the eCard API according to [TR-03112-1] (simplified)

At each of the five API interfaces, a Web service interface is to be provided. These Web service interfaces are in main parts identical to the Web services defined by [OASIS-DSS], [OASIS-EP] [ISO24727-3] and [ISO24727-4], amended by some management and support functions.

2.2 Card Info Files

In order to support specific card types on the ISO 24727 Service Interface [ISO24727-3] without the need to implement a card type specific code on top of the Generic Card Interface [ISO24727-2], the eCard API Framework employs the mechanism of Card Info files (CIFs).

A CIF contains a signed XML `CardInfo` structure that defines the mapping of generic ISO 24727 Service Interface function calls to card type specific application protocol data units (APDUs) as well as the means to recognize the respective card type by the card's answer to reset (ATR, for contact cards) or answer to select (ATS, for contactless cards) and other information.

The XML `CardInfo` structure is defined in Annex A of [TR-03112-4]. That structure needs

some amendments to be appropriate for the purposes of this Common PKI profile. Currently, there are efforts to integrate an appropriate `CardInfo` schema in an upcoming international standard via CEN. Future versions of this Common PKI specification will reference that schema in the international standard once it is finalized. For the time being, a preliminary redefinition of the `CardInfo` schema of [TR-03112-4] that is appropriate for the purposes of this profile is given in Annex C.

For the purpose of this Common PKI profile, an application MAY handle the `CardInfo` structure as an opaque data block, whereas clearly an implementation framework MUST be able to verify and use the content of a CIF.

2.3 Bindings

In implementation of the API specified MUST support a tightly coupled binding of the Web service functions via direct calls in the C/C++ or Java programming language. In this binding, the XML input/output data structures are not translated to the respective language-specific data type definitions, but rather the XML data are directly passed to wrapper functions in form of C unsigned char arrays resp. Java input/output streams.

At least one of the bindings as specified in Annex A (C/C++) and Annex B (Java) MUST be implemented, both SHOULD be implemented.

In addition, any other binding required by [TR-03112-1] such as the loosely coupled SOAP binding via HTTP (specified by [SOAPv11]) MAY also be implemented.

2.4 XML Schemas and Namespaces

The XML data type definitions in chapter 3 are based on the XML schemas shown in Table 1.

Table 1: XML Schemas and Namespaces

#	XML DATA FORMAT	NAMESPACE	COMMON PREFIX	REFERENCES	NOTES
1	XML Schema	`http://www.w3.org/2001/XMLSchema`	`xs` `xsd`	[XMLSchema]	
2	WSDL v1.1	`http://schemas.xmlsoap.org/wsdl/`	`wsdl`	[WSDLv1.1]	
3	SOAP v 1.1	`http://schemas.xmlsoap.org/wsdl/soap/`	`soap`	[SOAPv1.1]	
4	SAML v1.0	`urn:oasis:names:tc:SAML:1.0:assertion`	`saml`	[OASIS-SAML]	
5	XMLDSig	`http://www.w3.org/2000/09/xmldsig#`	`ds`	[XMLDSig]	
6	XMLEnc	`http://www.w3.org/2001/04/xmlenc#`	`xenc`	[XMLEnc]	
7	XAdES v1.2.2	`http://uri.etsi.org/01903/v1.2.2#`	`XAdES`	[XAdES]	
8	Trust Service Provider status information	`http://uri.etsi.org/02231/v2#`	`tsl`	[TS-102231]	[1]
9	OASIS DSS Core v1.0	`urn:oasis:names:tc:dss:1.0:core:schema`	`dss`	[OASIS-DSS]	
10	OASIS Advances Electronic Signature Profiles of DSS v1.0	`urn:oasis:names:tc:dss:1.0:profiles:AdES:schema#`		[OASIS-AdES]	
11	OASIS DSS Encryption Profile v0.2 (Draft)	`urn:oasis:names:tc:dss:1.0:profiles:encryption:schema#`	`dsse`	[OASIS-EP]	

12	ISO 24727 Web Service Binding	urn:iso:std:iso-iec:24727:tech:schema	iso	[ISO24727-3] [ISO24727-4] [TR-03112-4]	[2]
13	ISO 24727 IFD-API Web Service Binding	http://www.iso.org/24727	ec	[ISO24727-4]	[3]
14	OASIS DSS Verification Report v0.2 (Draft)	urn:oasis:names:tc:dss:1.0:profiles:verificationreport:schema#	vr	[OASIS-VR] [TR-03112-2]	[4]
15	eCard API data structures,	http://www.bsi.bund.de/ecard/api/1.0	ec	[TR-03112-2] [TR-03112-3] [TR-03112-5] [TR-03112-7]	[5]
16	Common PKI 2.0 Signature API	http://www.common-pki.org/xmlns/2.0/SignatureAPI/	cpsa		[6]

[1]	Available as file draft_ts102231v020101xsd.xsd from http://www.bsi.bund.de/literat/tr/tr03112/api/1.0/wsdl.zip
[2]	Available as files ISOCommon.xsd, ISO24727-3.xsd, ISO24727-3-Protocols.xsd, ISOIFD.xsd, ISOIFDCallback.xsd and CardInfo.xsd from http://www.bsi.bund.de/literat/tr/tr03112/api/1.0/wsdl.zip
[3]	Available as files ISOCommon.xsd and ISOIFD.xsd in Annex B of [ISO24727-4]
[4]	Available as file VerificationReport.xsd from http://www.bsi.bund.de/literat/tr/tr03112/api/1.0/wsdl.zip
[5]	Available as files eCard.xsd, eCard-Protocols.xsd, Support.xsd and Management.xsd from http://www.bsi.bund.de/literat/tr/tr03112/api/1.0/wsdl.zip
[6]	That namespace and prefix is used for schemas defined by or adapted to this Common PKI specification.

3 API Functions

3.1 General Request and Response Data Structures

Table 2: RequestType

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	<complexType name="RequestType"> <complexContent>	Generic Web service request data structure for the API framework.	++	++	[ISO24727-3] [TR-03112-1] 4.1.1		[1]
2	<restriction base="dss:RequestBaseType"> </restriction>	Based on the OASIS Digital Signature Service RequestBaseType.			[OASIS-DSS]	#4	
3	</complexContent> </complexType>						
4	<complexType name="RequestBaseType"> <sequence>				[OASIS-DSS]		
5	<element ref="dss:OptionalInputs" minOccurs="0"/>	Optional input parameters to a specific function request.	+-	++			
6	<element ref="dss:InputDocuments" minOccurs="0"/>	OPTIONALInput documents.	+-	++		T3	
7	</sequence> <attribute name="RequestID" type=„string" use="optional"/> <attribute name="Profile" type=„anyURI" use="optional"/> </complexType>						
[1]	**Common PKI Profile:** Generation requirements pertain to an application calling the API framework. Processing requirements pertain to the API Framework called by an application.						

Table 3: InputDocuments

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="InputDocuments">` `  <complexType>` `  <sequence>` `  <choice maxOccurs="unbounded">`	Generic input documents to an API function call.			[OASIS-DSS]		
2	`  <element ref="dss:Document"/>`		+-	++			
3	`  <element ref="dss:TransformedData"/>`		+-	++			
4	`  <element ref="dss:DocumentHash"/>`		+-	++			[2]
5	`  <element name="Other" type="dss:AnyType"/>`		+-	++			[1]
6	`  </choice>` `  </sequence>` `  </complexType>` `</element>`						
[1]	[TR-03112-2]: Possible `Other` document types are `dsse:StructuredDataType` and `dsse:OpaqueDataType`.						
[2]	[TR-03112-2]:In a `SignRequest`, his option MAY only be used for the request of time stamps.						

Table 4: ResponseType

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<complexType name="ResponseType">` `  <complexContent>`	Generic Web service response data structure for the API framework.	++	++	[ISO24727-3] [TR-03112-1] 4.1.2		[1] [2]
2	`    <restriction base="dss:ResponseBaseType">` `      <sequence>`	Based on the OASIS Digital Signature Service ResponseBaseType.			[OASIS-DSS]		
3	`        <element ref="dss:Result"/>`	Result of the API function call.	++	++	[OASIS-DSS]	T5	
4	`      </sequence>` `    </restriction>` `  </complexContent>` `</complexType>`						
[1]	**Common PKI Profile:** Generation requirements pertain to the API Framework called by an application. Processing requirements pertain to an application calling the API framework.						
[2]	[TR-03112-1]: This restricted `ResponseType` is used for all functions of the API framework with the exception of OASIS DSS compliant functions of the application level eCard interface, which use the original `dss:ResponseBaseType`.						

Table 5: Result

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="Result">` `  <complexType>` `  <sequence>`	Generic result code of an API function call.			[OASIS-DSS] [TR-03112-1] 4.1.2		
2	`  <element name="ResultMajor" type=„anyURI"/>`	Major result code indicating general success or failure.	++	++			[1]
3	`  <element name="ResultMinor" type=„anyURI" minOccurs="0"/>`	OPTIONAL minor result code indicating details on failure reasons or further information.	+-	+			[2]
4	`  <element name="ResultMessage" type="dss:InternationalStringType" minOccurs="0"/>`	OPTIONAL further result code URIs or human-readable message.	+-	+			
5	`  </sequence>` `  </complexType>` `</element>`						
[1]	[TR-03112-1]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning`						
[2]	[TR-03112-1]: `ResultMinor` MUST be included if `ResultMajor` code is `#error` or `#warning`.						

3.2 Application Level Functions

Table 6: InitializeFramework

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="InitializeFramework" type="iso:RequestType"/>`	Function call without input parameters	++	++	[TR-03112-3] 3.1.1	T2	[1]
2	`<element name="InitializeFrameworkResponse">` `  <complexType>` `  <complexContent>` `  <extension base="iso:ResponseType">` `  <sequence>`	Function output including mandatory result data structure.	++	++		T4, T5	[2]
3	`  <element name="Version" maxOccurs="1" minOccurs="1">` `  <complexType>` `  <sequence>`	Version number of the API framework started.					
4	`  <element name="Major" type="integer"/>`	Major version number.	++	+			[3]
5	`  <element name="Minor" type="integer" maxOccurs="1" minOccurs="0"/>`	Minor version number.	++ (TR: +-)	+			[3]
6	`  <element name="SubMinor" type="integer" maxOccurs="1" minOccurs="0"/>`	Sub version number below minor version.	++ (TR: +-)	+			[3]
7	`  </sequence>` `  </complexType>` `  </element>` `  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
[1]	[TR-03112-3] semantics is: Initialize the API framework including lower level API interfaces. MUST be the first API function called.						
[2]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#ParameterError`						

[3]	**Common PKI Profile:** The application calling the framework SHOULD check whether the version of the API framework. is acceptable.

Table 7: TerminateFramework

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="TerminateFramework" type="iso:RequestType"/>`	Function call without input parameters.	+	++	[TR-03112-3] 3.1.2	T2	[1]
2	`<element name="TerminateFrameworkResponse" type="iso:ResponseType"/>`	Function output including mandatory result data structure.	++	+		T4, T5	[2]
[1]	[TR-03112-3] semantics is: Terminate the current session between application and API framework interfaces. The only API function that MAY be called after TerminateFramework is another InitializeFramework..						
[2]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#ParameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#sessionTerminatedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#notInitialized`						

Table 8: SignRequest

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="SignRequest" type="dss:RequestBaseType"></element>`	Function call with input parameters.	+-	++	[OASIS-DSS] [OASIS-AdES] [OASIS-SigG] [TR-03112-2] 3.2.1	T2, T3	[1] [2] [3]
2	`<element name="SignRequestInput" type="ec:SignRequestInputType"/>` `<complexType name="SignRequestInputType">` `<sequence>`	Optional input parameters to `SignRequest`.	++	++		T2	
3	`<element name="ConnectionHandle" type="iso:ConnectionHandleType" maxOccurs="1" minOccurs="1"/>`	Reference to a connected card application.	++	++	[ISO24727-3] [TR-03112-4] 3.2.1	T26#5	[10]
4	`<element name="KeySelector" type="ec:KeySelectorType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL reference to cryptographic key and associated algorithm. Default is to use standard key and algorithm for encryption.	+-	++		#25	
5	`<choice>` `<element name="SignaturePolicy" type="anyURI"/>`	Reference to a signature policy.	+-	+-			[4]
6	`<element name="SignOptions" type="ec:SignOptionsType" maxOccurs="1" minOccurs="1"/>` `</choice>`	Explicit options for signature generation.	+-	++		#8	
7	`</sequence>` `</complexType>`						
8	`<complexType name="SignOptionsType">` `<sequence>`						
9	`<element name="SignatureForm" type="anyURI" maxOccurs="1" minOccurs="0"/>`	OPTIONAL reference to a particular XAdES or CAdES signature form, which is to be generated.	+-	++	[OASIS-AdES]		[5]

10	`<element name="SignatureType" type="anyURI" maxOccurs="1" minOccurs="0"/>`	OPTIONAL reference to a particular signature or time stamp message format, which is to be generated.	+-	++			[6]
11	`<element ref="dss:Properties" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instructions to embed particular signed on unsigned attributes in the signature to be generated.	+-	++	[OASIS-DSS] [OASIS-AdES]		[7]
12	`<element name="IncludeEContent" type="boolean" maxOccurs="1" minOccurs="0"/>`	`TRUE` for enveloping signatures. `FALSE` for detached signatures. Ignored, if given, for timestamps.	+-	++			
13	`<element ref="dss:IncludeObject" maxOccurs="unbounded" minOccurs="0"/>`	OPTIONAL list of objects to be included in an XML signature. Default is to sing the complete XML document.	+-	++	[OASIS-DSS]		
14	`<element ref="dss:SignaturePlacement" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instruction where to place the `signature` element in a signed XML document. Default is a new node at the end of the XML document.	+-	++	[OASIS-DSS]		
15	`<element ref="dss:Schemas" maxOccurs="1" minOccurs="0"></element>`	OPTIONAL set of XML schemas to be applied for validation of an XML form input document. Default is to use the configured standard schemas.	+-	++	[OASIS-DSS]		
16	`<element name="TrustedViewerInfo" type="ec:TrustedViewerInfoType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instruction to show the document to be signed in a trusted viewer before the signature is generated. Default is to skip the trusted viewer.	+-	++		#19	[3]
17	`</sequence>` `</complexType>`						
18	`<element name="PreviousTimeStampHash" type="XAdES:DigestAlgAndValueType"/>`	Data element for the previous timestamp hash signed attribute.	+-	++			[7]
19	`<complexType name="TrustedViewerInfoType">` `<sequence>`						
20	`<element name="TrustedViewerId" type="ec:TrustedViewerIdType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL reference to a trusted viewer. Default is to use the configured standard trusted viewer.	- (TR: +-)	+- (TR: ++)			[9]
21	`<element name="StyleSheet" type="ec:StyleSheetType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL style sheet for visualization of an XML document.	+-	++		#24	

22	`<element name="IncludeViewerManifest" type="boolean" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instruction whether to embed a reference to the style sheet in the signature manifest of XML signatures. Default is to embed the information for XML signatures and to omit such information for CMS signatures.	+-	++			
23	`</sequence>` `</complexType>`						
24	`<complexType name="StyleSheetType">` `<simpleContent>` `<extension base="base64Binary">` `<attribute name="StyleSheetId" type="anyURI" use="optional"/>` `</extension>` `</simpleContent>` `</complexType>`	Reference to a style sheet.					
25	`<complexType name="KeySelectorType">` `<sequence>`	Reference to cryptographic key and associated algorithm.	+-	++			
26	`<choice>` `<element ref="ds:KeyInfo"></element>`				[XMLDsig]		[8]
27	`<sequence>` `<element name="DIDName" type="iso:DIDNameType" maxOccurs="1" minOccurs="1"/>`	Name of a Differential-Identity (DID) in the card application referenced by the `ConnectionHandle` element.	+-	++			
28	`<element name="DIDScope" type="iso:DIDScopeType" maxOccurs="1" minOccurs="0"/>` `</sequence>`	OPTIONAL parameter to uniquely identify a DID. MAY be omitted if `DIDName` is already unique.	+-	++			
29	`</choice>` `<choice maxOccurs="1" minOccurs="0">`						
30	`<element name="SignInfo" type="iso:SignInfoType" maxOccurs="1" minOccurs="1"/>`	OPTIONAL reference to a particular signature algorithm or associated card command.	+-	++			
31	`<element name="EncryptionMethod" type="xenc:EncryptionMethodType"/>`	OPTIONAL reference to a particular key ancryption algorithm and associated parameters.	+-	++	[XMLEnc]		
32	`</choice>` `</sequence>` `</complexType>`						

33	`<element name="SignResponse">` `<complexType>` `<complexContent>` `<extension base="dss:ResponseBaseType">` `<sequence>`	Function output including mandatory result data structure.	+-	++	[OASIS-DSS] [TR-03112-2] 3.2.1	T5	[11] [12]
34	`<element ref="dss:SignatureObject" minOccurs="0" maxOccurs="unbounded"/>`	Signatures or timestamps, if successfully generated	++	+	[OASIS-DSS]		[13]
35	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						

[1]	[TR-03112-2] semantics is: Generate electronic signatures for the input documents. **Common PKI Profile:** The generated signatures can be qualified or advanced electronic signatures according to Common PKI Parts 1 to 8 or qualified electronic sighnatures according to Common PKI part 9 (SigG Profile).
[2]	[TR-03112-2]: Input documents are to be passed as elements of type `dss:Document`. Additionally input document type `dss:DocumentHash` MAY be used for generating timestamps. Other forms of input documents MUST NOT be used. **Common PKI Profile:** Large amounts of data (e. g. digital images) to be signed MAY be passed to the API function in form of a file by using the `dss:AttachmentReference` variant of `dss:Document`.
[3]	**Common PKI Profile:** If there are multiple input documents given for batch processing and one of these documents cannot be processed as requested, the whole batch is to be discarded. If display by a trusted viewer has been selected by the application but one of multiple documents in a batch cannot be displayed by the selected viewer (e. g. due to an unsupported document type), the whole batch has to be discarded, too.
[4]	[TR-03112-2]: An implementation of the API framework MAY support specific predefined signature policies. **Common PKI Profile:** Signature Policies MAY be supported. If an implementation of the API framework chooses to support signature policies, they MUST be specified in detail in the accompanying documentation. Signature policies MUST be compliant with all requirements and restrictions of the Common PKI profile.
[5]	**Common PKI Profile:** The signature forms `urn:oasis:names:tc:dss:1.0:profiles:AdES:forms:BES` and `urn:oasis:names:tc:dss:1.0:profiles:AdES:forms:ES-T` MUST be supported by an implementation of the API framework. Other signature forms MAY be supported. Unsupported signature form URIs will lead to the result code `/resultminor/il/signature#unknownSignatureForm`.

[6]	[TR-03112-2]: Permitted URIs are: `urn:ietf:rfc:3275` for XML-DSig (optionally XAdES) signatures `urn:ietf:rfc:3369` for CMS (optionally CAdES) signatures `http://ns.adobe.com/pdf` for PDF signatures `urn:ietf:rfc:3161` for timestamps according to [RFC3161] `urn:oasis:names:tc:dss:1.0:core: schema:XMLTimeStampToken` for XML timestamps `urn:ietf:rfc:4998` for archive timestamps according to [RFC4998] **Common PKI Profile:** XML-DSig/XAdES signatures according to the profile specified in Common PKI Part 8, CMS/CAdES signatures according to the profile specified in Common PKI Part 3 or RFC 3161 timestamps according to the profile specified in Common PKI Part 4 MUST be supported. PDF-signatures SHOULD be supported.. Other signature types MUST NOT be used and MUST lead to the result code `/resultminor/il/signature#signatureFormatNotSupported`.
[7]	[TR-03112-2]: In addition to the respective URIs specified in [OASIS-AdES], the URI `http://www.bsi.bund.de/ecard/api/1.0/properties/previousTimeStampHash` MAY be used to embed the hash value on the `Signature` respectively `TimeStampToken` element of previously generated timestamp as assigned attribute. This feature can be employed by an application to provide evidence that the signature in question has been generated after the point of time indicated by that particular timestamp. **Common PKI Profile:** Support for the signed attribute `urn:oasis:names:tc:dss:1.0:profiles:XAdES:SigningDataObjectProperties` is not required. All other signed aattributes defined in [OASIS-AdES] and [TR-03112-2] MUST be supported by an implementation of the API framework. **Common PKI Profile:** The timestamp to be used in `PreviousTimeStampHash` must be an RFC 3161 timestamp compliant with the profiling in Common PKI Part 4.
[8]	**Common PKI Profile:** Only X.509v3 public key certificates compliant with the Common PKI Part 1 or Part 9 (SigG Profile) and only signature algorithms compliant with Common PKI Part 6 MUST be used in `KeyInfo` elements.
[9]	[TR-03112-2]: `TrustedViewerId` is defined with `minOccurs="1"`. **Common PKI Profile:** The verbal description in [TR-03112-2] clearly states that this element is optional, hence `minOccurs="0"`. is considered a corrigendum.

[10]	According to [TR-03112-4] the `ConnectionHandle` is obtained by a call to the `CardApplicationConnect` function. **Common PKI Profile:** Using the functions defined in this profile, the `ConnectionHandle` can be constructed as follows: `PathSecurity` MUST be omitted. `ChannelHandle` MUST be omitted. `ContextHandle` SHOULD be omitted or otherwise left empty, see T20.[2]. `IFDName` can be obtained by a call to the `ListIFDs` function. `SlotIndex`: can be obtained by a call to the `GetStatus` function `CardApplicationIdentifier` can be omitted to reference the alpha card application by default. `CardHandle` can be obtained by a call to the `Connect` function. `RecognitionInfo` can be omitted.

[11]	[TR-03112-2]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#trustedChannelEstablishmentFailed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownProtocol` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownWebserviceBinding` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDIDName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDataSetName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDSIName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unknownSignaturePolicy` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#signatureFormatNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#PDFSignatureForNonPDFDocument` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unableToIncludeEContent` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#ignoredSignaturePlacementFlag` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateNotFound` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/service#timeStampServiceUnreachable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#resolutionOfObjectReferenceImpossible` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#transformationAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unknownViewer` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#signatureTypeDoesNotSupportSignatureFormClarificationWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unknownSignatureForm` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#includeObjectOnlyForXMLSignatureAllowed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/algorithm#hashAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/algorithm#signatureAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#securityConditionsNotSatisfied` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#noCard`
[12]	[TR-03112-2]: Enveloped XML signatures or PDF documents with embedded signatures MAY be returned as `OPTIONALOutputs` element of type `dss:DocumentWithSignature`.
[13]	[TR-03112-2]: XML-DSig/XAdES signatures are returned in form of a `ds:Signature` element. CMS/CAdES signatures are returned in form of a `dss:Base64Signature`. RFC 3161 timestamps are returned in form of a `RFC3161timeStampToken` element of a `dss:TimeStamp`.

Table 9: VerifyRequest

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="VerifyRequest">` `  <complexType>` `  <complexContent>` `  <extension base="dss:RequestBaseType">` `  <sequence>`	Function call with input parameters.	+-	++	[TR-03112-2] 3.2.2	T2	[1] [2] [3]
2	`  <element ref="dss:SignatureObject" maxOccurs="unbounded" minOccurs="0"/>`	Signatures and timestamps to verify.	++	++	[OASIS-DSS]		[3] [4]
3	`  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
4	`<element name="VerifyRequestInput" type="ec:VerifyRequestInputType"/>` `  <complexType name="VerifyRequestInputType">` `  <sequence>`	Optional input parameters to `VerifyRequest`.	++	++		T2	
5	`  <element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
6	`  <choice>` `  <element name="SignaturePolicy" type="anyURI"/>`	Reference to a signature policy.	+-	++		T8#5	
7	`  <element name="VerifyOptions" type="ec:VerifyOptionsType" maxOccurs="1" minOccurs="0"/>` `  </choice>`	OPTIONAL explicit options for signature verification. Default are the configured standard options.	+-	++		#10	[5]
8	`  <element ref="dss:AdditionalKeyInfo" maxOccurs="1" minOccurs="0"/>`	OPTIONAL further certificates required for signature verification.	+-	++	[OASIS-DSS]		
9	`  </sequence>` `  </complexType>`						
10	`  <complexType name="VerifyOptionsType">` `  <sequence>`						
11	`  <element name="UseVerificationTime" type="dss:UseVerificationTimeType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL point of time that serves as reference time for the verification process.	+-	++	[OASIS-DSS]		[6]

12	`<element name="ReturnOptions" maxOccurs="1" minOccurs="0"> <complexType> <sequence>`	OPTIONAL instructions whether the signature to be verified is to be amended by particular unsigned attributes and/or which information is to be included in the verification report. Default are the configured standard options.	+-	++			
13	`<element ref="dss:ReturnUpdatedSignature" maxOccurs="1" minOccurs="0"/>`	Instructions how the signature to be verified is to be amended by particular unsigned attributes.	+-	++	[OASIS-DSS] [OASIS-AdES]		
14	`<element name="ReportOptions" maxOccurs="1" minOccurs="0" type="vr:ReportOptionsType"/>`	Instructions, which information is to be included in the verification report.	+-	++	[OASIS-VR]		
15	`</sequence> </complexType> </element>`						
16	`<element name="CheckOptions" maxOccurs="1" minOccurs="0" type="vr:CheckOptionsType"/>`	OPTIONAL instructions which verification step are to be performed. Default are the configured standard options.	+-	++	[OASIS-VR]		
17	`<element name="SignVerificationReport" type="anyURI" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instruction whether the verification report is to be protected by a signature or timestamp of a certain `SignatureType`. Default is not to sign the verification report.	+-	++	[OASIS-VR]	T8#10	[7]
18	`<element name="TrustedViewerInfo" type="ec:TrustedViewerInfoType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL instruction to show the document to verification result in a trusted. Default is to skip the trusted viewer.	+-	++		T8#19	[3]
19	`</sequence> </complexType>`						
20	`<element name="VerifyResponse" type="dss:ResponseBaseType"/>`	Function output including mandatory result data structure.	++	+	[OASIS-DSS] [TR-03112-2] 3.2.2	T5	[8]
21	`<complexType name="VerifyRequestOutputType"> <sequence>`	Optional output parameters to `VerifyRequest`.	++	+	[TR-03112-2] 3.2.2		
22	`<element ref="dss:DocumentWithSignature" maxOccurs="unbounded" minOccurs="0"/>`	OPTIONAL documents with embedded signatures, amended by unsigned signature attributes if such an amendment was requested.	+-	+	[OASIS-DSS]		

23	`<element ref="dss:UpdatedSignature" maxOccurs="unbounded" minOccurs="0"/>`	OPTIONAL signatures, amended by unsigned signature attributes if such an amendment was requested.	+-	+	[OASIS-DSS]		
24	`<element name="VerificationReport" type="vr:VerificationReportType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL verification report if requested.	+-	+	[OASIS-VR]		[9]
25	`<element ref="dss:SignatureObject" maxOccurs="1" minOccurs="0"/>`	OPTIONAL signature or timestamp on the verification report if requested.	+-	+	[OASIS-DSS]		
26	`</sequence>` `</complexType>`						

[1] [TR-03112-2] semantics is: Verify signed objects (document signatures, timestamps, certificates etc.). Depending on the `VerifyOptions` input parameter, a partial result pertaining to certain verification steps MAY be returned.

[2] [TR-03112-2]: Documents required to verify signatures or timestamps MAY be passed to the API function as `InputDocuments` element if they are not part of `dss:SignatureObject`.

Common PKI Profile: Large amounts of signed data (e. g. digital images) to be used for verification MAY be passed to the API function in form of a file by using the `dss:AttachmentReference` variant of `dss:Document`.

[3] **Common PKI Profile:** If there are multiple input documents given for batch processing and one of these documents cannot be processed as requested, the whole batch is to be discarded. If display by a trusted viewer has been selected by the application but one of multiple documents in a batch cannot be displayed by the selected viewer (e. g. due to an unsupported document type), the whole batch has to be discarded, too.

[4] [TR-03112-2]: Other PKI data structures such as public key certificates, attribute certificates, CRLs or OCSP responses MAY be verified using the `VerifyRequest` API function by embedding them in the appropriate data element within a "dummy" XAdES signature.

[5] [TR-03112-2]: This element is defined with `minOccurs="1"`.

Common PKI Profile: The verbal description in [TR-03112-2] clearly states that this element is optional, hence `minOccurs="0"` is considered a corrigendum.

[6] [TR-03112-2]: If is `UseVerificationTime` omitted the API framework MUST determine the reference time for verification by an available time stamp or other trustworthy indication of the signature generation time. In absence of such a trustworthy time indication, the current time of verification MUST be used as reference time. In the latter case, a trustworthy timestamp or other time indication must be amended to the verification data so that a subsequent verification processes will end up with the same result.

[7] **Common PKI Profile:** The same restrictions as for the `SignatureType` element in a `SignRequest` (T8.[5]) apply.

[8]	[TR-03112-2]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#trustedChannelEstablishmentFailed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownProtocol` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownWebserviceBinding` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDataSetName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDSIName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateNotFound` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateFormatNotCorrectWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#invalidCertificateReference` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateChainInterrupted` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#resolutionOfObjectReferenceImpossible` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#transformationAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unknownViewer` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificatePathNotValidatedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateStatusNotCheckedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#signatureManifestNotCheckedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#suitabilityOfAlgorithmsNotCheckedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#detachedSignatureWithoutEContent` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#improperRevocationInformationWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#SignatureManifestNotCorrect` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/algorithm#hashAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/algorithm#signatureAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#signatureAlgorithmNotSuitable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#hashAlgorithmNotSuitable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#securityConditionsNotSatisfied`
[9]	**Common PKI Profile:** The same restrictions as for the signed properties associated with the `Properties` element in a `SignRequest` (T8.[6]) apply to the `SignedProperties` element in the verification report. **Common PKI Profile:** The verification report MUST indicate which validation model (the PKIX model according to Common PKI Part 5 or the SigG model according to Common PKI Part 9) was used in the verification process.

Table 10: EncryptRequest

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	<element name="EncryptRequest" type="dsse:EncryptRequestType"/>	Function call with input parameters.	+-	++	[OASIS-EP] [TR-03112-2] 3.3.1	T2, T3	[1] [2] [3]
2	<complexType name="EncryptionRequestInputType"> <sequence>	Optional input parameters to EncryptRequest.	+-	++			
3	<element name="ConnectionHandle" type="iso:ConnectionHandleType" maxOccurs="1" minOccurs="0"/>	OPTIONAL reference to a connected card application.	-- (TR +-)	+	[ISO24727-3] [TR-03112-4] 3.2,1	T26#5, T8.[10]	[4]
4	<element name="KeySelector" type="ec:KeySelectorType" maxOccurs="1" minOccurs="0"></element>	OPTIONAL reference to cryptographic key and associated algorithm. Default is to use standard key and algorithm for encryption.	-- (TR +-)	+		T8#25	[4]
5	<element name="encryptionMethod" type="xenc:EncryptionMethodType" minOccurs="0" />	OPTIONAL reference to an encryption method to be used.	+- (TR --)	++			[5]
6	<element name="RecipientCertificate" type="ds:X509DataType" maxOccurs="unbounded" minOccurs="0" />	Encryption certificate(s) of one or more intended recipients.	++ (TR --)	++			[5]
7	</sequence> </complexType>						
8	<element name="EncryptResponse" type="dsse:EncryptResponseType"/>	Function output including mandatory result data structure.	++	+	[OASIS-EP] [TR-03112-2] 3.3.1	T5	[6]
9	<complexType name=”EncryptResponseType”> <complexContent> <extension base= »dss :ResponseBaseType »> <sequence> </sequence>		++	+	[OASIS-EP]		[7]

10	`<element name="OutputDocuments"> <complexType> <sequence maxOccurs="unbounded"> <element ref= »dss :Document »/> </sequence> </complexType> </element>`	Encrypted output documents, if encryption was successful.	+-	+	[OASIS-EP] [OASIS-DSS]	T3	[8]
11	`</extension> </complexContent> </complexType>`						

[1]	[TR-03112-2] semantics is: Encrypt the input documents.
[2]	[TR-03112-2], [OASIS-EP]: Input documents of type `dss:Document` are to be encrypted as a whole. **Common PKI Profile:** For XML input documents, the encrypted message format MUST be XML-Enc according to the profile specified in Common PKI Part 8. For non-XML input documents, the encrypted message format MUST be CMS according to the profile specified in Common PKI Part 3. **Common PKI Profile:** Large amounts of data (e. g. digital images) to be encrypted MAY be passed to the API function in form of a file by using the `dss:AttachmentReference` variant of `dss:Document`.
[3]	[TR-03112-2], [OASIS-EP]: By using input documents of type `dsse:StructuredDataType`, parts of XML documents can be encrypted or opaque encrypted data can be inserted into XML documents. **Common PKI Profile:** The encrypted message format MUST be XML-Enc according to the profile specified in Common PKI Part 8.
[4]	**Common PKI Profile:** The `ConnectionHandle` and `KeySelector` elements MUST be omitted.
[5]	**Common PKI Profile:** The `EncryptionMethod` and `RecipientCertificate` elements are Common PKI extensions to the `EncryptRequest` function input.

[6]	[TR-03112-2]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#trustedChannelEstablishmentFailed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownProtocol` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownWebserviceBinding` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDataSetName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDSIName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateNotFound` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateFormatNotCorrectWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#invalidCertificateReference` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateChainInterrupted` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/service#ocspResponderUnreachable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/service#directoryServiceUnreachable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificatePathNotValidatedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateStatusNotCheckedWarning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#digitalSignatureNotCorrect` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#signatureAlgorithmNotSui table` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#invalidCertificatePath` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#certificateRevoked` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#referenceTimeNotWithinCertificateValidityPeriod` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/encryption#encryptionOfCertainNodesOnlyForXMLDocuments` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/encryption#encryptionFormatNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/encryption#invalidCertificate` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/key#keyGenerationNotPossible` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/key#encryptionAlgorithmNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#securityConditionsNotSatisfied`
[7]	[TR-03112-2],[OASIS-EP]: If input documents were of type `dsse:StructuredDataType`, output documents with encrypted parts are returned as `dss:OPTIONALOutputs` element of the response in form of `dsse:EncryptedEstructuredData`.
[7]	[TR-03112-2],[OASIS-EP]: Output documents are encrypted `dss:Document` elements.

Table 11: DecryptRequest

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="DecryptRequest" type="dsse:DecryptRequestType"/>`	Function call with input parameters.	+-	++	[OASIS-EP]	T2, T3	[1]
2	`<complexType name="DecryptionRequestInputType"> <complexContent> <extension base="ec:EncryptionRequestInputType"> <sequence>`	Optional input parameters to `DecryptRequest`. `ConnectionHandle` and `KeySelector` elements are inherited from `EncryptionRequestInputType`.	++	++	[OASIS-EP] [TR-03112-2] 3.3.2	T10#2	[2]
3	`<element ref="dsse:EncryptedStructuredData" maxOccurs="unbounded" minOccurs="0"/>`		+-	++			[3]
4	`</sequence> </extension> </complexContent> </complexType>`			++			
5	`<element name="DecryptResponse" type="dsse:DecryptResponseType"/>`	Function output including mandatory result data structure.	++	+	[OASIS-EP] [TR-03112-2] 3.3.2	T5	[4]
6	`<complexType name="DecryptResponseType"> <complexContent> <extension base="dss:ResponseBaseType"> <sequence>`		++	+			
7	`<element name="OutputDocuments"> <complexType> <sequence maxOccurs="unbounded"> <element ref="dss:Document"/> </sequence> </complexType> </element>`	Decrypted output documents, if output was successful.	++	+	[OASIS-DSS]		
8	`</sequence> </extension> </complexContent> </complexType>`						
[1]	[TR-03112-2] semantics is: Decrypt encrypted documents.						

[2]	[TR-03112-2], [OASIS-EP]: Encrypted input documents of type `dss:Document` are passed to the API function as `InputDocuments` element of the request. **Common PKI Profile:** Large amounts of encrypted data (e. g. digital images) to be decrypted MAY be passed to the API function in form of a file by using the `dss:AttachmentReference` variant of `dss:Document`.
[3]	[TR-03112-2], [OASIS-EP]: Partially encrypted input documents of type `dsse:EnhcryptedStructuredData` are passed to the API function as optional input of the request.
[4]	[TR-03112-2]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#trustedChannelEstablishmentFailed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownProtocol` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownWebserviceBinding` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDIDName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/encryption#encryptionFormatNotSupported` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#decryptionNotPossible` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#securityConditionsNotSatisfied` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#fileNotFound` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#noCard`

Table 12: ShowViewer

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="ShowViewer">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-2] 3.2.3	T2	[1]
2	`<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`<element name="TrustedViewerId" type="ec:TrustedViewerIdType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL identifier for selecting a certain trusted viewer. Default is selecting the configured default viewer.	+-	++		#11	[2]
4	`<element name="Document" type="dss:DocumentType" maxOccurs="unbounded" minOccurs="0"/>`	OPTIONAL document(s) to be displayed. Subject to on the effective security policy, a trusted viewer MAY decide to display only a subset or overview of multiple similar documents.	+-	++	[OASIS-DSS]		
5	`<element name="StyleSheetContent" type="base64Binary" maxOccurs="1" minOccurs="0"/>`	OPTIONAL XSL style sheet that the framework MAY use to display XML content.	+-	+-	[XSLv1.1]		
6	`<element name="ViewerMessage" maxOccurs="1" minOccurs="0">` `<complexType>` `<sequence>`	OPTIONAL message to be displayed by the trusted Viewer. Default is to use standard messages of the trusted viewer.	+-	+			
7	`<element name="FrameMessage" type="string" maxOccurs="1" minOccurs="0"/>`	OPTIONAL message to be displayed in the Windows title or heading.	+-	+			
8	`<element name="BodyMessage" type="string" maxOccurs="1" minOccurs="0"/>` `</sequence>` `</complexType>` `</element>`	OPTIONAL message to be displayed in the windows body-	+-	+			

9	<element name="Timeout" type="nonNegativeInteger" maxOccurs="1" minOccurs="0"/>	OPTIONAL number of seconds, after which the trusted viewer windows(s) will be automatically closed without user interaction. Default SHOULD be to close the viewer window(s) after 30 seconds.	+-	+			
10	</sequence> </extension> </complexContent> </complexType> </element>						
11	<simpleType name="TrustedViewerIdType"> <restriction base="string"> <maxLength value="64"/> </restriction> </simpleType>						
12	<element name="ShowViewerResponse" type="iso:ResponseType">	Function output including mandatory result data structure.	++	+	[TR-03112-2] 3.2.3	T4, T5	[3]
[1]	[TR-03112-2] semantics is: Show signed objects and/or verification results in a trusted viewer component.						
[2]	[TR-03112-2]: Reference to an unknown/unsupported viewer will result in the error message `signature#unknownViewer`.						
[3]	[TR-03112-2]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#warning` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#trustedChannelEstablishmentFailed` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownProtocol` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownWebserviceBinding` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/viewer#timeout` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/viewer#cancelationByUser` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/signature#unknownViewer` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/viewer#unsuitableSylesheetForDocument` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/il/viewer#viewerMessageTooLong`						

Table 13: GetTrustedViewerList

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="GetTrustedViewerList">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-3] 3.4.1	T2	[1]
2	`<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
4	`<element name="GetTrustedViewerListResponse">` `<complexType>` `<complexContent>` `<extension base="iso:ResponseType">`	Function output including mandatory result data structure.	++	+	[TR-03112-3] 3.4.1	T4, T5	[2]
5	`<sequence>` `<element name="TrustedViewerId" type="ec:TrustedViewerIdType" maxOccurs="unbounded" minOccurs="0"/>` `</sequence>`	List of the available trusted viewer components.	++	+		T12#11	
6	`</extension>` `</complexContent>` `</complexType>` `</element>`						
[1]	[TR-03112-3] semantics is: Retrieve a list of all available trusted viewer components.						
[2]	[TR-03112-2]: Reference to an unknown/unsupported viewer will result in the error message `signature#unknownViewer`.						
[2]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`						

Table 14: GetTrustedViewerConfiguration

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	<element name="GetTrustedViewerConfiguration"> <complexType> <complexContent> <extension base="iso:RequestType"> <sequence>	Function call with input parameters.	+-	++	[TR-03112-3] 3.4.2	T2	[1]
2	<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	<element name="TrustedViewerId" type="ec:TrustedViewerIdType"/>	ID of the trusted viewer for which configuration information is to be retrieved.	++	++		T12#11	[2]
4	</sequence> </extension> </complexContent> </complexType> </element>						
5	<element name="GetTrustedViewerConfigurationResponse"> <complexType> <complexContent> <extension base="iso:ResponseType"> <sequence>	Function output including mandatory result data structure.	++	+	[TR-03112-3] 3.4.2	T4, T5	[3]
6	<element name="ViewerConfiguration" type="ec:ViewerConfigurationType"/>	Trusted viewer configuration information as it could be retrieved.	++	+		#8	
7	</sequence> </extension> </complexContent> </complexType> </element>						
8	<complexType name="ViewerConfigurationType"> <sequence>						
9	<element name="SupportedDocumentTypes" maxOccurs="unbounded" minOccurs="0"> <complexType> <sequence>	Information which document types the viewer supports.	+-	++			[4]

10	`<element name="MimeType" type="string"/>`	MIME content type associated with a document type supported by the viewer.	++	++	[RFC2045]		
11	`<element name="Application" type="string" maxOccurs="1" minOccurs="0"/>`	OPTI'ONAL name of an application associated with the MIME type above.	+-	+			
12	`<element name="StyleSheet" type="dss:InlineXMLType" maxOccurs="unbounded" minOccurs="0"/>`	OPTIONAL set of style sheets that the viewer employs to display particular content.	+-	+			
13	`</sequence>` `</complexType>` `</element>`						
14	`<element name="IFDName" type="string" maxOccurs="1" minOccurs="0"/>`	OPTIONAL reference to a card terminal that is logically associated with the trusted viewer.	+-	+			
15	`</sequence>` `</complexType>`						

[1]	[TR-03112-3] semantics is: Retrieve the configuration information about a particular trusted viewer.
[2]	`[TR-03112-2]: Reference to an unknown/unsupported viewer will result in the error message signature#unknownViewer.`
[3]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/TrustedViewer#invalidID` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`
[4]	

Table 15: GetCardInfoList

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="GetCardInfoList">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-3] 3.2.1	T2	[1]
2	`<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
4	`<element name="GetCardInfoListResponse">` `<complexType>` `<complexContent>` `<extension base="iso:ResponseType">`	Function output including mandatory result data structure.	++	+	[TR-03112-3] 3.2.1	T4, T5	[2]
	`<sequence maxOccurs="1" minOccurs="1">` `<element name="CardInfo" type="iso:CardInfoType" maxOccurs="unbounded" minOccurs="0"/>` `</sequence>`	List of registered `CardInfo` structures.	++	+	[TR-03112-4] Annex A	Section 2.2	
5	`</extension>` `</complexContent>` `</complexType>` `</element>`						
[1]	[TR-03112-3] semantics is: List all card types known by means of `CardInfo` files.						
[2]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`						

Table 16: SetCardInfoList

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="SetCardInfoList">` `  <complexType>` `  <complexContent>` `  <extension base="iso:RequestType">` `  <sequence>`	Function call with input parameters.	+-	++	[TR-03112-3] 3.2.2	T2	[1]
2	`  <element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`  <element name="CardInfo" type="iso:CardInfoType" maxOccurs="unbounded" minOccurs="0"/>`	List of `CardInfo` structures to register..	+-	++	[TR-03112-4] Annex A	Section 2.2	[2]
4	`  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
5	`<element name="SetCardInfoListResponse" type="iso:ResponseType"/>`	Function output including mandatory result data structure.			[TR-03112-3] 3.2.2	T4, T5	[3]
[1]	[TR-03112-3] semantics is: Store a list of `CardInfo` structures. The order of these structures is relevant for the recognition of card types.						
[2]	[TR-03112-3]: The list MAY be empty. That feature can be employed by an application to clear the current `CardInfo` list before registering new CIFs using the `AddCardInfoFiles` function.						
[3]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/CardInfo#incorrectFile` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`						

Table 17: AddCardInfoFiles

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="AddCardInfoFiles">` `  <complexType>` `  <complexContent>` `  <extension base="iso:RequestType">` `  <sequence>`	Function call with input parameters.			[TR-03112-3] 3.2.3	T2	[1]
2	`  <element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`  <element name="CardInfo" type="iso:CardInfoType" maxOccurs="unbounded" minOccurs="1"/>`	`CardInfo` structures to add to the current list, if not yet present in that list.	++	++	[TR-03112-4] Annex A	Section 2.2	
4	`  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
5	`<element name="AddCardInfoFilesResponse" type="iso:ResponseType"/>`	Function output including mandatory result data structure.			[TR-03112-3] 3.2.3	T4, T5	[2]
[1]	[TR-03112-3] semantics is: Append `CardInfo` structures from files for additional card types to the `CardInfo` list. During the import consistency of the card information and signatures, if available, on the `CardInfo` files MUST be verified. **Common PKI Profile:** All `CardInfo` files (CIF) must be signed.						
[2]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/CardInfo#addNotPossible` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/CardInfo#alreadyExisting` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/CardInfo#incorrectFile` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `  http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`						

Table 18: DeleteCardInfoFiles

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="DeleteCardInfoFiles">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.			[TR-03112-3] 3.2.4	T2	[1]
2	`<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)	[TR-03112-4] 3.1.3		
3	`<element name="CardTypeIdentifier" type="anyURI" maxOccurs="unbounded" minOccurs="0"/>`	Unique identifiers of `CardInfo` structures to remove from the currently registered list.	+-	++	[TR-03112-4] Annex A.3	Section 2.2	[2]
4	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
5	`<element name="DeleteCardInfoFilesResponse" type="iso:ResponseType"/>`	Function output including mandatory result data structure.			[TR-03112-3] 3.2.4	T4, T5	[3]
[1]	[TR-03112-3] semantics is: Delete zero or more `CardInfo` files.						
[2]	[TR-03112-3] Annex A.3: The unique identifying URI of a `CardInfo` structure is the sub-element `ObjectIdentifier` of the `CardType` data element.						
[3]	[TR-03112-3]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0 /resultminor/al/CardInfo#notExisting` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/CardInfo#deleteNotPossible` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChannelHandle`						

Table 19: GetProductInfo

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="GetProductInfo" type="iso:RequestType"/>`	Function call without input parameters.	+-	++		T2	[1]
2	`<element name="GetProcuctInfoResponse">` `  <complexType>` `  <complexContent>` `  <extension base="iso:ResponseType">` `  <sequence>`	Function output including mandatory result data structure.	++	+		T4, T5	[2]
3	`  <element name="ProducerName" type="string"/>`	Manufacturer of the API framework implementation	++	+			
4	`  <element name="ProductName" type="string"/>`	Product name of the API framework implementation	++	+			
5	`  <element name="Version" type="string"/>`	Version number of the API framework implementation	++	+			
6	`  <element name="BuildNo" type="string" minOccurs="0"/>`	OPTIONAL build number of the API framework implementation	+-	+			
7	`  <element name="ProducerAdditions" type="anyType" minOccurs="0"/>`	OPTIONAL additional data provided by the manufacturer of the API framework implementation	+-	+			
8	`  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
[1]	**Common PKI Profile:** This is an additional function. Semantics is: Provide information about the API framework implementation.						
[2]	**Common PKI Profile:** Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` or `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common# UnknownAPIFunction` if the API framework implementation does not support this function.						

3.3 Card and Reader Service Level Functions

Table 20: ListIFDs

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="ListIFDs">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-6] 3.1.3	T2	[1]
2	`<element name="ContextHandle" type="iso:ContextHandleType"/>`	Reference to a terminal layer session.	++	++	[ISO24727-4] [TR-03112-6] 3.1.1	#4	[2]
3	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
4	`<simpleType name="ContextHandleType">` `<restriction base="hexBinary"> </restriction>` `</simpleType>`						
5	`<element name="ListIFDsResponse">` `<complexType>` `<complexContent>` `<extension base="iso:ResponseType">`	Function output including mandatory result data structure.	++	+	[TR-03112-6] 3.1.3	T4, T5	[3]
6	`<sequence>` `<element name="IFDName" maxOccurs="unbounded" minOccurs="0" type="string"/>` `</sequence>`	Unique names of available card terminals	++	+			
7	`</extension>` `</complexContent>` `</complexType>` `</element>`						
[1]	[TR-03112-6] semantics is: List all card terminals available to the API framework.						
[2]	According to [TR-03112-6] the `ContextHandle` is obtained by a call to the `EstablishContext` function. **Common PKI Profile:** An empty `ContectHandle` is used to reference the default context established by the `InitializeFramwork` function. In functions where this element is optional it SHOULD be omitted.						

[2]	[TR-03112-6]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#unknownContextHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#timeout`

Table 21: GetStatus

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	<element name="GetStatus"> <complexType> <complexContent> <extension base="iso:RequestType"> <sequence>	Function call with input parameters.	+-	++	[TR-03112-6] 3.1.5	T2	[1]
2	<element name=”ContextHandle” type=”iso:ContextHandleType” maxOccurs=”1” minOccurs=”1”/>	Reference to a terminal layer session.	++	++	[ISO24727-4] [TR-03112-6] 3.1.1	T20#4 T20.[2]	
3	<element name=”IFDName” type=”string” maxOccurs=”1” minOccurs=”0”/>	Name of	+-	++			[2]
4	</sequence> </extension> </complexContent> </complexType> </element>						
5	<element name="GetStatusResponse"> <complexType> <complexContent> <extension base="iso:ResponseType">	Function output including mandatory result data structure.	++	+	[TR-03112-6] 3.1.5	T4, T5	[3]
6	<sequence maxOccurs=”1” minOccurs=”1”> <element name=”IFDStatus” maxOccurs=”unbounded” minOccurs=”0” type=”iso:IFDStatusType”/> </sequence>	Card terminal status.	++	+		#8	
7	</extension> </complexContent> </complexType> </element>						
8	complexType name="IFDStatusType"> <sequence>	Status of a single card terminal.			[TR-03112-6] 3.1.5		
9	<element name="IFDName" type="string" maxOccurs="1" minOccurs="0"/>	Unique name of the card terminal.	++	+			

10	`<element name="Connected" type="boolean" maxOccurs="1" minOccurs="0"/>`	OPTIONAL indication whether a connection to the card terminal is available. MAY be omitted if the card terminal is permanently attached to the local system.	+-	+			
11	`<element minOccurs="1" maxOccurs="unbounded" name="SlotStatus" type="iso:SlotStatusType"/>`	Status of the slot(s) available in the card terminal.	++	+		#17	
12	`<element name="ActiveAntenna" type="boolean" maxOccurs="1" minOccurs="0"/>`	Indication whether a coupling antenna for contactless cards is activated. MUST be omitted if the card terminal is for contact cards only.	+-	+			
13	`<element minOccurs="0" maxOccurs="unbounded" name="DisplayStatus" type="iso:SimpleFUStatusType"/>`	Status information about the available display(s). MUST be omitted if there is no display available in the card terminal.	+-	+		#22	
14	`<element minOccurs="0" maxOccurs="unbounded" name="KeyPadStatus" type="iso:SimpleFUStatusType"/>`	Status information about the available key pad(s). MUST be omitted if there is no key pad available in the card terminal.	+-	+		#22	
15	`<element minOccurs="0" maxOccurs="unbounded" name="BioSensorStatus" type="iso:SimpleFUStatusType"/>`	Status information about the available biometric sensor(s). MUST be omitted if there is no biometric sensor available in the card terminal.	+-	+		#22	
16	`</sequence>` `</complexType>`						
17	`<complexType name="SlotStatusType">` `<sequence>`	Status of a slot within the card terminal.			[TR-03112-6] 3.1.5		
18	`<element name="Index" type="nonNegativeInteger" maxOccurs="1" minOccurs="1"/>`	Slot index within the card terminal.	++	+			
19	`<element minOccurs="1" maxOccurs="1" name="CardAvailable" type="boolean"/>`	`TRUE` if there is a card available in the slot.	++	+			
20	`<element name="ATRorATS" type="hexBinary" maxOccurs="1" minOccurs="0"/>`	MUST hold the card's ATR (Answer To Reset for contact cards) or ATS (Answer To Select for contactless cards) if `CardAvailable` is `True`. Otherwise the element MUST be omitted.	+-	+			
21	`</sequence>` `</complexType>`						
22	`<complexType name="SimpleFUStatusType">` `<sequence>`	Status of available devices (displays, key pad or biometric sensors) within a card terminal.			[TR-03112-6] 3.1.5		

23	`<element name="Index" type="nonNegativeInteger"/>`	Index of the device.	++	+			
24	`<element name="Available" type="boolean"/>`	`TRUE` if the device is available to the application.	++	+			
25	`</sequence>` `</complexType>`						
[1]	[TR-03112-6] semantics is: Retrieve the status of one or all card terminals available to the API framework.						
[2]	The `IFDName` can be obtained by a call to the `ListIFDs` function.						
[3]	[TR-03112-6]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#unknownContextHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#unknownIFDName`						

Table 22: GetCardInfo

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="GetCardInfo">` `<complexType>` `<choice>`	Function call with input parameters.	+-	++	[TR-03112-5] 3.4	T2	[1]
2	`<element name="ConnectionHandle" type="iso:ConnectionHandleType"/>`	Reference to a connected card application. If given, the `CardInfo` structure for the respective card type will be returned.	+-	++	[ISO24727-3] [TR-03112-4] 3.2,1	T26#5, T8.[10]	
3	`<sequence>` `<element name="Action" type="anyURI" maxOccurs="1" minOccurs="0"/>`	OPTIONAL indication how the CIFs to retrieve from the repository are to be selected.	+-	++			[2]
4	`<element name="CardTypeIdentifier" type="anyURI" maxOccurs="unbounded" minOccurs="0"/>` `</sequence>`	If given, unique identifiers of `CardInfo` structures to retrieve (or exclude from retrieval) from the repository server.	+-	++	[TR-03112-4] Annex A.3	Section 2.2	[3]
5	`</choice>` `</complexType>` `</element>`						
6	`<element name="GetCardInfoResponse">` `<complexType>` `<complexContent>` `<extension base="iso:ResponseType">` `<sequence minOccurs="1" maxOccurs="1">`	Function output including mandatory result data structure.	++	+	[TR-03112-5] 3.4	T4, T5	[4]
7	`<element name="CardInfo" type="iso:CardInfoType" maxOccurs="unbounded" minOccurs="0"/>`	The requested `CardInfo` structure(s) if they could be retrieved.	++	+	[TR-03112-4] Annex A	Section 2.2	
8	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
[1]	[TR-03112-5] semantics is: Retrieve Information about a card currently available to the API framework or retrieve `CardInfo` files (CIF) from a designated repository server.						

[2]	[TR-03112-3]: The possible actions are: `http://www.bsi.bund.de/ecard/api/1.0/cardinfo/action#getSepcifiedFile` (default) to get the CIF specified by the subsequent card type identifier(s), `http://www.bsi.bund.de/ecard/api/1.0/cardinfo/action#getRelatedFiles` to get all CIFs related to the subsequent card type identifier(s) and `http://www.bsi.bund.de/ecard/api/1.0/cardinfo/action#getOtherFiles` to get alls the CIFs available at the repository, except for those specified by the subsequent card type identifier(s).
[3]	[TR-03112-3] Annex A.3: The unique identifying URI of a `CardInfo` structure is the sub-element `ObjectIdentifier` of the `CardType` data element.
[4]	[TR-03112-5]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownConnectionHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownCardType` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal/support#cardInfoRepositoryUnreachable`

Table 23: Connect

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	<element name="Connect"> <complexType> <complexContent> <extension base="iso:RequestType"> <sequence>	Function call with input parameters.	+-	++	[TR-03112-6] 3.2.1	T2	[1]
2	<element name="ContextHandle" type="iso:ContextHandleType" maxOccurs="1" minOccurs="1"/>	Reference to a terminal layer session.	++	++	[ISO24727-4] [TR-03112-6] 3.1.1	T20#4, T20,[2]	
3	<element name="IFDName" type="string"/>	Name of the card terminal.	++	++			[2]
4	<element name="Slot" type="nonNegativeInteger"/>	Index of a slot within the card terminal.	++	++			[3]
5	<element name="Exclusive" type="boolean" maxOccurs="1" minOccurs="0"/>	Set to TRUE if the card is to be blocked exclusively for the application.	++	++			
6	</sequence> </extension> </complexContent> </complexType> </element>						
7	element name="ConnectResponse"> <complexType> <complexContent> <extension base="iso:ResponseType"> <sequence>	Function output including mandatory result data structure.	++	+	[TR-03112-6] 3.2.1	T4, T5	[4]
8	<element name="CardHandle" type="iso:CardHandleType" maxOccurs="1" minOccurs="0"/>	Reference to the connected card if the connection attempt was successful.	++	+		#10	
9	</sequence> </extension> </complexContent> </complexType> </element>						
10	<simpleType name="CardHandleType"> <restriction base="hexBinary"> </restriction> </simpleType>	Reference to a connected card.					

[1]	[TR-03112-6] semantics is: Establish a connection to a card.
[2]	The `IFDName` can be obtained by a call to the `ListIFDs` function.
[3]	The `SlotIndex` can be obtained by a call to the `GetStatus` function.
[4]	[TR-03112-6]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#unknownContextHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#exclusiveNotAvailable` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#unknownIFDName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#slotIndexNotExisting` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#noCard`

Table 24: Disconnect

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="Disconnect">` `  <complexType>` `  <complexContent>` `  <extension base="iso:RequestType">` `  <sequence>`	Function call with input parameters.	+	++	[TR-03112-6] 3.2.2	T2	[1]
2	`  <element name="CardHandle" type="iso:CardHandleType"/>`	Reference to the connected card.	++	++		T23#10	
3	`  <element name="Action" type="iso:ActionType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL indication of the action to be performed with the card.	+-	++		#5	
4	`  </sequence>` `  </extension>` `  </complexContent>` `  </complexType>` `</element>`						
5	`<simpleType name="ActionType">` `  <restriction base="string">` `  <enumeration value="Reset"/>` `  <enumeration value="Unpower"/>` `  <enumeration value="Eject"/>` `  <enumeration value="Confiscate"/>` `  </restriction>` `</simpleType>`	Indication of the action to be performed with the card upon deactivation.	+-	++	[TR-03112-6] 3.2.2		[2]
6	`<element name="DisconnectResponse" type="iso:ResponseType"/>`	Function output including mandatory result data structure.			[TR-03112-6] 3.2.2	T4, T5	[3]
[1]	[TR-03112-6] semantics is: Terminate the connection to a card.						
[2]	**Common PKI Profile:** `Reset` and `Unpower` MUST be supported by the framework. `Eject` and `Confiscate` SHOULD be supported, if the IFD provides the respective mechanical capability. An application SHOULD NOT rely on `Eject` or `Confiscate` actions.						

[3]	[TR-03112-6]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#invalidCardHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#timeout` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#mechanicalFunctionNotSupported`

Table 25: VerifyUser

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="VerifyUser">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-6] 3.3.1	T2	[1]
2	`<element name="CardHandle" type="iso:CardHandleType"/>`	Reference to a connected card.	++	++			[2]
3	`<element name="InputUnit" type="iso:InputUnitType"/>`	Device and method to be used for user authentication.	++	++		#10	
4	`<element name="DisplayIndex" type="nonNegativeInteger" maxOccurs="1" minOccurs="0 /">`	OPTIONAL index of the display device in the card terminal to be used for user interface messages.	+-	++			
5	`<element name="AltVUMessages" type="iso:AltVUMessagesType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL alternative user interface messages provided by the application. Default is to use standard messages.	+-	+			
6	`<element name="TimeoutUntilFirstKey" type="positiveInteger" maxOccurs="1" minOccurs="0"/>`	OPTIONAL timeout if the user does not type a key, in milliseconds.	+-	+			[3]
7	`<element name="TimeoutAfterFirstKey" type="positiveInteger" maxOccurs="1" minOccurs="0"/>`	OPTIONAL timeout if the user types an insufficient number of keys, in milliseconds.	+-	+			[3]
8	`<element name="Template" type="hexBinary"/>`	APDU template for the verify command according to [ISO7816-4]	++	++			
9	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
10	`<complexType name="InputUnitType">` `<choice>`				[TR-03112-6] 3.3.1		
11	`<element name="PinInput" type="iso:PinInputType"/>`	Use a PIN input device.	++	++		#15	

12	`<element name="BiometricInput" type="iso:BiometricInputType"/>`	Use a biometric input device.	-- (TR +-)	+- (TR ++)		#31	
13	`</choice>` `</complexType>`						
14	`<complexType name="PinInputType">` `<sequence>`	Details of a PIN input device.	++	++	[TR-03112-6] 3.3.1		
15	`<element name="Index" type="nonNegativeInteger"/>`	Index of the input device within the card terminal.	++	++			
16	`<element name="PasswordAttributes" type="iso:PasswordAttributesType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL PIN/password attributes according to [ISO7816-15] and [ISO7816-15AM2].	+-	++			
17	`</sequence>` `</complexType>`						
18	`<simpleType name="PadCharType">` `<restriction base="hexBinary">` `<length value="1" fixed="true"/>` `</restriction>` `</simpleType>`	Padding Character.					
19	`<complexType name="PasswordAttributesType">` `<sequence>`	PIN/password policy.			[ISO7816-15] [ISO7816-15AM2] [TR-03112-6] 3.3.1		
20	`<element name="pwdFlags" type="iso:PasswordFlagsType"/>`	Information about the nature of the PIN.	++	++		#28	
21	`<element name="pwdType" type="iso:PasswordTypeType"/>`	Character set used for the PIN.	++	++		#29	
22	`<element name="minLength" type="nonNegativeInteger"/>`	Minimal number of characters.	++	++			
23	`<element name="storedLength" type="nonNegativeInteger"/>`	Number of PIN characters stored in the card.	++	++			
24	`<element name="maxLength" type="nonNegativeInteger" maxOccurs="1" minOccurs="0"/>`	Maximal number of characters.	+-	++			
25	`<element name="padChar" type="iso:PadCharType" maxOccurs="1" minOccurs="0"/>`	OPTIONAL padding character used if more then `minLength` characters are stored in the card.	+-	++		#18	

26	`element name="lastPasswordChange" type="dateTime" maxOccurs="1" minOccurs="0"/>`	OPTIONALtime of last PIN change.	+-	++			
27	`</sequence>` `</complexType>`						
28	`<simpleType name="PasswordFlagsType">` `<union memberTypes="iso:BitString">` `<simpleType>` `<list>` `<simpleType>` `<restriction base="token">` `<enumeration value="case-sensitive"/>` `<enumeration value="local"/>` `<enumeration value="change-disabled"/>` `<enumeration value="unblock-disabled"/>` `<enumeration value="initialized"/>` `<enumeration value="needs-padding"/>` `<enumeration value="unblockingPassword"/>` `<enumeration value="soPassword"/>` `<enumeration value="disable-allowed"/>` `<enumeration value="integrity-protected"/>` `<enumeration value="confidentiality-protected"/>` `<enumeration value="exchangeRefData"/>` `<enumeration value="resetRetryCounter1"/>` `<enumeration value="resetRetryCounter2"/>` `</restriction>` `</simpleType>` `</list>` `</simpleType>` `</union>` `</simpleType>`	Password attribute flags.	+-	++	[ISO7816-15] [ISO7816-15AM2] [TR-03112-6] 3.3.1		
29	`<simpleType name="PasswordTypeType">` `<restriction base="string">` `<enumeration value="bcd"/>` `<enumeration value="ascii-numeric"/>` `<enumeration value="utf8"/>` `<enumeration value="half-nibble-bcd"/>` `<enumeration value="iso9564-1"/>` `</restriction>` `</simpleType>`	Character set used for the PIN.	+-	++	[ISO7816-15] [TR-03112-6] 3.3.1		[4]

30	<simpleType name="BitString"> <restriction base="string"> <pattern value="[0-1]{0,}"/> </restriction> </simpleType>						
31	<complexType name="BiometricInputType"> <sequence>	Details of a biometric input device.	-- (TR +-)	+- (TR ++)	[TR-03112-6] 3.3.1		
32	<element name="Index" type="nonNegativeInteger"/>	Index of the input device within the card terminal.	-- (TR +-)	+- (TR ++)			
33	<element name="BiometricSubtype" type="nonNegativeInteger"/>	Subtype of biometric method according to the BioAPI specification.	-- (TR +-)	+- (TR ++)	[ISO19784-1]		
34	</sequence> </complexType>						
35	<complexType name="AltVUMessagesType"> <sequence>	Alternative user interface messages provided by the application.					
36	<element name="AuthenticationRequestMessage" type="string" maxOccurs="1" minOccurs="0"/>	Initial prompt for authentication.	+-	+			
37	<element name="SuccessMessage" type="string" maxOccurs="1" minOccurs="0"/>	Successful authentication response to the user.	+-	+			
38	<element name="AuthenticationFailedMessage" type="string" maxOccurs="1" minOccurs="0"/>	Failed authentication response to the user. SHOULD indicate that the card may be blocked due to the failed attempt.	+-	+			
39	<element name="RequestConfirmationMessage" type="string" maxOccurs="1" minOccurs="0"/>	Prompt to repeat the last input.	+-	+			
40	<element name="CancelMessage" type="string" maxOccurs="1" minOccurs="0"/>	Cancelled authentication response to the user.	+-	+			
41	</sequence> </complexType>						
42	<element name="VerifyUserResponse"> <complexType> <complexContent> <extension base="iso:ResponseType"> <sequence>	Function output including mandatory result data structure.			[TR-03112-6] 3.3.1	T4, T5	[5]
43	<element name="Response" type="hexBinary" maxOccurs="1" minOccurs="1"/>	Return code of the card (e. g. 90 00 for successful authentication)					

44	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						

[1]	[TR-03112-6] semantics is: Initiate a user authentication to the card via PIN or biometric means.
[2]	The `CardHandle` can be obtained by a call to the `Connect` function.
[3]	**Common PKI Profile:** If the card terminal comprises a display, an appropriate cancel message SHOULD be displayed by the card terminal if a timeout during PIN entry occurred.
[4]	**Common PKI Profile:** The additional value `iso9564-1` means that the PIN is to be encoded in the 2 PIN Block format according to [ISO9564-1].
[5]	[TR-03112-6]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl#cancelationByUser` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#timeout` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#invalidCardHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#noCard` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/terminal#IFDBusy` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/IO#unknownInputDevice` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/IO#unknownBiometricSubtype`

Table 26: DSIRead

#	XML SCHEMA DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	BASE STANDARDS.	COMMON PKI	
1	`<element name="DSIRead">` `<complexType>` `<complexContent>` `<extension base="iso:RequestType">` `<sequence>`	Function call with input parameters.	+-	++	[TR-03112-4] 3.4.9	T2	[1]
2	`<element name="ConnectionHandle" type="iso:ConnectionHandleType"/>`	Reference to a connected card application.	++	++	[ISO24727-3] [TR-03112-4] 3.2,1	#5, T8.[10]	
3	`<element name="DSIName" type="iso:DSINameType"/>`	Name of the DSI that is to be read.	++	++		#20	[2]
4	`</sequence>` `</extension>` `</complexContent>` `</complexType>` `</element>`						
5	`<complexType name="ConnectionHandleType">` `<complexContent>`	Reference to a connected card application.	++	++	[TR-03112-4] 3.2,1		
6	`<extension base="iso:CardApplicationPathType">` `<sequence>`	Reference to a card application.				#13	
7	`<element name="CardHandle" type="iso:CardHandleType"/>`	Reference to a connected card.	++	++		T23#10	[3]
8	`<element name="RecognitionInfo" maxOccurs="1" minOccurs="0">` `<complexType>` `<sequence>`	OPTIONAL additional info for selecting the card that contains the DSI to be read.	-- (TR +-)	+- (TR ++)			
9	`<element name="CardType" type="anyURI" maxOccurs="1" minOccurs="0"/>`	OPTIONAL card type identifier.	-- (TR +-)	+- (TR ++)			
10	`<element name="CaptureTime" type="dateTime" maxOccurs="1" minOccurs="0"/>`	OPTIONAL specification of the time when the card was recognized.	-- (TR +-)	+- (TR ++)			
11	`<element name="ICCSN" type="string" maxOccurs="1" minOccurs="0"/>`	OPTIONAL card serial number.	-- (TR +-)	+- (TR ++)			

12	</sequence> </complexType> </element> </sequence> </extension> </complexContent> </complexType>						
13	<complexType name="CardApplicationPathType"> <sequence>	Reference to a card application.			[TR-03112-4] 3.1.3		
14	<element name="ChannelHandle" type="iso:ChannelHandleType" maxOccurs="1" minOccurs="0"/>	OPTIONAL parameter for addressing remote systems. Default is addressing local system.	-- (TR +-)	+- (TR ++)			
15	<element name="ContextHandle" type="iso:ContextHandleType" maxOccurs="1" minOccurs="0"/>	OPTIONAL reference to a terminal layer session.	- (TR +-)	++	[ISO24727-4] [TR-03112-6] 3.1.1	T20#4, T20.[2]	
16	<element name="IFDName" maxOccurs="1" minOccurs="0" type="string"/>	OPTIONAL name of the card terminal in which the card that contains the DSI to be read is inserted.	+-	++			[4]
17	< element name="SlotIndex" type="nonNegativeInteger" maxOccurs="1" minOccurs="0" >	OPTIONAL index of a slot within the card terminal.	+-	++			[5]
18	<element name="CardApplicationIdentifier" maxOccurs="1" minOccurs="0" type="iso:ApplicationIdentifierType"/>	OPTIONAL identifier of a card application.	-- (TR +-)	+- (TR ++)	[ISO24727-3]		
19	</sequence> </complexType>						
20	simpleType name="DSINameType"> <restriction base="string"> <minLength value="1"/> <maxLength value="255"/> </restriction> </simpleType>				[ISO24727-3]		
21	<element name="DSIReadResponse"> <complexType> <complexContent> <extension base="iso:ResponseType"> <sequence>	Function output including mandatory result data structure.	++	+	[TR-03112-4] 3.4.9	T4, T5	[6]
22	<element name="DSIContent" type="hexBinary" maxOccurs="1" minOccurs="0"/>	Value of the DSI, if it could be read successfully.	++	+			

23	`        </sequence>` `      </extension>` `    </complexContent>` `  </complexType>` `</element>`						

[1]	[TR-03112-4] semantics is: Read a Data Structure for Interoperability (DSI) in the selected data set of a card application. **Common PKI Profile:** This function is required to read certificates from a card.
[2]	**Common PKI Profile:** Only DSIs representing certificates SHOULD be read.
[3]	The `CardHandle` can be obtained by a call to the `Connect` function.
[4]	The `IFDName` can be obtained by a call to the `ListIFDs` function.
[5]	The `SlotIndex` can be obtained by a call to the `GetStatus` function.
[6]	[TR-03112-4]: Possible major and minor result codes are: `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#ok` `http://www.bsi.bund.de/ecard/api/1.0/resultmajor#error` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#noPermission` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#internalError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/al/common#parameterError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#communicationError` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/dp#unknownChanelHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownConnectionHandle` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#notInitialized` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#unknownDSIName` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#prerequisitesNotSatisfied` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/sal#securityConditionsNotSatisfied` `http://www.bsi.bund.de/ecard/api/1.0/resultminor/ifdl/common#unknownContextHandle`

Annexes

Annex A: C/C++ Binding

This annex provides the contents of a header file `cpsigapi.h` for the C/C++ binding of the Common PKI Signature API.

Listing 1: File `CPSigAPI.h`

```
#ifndef CPSIGAPI_H
#define CPSIGAPI_H

/*
 * Copyright (c) 2008, T7 e.V.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * - Neither the name of T7 e.V. nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
```

```
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */

#ifdef __cplusplus
extern "C" {
#endif

#if defined(WIN32)
  #define CPSIGAPIEXPORT_FOR_WIN32
#endif

#ifndef CPSIGAPIEXPORT_INEXPORT
  #ifdef CPSIGAPIEXPORT_FOR_WIN32
    #define CPSIGAPIEXPORT_INEXPORT __declspec(dllimport)
  #else
    #define CPSIGAPIEXPORT_INEXPORT
  #endif
#endif

#define CPSIGAPIEXPORT_RET(ret) CPSIGAPIEXPORT_INEXPORT ret _stdcall

/**
  *
  * \brief Common PKI Signature API Context
  * Contect definition
  *
  */
typedef void* CPSigAPIContext;

/**
  *
  * \brief Aquire Common PKI Signature API Context
  *
  */
```

```
CPSIGAPIEXPORT_RET(CPSigAPIContext)
CPSigAPIAquireContext
        (
        );

/**
  *
  * \brief Free Common PKI Signature API Context
  *
  */
CPSIGAPIEXPORT_RET(void)
CPSigFreeContext
        (
        CPSigAPIContext context
        );

/**
  *
  * \brief Excecute Common PKI Signature API Context
  *
  * Execution of a Common PKI Signature API function names 'function'.
..* Function call parameter is the context handler 'context'.
..* This contect handler must previously be allocated via CPSigAPIAquireContext.
..* The function call to 'function'is then execured in that context.
..* Input and output parameter to 'function' are passed as XML structures as specified
..* in the Common PKI Signature API.
..* Input parameters are passed in buffer 'xmlInput' with size 'xmlInputSize'.
..* For the output parameters of the function, the caller  allocates and passes
  * a buffer 'xmlOutput' of 'xmlOutputSize' bytes.
  * If that buffer is suffucient, it will be used. If the buffer is too small,
  * the function will return with an error and indicate the required output buffer
  * size in 'xmlOutputSize'.The XML function result may then be retrieved by a second
  * function call using no input ('xmlInput' a Null pointer and 'xmlInputSize' zero)
  * and a reallocates, sufficiently large output buffer.
  *
  * \param   context       Context in which the function 'function' is to be executed
  * \param   function      Name of the Common PKI Signature APi function to be executed
```

```
  * \param   xmlInput      Buffer containing the input parameters to 'function' in
  *                       form of an XML structure
  * \param   xmlInputSize  Size of the input buffer
  * \param   xmlOutput     Buffer fort he XMl result structure
  * \param   xmlOutputSize Size of the result buffer; will be set to the size of the XML result
  *                       structure upon success and upon a CPSIGAPI_BUFFERTOSMALL error
  * \param   error         Detailed error code if the function result is -1
  *
  * \return  0             Function could be executed successfully. The result of the call
  *                        is placed in 'xmlOutput'. the size of the XML result structure is
  *                        placed in 'xmlOutputSize'.
  *         -1             An error occured. The variable 'error' contains a detailed
  *                        error code. Possible error codes are CPSIGAPI_SUCCESS,
  *                        CPSIGAPI_BUFFERTOSMALL, CPSIGAPI_UNKNOWNFUNCTION and
  *                        CPSIGAPI_UNKNOWNERROR.
  */
#define CPSIGAPI_SUCCESS          0    /**< no error                           */
#define CPSIGAPI_UNKNOWNERROR     1    /**< internal error                     */
#define CPSIGAPI_BUFFERTOSMALL    2    /**< output buffer too small for result */
#define CPSIGAPI_UNKNOWNFUNCTION  3    /**< unknown function name              */

CPSIGAPIEXPORT_RET(int)
CPSigAPIExecute
        (
        CPSigAPIContext context,
        unsigned char const* xmlInput,
        int xmlInputSize,
        unsigned char const* xmlOutput,
        int *xmlOutputSize,
        unsigned long* error
        );

/**
  *
  * \brief GetErrorMessage
  *
  * If an error occured during CPSigAPIExecute, i. e. its return value is -1 and the error
```

```
  * code in the 'error' parameter is CPSIGAPI_UNKNOWNERROR, i. e. an internal error of the
  * C wrapper layer occurred, the C wrapper interal error messaage can be retrieved using this
  * function.
  *
  * \return               Internal C wrapper error message (null-terminated string)
  */
CPSIGAPIEXPORT_RET(char *)
CPSigAPIGetErrorMessage
        (
        );

#ifdef __cplusplus
}
#endif

#endif
```

Annex B: Java Binding

This annex provides the contents of definition files of the package `org.common-pki.signatureapi` for the Java binding of the Common PKI Signature API.

Listing 2: File `ECardApiService.java`

```
/*
 * Copyright (c) 2008, T7 e.V.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * - Neither the name of T7 e.V. nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
```

```
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.common-pki.signatureapi;

import java.util.Iterator;

import sun.misc.Service;

/**
 * Access the VM singleton for {@link IECardApiService}.
 * <p>
 * To make this work, just do one of the following:
 * <ul>
 * <li> set a {@link IECardApiService} of your choice in {@link ECardApiService}.</li>
 * <li> include a service provider file
 * "META-INF/services/org.common-pki.signatureapi.IECardApiService" contain just the
 * class name of your implementation in your deployment (jar-file). </li>
 * </ul>
 *
 */
public class ECardApiService {

  private static IECardApiService ACTIVE;

  private static IECardApiService findNativeInterface() {
    ClassLoader loader = Thread.currentThread().getContextClassLoader();
    if (loader == null) {
      loader = ECardApiService.class.getClassLoader();
    }
    IECardApiService impl = null;
    Iterator ps = Service.providers(IECardApiService.class, loader);
    if (ps.hasNext()) {
      impl = (IECardApiService) ps.next();
    }
    return impl;
  }
```

```
  public static synchronized IECardApiService get() {
    if (ACTIVE == null) {
      set(findNativeInterface());
    }
    return ACTIVE;
  }

  public static synchronized void set(IECardApiService eCardApiServiceImpl) {
    ACTIVE = eCardApiServiceImpl;
  }

}
```

Listing 3: File `IECardApiService.java`

```
/*
 * Copyright (c) 2008, T7 e.V.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * - Neither the name of T7 e.V. nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without specific
```

```
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.common-pki.signatureapi;

import java.io.IOException;

/**
 * The interface IECardApiService provides direct access to aCommon PKI Signature API
 * implementation. Calls to the eCard API implementation are stripped down to
 * the SOAP message's body content only.
 *
 */
public interface IECardApiService {

  public void service(IECardApiRequest request, IECardApiResponse response)
      throws IOException;

}
```

Listing 4: File `IECardApiRequest.java`

```
/*
 * Copyright (c) 2008, T7 e.V.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * - Neither the name of T7 e.V. nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.common-pki.signatureapi;

import java.io.InputStream;

/**
 * A eCard API service request providing just the contents of the SOAP request's
```

```
     * &lt;body> tag.
     *
     * For example:
     *
     * <pre>
     *  &lt;m:GetCertificate xmlns:m=&quot;http://www.bsi.bund.de/ecard/api/1.0&quot;&gt;
     *  &lt;m:GetCertificateRequest&gt;
     *   ...
     *  &lt;/m:GetCertificateRequest&gt;
     *  &lt;/m:GetCertificate&gt;
     * </pre>
     *
     */
    public interface IECardApiRequest {

      /**
       * @return a InputStream containing the contents of the SOAP request's
       *         &lt;body> tag
       */
      public InputStream getInputStream();

    }
```

Listing 5: File `IECardApiResponse.java`

```
    /*
     * Copyright (c) 2008, T7 e.V.
     *
     * Redistribution and use in source and binary forms, with or without
     * modification, are permitted provided that the following conditions are met:
```

```
 *
 * - Redistributions of source code must retain the above copyright notice,
 *   this list of conditions and the following disclaimer.
 *
 * - Redistributions in binary form must reproduce the above copyright notice,
 *   this list of conditions and the following disclaimer in the documentation
 *   and/or other materials provided with the distribution.
 *
 * - Neither the name of T7 e.V. nor the names of its contributors may be used
 *   to endorse or promote products derived from this software without specific
 *   prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
 * POSSIBILITY OF SUCH DAMAGE.
 */
package org.common-pki.signatureapi;

import java.io.OutputStream;

/**
 *
 * A eCard API service response providing just the contents of the SOAP
 * responses' &lt;body> tag.
 *
 * For example:
 *
 * <pre>
 *  &lt;m:GetCertificateResponse xmlns:m=&quot;http://www.bsi.bund.de/ecard/api/1.0&quot;&gt;
```

```
 *  ...
 *  &lt;/m:GetCertificateResponse&gt;
 * </pre>
 *
 */
public interface IECardApiResponse {

  public OutputStream getOutputStream();

}
```

Annex C: Schema for Card Information Files

The following schema is a redefinition of the one available in files CardInfo.xsd, ISO24727-3.xsd, ISO24727-Protocols.xsd , ISOCommon.xsd and ISOIFD.xsd from http://www.bsi.bund.de/literat/tr/tr03112/api/1.0/wsdl.zip. See also the remarks in section 2.2.

Listing 6: File `CommonPKICardInfo.xsd`

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema" xmlns:iso="urn:iso:std:iso-iec:24727:tech:schema"
targetNamespace="urn:iso:std:iso-iec:24727:tech:schema">

  <!-- ========================== -->
  <!--      BEGIN <redefine>      -->
  <!-- ========================== -->

  <redefine schemaLocation="CardInfo.xsd">


    <!-- ========================== -->
    <!--        PIN Compare         -->
    <!-- ========================== -->

    <complexType name="PinCompareQualifierType">
      <complexContent>
        <extension base="iso:PinCompareQualifierType">
          <sequence>
            <element name="RetryCounterProtocol" type="anyURI" minOccurs="0">
              <annotation>
                <documentation>
                  Protocol for determining the current value of the PIN retry counter
                  URIs will be assigned for strictly ISO 7816 compatible cards,
                  CardOS, StarCOS and other card operating systems if required
                  Protocol 'urn:t7:cards:pin:eci:none'
                    describes the fact that a card does not provide information about the retry counter
```

```
              </documentation>
            </annotation>
          </element>
          <element name="OperationUsageCounter" type="iso:OperationUsageCounterType" minOccurs="0"
maxOccurs="unbounded">
            <annotation>
              <documentation>
                Information about the allowed number of certain operations
                before a new validation of the PIN is required
              </documentation>
            </annotation>
          </element>
          <element name="PinInitializationInfo" type="iso:PinInitializationInfoType" minOccurs="0"/>
          <element name="PinInitializationCheck" type="iso:PinInitializationCheckType" minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>


  <!-- ========================= -->
  <!--   Signature Generation    -->
  <!-- ========================= -->

  <complexType name="SignInfoType">
    <complexContent>
      <extension base="iso:SignInfoType">
        <sequence>
          <element name="SignatureGenerationSequence" type="iso:SignatureGenerationSequenceType"
minOccurs="0"/>
        </sequence>
      </extension>
    </complexContent>
  </complexType>
</redefine>

<!-- ========================= -->
```

```
<!--       END <redefine>      -->
<!-- ========================= -->

<!-- ====================== -->
<!--     Usage Counter      -->
<!-- ====================== -->

<complexType name="OperationUsageCounterType">
  <simpleContent>
    <annotation>
      <documentation>
          Type of an operation and number of these operations that may
          be performed upon a single validation of the PIN,
          for the 'signature' operation this corresponds to the SSEC value
        </documentation>
    </annotation>
    <extension base="nonNegativeInteger">
      <attribute name="Operation">
        <simpleType>
          <restriction base="string">
            <enumeration value="signature"/>
            <enumeration value="decryption"/>
            <enumeration value="authentication"/>
            <enumeration value="encryption"/>
          </restriction>
        </simpleType>
      </attribute>
    </extension>
  </simpleContent>
</complexType>


<!-- ====================== -->
<!--   PIN Initialization   -->
<!-- ====================== -->

<complexType name="simpleDataMaskType">
```

```
      <sequence>
        <annotation>
          <documentation>
            Used for checking the status bytes in a cards response APDU.
            If the status bytes in a bit-wise logical AND conjunction
            with the Mask element correspond to the Value element,
            the response is considered consistent
          </documentation>
        </annotation>
        <element name="Value" type="hexBinary"/>
        <element name="Mask" type="hexBinary" minOccurs="0"/>
      </sequence>
    </complexType>
    <complexType name="PinInitializationInfoType">
      <complexContent>
        <restriction base="iso:DIDAbstractQualifierType">
          <annotation>
            <documentation>
              Protocol 'urn:t7:cards:pin:init:fixed'
                describes initialization with a fixed transport PIN
                TransportPinDID is optional. If it is given, the conversion of the transport PIN
                  must be done in two steps: first a VERIFY command for PIN verification,
                  the a CHANGE REFERENCE DATA command for changing the signature PIN.
                  The format for transmission of transport and signature PIN depends on the
                  respective DIDInfo elements. If the TransportPinDID element is omitted,
                  the conversion of the transport PIN must be performed by a
                  CHANGE REFERENCE DATA command that contains both transport PIN
                  and new signature PIN. The format for transmission of transport and signature PIN
                  depends on the DIDInfor element of the signature PIN.
                BulkInitialization
                  If multiple PINS are stored and are to be initialized using the same transport PIN,
                  the respective DIDInfo elements can express this by referencing the same
                  transport PIN object in the TransportPinDID element and a 'true' value in the optional
                  BulkInitialization element. In that case all PINs referencing that transport PIN must be
                  initialized after verification of the transport PIN.
                  Default if the element is omitted shall be 'false'.
                TransportPINValue is optional. I fit is given, it contains the value of the transport PIN
```

```
            as hexadecimal string.If necessary that value must be embedded in a 2PIN Block
            or padded before transmitting it to the card. If an empty hex string is given, an empty
            value must be transmitted to the card. If the element is omitted, the PIN can be
            initialized immediately, without using a transport PIN. For that end, APDU parameter
            P1 must be set to 0x01.
        Protocol 'urn:t7:cards:pin:init:oncard'
          describes the case that the transport PIN is stored in a file on the card.
          TransportPinDID as in the 'urn:t7:cards:pin:init:fixed' protocol
          BulkInitialization as in the 'urn:t7:cards:pin:init:fixed' protocol.
          TransportPinDSI references the data object on the card that contains the transport PIN.
            That data object may be a binary EF or a record.
        Protocol 'urn:t7:cards:pin:init:user'
          describes the case that the transport PIN is transferred to the user by an out-of-band
          mechanism, e. g. a PIN letter.
          TransportPinDID as in the 'urn:t7:cards:pin:init:fixed' protocol.
          BulkInitialization as in the 'urn:t7:cards:pin:init:fixed' protocol.
      </documentation>
      </annotation>
      <sequence>
        <element name="TransportPinDID" type="string" minOccurs="0"/>
        <element name="BulkInitialization" type="boolean" minOccurs="0"/>
        <element name="TransportPinValue" type="hexBinary" minOccurs="0"/>
        <element name="TransportPinDSI" type="string" minOccurs="0"/>
        <element name="PostInitializationCommands" minOccurs="0">
          <annotation>
            <documentation>
              PostInitializationCommands can be used to store the state resulting from
              PIN initialization in a file. The optional data element consists of a sequence
              of command and response APDUs. The command aPDUs are sent to the card
              and the answers are checked for conformity with the respective resporse APDUs.
              If an error occurs, the execurtion of post initialization commands is interrupted.
              In any case the card is reset after post initialization commands have been executed.
            </documentation>
          </annotation>
          <complexType>
            <sequence minOccurs="0" maxOccurs="unbounded">
              <element name="CommandApdu" type="hexBinary"/>
```

```
              <element name="ResponseApdu" type="iso:simpleDataMaskType"/>
            </sequence>
          </complexType>
        </element>
      </sequence>
    </restriction>
  </complexContent>
</complexType>


<!-- ============================ -->
<!--   PIN Initialization Check   -->
<!-- ============================ -->

<complexType name="PinInitializationCheckType">
  <complexContent>
    <restriction base="iso:DIDAbstractQualifierType">
      <annotation>
        <documentation>
        Protocol 'urn:t7:cards:pin:initcheck:verify'
          determines the PIN initialization state by sending a VERIFY command with empty data field to the
card.
          PinUsable defines the expected status byte response if the PIN is in a usable state.
          PinTransportState defines the expected status byte response if the PIN has not yet been
initialized.
          Both elements are mutually exclusive.
        Protocol 'urn:t7:cards:pin:initcheck:file'
          determines the PIN initialization state based on a file on the card.
          PinStatusDSI references the DSI that stores information about the PIN inititalization state.
          PinUsable defines the expected status byte response after evaluation of the DSI,
            if the PIN is in a usable state.
          PinTransportState defines the expected status byte response after evaluation of the DSI,
            if the PIN has not yet been initialized.
          The last two elements are mutually exclusive.
        </documentation>
      </annotation>
      <sequence>
```

```
        <element name="PinStatusDSI" type="string" minOccurs="0"/>
        <choice>
          <element name="PinUsable" type="iso:simpleDataMaskType"/>
          <element name="PinTransportState" type="iso:simpleDataMaskType"/>
        </choice>
      </sequence>
    </restriction>
  </complexContent>
</complexType>


<!-- ======================= -->
<!--   SignatureGeneration   -->
<!-- ======================= -->

<complexType name="CommandType">
  <annotation>
    <documentation>
      placeholder for command references
    </documentation>
  </annotation>
  <sequence>
  </sequence>
</complexType>
<complexType name="SignatureGenerationSequenceType">
  <annotation>
    <documentation>
      Required sequence of commands for generating a qualified signature
    </documentation>
  </annotation>
  <sequence>
    <choice minOccurs="0">
      <element name="MSE_RESTORE_ONCE" type="iso:CommandType"/>
      <element name="MSE_RESTORE_ALWAYS" type=" iso:CommandType"/>
    </choice>
    <element name="MSE_HASH" type="iso:CommandType" minOccurs="0"/>
    <element name="PSO_HASH" type="iso:CommandType" minOccurs="0"/>
```

```
      <choice minOccurs="0">
        <element name="MSE_KEY" type="iso:CommandType"/>
        <element name="MSE_DS" type="iso:CommandType"/>
        <element name="MSE_KEY_DS" type="iso:CommandType"/>
      </choice>
      <element name="PSO_CDS" type="iso:CommandType" minOccurs="0"/>
    </sequence>
    <attribute name="id" type="integer"/>
  </complexType>
</schema>
```

References

[CAdES] ETSI TS 101 733 v1.7.4: Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES), July 2008

[ISO19784-1] ISO/IEC 19784-1: "Information technology – Biometric application programming interface – Part 1: BioAPI specification", (FDIS) Version 2005-03-06

[ISO24727-2] ISO/IEC 24727-2: "Identification Cards – Integrated Circuit Cards Programming Interfaces – Part 2: Generic card interface", (FDIS-Ballot) Version 2007-10-25

[ISO24727-3] ISO/IEC 24727-3: "Identification Cards – Integrated Circuit Cards Programming Interfaces – Part 3: Application Interface", (FCD) Version 2007-09-14

[ISO24727-4] ISO/IEC 24727-4: "Identification Cards – Integrated Circuit Cards Programming Interfaces – Part 4: API Administration", (FCD) Version 2007-10-31

[ISO7816-4] ISO/IEC 7816-4: "Identification cards – Integrated circuit cards – Part 4:Organization, security and commands for interchange", Version 2005-01-15

[ISO7816-15] ISO/IEC 7816-15: "Identification cards – Integrated circuit(s) cards with contacts – Part 15: Cryptographic information application", (FDIS) Version 2003-02-12

[ISO7816-15AM2] ISO/IEC 7816-15 Amendment 2: "Identification cards – Integrated circuit(s) cards with contacts – Part 15: Cryptographic information application – Amendment for modifications and error corrections on ISO/IEC 7816- 15", (FDIS-Ballot) Version 2007-09-18

[ISO9564-1] ISO 9564-1: Banking – Personal Identification Number (PIN) management and security – Part 1: Basic principles and requirements for online PIN handling in ATM and POS systems

[OASIS-AdES] OASIS: "Advanced Electronic Signature Profiles of the OASIS Digital Signature Service", Version 1.0, http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles-AdES-spec-v1.0-os.pdf

[OASIS-DSS] OASIS: "Digital Signature Service Core Protocols, Elements, and Bindings", Version 1.0, http://docs.oasis-open.org/dss/v1.0/oasis-dss-core-spec-v1.0-os.pdf

[OASIS-EP] OASIS / C. Orthacker (A-SIT): "Proposal for an Encryption Profile for OASIS DSS", A-SIT Contribution 01, 20 September 2007, http://www.oasis-open.org/committees/download.php/25384/oasis-dss_profile-encryption_A-SIT_v0.1.doc

[OASIS-SAML] OASIS: "Security Assertion Markup Language (SAML)", Version 1.0, http://www.oasis-open.org/committees/download.php/2290/oasis-sstc-saml-1.0.zip

[OASIS-SigG] OASIS: "German Signature Law Profile of the OASIS Digital Signature Service", Version 1.0, http://docs.oasis-open.org/dss/v1.0/oasis-dss-profiles_german_signature_law-spec-v1.0-os.pdf

[OASIS-VR] OASIS / D. Hühnlein: "Profile for comprehensive multi-signature verification reports for OASIS Digital Signature Services Version 1.0", working draft, 05 May 2008, http://www.oasis-open.org/committees/download.php/28182/2008_05_05_oasis-dss-profile-for-comprehensive-signature-verification-report.doc

[PAOSv1.1] Liberty Alliance Project: "Liberty Reverse HTTP Binding for SOAP Specification, Version v1.1", http://www.projectliberty.org/liberty/content/download/1219/7957/file/liberty-paos-v1.1.pdf

[RFC2045] N. Freed, N. Borenstein: "RFC 2045: Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", http://www.ietf.org/rfc/rfc2045.txt

[RFC3161] C. Adams, P. Cain, D. Pinkas, R. Zuccherato: "RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP)", http://www.ietf.org/rfc/rfc3161.txt

[RFC4998] T. Gondrom, R. Brandner, U. Pordesch: "RFC 4998: Evidence Record Syntax (ERS)", http://www.ietf.org/rfc/rfc4998.txt

[SOAPv1.1] W3C Note: "Simple Object Access Protocol (SOAP) 1.1", 08 May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508

[TR-03112-1] BSI - Technische Richtlinie TR-03112-1: "eCard-API-Framework – Teil 1 – Überblick und übergreifende Mechanismen", Version 1.0, 03 March 2008, (in German language)

[TR-03112-2] BSI - Technische Richtlinie TR-03112-2: "eCard-API-Framework – Teil 2 – eCard-Interface", Version 1.0, 03 March 2008, (in German language)

[TR-03112-3] BSI - Technische Richtlinie TR-03112-3: "eCard-API-Framework – Teil 3 – Management-Interface", Version 1.0, 03 March 2008, (in German language)

[TR-03112-4] BSI - Technische Richtlinie TR-03112-4: "eCard-API-Framework – Teil 4 – ISO24727-3-Interface", Version 1.0, 03 March 2008, (in German language)

[TR-03112-5] BSI - Technische Richtlinie TR-03112-5: "eCard-API-Framework – Teil 5 – Support-Interface", Version 1.0, 03 March 2008, (in German language)

[TR-03112-6] BSI - Technische Richtlinie TR-03112-6: "eCard-API-Framework – Teil 6 – Reader-Interface", Version 1.0, 03 March 2008, (in German language)

[TR-03112-7] BSI - Technische Richtlinie TR-03112-7: "eCard-API-Framework – Teil 7 – Protokolle", Version 1.0, 03 March 2008, (in German language)

[TS-102231] ETSI TS 102 231: "Provision of harmonized Trust Service Provider (TSP) status information"

[WSDLv1.1] W3C Recommendation: "Web Services Description Language (WSDL) Version 1.1", http://www.w3.org/TR/wsdl

[XAdES] ETSI TS 101 903: "XML Advanced Electronic Signatures (XAdES)", V1.2.2 (2004-04)

[XMLDSig] W3C Recommendation: "XML Signature Syntax and Processing", 10 June 2008, http://www.w3.org/TR/xmldsig-core/

[XMLEnc] W3C Recommendation: "XML Encryption Syntax and Processing", 10 December 2002, http://www.w3.org/TR/xmlenc-core/

[XMLSchema] W3C Recommendation. "XML Schema" Part 0 to Part 2, 28 October 2004, http://www.w3.org/TR/xmlschema-0/, -1/ and-2/

[XSLv1.1] W3C Recommendation "Extensible Stylesheet Language (XSL) Version 1.1", 05 December 2006, http://www.w3.org/TR/xsl

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 8

XML BASED MESSAGE FORMATS

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.2 27.10.2003	First public edition
1.1 16.03.2004	Several editorial changes. 1) Chapters 1 and 2 have been combined and renamed as Preface. 2) Chapter 5 has been integrated into Part 6 with the exception of canonicalization, transforms, and decoding. 3) Superfluous references have been deleted.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Renamed to "XML based Message Formats"- Adapted to new versions of the base standards: - ETSI TS 101 903 v1.3.2 - OASIS Standard 200401 (WS-Security 2004) - RFC 3852 - X.509:2005 - http://www.w3.org/TR/2001/REC-xmlschema-1-20041028/ - http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/ - http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/ Various corrections and clarifications.

Table of Contents

1 Preface

This part of the Common PKI specification provides the Common PKI profile for XML signatures. The XML signature format conforms to the most widely accepted international XML_DSIG standard [XML_DSIG] and to the OSCI-profile [OSCI]. OSCI has been issued to trim the XML_DSIG format to the needs of eGovernment and allows wide interoperability of the applications by restricting the formats and contents to a well-defined subset of possible options allowed by XML_DSIG.

The Common PKI profile for XML signatures is based on [XML_DSIG], [XML_ENC], and [XAdES]. It is also a general signature profile that is coherent to [OSCI]. OSCI as a SOAP dialect is a specification that has strong roots in the public sector in Germany (and beyond) and one of the aims of this profile is to harmonize Common PKI and OSCI. OSCI can now be redefined as a special signature profile based on this Common PKI general XML signature profile without any essential changes.

This Common PKI profile makes use of the redefine mechanism defined in [XML-SCHEMA]. The redefine definitions in chapter 5 are the normative part of the specification. The tables before that are the descriptive part. A difference to the other parts of Common PKI is the fact that only those elements are described in tables that are actually profiled i.e. restricted or re-defined.

The XAdES part is not profiled for the time being. It may become necessary to add more definitions to this part with more experience and when requirements will become clearer.

A few notes on Web Service Security 1.0 [WS_SEC_2004]. There again a XML_DSIG signature profile is defined much related to the special requirements of SOAP. Essentially the distinctions to this profile are

1. Enveloped Signature and Enveloped Signature Transform are discouraged ("SHOULD NOT") by [WS_SEC_SOAP] because otherwise changes in SOAP headers might destroy the signature.
2. SecurityTokenReference is a new field in the ds:KeyInfo element. There a WSS Security Token may be inserted which can transport X.509 certificates as well as kerberos tickets. The specification of WS Security – X.509 Token Profile [WS_SEC_CERT] includes more data types to be contained in wsse:SecurityTokenReference.
3. [WS_SEC_SOAP] recommends Exclusive XML Canonicalization and permits XML Decryption Transform.
4. A special STR Dereference Transform in WS Security – SOAP Message Specification [WS_SEC_SOAP] of OASIS

While all these are important features for the Web Service Security context they should not be mandated in a general XML signature context because then all applications would have to support the entire WS Security syntax. Also there is no reason for a general signature context to forbid the enveloped form.

Finally a few notes on PDF and MS Office

- Current work at ETSI ESI aims at establishing PDF [ISO32000] as a third message format for advanced signatures alongside CMS [CAdES] and XML [XAdES]. Once that standardization process is stable and subject to a sufficient demand for further profiling in that area, a Common PKI message profile for PDF may be specified in addition to Parts 3 and 8.
- Microsoft has an implementation of XML_DSIG signatures in Infopath. As far as we know so far this profile can be used in this environment.

In the following the format of XML digital signatures will be specified by means of XML (Extensible Markup Language) and derived variants. Since it is the intention to profile the W3C XMLDSIG recommendations we make use of the redefinition mechanism as in [XML_SCHEMA]. In order to make the definitions made in this specification as transparent as ever possible we make use of the same table oriented notification (see Introduction of the Common PKI Specification) as in the other parts of the Common PKI specification. Inside the tables we note the desired results, the normative schema redefinitions on XMLDSIG are given in chapter 5 .

2 XML Signature Format

The following tables show the profile to XML_DSIG and XML_ENC in detail. Most of the differences are restrictions in the usage of elements, attributes and algorithms in order to provide a narrow enough profile without loosing the flexibility of XML in general.

There are no restrictions in this profile on the use of enveloped, enveloping or detached forms of XML signatures. All three signature forms MUST be supported.

An area of concern is the usage of the RIPEMD algorithm. The Common PKI board would like to exclude RIPEMD for interoperability reasons. So wherever you find RIPEMD referenced in the current document this is subject to exclusion in later versions. But we would like to invite comments on this special issue by all those who may need to have the algorithm included.

Please note that in the “References” column you will find references to OSCI only for elements with differences between this Common PKI specification and OSCI. It is one of the goals of this document to harmonize Common PKI and OSCI in a way that OSCI can (almost) without changes become a profile of this Common PKI document. Hence only those definitions of OSCI have been discarded that do not fit into a general signature and encryption profile.

2.1 Signature Element

Table 1: Signature Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML DSIG	
1	“SignatureType" <sequence>	No changes			4.1	
2	<element ref="ds:SignedInfo"/>	No changes			4.3	
3	<element ref="ds:SignatureValue"/>	No changes			4.2	
4	<element ref="ds:KeyInfo” minOccurs="0"/>	[minOccurs="0"] Excluded.	++	++	4.4	[1]
5	<element ref="ds:Object" minOccurs="0" maxOccurs="unbounded"/> </sequence>	No changes			4.5	
6	<attribute name="Id" type="ID" use="optional"/>	No changes	+	+	4.1	
[1]	A KeyInfo element MUST be present in any signature conforming with Common PKI.					

2.2 SignatureValue Element

No changes to the SignatureValue Element.

2.3 SignedInfo Element

No changes to the SignedInfo Element itself, only to children.

2.3.1 CanonicalizationMethod Element

Table 2: CanonicalizationMethod Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES		NOTES
			GEN	PROC	XML DSIG	OSCI	
1	CanonicalizationMethodType <sequence>	No change			4.3.1		
2	<any namespace="##any" minOccurs="0" maxOccurs="unbounded"/> </sequence>	No Change					[1]
3	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315/"/>	-	++		++	[2] [4]
4		<xsd:enumeration value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315/#WithComments"/>					[2] [4]
5		<xsd:enumeration value="http://www.w3.org/2001/10/xml-exc-c14n#"/>	++	++		--	[2] [3]
6		<xsd:enumeration value="http://www.w3.org/2001/10/xml-exc-c14n#WithComments"/>					[2] [3]
7		<xsd:enumeration value="http://www.w3.org/2006/12/xml-c14n11"/>	+	++			[2] [4]
8		<xsd:enumeration value="http://www.w3.org/2006/12/xml-c14n11#WithComments"/>					[2] [4]
[1]	This restriction is suitable, because only [XML_C14N] and [XML_EXCAN] are allowed.						

[2]	Canonicalization is the standard serialization method of XML. For definitions and usage of canonicalization in XML see [XML_C14N], [XML_C14N11] and [XML_EXCAN] or follow the links noted in #3, #7 and #5. All three algorithms MUST be supported by processing applications. Other canonicalization algorithms MUST NOT be used in conformance with Common PKI. This delimits usage to the most common types and specifically rules out any proprietary algorithms. **Note:** Although [XML_C14N] has proved to be a valuable algorithm the fact that it includes ancestor namespace information makes it impractical in contexts where a signed subdocument is to be extracted and used in some other context without breaking the signature. This has lead to the definition of [XML_EXCAN] where ancestor context is excluded from serialization. For compatibility reasons with regard to many XML implementations [XML_C14N] is still to be supported but [XML_EXCAN] should be used wherever applicable.
[3]	**Note:** Exclusive Canonicalization was not existent when OSCI was defined. It has not yet been incorporated.
[4]	[XML_DSIG] REQUIRES implementation of both Canonical XML 1.0 [XML_C14N] and Canonical XML 1.1 [XML_C14N11], but RECOMMENDS that applications that generate signatures choose Canonical XML 1.1 [XML_C14N11] when inclusive canonicalization is desired.

2.3.2 SignatureMethod Element

Table 3: SignatureMethod Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML DSIG	
1	SignatureMethodType <sequence>	No change			4.3.2	
2	<element name="HMACOutputLength" minOccurs="0" type="ds:HMACOutputLengthType"/>	No change	-	-		[1]
3	<any namespace="##other" minOccurs="0" maxOccurs="unbounded"/> </sequence>	No change				
4	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />	++	++		[2]
		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#ripemd160" />	-	-		[2] [3]
		<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />	++	++		[2]

[1]	**Common PKI Profile:** As noted in chapter 2.4.2 the only way to handle keys in this profile is X.509 certificates. This makes HMAC obsolete and we discourage usage of HMAC entirely for the time being. Conforming clients SHOULD NOT make use of HMAC. The reason why we do not exclude the element in this profile is the fact that it is used with good reasons in [XKMS_REQ]. It may happen that in the future XKMS will become important for Common PKI and thus HMAC may return. So leaving it here will perhaps then make things a little easier.
[2]	Delimits the possible algorithms toDSA-SHA1, RSA-SHA1 and RSA-RIPEMD160.

[3]	**Common PKI Profile:** Although RIPEMD160 remains a suitable hash algorithm it will no longer be included as mandatory neither on the generating nor on the processing side in the next version of Common PKI. This is due to the fact that a great number of applications are practically declared non conforming to this profile because they do not implement RIPEMD160. So we discourage the usage of RIPEMD160 already in this version of the profile. Conforming clients SHOULD NOT make use of RIPEMD160. Conforming clients are not expected to support RIPEMD160 except for those that support OSCI 1.2. **Important Note:** For a coherent status in OSCI 1.2 and this profile RIPEMD160 will stay in this specification until it will be excluded from OSCI in the new upcoming version which is announced for beginning of 2005. **Note** that in OSCI a different URI is defined: http://www.osci.de/2002/04/osci#ripemd160 There is no difference in their meaning so Clients SHOULD interpret this URI as being identical to the URI named in this specification.

2.3.3 Reference Element

Table 4: Reference Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML DSIG	
1	ReferenceType <sequence>	No change			4.3.3	
2	<element ref="ds:Transforms" minOccurs="0"/>	No change				
3	<element ref="ds:DigestMethod"/>	No change				
4	<element ref="ds:DigestValue"/> </sequence>	No change				
5	<attribute name="Id" type="ID" use="optional"/>	No change				
6	<attribute name="URI" type="anyURI" use="optional"/>	No change				[1]
7	<attribute name="Type" type="anyURI" use="optional"/>	No change				
[1]	XML_DSIG: The allowed types of the URI are not specified in [XML_DSIG]. HTTP is RECOMMENDED. **Common PKI Profile:** URIs of types HTTP, HTTPS and LDAP are RECOMMENDED. For LDAP see also Common PKI Part 4, Chapter 7.					

2.3.3.1 DigestMethod Element

Table 5: DigestMethod Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML DSIG	
1	DigestMethodType <sequence>	No changes			4.3.3.5	
2	<any namespace="##other" processContents="lax" minOccurs="0" maxOccurs="unbounded"/> </sequence>	Excluded				
3	<attribute name="Algorithm" type="anyURI" use="required"/>	<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#sha1" />	++	++		[1]
4		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#ripemd160" />	-	-		[2]
[1]	Delimits the possible algorithms to SHA1 and RIPEMD160.					
[2]	**Common PKI Profile:** See Table 3 Annotation [3] on exclusion of RIPEMD160 Note that in OSCI a different URI is defined: http://www.osci.de/2002/04/osci#ripemd160 There is no difference in their meaning, so Clients SHOULD interpret this URI as being identical to the URI named in this specification.					

2.4 KeyInfo Element

Note that the restriction to allow in this element only X.509 type key data is a restriction not only for this element but also for the entire profile. X.509 certificates and related protocols to be used are described in Common PKI Parts 1-5 and 7 and possibly in the optional SigG profile.

Table 6: KeyInfo Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML DSIG	
1	KeyInfoType <choice maxOccurs="unbounded">	No change				
2	<element ref="ds:KeyName"/>	Excluded	--	--		
3	<element ref="ds:KeyValue"/>	Excluded	--	--		
4	<element ref="ds:RetrievalMethod"/>	No change				[1]
5	<element ref="ds:X509Data"/>	No change				[2]
6	<element ref="ds:PGPData"/>	Excluded	--	--	4.4	
7	<element ref="ds:SPKIData"/>	Excluded	--	--		
8	<element ref="ds:MgmtData"/>	Excluded	--	--		
9		<xsd:element ref="xenc:EncryptedKey" />	++	++		[3]
10		<xsd:element ref="xenc:AgreementMethod" />	+-	+-		[4]
11	<any processContents="lax" namespace="##other"/> </choice>	Excluded				
[1]	This leaves the usage of RetrievalMethod open, which will in turn be delimited to X509Data in T7.#4					
[2]	**Common PKI Profile:** The only way of storing KeyInfo data is X509Data for coherence with the rest of the Common PKI specification.					
[3]	OSCI conformance; to be clarified by OSCI					
[4]	XML_ENC: To support Diffie-Hellman key agreement for encrypting data, see also P6.T6 and [XML_ENC] chapter 5.5.					

2.4.1 RetrievalMethod Element

Table 7: RetrievalMethod Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XMLDSIG	
1	RetrievalMethodType <sequence>	No change				
2	<element ref="ds:Transforms" minOccurs="0"/> </sequence>	No change				
3	<attribute name="URI" type="anyURI"/>	use=”required”	++	++		[1]
4	<attribute name="Type" type="anyURI" use="optional"/>	<xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#X509Data" />	++	++		[1]
[1]	**Common PKI Profile:** Any usage of the (optional) RetrievalMethod MUST use X509Data. All other types MUST NOT be used.					

2.4.2 X509Data Element

Table 8: X509Data Type

#	XML_DSIG DEFINITION	RESTRICTION	SUPPORT		REFERENCES		NOTES
			GEN	PROC	XMLDSIG	OSCI	
1	X509DataType <sequence maxOccurs="unbounded"> <choice>	No change	++	++	4.4.4	++	[1]
2	<element name="X509IssuerSerial" type="ds:X509IssuerSerialType"/>	No change	+	+		--	[2]
3	<element name="X509SKI" type="base64Binary"/>	Excluded	--	--		--	
4	<element name="X509SubjectName" type="string"/>	Excluded	--	--		--	
5	<element name="X509Certificate" type="base64Binary"/>	No change	++	++		++	[3]
6	<element name="X509CRL" type="base64Binary"/>	No change	+	+		--	[4]
8	<any namespace="##other" processContents="lax"/> </choice> </sequence>	Excluded	--	--		--	

[1]	Note that OSCI delimits the number of possible entries to 1.
[2]	Note that OSCI does not allow this element.
[3]	This is the place to store the certificate chain in the same way as described in Common PKI Part 3, T2.#4.
[4]	For coherence with the rest of the Common PKI specification not only CRLs need to be stored but also OCSP responses. Rather than introducing a new type we RECOMMEND usage of XAdES (see following chapter) in case an OCSP response needs to be stored.

2.5 Object Element

For compatibility reasons the above definitions are strictly delimited to profiling [XML_DSIG]. Still there are a number of data elements present in other message formats (CMS as in Common PKI part 3) like e.g. signed and unsigned attributes which are not part of [XML_DSIG]. In order to provide this information also in the XML signature world [XAdES] has been defined as an enriching profile to [XML_DSIG]. Rather than to start new definition work in this area Common PKI references [XAdES]. Common PKI conforming applications SHOULD make use of [XAdES] as an optional extension of [XML_DSIG].

[XAdES] introduces additional structures within the Object element in much the same way as they are handled in CMS [RFC3852]. It also supports additional variants for long-term archival of signatures etc. Since all these elements are handled within the present Object element in a coherent and well defined way they do not interfere with any of the above definitions.

Note: Usually an optional element on the signature generation side has to be mandatory on the processing side since there is no way of knowing what kind of a signature will have to be processed. The intention here is a little weaker than this: If for an application additional data are important we want to impose the usage of [XAdES] for this purpose rather than usage of proprietary or other definitions. But an application not making use of [XAdES] at all can still claim conformance with this profile. The result will be two different types of Common PKI signatures: with and without [XAdES]. We think that this is a valid approach at the time being but we would like to invite for comments on this issue.

As a processing rule Common PKI conforming clients that support XAdES MUST be able to process signatures without any of the XAdES elements present. Also Common PKI conforming clients SHOULD include the signing certificate data into the KeyInfo element. This enables non-XAdES clients to process “raw” XML signatures without being able to process the special XAdES elements. But we would not usually encourage clients to do so because it can be assumed that the additional XAdES signature attributes are of importance and there is no way of correct interpretation without understanding the format.

3 XML Encryption format

3.1 EncryptedKey Element

Table 9: EncryptedKeyType

#	XML_ENC DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML ENC	
1	EncryptedKeyType	No change			3.5.1	
2	<extension base='xenc:EncryptedType'> <sequence>	No change				
3	<complexType name='EncryptedType' abstract='true'> <sequence>	No change			3.1	
4	<element name='EncryptionMethod' type='xenc:EncryptionMethodType' minOccurs='0'/>	minOccurs="1"	++	++		[1]
5	<element ref='ds:KeyInfo' minOccurs='0'/>	minOccurs="1"	++	++		[1]
6	<element ref='xenc:CipherData'/>	minOccurs="1"	++	++		[1]
7	<element ref='xenc:EncryptionProperties' minOccurs='0'/> </sequence>	Excluded	--	--		[1]
8	<attribute name='Id' type='ID' use='optional'/>	No change	+-	++		[1]
9	<attribute name='Type' type='anyURI' use='optional'/>	Excluded	--	--		[1]
10	<attribute name='MimeType' type='string' use='optional'/>	Excluded	--	--		[1]

11	<attribute name='Encoding' type='string' use='optional'/>	Excluded	--	--		[1]
12	<element ref='xenc:ReferenceList' minOccurs='0'/>	Excluded	--	--	3.5.1	[1]
13	<element name='CarriedKeyName' type='string' minOccurs='0'/> </sequence>	Excluded	--	--		[1]
14	<attribute name='Recipient' type='string' use='optional'/> </extension>	Excluded	--	--		[1]
[1]	**Common PKI Profile:** In order to create strict interoperability rules encrypted keys plus their reference data MUST be stored in #4 - #6. All other ways MUST NOT be used.					

3.2 EncryptedDataType

Table 10: EncryptedData Type

#	XML_ENC DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML ENC	
1	EncryptedDataType	No change			3.4	
2	<extension base='xenc:EncryptedType'>	No change				
3	<complexType name='EncryptedType' abstract='true'> <sequence>	No change				[1]
4	<element name='EncryptionMethod' type='xenc:EncryptionMethodType' minOccurs='0'/>	No change				[1]
5	<element ref='ds:KeyInfo' minOccurs='0'/>	No change				[1]
6	<element ref='xenc:CipherData'/>	minOccurs="1"	++	++		[1]
7	<element ref='xenc:EncryptionProperties' minOccurs='0'/> </sequence>	Excluded				[1]
8	<attribute name='Id' type='ID' use='optional'/>	No change				[1]
9	<attribute name='Type' type='anyURI' use='optional'/>	Excluded				[1]
10	<attribute name='MimeType' type='string' use='optional'/>	No change				
11	<attribute name='Encoding' type='string' use='optional'/>	Excluded				[1]

[1]	**Common PKI Profile:** In order to create strict interoperability rules encrypted keys plus their reference data MUST be stored in #3 - #5. All other ways MUST NOT be used.

3.3 EncryptionMethodType

Table 11: EncryptionMethod Type

#	XML_ ENC DEFINITION	RESTRICTION	SUPPORT		REFERENCES	NOTES
			GEN	PROC	XML ENC	
1	EncryptionMethodType <sequence>	No change			3.2	
2	<element name='KeySize' minOccurs='0' type='xenc:KeySizeType'/>	No change				
3	<any namespace='##other' minOccurs='0' maxOccurs='unbounded'/> </sequence>	Excluded				[1]
4	<attribute name='Algorithm' type='anyURI' use='required'/>	No change				
5		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />				[1]
6		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />				[1]
7		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#aes192-cbc" />				[1]
8		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />				[1]
9		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />				[1]
10		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" />				[1]

11		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#kw-tripledes" />				[1]
12		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#kw-aes128" />				[1]
13		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#kw-aes192" />				[1]
14		<xsd:enumeration value="http://www.w3.org/2001/04/xmlenc#kw-aes256" />				[1]
[1]	**Common PKI Profile:** Encryption methods are delimited to the enumerated list provided in #5 - #9. Other methods MUST NOT be used.					

4 Algorithm Support

4.1 Cryptographic Algorithms

Cryptographic algorithms required and/or recommended by this part of the Common PKI specification are listed in Part 6 “Cryptographic Algorithms” of the Common PKI Specification.

Most of the algorithms required for XML are referenced in [XML_DSIG] and [XML_ENC].

4.2 Canonicalization

Table 12: Canonicalization Algorithms

ALGORITHMS			REFERENCES	COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS		GEN	PROC	VALUES	
1.1	Canonical XML	Canonicalization algorithm	[XML_DSIG] [XML_ C14N]	-	++	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	[2]
1.2	Canonical XML with Comments	Canonicalization algorithm	[XML_DSIG] [XML_ C14N]	-	+	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments	[2]
1.3	Exclusive XML Canonicalization	Canonicalization algorithm	[XML_ENC] [XML_EXCAN]	++	++	http://www.w3.org/2001/10/xml-exc -c14n#	[1]
1.4	Exclusive XML Canonicalization with Comments	Canonicalization algorithm	[XML_ENC] [XML_EXCAN]	+-	+	http://www.w3.org/2001/10/xml-exc -c14n#WithComments	[1]
1.5	Canonical XML Version 1.1	Canonicalization algorithm	[XML_DSIG] [XML_C14N11]	+	++	http://www.w3.org/2006/12/xml-c14n11	[2]
1.6	Canonical XML Version 1.1 with Comments	Canonicalization algorithm	[XML_DSIG] [XML_C14N11]	+-	+	http://www.w3.org/2006/12/xml-c14n11#WithComments	[2]
[1]	Not specified in [XML_DSIG]						
[2]	[XML_DSIG] REQUIRES implementation of both Canonical XML 1.0 [XML_C14N] and Canonical XML 1.1 [XML_C14N11], but RECOMMENDS that applications that generate signatures choose Canonical XML 1.1 [XML_C14N11] when inclusive canonicalization is desired.						

4.3 Transforms

Transforms are processing steps that convert the input after dereferencing the URI into another representation that is to be signed/verified. As Transforms are a very powerful tools to transform content, it is important to operate only on the transformed content after a signature validation, because only the transformed content is secured by the signature. See [XML_DSIG, Chapter 8.1].

Table 13: Transform Algorithms

ALGORITHMS			REFERENCES	COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS		GEN	PROC	VALUES	
1.1	Canonical XML	Canonicalization algorithm	[XML_DSIG] [XML_C14N]	-	++	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	[2]
1.2	Base64	Base 64 Decoding	[XML_DSIG] [MIME]	++	++	http://www.w3.org/2000/09/xmldsig#base64	
1.3	XPath	XML Path Language	[XML_DSIG] [XPATH]	+	+	http://www.w3.org/TR/1999/REC-xpath-19991116	
1.4	XPath Filter 2.0	XML Signature XPath Filter 2.0	[XPATH_FILT]	+-	+-	http://www.w3.org/2002/06/xmldsig-filter2	
1.5	Enveloped Signature Transform		[XML_DSIG]	++	++	http://www.w3.org/2000/09/xmldsig#enveloped-signature	
1.6	XSLT	XSL Transform	[XML_DSIG] [XSLT]	+	+	http://www.w3.org/TR/1999/REC-xslt-19991116	[1]
1.7	Exclusive XML Canonicalization	Canonicalization algorithm	[XML_EXCAN]	++	++	http://www.w3.org/2001/10/xml-exc-c14n#	
1.8	Canonical XML Version 1.1	Canonicalization algorithm	[XML_DSIG] [XML_C14N11]	+	++	http://www.w3.org/2006/12/xml-c14n11	[2]

[1]	Note that when XSLT is used it is particularly important to rely only on those portions of an XML document that are actually secured by the signature.
[2]	[XML_DSIG] REQUIRES implementation of both Canonical XML 1.0 [XML_C14N] and Canonical XML 1.1 [XML_C14N11], but RECOMMENDS that applications that generate signatures choose Canonical XML 1.1 [XML_C14N11] when inclusive canonicalization is desired.

4.4 Decoding

Table 14: Decoding Algorithms

ALGORITHMS			REFERENCES	COMMON PKI SUPPORT			NOTES
#	NAME	SEMANTICS		GEN	PROC	VALUES	
1.1	Base64	Decoding algorithm	[XML_DSIG] [MIME]	++	++	http://www.w3.org/2000/09/xmldsig#base64	

5 XML Schema Redefines

5.1 XML_DSIG Redefine

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="http://www.w3.org/2000/09/xmldsig#"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#" elementFormDefault="qualified">
        <xsd:import namespace="http://www.w3.org/2001/04/xmlenc#"
                schemaLocation="oscienc.xsd" />
        <xsd:annotation>
                <xsd:documentation xml:lang="de">
                        Common PKI – Restrictions for XML DSIG
                        Based on OSCI 1.2
                </xsd:documentation>
        </xsd:annotation>
        <!-- ### redefinitions ### -->
        <xsd:redefine schemaLocation="http://www.w3.org/TR/2008/CR-xmldsig-core-20080610/xmldsig-core-schema.xsd">
                <xsd:complexType name="KeyInfoType">
                        <xsd:complexContent>
                                <xsd:restriction base="ds:KeyInfoType">
                                        <xsd:choice>
                                                <xsd:element ref="xenc:EncryptedKey" />
                                                <xsd:element ref="ds:RetrievalMethod" />
                                                <xsd:element ref="ds:X509Data" />
                                        </xsd:choice>
                                        <xsd:attribute name="Id" type="xsd:ID" use="optional" />
                                </xsd:restriction>
                        </xsd:complexContent>
                </xsd:complexType>
                <xsd:complexType name="SignatureType">
                        <xsd:complexContent>
                                <xsd:restriction base="ds:SignatureType">
                                        <xsd:sequence>
                                                <xsd:element ref="ds:SignedInfo" />
                                                <xsd:element ref="ds:SignatureValue" />
```

```
                                        <xsd:element ref="ds:KeyInfo" />
                                        <xsd:element ref="ds:Object"
                                                minOccurs="0" maxOccurs="unbounded"/>
                                </xsd:sequence>
                        </xsd:restriction>
                </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="RetrievalMethodType">
                <xsd:complexContent>
                        <xsd:restriction base="ds:RetrievalMethodType">
                                <xsd:attribute name="URI" type="xsd:anyURI" use="required" />
                                <xsd:attribute name="Type">
                                        <xsd:simpleType>
                                                <xsd:restriction base="xsd:anyURI">
                                                        <xsd:enumeration value="http://www.w3.org/2000/09/xmldsig#X509Data" />
                                                </xsd:restriction>
                                        </xsd:simpleType>
                                </xsd:attribute>
                        </xsd:restriction>
                </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="X509DataType">
                <xsd:complexContent>
                        <xsd:restriction base="ds:X509DataType">
                                <xsd:sequence maxOccurs="1">
                                        <xsd:choice>
                                                <xsd:element name="X509IssuerSerial" type="ds:X509IssuerSerialType"/>
                                                <xsd:element name="X509Certificate" type="xsd:base64Binary" />
                                                <xsd:element name="X509CRL" type="xsd:base64Binary" />
                                        </xsd:choice>
                                </xsd:sequence>
                        </xsd:restriction>
                </xsd:complexContent>
        </xsd:complexType>
        <xsd:complexType name="CanonicalizationMethodType">
                <xsd:complexContent>
                        <xsd:restriction base="ds:CanonicalizationMethodType">
                                <xsd:attribute name="Algorithm" use="required">
                                        <xsd:simpleType>
```

```
                                        <xsd:restriction base="xsd:anyURI">
                                                <xsd:enumeration
                                                        value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments" />

                                                <xsd:enumeration
                                                        value="http://www.w3.org/2001/10/xml-exc-c14n#" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/2001/10/xml-exc-c14n#WithComments" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/2006/12/xml-c14n11" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/2006/12/xml-c14n11#WithComments" />
                                        </xsd:restriction>
                                </xsd:simpleType>
                        </xsd:attribute>
                </xsd:restriction>
        </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="TransformType" mixed="true">
        <xsd:complexContent>
                <xsd:restriction base="ds:TransformType">
                        <xsd:attribute name="Algorithm" use="required">
                                <xsd:simpleType>
                                        <xsd:restriction base="xsd:anyURI">
                                                <xsd:enumeration
                                                        value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/2000/09/xmldsig#base64" />
                                                <xsd:enumeration
                                                        value="http://www.w3.org/TR/1999/REC-xpath-19991116" />
                                                <xsd:enumeration
```

```
                                                value="http://www.w3.org/2002/06/xmldsig-filter2" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2000/09/xmldsig#enveloped-signature" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/TR/1999/REC-xslt-19991116" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2001/10/xml-exc-c14n#" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2006/12/xml-c14n11" />
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:attribute>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="DigestMethodType">
    <xsd:complexContent>
        <xsd:restriction base="ds:DigestMethodType">
            <xsd:attribute name="Algorithm" use="required">
                <xsd:simpleType>
                    <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/xmlenc#sha512" />
                        <xsd:enumeration
                            value="http://www.w3.org/2001/04/xmlenc#ripemd160" />
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SignatureMethodType">
    <xsd:complexContent>
```

```
                        <xsd:restriction base="ds:SignatureMethodType">
                                <xsd:attribute name="Algorithm" use="required">
                                        <xsd:simpleType>
                                                <xsd:restriction base="xsd:anyURI">
                                                        <xsd:enumeration
                                                                value="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
                                                        <xsd:enumeration
                                                                value=" http://www.w3.org/2001/04/xmldsig-more#rsa-sha256" />
                                                        <xsd:enumeration
                                                                value=" http://www.w3.org/2001/04/xmldsig-more#rsa-sha512" />
                                                        <xsd:enumeration
                                                                value=" http://www.w3.org/2001/04/xmldsig-more/rsa-ripemd160" />
                                                        <xsd:enumeration
                                                                value="http://www.w3.org/2000/09/xmldsig#dsa-sha1" />
                                                </xsd:restriction>
                                        </xsd:simpleType>
                                </xsd:attribute>
                        </xsd:restriction>
                </xsd:complexContent>
        </xsd:complexType>
</xsd:redefine>
</xsd:schema>
```

5.2 XML_ENC Redefine

```
<?xml version="1.0" encoding="utf-8"?>
<xsd:schema targetNamespace="http://www.w3.org/2001/04/xmlenc#"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
                xmlns:ds="http://www.w3.org/2000/09/xmldsig#" elementFormDefault="qualified">
        <xsd:annotation>
                <xsd:documentation xml:lang="de">
                        Common PKI – Restrictions for XML Encryption
                        Based on OSCI 1.2
                </xsd:documentation>
        </xsd:annotation>
        <!-- ### redefinitions ### -->
        <xsd:redefine schemaLocation="http://www.w3.org/TR/xmlenc-core/xenc-schema.xsd">
                <xsd:complexType name="EncryptionMethodType">
                        <xsd:complexContent>
                                <xsd:restriction base="xenc:EncryptionMethodType">
                                        <xsd:sequence>
                                                <xsd:element name="KeySize" minOccurs="0" type="xenc:KeySizeType" />
                                                <xsd:element name='OAEPparams' minOccurs='0' type='base64Binary'/>
                                        </xsd:sequence>
                                        <xsd:attribute name="Algorithm" use="required">
                                                <xsd:simpleType>
                                                        <xsd:restriction base="xsd:anyURI">
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#tripledes-cbc" />
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#aes128-cbc" />
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#aes192-cbc" />
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#aes256-cbc" />
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#rsa-1_5" />
                                                                <xsd:enumeration
                                                                        value="http://www.w3.org/2001/04/xmlenc#rsa-oaep-mgf1p" />
```

```
                                        <xsd:enumeration
                                                value="http://www.w3.org/2001/04/xmlenc#kw-tripledes" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2001/04/xmlenc#kw-aes128" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2001/04/xmlenc#kw-aes192" />
                                        <xsd:enumeration
                                                value="http://www.w3.org/2001/04/xmlenc#kw-aes256" />
                                </xsd:restriction>
                        </xsd:simpleType>
                </xsd:attribute>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EncryptedDataType">
    <xsd:complexContent>
        <xsd:restriction base="xenc:EncryptedDataType">
            <xsd:sequence>
                <xsd:element name="EncryptionMethod"
                        type="xenc:EncryptionMethodType" minOccurs="0" />
                <xsd:element ref="ds:KeyInfo" minOccurs="0" />
                <xsd:element ref="xenc:CipherData" minOccurs="1" />
            </xsd:sequence>
            <xsd:attribute name="MimeType" type="xsd:string" use="optional" />
            <xsd:attribute name='Id' type='ID' use='optional'/>
        </xsd:restriction>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="EncryptedKeyType">
    <xsd:complexContent>
        <xsd:restriction base="xenc:EncryptedKeyType">
            <xsd:sequence>
                <xsd:element name="EncryptionMethod"
                        type="xenc:EncryptionMethodType" minOccurs="1" />
                <xsd:element ref="ds:KeyInfo" minOccurs="1" />
                <xsd:element ref="xenc:CipherData" minOccurs="1" />
```

```
                                        </xsd:sequence>
                                        <xsd:attribute name='Id' type='ID' use='optional'/>
                                    </xsd:restriction>
                                </xsd:complexContent>
                            </xsd:complexType>
                        </xsd:redefine>
                    </xsd:schema>
```

References

[CAdES]	ETSI TS 101 733 v1.7.4: Electronic Signatures and Infrastructures (ESI); CMS Advanced Electronic Signatures (CAdES), July 2008
[ISO32000]	ISO 32000-1:2008: Document management -- Portable document format -- Part 1: PDF 1.7
[MIME]	N. Freed & N. Borenstein: „Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies.“, RFC2045, November 1996
[OSCI]	OSCI Leitstelle: OSCI Transport, Version 1.2, Bremen, 6. Juni 2002
[RFC3852]	R. Housley: Cryptographic Message Syntax (CMS), RFC 3852, July 2004
[WS_SEC_2004]	OASIS Open: Web Services Security 1.0, OASIS Standard 200401, March 2004, http://www.oasis-open.org/specs/#wssv1.0
[WS_SEC_SOAP]	OASIS Open: Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), OASIS Standard 200401, March 2004
[WS_SEC_CERT]	OASIS Open: Web Services Security X.509 Certificate Token Profile, OASIS Standard 200401, March 2004
[X.509]	ITU-T X.509: Information technology – Open Systems Interconnection – The Directory: Public-key and attribute certificate frameworks, 2005
[XAdES]	ETSI TS 101 903 V1.3.3 (2007-02): XML Advanced Electronic Signatures (XAdES), Technical Specification
[XKMS_REQ]	W3C: XML Key Management (XKMS 2.0) Requirements, 05 May 2003 http://www.w3.org/TR/2003/NOTE-xkms2-req-20030505
[XML_C14N]	W3C: Canonical XML 1.0, 15 March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315
[XML_C14N11]	W3C: Canonical XML 1.1. 2 May 2008, http://www.w3.org/TR/2008/REC-xml-c14n11-20080502/
[XML_DSIG]	W3C: XML-Signature Syntax and Processing (Second Edition), 10 June 2008, http://www.w3.org/TR/2008/REC-xmldsig-core-20080610/
[XML_ENC]	W3C: „XML Encryption Syntax and Processing“, 10 December 2002, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/
[XML_EXCAN]	W3C: Exclusive XML Canonicalization 1.0, 18 July 2002, http://www.w3c.org/TR/2002/REC-xml-exc-c14n-20020718/
[XML_SCHEMA]	W3C: XML Schema Part 1: Structures Second Edition, 28 October 2004. http://www.w3.org/TR/2004/REC-xmlschema-1-20041028/
[XML_SCHEMA2]	W3C: XML Schema Part 2: Datatypes, 02 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/
[XPATH]	W3C: XML Path Language, Version 1.0, October 1999 http://www.w3.org/TR/1999/REC-xpath-19991116
[XPATH_FILT]	W3C: XML-Signature XPath Filter 2.0, November 2002 http://www.w3.org/TR/2002/REC-xmldsig-filter2-20021108/
[XSLT]	W3C: XSL Transforms, Version 1.0, November 1999 http://www.w3.org/TR/1999/REC-xslt-19991116

COMMON PKI SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 9

SIGG-PROFILE

VERSION 2.0 – 20 JANUARY 2009

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

Editors of Common PKI specifications:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0 30.09.2001	First public edition
1.0.1 15.11.2001	A couple of editorial and stylistic changes: - references to SigG-specific issues eliminated from core documents - core documents (Part 1-7) and optional profiles have been separated in different PDF documents.
1.0.2 19.07.2002	Several editorial changes and bug-fixes. The most relevant changes affecting technical aspects are: 1) The DName attribute *nameDistinguisher*, used in legacy systems and older PKCs, MUST be supported by processing applications in *issuer* and *subject* names. (T1.#18) 2) DName attributes in *Procuration* limited to RFC3039 attributes. (T4.#7,[2]) 3) *QcEuLimitValue* may be included in an AC as attribute in place of *MonetaryLimit*. (T10.#10,[2]) 4) Table 12 contains all OIDs defined for ISIS-MTT 5) Profiling information with respect to Part 5 added to adopt validity model to SigG. (Section 3)
1.0.2 11.08.2003	Incorporated all changes from Corrigenda version 1.2
1.1 16.03.2004	Several editorial changes and bug-fixes. The most relevant changes affecting technical aspects are: 1) The policy identifier *id-Co. PKI-cp-sigGconform* has been renamed to *id-Co. PKI-cp-accredited* in order to better reflect the correct semantics. 2) Added a new extension/attribute *AdditionalInformation*. 3) Added a new section about algorithms 4) The permitted size if an *ICCSN* was increased to 20 octets (corresponding to the decimal character representation of a 64 bit value). 5) Definitions of ISIS-MTT private attributes for attribute certificates have been moved from the optional SigG Profile to core Part 1. 6) Key usage has been aligned with ETSI TS 102 280. 7) The qualified certificate statement QcSSCD has been added to the list of QCs. 8) Added profession OID values for the *Admission* attribute.
1.1 13/10/2008	Incorporated all changes from Corrigenda to ISIS-MTT 1.1
2.0 20/Jan/2009	Name change from ISIS-MTT to Common PKI. Included optional private OCSP extensions of the ISIS-MTT v1.1 SigG Options profile. Reflected name change of RegTP to BNetzA. Adapted to new versions of the base standards: - RFC 3739 - RFC 5280 - BNetzA Notification 2008 - ETSI TS 101 456 v1.4.3 - ETSI TS 101 862 v1.3.3 Various corrections and clarifications.

Table of Contents

1 Preface

The *German Signature Act* (SigG) and the *Ordinance on Digital Signatures* (SigV) raise a couple of special requirements on technical components as well as on the certificate policy of certification service providers (CSPs). This profile addresses these technical requirements. These requirements affect certificate contents, CSP service protocols as well as the validity model, implied by the SigG. Besides providing means to fulfil technical requirements, induced by the SigG, this profile specifies new certificate contents, in form of private extensions and attributes, required in common business cases that rely on the legal instruments of the SigG.

This profile is intended for system and application developers who intend to design components that:

- fulfil the requirements induced by the SigG and the SigV on technical means;
- should either be employed in the technical arsenal of CSPs that provide qualified services in the context of SigG and either aspire an *accreditation* in the sense of the SigG, or intend to operate without an accreditation;
- or in end-entity components in SigG-related applications that rely on the qualified services of either accredited or non-accredited CSPs.
- interoperate with PKIs and components designed to comply with the other Common PKI Parts.

The association T7 e.V. of accredited CSP commits itself to this profile, i.e. services and technical components provided by accredited CSP MUST comply with this profile. Non-accredited CSP and third-party software manufactures MAY choose to comply with this profile.

The SigG Profile in this Part of Common PKI is defined in form of a delta-specification with regard to the general Common PKI profile as laid down in Part 1 to 8. That latter general Common PKI profile is hereinafter referenced as "Core" profile. For reference purposes, different requirements in the Core profile of Part 1 to 8 are marked by the prefix **CORE** below.

1.1 Interoperability Aspects

The German Signature Act (Signaturgesetz, [SigG]) defines the general framework for so-called qualified electronic signatures that can be used in legal actions. The SigG has been first passed in 1997 and has been modified in 2001 to comply with *the Directive on Electronic Signatures of the European Community* [ECDir]. The signature law and the ordinance on its technical realization (Signaturverordnung, [SigV01]) put very strong security requirements on the entire public key infrastructure providing means for "qualified electronic signatures", i.e. on signature devices, signature software as well as CA services. The GISA – German IT Security Agency (Bundesamt für Sicherheit in der Informationstechnik, BSI) has issued a "Signature Interoperability Specification" (SigI), promoting uniform signature and certificate formats for SigG-related applications. Companies providing qualified CA services have founded the association "T7" and have issued the standard "Industrial Signature Interoperability Standard" (ISIS), which is an enhancement of a subset of SigI.

The EU-Directive and the German Signature Act classifies electronic signatures as follows:

1. "**electronic signature**" means data in electronic form which are attached to or logically associated with other electronic data and which serve as a method of authentication;

2. "**advanced electronic signature**" means an electronic signature which meets the following requirements:
   (a) it is uniquely linked to the signatory;
   (b) it is capable of identifying the signatory;
   (c) it is created using means that the signatory can maintain under his sole control; and
   (d) it is linked to the data to which it relates in such a manner that any subsequent change of the data is detectable;
3. "**qualified electronic signature**" means an advanced electronic signatures which:

(a) is based on a "qualified certificate" that was valid at the time of signature-creation;

(b) was generated by a "secure-signature-creation device";

Based on MailTrusT (a specification of TeleTrusT for PKI-based secure email), SigI and ISIS, this Common PKI specification aims to provide a common specification for client applications that integrate secure email or other functions and qualified (i.e. SigG-conforming) signature functions. Interoperability among client components as well as CA-services should be provided regardless of the aspired level of security or trust. This characteristic is also referred to as *vertical interoperability.*

More in detail this means:

- components offering the same security level MUST be unconditionally interoperable;
- components offering a different security level must be interoperable as far as possible: qualified components MUST conform to any lower security levels. For example, certified client software (implementing a secure "signature-application component" in the sense of SigG) MUST be able to verify signatures generated by any other Common PKI-compliant components, where the user must be given a note about the actual assumable level of trust. Non-certified components are STRONGLY RECOMMENDED to support data structures (e.g. qualified certificates) and CA services as described in this document. Accordingly, non-certified client software should be able to verify qualified signatures, where of course, the verification can be trusted only to the same extent as the client environment can be trusted.
- Interoperability with common Internet components and data formats based on PKIX standards is enforced.
- Components that are certified or declared as conforming to the German Signature Law and related data formats (the subject of this SigG-Profile) are specified in a manner to meet the requirements of the SigG and of the SigV and to fully comply with the standards of ETSI (European Communications Standards Institute).

In order to achieve the above interoperability and conformity goals, a special "sub"-profile of Common PKI for components and services related to qualified signatures will be defined in this document.

1.2 Requirements on technical components

The SigG and the SigV induces a couple of special requirement on technical components (especially certificates and directory services) used SigG-conforming services or SigG-related applications. Among many others, the following requirements apply:

(1) the validity time (as indicated by the corresponding X.509 data object) of qualified certificates is limited to 5 years (SigV §14 (3))

(2) long term verifiability: it must be possible to verify a signature after expiry and even after revocation of relevant certificates. This period is set at a minimum of 5 years for non-accredited CAs and at a minimum of 30 years for accredited CAs (SigV §4 (1) and (2))

(3) a flat, 3-layer certification hierarchy for accredited CAs: a governmental agency at the top level (responsible for policies, accreditation and subsequent supervision), certification service providers at the middle level (providing CA services for end entities, but not permitted to issue certificates for other CAs) and end entities at the bottom.

(4) SigG §19 (5): The user certificates issued by a conforming CA remain valid even if the accreditation of the issuing CA gets revoked. In this case all certificates of the CA must be revoked.

(5) SigG §8 (1): A back-dated revocation of certificates is forbidden.

(6) SigG §5 (1) distinguishes between *confirming the status* of certificates from keeping them *accessible for downloading*. While a conforming CA is obliged to provide status information about all certificates, its directory service may only publish a qualified certificate with the approval of its owner.

2 Certificate and CRL Formats

The special requirements on certificate and CRL contents are collected in the following tables. Profiling information on specific data components are linked via references to corresponding definitions in Part 1. Note that certificate and CRL formats conforming this SigG-Profile are fully compliant with the more general Common PKI Core profile, laid down in Part 1.

2.1 Public Key Certificate Format

Table 1 : Special requirements on SigG-conforming qualified PKCs

#	DATA FIELD	SEMANTICS AND SIGG PROFILING INFORMATION (CONSTRAINT OR ENHANCEMENT WITH RESPECT TO CORE)	CRITI-CAL	SUPPORT		REFE-RENCE	NO TES
				GEN	PROC		
0	Validity	According to the ordinance on signatures [SigV01], §14, the interval defined by the validity time data field of qualified certificates MUST NOT exceed 5 years.		++	++	P1.T2.#6	
	STANDARD EXTENSIONS						
1	KeyUsage	The following restriction applies in end-entity qualified signature certificates: the *contentCommitment* bit and only this bit MUST be set if these certificates are used to validate commitment to signed content, such as electronic signatures on agreements and/or transactions. These certificates MUST NOT be used for other purposes, like authentication or encryption.	++ (RFC 3739 +)	++ (RFC 3739 ++)	++ (RFC n.a.)	P1.T12	
2	CertificatePolicies	Legacy systems use the *CertificatePolicies* extension to mark qualified certificates and to recognize this fact in components.	- (RFC 5280 +-)	+-	++	P1.T14	[1]

3	`id-commonpki-cp-accredited`	The *id-commonpki-cp-accredited* OID indicates that the certificate is a qualified certificate according to [EUDIR], which additionally conforms the special requirements of the SigG and has been issued by an accredited CA. This latter means that the security of all relevant components (CA, DIR, smartcards etc.) has been proven by an independent accredited laboratory and provides an appropriately high level of trust according to ITSEC. The voluntary accreditation process for CAs is described in §15 and §16 of the novel signature act [SigG] from 2001. Since many of the currently used QCs do not include a *QCStatement*, SigG-conforming components MUST be able to evaluate both the *id-commonpki-cp-accredited* policy OID and *QCStatements*. New qualified certificates MUST be issued with a proper *QCStatement* (see #6) and MAY include the *id-commonpki-cp-accredited* policy OID to indicate *voluntary accreditation* of the issuing CA. Non-accredited CAs issuing SigG-conforming certificates MUST NOT use this OID, but SHOULD mark the certificate by including a proper policy OID in *QCStatements*. ATTENTION! Currently used qualified certificates have been issued including merely the *id-commonpki-cp-accredited* policy OID (i.e. no *QCStatement* present). As *voluntary accreditation* of the CA implies that all issued certificates are qualified ones, components MUST be able to recognize this fact in the absence of a *QCStatement*.		+-	++	P1.T14	
4	`SubjectDirectoryAttributes`	Qualified PKCs MAY include legal identification data of the subject in the *subjectDirectoryAttributes* extension. The same kind of information MAY be included in attribute certificates as separate attribute (i.e. in the 'attributes' field instead of an extension) but using the same *SubjectDirectoryAttributes* syntax. The attributes that can be inserted by compliant CAs MUST be selected from the following list: Standard attributes: *commonName, surname, givenName, title, postalAddress* (with the address of permanent residence) RFC3739 attributes: *dateOfBirth, placeOfBirth, gender, countryOfCitizenship,* *countryOfResidence,* Common PKI attribute: *nameAtBirth* SigG-conforming components MUST be prepared to process all these DName attribute types. Clients SHOULD be able to process all these attribute types that may occur in the subject field. According to the German law, the following items are required for a legally valid identification record: *surname, givenName, title, dateOfBirth, placeOfBirth, nameAtBirth, countryOfCitizenship, postalAddress*. No attributes have yet been introduced for further data items of a German ID card, like ID card number, height, colour of eyes, issuing institution, issuing date.	--	+-	(CORE+) P9++	P1.T17	

	RFC3739 (QC) **PRIVATE EXTENSIONS**						
5	QCStatements	*QCStatements* (Qualified Certificate Statements) extension MUST be recognized and evaluated by SigG-conforming components.	- (RFC 3739 +-)	(CORE+-) P9++	(CORE+) P9++	P1.T25	[1]
6	id-etsi-qcs-QcCompliance	In accordance with [ETSI-QC], qualified certificates to be used in the context of the signature act (SigG) MUST include a *QCStatement* (Qualified Certificate Statement) extension with this OID. This applies to end entity as well as to CA certificates. The meaning of this OID is that the certificate policy is compliant with the policy described in [ETSI-POL]. This QC statement was RECOMMENDED to be included in SigG-conforming certificates issued until June 30, 2005 and it MUST be present in certificates issued later.		(CORE+) P9++	(CORE+) P9++	P1.T25	
6a	id-etsi-qcs-QcSSCD	In accordance with [ETSI-QC], qualified certificates to be used in the context of the signature act (SigG) MAY include a *QCStatement* (Qualified Certificate Statement) extension with this OID. This applies to end entity as well as to CA certificates. The meaning of this OID is to indicate that the CA warrants that the private key associated with the public key in the certificate is stored in an SSCD according to Annex III of [ECDIR].		+-	(CORE+) P9++	P1.T25	
7	id-etsi-qcs-QcLimitValue	The *QcLimitValue* statement SHOULD be used in new certificates in place of the extension/attribute *MonetaryLimit*. Nevertheless, *MonetaryLimit* was allowed until December 31, 2003. After this date, *MonetaryLimit* MUST NOT be used any longer. For the sake of backward compatibility with certificates already in use, components MUST support *MonetaryLimit* (as well as *QcEuLimitValue*). If both *QcEuLimitValue* and *MonetaryLimit* occur in the same certificate, they MUST assert the same value and currency. A certificate SHOULD use only one form.		+-	(CORE+) P9++	P1.T25	
8	id-etsi-qcs-QcRetentionPeriod	The *QcRetentionPeriod* statement indicates CAs or a relevant name registration authority retains external information (i.e. registration documents) about the owner of qualified certificates. This information allows identifying the physical person in case of dispute. SigG-compliant client MUST support this statement.		+-	(CORE+) P9++	P1.T25	
	RFC2560 (OCSP) **PRIVATE EXTENSIONS**						
9	OCSPNocheck	OCSP clients need to know how to check that an authorized OCSP responder's certificate has not been revoked. A CA MAY specify that an OCSP client can trust a responder for the lifetime of the responder's certificate, i.e. the client need no CRL information. The CA does so by including the extension *OCSPNocheck*. SigG-compliant CAs MUST provide status information on the responder's certificate. Hence, this extension MUST NOT be included in qualified certificates.	-	(CORE+-) P9--	+	P1.T26	

	COMMON PKI SIGG-PROFILE PRIVATE EXTENSIONS						[2]
10	LiabilityLimitationFlag	Indicates that an attribute certificate exists, which restricts the application of this public key certificate. Whenever verifying a signature with the help of this certificate, the content of the corresponding attribute certificate should be concerned. This extension MUST be included in a PKC, if a corresponding attribute certificate (having the PKC as base certificate) contains some attribute that restricts the usability of the PKC too. Attribute certificates with restricting content MUST always be included in the signed document.	-	P9+-	P9++	P9.T2	[1]
11	DateOfCertGen	The CA MAY include the *DateOfCertGen* extension, if the certificate is issued right before its validity period, i.e. the signing time T_s lies before *validity.notBefore*. Otherwise the extension SHOULD NOT be included. This information plays a role, if a relying component decides to validate the certificate according to the SigG-specific validity model, described in Section 6. Note that in the context of the SigG Profile, a certificate MUST be considered valid, despite of a later revocation of the issuing CA's certificate, if the issuing CA's certificate was valid at the issued certificate's DateOfCertGen time. Note also that any signature made before the NotBefore time of the corresponding signature certificate is not valid and does not ever become valid, regardless of a DateOfCertGen time included in the signature certificate.	--	P9+-	P9++	P9.T3	
12	Procuration	This attribute may also be used as an extension. As an extension it is single-valued. At the current legal situation, only natural persons and no legal persons (organizations) may be substituted.	--	P9+-	P9++	P1.T29a	
13	Admission	This attribute may also be used as an extension.	--	P9+-	P9++	P1.T29b	[3]
14	MonetaryLimit	The QcEuMonetaryLimit QC statement MUST be used in new certificates in place of the extension/attribute MonetaryLimit since January 1, 2004. For the sake of backward compatibility with certificates already in use, SigG conforming components MUST support MonetaryLimit (as well as QcEuLimitValue).	-	P9--	P9++	P1.T29c	[1]
15	DeclarationOfMajority	This attribute may also be used as an extension.	--	P9+-	P9++	P1.T29d	
16	Restriction	This attribute may also be used as an extension.	-	P9+-	P9++	P1.T29e	[1]
16a	AdditionalInformation	This attribute may also be used as an extension.	-	P9+-	P9++	P1.T29f	[1]
17	ICCSN	Smartcard serial number, to bind a public key to a smart card that stores the corresponding private key.	--	P9+-	P9+-	P9.T9	
	DNAME ATTRIBUTES						
18	nameDistinguisher	Legacy systems, software and certificates use this DName attribute in conjunction with the OID id-commonpki-*at-nameDistinguiser* to distinguish DNames if different entities, if their DNames are otherwise identical. [RFC3739] and Common PKI recommends using the attribute *serialNumber* for this purpose. For backward compatibility, S	--	--	P9++		

[1]	Notes on criticality: For the sake of *vertical interoperability*, these extensions SHOULD NOT be marked critical, in spite of the fact that their contents restrict the usability of the certificate in some way. As these information are extremely relevant in verifying the legal validity of the signature, SigG-conforming components MUST evaluate them.
[2]	All SigG-specific extensions, except ICCSN, MUST be processed by SigG-conforming components.

[3] Profession OIDs SHOULD always be defined under the OID branch of the responsible naming authority.

At the time of this writing, the work group “Recht, Wirtschaft, Steuern” (“Law, Economy, Taxes”) is registered as the first naming authority under the OID *id-commonpki-at-namingAuthorities* and defined the following profession OIDs:

`id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern {id-commonpki-at-namingAuthorities 1}`

Rechtsanwältin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 1}`
Rechtsanwalt	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 2}`
Rechtsbeistand	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 3}`
Steuerberaterin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 4}`
Steuerberater	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 5}`
Steuerbevollmächtigte	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 6}`
Steuerbevollmächtigter	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 7}`
Notarin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 8}`
Notar	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 9}`
Notarvertreterin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 10}`
Notarvertreter	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 11}`
Notariatsverwalterin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 12}`
Notariatsverwalter	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 13}`
Wirtschaftsprüferin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 14}`
Wirtschaftsprüfer	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 15}`
Vereidigte Buchprüferin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 16}`
Vereidigter Buchprüfer	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 17}`
Patentanwältin	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 18}`
Patentanwalt	`{id-commonpki-at-namingAuthorities-RechtWirtschaftSteuern 19}`

See http://www.teletrust.de/fileadmin/files/oid/oid_Antrag.pdf for an application form and http://www.teletrust.de/index.php?id=524for an overview of registered naming authorities.

However a naming authority is NOT REQUIRED to register under the OID *id-commonpki-at-namingAuthorities* in order to define profession OIDs.

At the time of this writing, profession OIDs for the German health care system are defined in the OID sub tree under (1 2 276 0 76 4), see http://www.dimdi.de/dynamic/de/ehealth/oid/verzeichnis.html .

Note that e.g. the profession OIDs *Rechtsanwältin* and *Rechtsanwalt* MUST be considered as equal. The same applies to the other OIDs.

Table 2: LiabilityLimitationFlag

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-LiabilityLimitationFlag** OBJECT IDENTIFIER ::= {0 2 262 1 10 12 0}	OID for extension *LiabilityLimitationFlag*			n.a.	P9.T12	
2	**LiabilityLimitationFlagSyntax** ::= BOOLEAN	The extension SHOULD only be present, if it has value *true*.	P9+-	P9++	n.a.		

Table 3: DateOfCertGen

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-dateOfCertGen** OBJECT IDENTIFIER ::= {id-commonpki-at 1}	OID for extension *DateOfCertGen*			n.a.	P9.T12	
2	**DateOfCertGenSyntax** ::= GeneralizedTime	Date of the generation of the certificate. The format YYYYMMDDhhmmssZ MUST be used.	P9+-	P9++	n.a.		

Table 4: Obsoleted by Part 1 Table 29a

Table 5: Obsoleted by Part 1 Table 29b

Table 6: Obsoleted by Part 1 Table 29c

Table 7: Obsoleted by Part 1 Table 29d

Table 8: Obsoleted by Part 1 Table 29e

Table 9: ICCSN

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC	CO. PKI	
1	**id-commonpki-at-iCSSN** OBJECT IDENTIFIER ::= {id-commonpki-at 6}	OID for extension *ICCSN*			n.a.	P9.T12	
2	**ICCSNSyntax** ::= OCTET STRING (SIZE(8..20))	Serial number of the smart card containing the corresponding private key	+-	+-	n.a.		[1]
[1]	**COMMON PKI PROFILE:** This information may be particularly useful in business applications, where the workflow of issuing a smartcard starts with producing the card, that will be bound to a person only a later stage. In such applications, the ICCSN can serve as the main reference to the client's data during the entire life cycle of the smartcard, e.g. for logging or billing particular transactions carried out by the card holder.						

2.2 Attribute Certificate Format

Table 10: Special requirements on SigG-conforming qualified attribute certificates

#	DATA FIELD	SIGG PROFILING INFORMATION (CONSTRAINT OR ENHANCEMENT WITH RESPECT TO CORE)	CRITICAL O. MULTI-VALUED	SUPPORT		REFE-RENCE	NOTES
				GEN	PROC		
	BASIC AC FIELDS						
1	Subject	SigG-conforming attribute certificates may exist only in conjunction with a key certificate (the base certificate) of the subject. Hence, such certificates MUST use the *baseCertificateID* option when filling the subject field.		++	++	P1.T28.#3	
2	attrCertValidityPeriod	According to the ordinance on signatures [SigV01], §7, the validity of an attribute certificate ends with the validity of the accompanying base certificate. Therefore the maximum validity period is 5 years.		++	++	P1.T28.#9	
	COMMON PKI SIGG-PROFILE PRIVATE EXTENSIONS		**CRITICAL**				
3	DateOfCertGen	The same applies as to the corresponding PKC extension. See T1.#11	--	P9+-	P9++	T1.#11, P9.T3	
	COMMON PKI PRIVATE ATTRIBUTES		**MULTI-VALUED**				
4	Procuration	The same applies as to the corresponding PKC extension. See T1.#12	Y	+-	(CORE +-) P9++	T1.#12, P1.T29a	
5	Admission	The same applies as to the corresponding PKC extension. See T1.#13	N	+-	(CORE +-) P9++	T1.#13, P1.T29b	
6	MonetaryLimit	The same applies as to the corresponding PKC extension. See T1.#14	N	--	(CORE +-) P9++	T1.#14, P1.T29c	[1]
7	DeclarationOfMajority	The same applies as to the corresponding PKC extension. See T1.#15	N	+-	(CORE +-) P9++	T1.#15, P1.T29d	
8	Restriction	The same applies as to the corresponding PKC extension. See T1.#16	Y	+-	(CORE +-) P9++	T1.#11, P1.T29e	[1]
8a	AdditionalInformation	The same applies as to the corresponding PKC extension. See T1.#16a	Y	+-	(CORE +-) P9++	T1.#11, P1.T29f	[1]
9	SubjectDirectoryAttributes	The same applies as to the corresponding PKC extension. See T1.#4	N	+-	(CORE +-) P9++	T1.#4 P1.T17	
10	QcEuLimitValue id-etsi-qcs-QcLimitValue	This attribute MUST be processed by conforming applications.	N	+-	(CORE +-) P9++	P1.T25 .#13	

[1]	**SIGG-PROFILE:** In conjunction with setting the *LiabilityLimitationFlag* in the base certificate, this specification allows issuing attribute certificates that restrict the usability of the base certificate.

2.3 CRL Format

Table 11: Special requirements on CRLs of SigG-conforming qualified certificates

#	DATA FIELD	SIGG PROFILING INFORMATION (CONSTRAINT OR ENHANCEMENT WITH RESPECT TO CORE)	CRITI-CAL	SUPPORT		REFE-RENCE	NO TES
				CA	CLIENT		
	CRL ENTRY EXTENSIONS						
1	`CRLReason`	Only the reason codes *keyCompromise*, *cACompromise*, *affiliationChanged*, *cessationOfOperation* are allowed. As revoked SigG-conforming certificates cannot be released again, the reasons *certificateHold* and *removeFromCRL* never apply.	--	+-	+-	P1.T38	
2	`HoldInstruction`	As SigG-conforming certificates MUST NOT be suspended (status *certificateHold*) in directories, this extension MUST NOT occur in CRL entries corresponding to such certificates.	--	(CORE+-) P9--	+-	P1.T39	

2.4 Common PKI Object Identifiers

The following table lists all ASN.1 object identifiers introduced in the Common PKI Specification Core and in this SigG-Profile. Furthermore, obsolete OIDs, defined in [ISIS] or earlier Common PKI versions, are listed too. These OID values are reserved and MUST NOT be used for any other purpose. The `id-commonpki` branch of the OID tree was previously known under the name `id-isismtt` and before that under the name `id-sigi`, the name but not the meaning has been changed in this version.

Table 12: Common PKI Object Identifiers

#	ASN.1 DEFINITION		SEMANTICS	SUPPORT		REFERENCES		NOTES
				GEN	PROC	RFC	CO. PKI	
1	**id-commonpki**	OBJECT IDENTIFIER ::= {1 3 36 8 }		++	++	n.a.		
2	**id-commonpki-cp**	OBJECT IDENTIFIER ::= {id-commonpki 1}	Branch for policies			n.a.	#1	
3	**id-commonpki-cp-accredited**	OBJECT IDENTIFIER ::= {id-commonpki-cp 1}		+-	++	n.a.	#2	
4	**id-commonpki-at**	OBJECT IDENTIFIER ::= {id-commonpki 3}	Branch for attributess and extensions			n.a.	#1	
4	**id-commonpki-at-dateOfCertGen**	OBJECT IDENTIFIER ::= {id-commonpki-at 1}		+-	++	n.a.	P9.T3	
5	**id-commonpki-at-procuration**	OBJECT IDENTIFIER ::= {id-commonpki-at 2}		+-	++	n.a.	P1.T29a	
6	**id-commonpki-at-admission**	OBJECT IDENTIFIER ::= {id-commonpki-at 3}		+-	++	n.a.	P1.T29b	
7	**id-commonpki-at-monetaryLimit**	OBJECT IDENTIFIER ::= {id-commonpki-at 4}		+-	++	n.a.	P1.T29c	
8	**id-commonpki-at-declarationOfMajority**	OBJECT IDENTIFIER ::= {id-commonpki-at 5}		+-	++	n.a.	P1.T29d	
9	**id-commonpki-at-iCSSN**	OBJECT IDENTIFIER ::= {id-commonpki-at 6}		+-	++	n.a.	P9.T9	
10	**id-commonpki-at-pKReference**	OBJECT IDENTIFIER ::= {id-commonpki-at 7}	obsolete	--	-	n.a.	obsolete	
11	**id-commonpki-at-restriction**	OBJECT IDENTIFIER ::= {id-commonpki-at 8}		+-	++	n.a.	P1.T29e	
12	**id-commonpki-at-retrieveIfAllowed**	OBJECT IDENTIFIER ::= {id-commonpki-at 9}		CORE-- P9+-	-	n.a.	P9.T15	
13	**id-commonpki-at-requestedCertificate**	OBJECT IDENTIFIER ::= {id-commonpki-at 10}		CORE -- P9+-	-	n.a.	P9.T16	
14	**id-commonpki-at-namingAuthorities**	OBJECT IDENTIFIER ::= {id-commonpki-at 11}		+-	++	n.a.	P1.T29b	
16	**id-commonpki-at-certHash**	OBJECT IDENTIFIER ::= {id-commonpki-at 13}		++	++	n.a.	P4.T15	
17	**id-commonpki-at-nameAtBirth**	OBJECT IDENTIFIER ::= {id-commonpki-at 14}		+-	++	n.a.	P1.T7	
17a	**id-commonpki-at-additionalInformation**	OBJECT IDENTIFIER ::= {id-commonpki-at 15}		+-	++	n.a.	P1.T29f	
18	**id-commonpki-at-liabilityLimitationFlag**	OBJECT IDENTIFIER ::= {0 2 262 1 10 12 0}		+-	++	n.a.	P9.T2	
19	**id-commonpki-at-nameDistinguisher**	OBJECT IDENTIFIER ::= {0 2 262 1 10 7 20}	obsolete, backward compatibility!	--	++	n.a.	T1.#18	

3 LDAP

Common PKI-compliant certification authorities MUST publish end entity and CA certificates. It is RECOMMENDED that certificates are downloadable from an LDAP server. No specific requirements apply for SigG-conforming systems and thus no profiling information is added here with respect to the Core Document Part 4.

4 OCSP

For SigG-conforming applications, the primary means of providing and obtaining revocation status information is declared by this profile to be OCSP. CSPs that are accredited according to the German Signature Law MUST provide an OCSP service, other CSPs MAY choose to provide one.

For the sake of long term validation (Requirement (2) of Section 1.2), SigG-conforming directories MUST retain status information for a so called *retention period* of time after the end of the expiry year. The retention period is as long as 5 years for non-accredited CSPs and 30 years for accredited ones. Certificates MAY include the *RetentionPeriod* extension. Certificates MUST be kept in the directory for this period and OCSP responders MUST be able to deliver status information after the expiry of certificates. For the same reason, this profile RECOMMENDS against deleting revoked certificates from CRLs, which is common practice. The means for downloading certificates SHOULD be LDAP.

If requesting status information from a standard OCSP responder beyond the retention period, standard OCSP products may deliver the response '*good*' (indicating a positive response to the status inquiry and meaning at minimum 'not known to be revoked' according to [RFC2560]). This may falsely lead to successful validation of a certificate. It is therefore crucial that the directory service of a CA is able to send a '*positive statement of availability*' to the clients, indicating that the requested certificate is kept in the queried directory and the revocation information is thus reliable (i.e. help the client to be able to interpret '*good*' as 'certificate is known to the responder and has certainly not been revoked'). Each OCSP response given for SigG-conforming signature certificates MUST contain such a positive statement in form of the *CertHash* extension.

Additionally, the retention period MAY be explicitly sent in the response, so that clients, querying the status of a certificate beyond the retention period, can detect that status information is no longer available. OCSP responders MAY send this information in a *ArchiveCutoff* extension of the response.

Relying components MUST be able to interpret the positive statement and the retention information and MUST involve them in the signature validation process.

Table 13: Special requirements on OCSP protocol elements

#	DATA FIELD	PROFILING INFORMATION (CONSTRAINT OR ENHANCEMENT WITH RESPECT TO COMMON PKI)	CRITI-CAL	SUPPORT		REFE-RENCE	NO TES
				GEN	PROC		
	BASIC OCSPRESPONSE FIELDS						
1	`signature`	[RFC2560]: All definitive response messages (*responseStatus=successful*) MUST be digitally signed. The key used to sign the response MUST belong to one of the following: (a) the CA who issued the certificate(s) in question (b) a Trusted Responder whose public key is trusted by the responder (and installed directly at the client), affected certificates include the *OCSPNocheck* extension (see Table 1.#5) (c) a CA Designated Responder (Authorized Responder) who holds a specially marked certificate issued directly by the CA, indicating in the *ExtendedKeyUsage* extension that the responder may issue OCSP responses for that CA. [RFC2560]: The above list is extended with the following option: (d) a key associated with the CA (i.e. a CA's 'OCSP Signing' key) **COMMON PKI PROFILE:** As described in (d) above, the responder's certificate MAY be issued for the CA by some other trusted authority. This set-up allows relying components to obtain reliable status information even if the key of the issuing CA has been compromised. SigG-conforming accredited CAs MUST obtain responder certificates from the German Federal Network Agency for Electricity, Gas, Telecommunications, Post and Railway (BNetzA), which contains an 'OCSP signing' key. ATTENTION! Currently, the certificates issued by the BNetzA for OCSP responders are marked with the *CRLSign*-bit in the *KeyUsage* extension, whereas the *ExtKeyUsage* extension is not included. Clients MUST temporarily accept this kind of flagging as authorization for OCSP signing.		++	++	P4.T8.#5	
2	`CertStatus 'good'`	[RFC2560]: ATTENTION! As status information delivered by OCSP may be obtained from CRLs, '*good*' does not necessarily mean that the certificate was ever issued or that the response time lies within the certificate's validity interval. Additional information regarding the status, such as positive statement about issuance, validity, may be included in response extensions. SigG-conforming CAs MUST provide positive statement about the issuance of a certificate. This Common PKI Specification provides means for that by defining the private single response extension *CertHash*. See also #4.				P4.T8.#24	

	RFC 2560 EXTENSIONS						
3	`ArchiveCutoff`	extension in ResponseData: a responder MAY choose to retain revocation information beyond the certificate's expiry date. In this case, the responder SHOULD include the certificate's "cutoff" date, which is obtained by subtracting the retention period from the *producedAt* time. According to the SigG, compliant directory services are obliged to retain information for a period of 30 years in accredited directories and respectively for 7 years in non-accredited ones. The *ArchiveCutoff* extension with appropriate content SHOULD be present, independent of whether *CertHash* is present or not.	--	+	++ (RFC+-)	P4.T13	
	COMMON PKI SIGG-PROFILE PRIVATE EXTENSIONS						
4	`CertHash (Positive Statement)`	*SingleResponse* extension: the responder may include this extension in a response to send the hash of the requested certificate to the requestor. This hash serves as evidence that the certificate is known to the responder (i.e. it has been issued) and will be used as means to provide a '*positive statement on issuance*'. According to the SigG (§ 5 (1)), compliant directory services MUST provide positive statement about the issuance of signature certificates. Hence, SigG-compliant responders MUST always include this extension in single responses.	--	(CORE+-) P9++	++	P4.T15	
5	`RetrieveIfAllowed`	(Single)*Request* extension: Clients may include this extension in a (single) *Request* to request the responder to send the certificate in the response message along with the status information. Besides the LDAP service, this extension provides another mechanism for the distribution of certificates, which MAY optionally by provided by certificate repositories.	--	(CORE–) P9+-	+-	T15	
6	`RequestedCertificate`	*SingleOCSPResponse* extension: The certificate requested by the client by inserting the *RetrieveIfAllowed* extension in the request, will be returned in this extension. The SigG allows publishing certificates only then, when the certificate owner gives his explicit permission. Accordingly, there may be '*non-downloadable*' certificates, about which the responder must provide status information, but MUST NOT include in the response. Clients may get therefore the following three kind of answers on a single request including the *RetrieveIfAllowed* extension: (a) the responder supports the extension and is allowed to publish the certificate: *RequestedCertificate* returned including the requested certificate (b) the responder supports the extension but is NOT allowed to publish the certificate: *RequestedCertificate* returned including an empty OCTET STRING (c) the responder does not support the extension: *RequestedCertificate* is not included in the response Clients requesting *RetrieveIfAllowed* MUST be able to handle these cases.	--	(CORE–) P9+-	+-	T16	

Table 14: RetrieveIfAllowed

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	CO. PKI	
1	**id-commonpki-at-retrieveIfAllowed** OBJECT IDENTIFIER ::= {1 3 36 8 3 9}						
2	**RetrieveIfAllowed** ::= BOOLEAN		+-	+-			[1]
[1]	Clients may include this extension in a (single) *Request* to request the responder to send the certificate in the response message along with the status information. Besides the mandatory LDAP service, this extension provides another mechanism for the distribution of certificates, which MAY optionally by provided by certificate repositories.						

Table 15: RequestedCertificate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC 2560	CO. PKI	
1	**id-commonpki-at-requestedCertificate** OBJECT IDENTIFIER ::= {1 3 36 8 3 10}						
2	**RequestedCertificate** ::= CHOICE { Certificate Certificate, publicKeyCertificate [0] EXPLICIT OCTET STRING, attributeCertificate [1] EXPLICIT OCTET STRING }		+-	+-			[1]
[1]	The certificate requested by the client by inserting the *RetrieveIfAllowed* extension in the request, will be returned in this extension. The signature act allows publishing certificates only then, when the certificate owner gives his explicit permission. Accordingly, there may be *'non- downloadable'* certificates, about which the responder must provide status information, but MUST NOT include them in the response. Clients may get therefore the following three kind of answers on a single request including the RetrieveIfAllowed extension: a) the responder supports the extension and is allowed to publish the certificate: *RequestedCertificate* returned including the requested certificate b) the responder supports the extension but is NOT allowed to publish the certificate: *RequestedCertificate* returned including an empty OCTET STRING c) the responder does not support the extension: *RequestedCertificate* is not included in the response Clients requesting *RetrieveIfAllowed* MUST be able to handle these cases. If any of the *OCTET STRING* options is used, it MUST contain the DER encoding of the requested certificate.						

5 TSP

SigG-conforming certification authorities MAY offer time-stamping services. For the sake of interoperability, Common PKI specifies a time stamp protocol (TSP) to acquire and obtain time stamp from a server. This protocol is fully compatible with the one defined in the PKIX standard [RFC3161]. No profiling information with respect to the Common PKI Core Document Part 4 is added here for SigG-conforming applications.

6 Certificate Path Validation

Part 5 of the Common PKI Specification describes a certificate path validation algorithm that complies with [RFC5280] and the validity implied by that PKIX profile. This model allows verifying long term signatures, even after the validity period respectively after the revocation of a signature certificate. This situation is illustrated in Figure 1. If a relying user wants to validate a signature at T_{val}, he/she/it must mathematically verify the signature over the document using the public key in the certificate of the signer and check whether this certificate and all certificates of its path were valid at the time T_{sig} of signing the document.

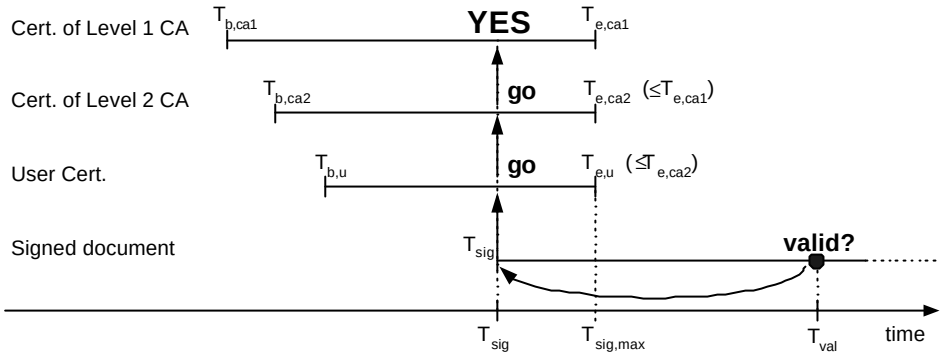


Figure 1: Successful validation of a signature according to the PKIX model

If a CA certificate in the path of the signing certificate has been revoked before the signing time T_{sig}, the signature is considered to be invalid in the PKIX model, as depicted in Figure 2. This also means that the latest time $T_{sig,max}$ a user can provide a valid signature is the of the revocation time $T_{rev,ca2}$ of the CA certificate in the path. After this time the user cannot generate valid signatures with its private key in conjunction with this user certificate, even if the certificate was not explicitly revoked.

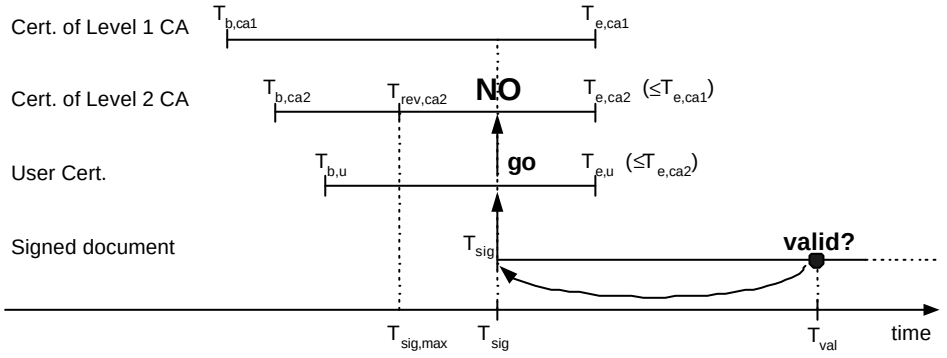


Figure 2: Signatures created after a revocation are invalid in the PKIX model

This PKIX validity model is used throughout Part 1 to 8 of Common PKI. There are however interpretations of the validity and invalidity of signatures and certificates that differ from this notion. Notably, the SigG raises a different requirement in §19 (5), saying:

"§ 19: Supervision Measures …

(5) The validity of qualified certificates issued by a certification service provider shall not be affected by a ban on his operations and cessation of operations or by withdrawal and revocation of an accreditation." (unofficial translation, by courtesy of BnetzA)

Well, there have been disputes for a long time, what the purpose of this clause could be and whether the legislator actually meant the validity of the signature (not the certificate) to remain unaffected after cessation of the CSP, in which case the PKIX model would exactly fit the legal requirements. Compared with the current formulation of §19 (5), the PKIX model is too restrictive: in case of cessation of a CSP it delivers a negative *technical* judgement for a signature that is valid in the *juridical* sense. CSPs MAY take this into account and promote the PKIX model to be used in conjunction with their certificates. The reverse situation, i.e. interpreting a legally valid signature as technically invalid, can never occur.

Note furthermore, that if the CSP commits itself to a policy of revoking all user certificates before its own certificate gets revoked, the situation can never occur and the PKIX model always delivers a technical judgement of validity which is identical with the juridical one. It is being discussed whether such a revocation policy should be seen as an infringement of the law.

In the current vague situation, CSPs wanting to provide technical products that exactly fulfil the validity requirements of the SigG, MAY implement a slightly different variant of the PKIX model, called here the *SigG-model*. According to this model, validation follows exactly the "normal way" induced by the PKIX model and delivers the same results in the normal case. If, however, the relying component detects that the certificate of the CA that issued the user's signing certificate was revoked before the signing time T_{sig}, it shall not to cease with negative result, but try to validate the CA certificate with respect to the issuing time $T_{b,u}$ of the user's certificate. If it succeeds with this, the user's certificate shall be considered valid. This procedure is illustrated in Figure 3. If the time of issuance is different from the beginning of the validity period (e.g. a certificate is issued with validity period in the future), the issuance time SHOULD be indicated in a *DateOfCertGen* extension of the user certificate. Note that signatures made before the NotBefore time of the corresponding signature certificate are not valid according to the Signature Law, regardless of a DateOfCertGen time included in the signature certificate.

Note that the "escape route" can only be taken, if the secret key of the CSP has not be compromised, but revoked for some other reason, which does not affect the reliability of the issued certificates. If the reason of revocation cannot be reliably determined, the component SHOULD consider the signature to be invalid.

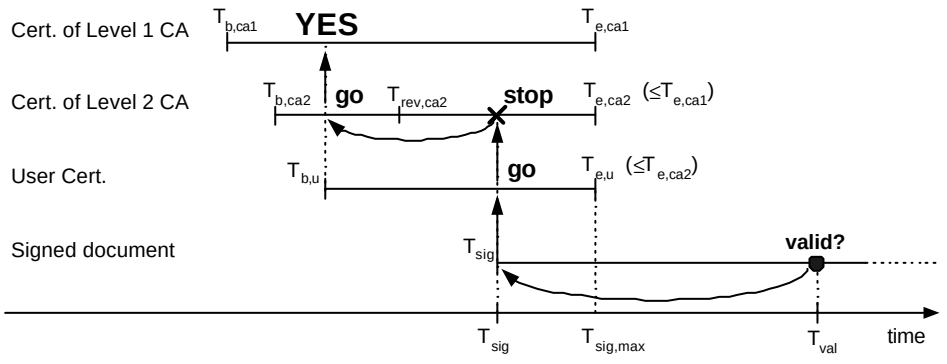


Figure 3: Signatures created after cessation of a CA are valid in the SigG model

In the following, we give a formal description of a path validation algorithm that implements the SigG-model. The algorithm is almost identical with the one specified in Section 2.2 of Part 5. Actually, one single step of the *ValidateCertPath*() function, namely Step #12 of P5.T4, needs to be altered to adopt the algorithm to the SigG-model. The description of this step is given in Table 16, using the same tabular form and notation as in Part 5.

Table 16: ValidateCertPath()

#	PSEUDO-CODE	COMMENTS	REF. TO PART 4	NOTES
1	`if( CheckRevocationStatus( tbvCert,` `tbvCerts,` `refTime,` `pathConstraints,` `trustedCerts,` `trustedCrls )==false )` `{` `if( (tbvCert.certType==SelfIssuedCACert ||` `tbvCert.certType==CACert ||` `tbvCert.certType==CrossCACert) &&` `(tbvCert.revoked==true) &&` `(tbvCert.revocationReason!='keyCompromise' &&` `tbvCert.revocationReason!='cACompromise') )` `{` `Certificate &eeCert = tbvCertPath.GetItem(n);` `Time eeCertSigningTime;` `if( eeCert.ContainsDateOfCertGen() )` `eeCertSigningTime = eeCert.GetDateOfCertGen();` `else` `eeCertSigningTime = eeCert.GetValidityNotBefore();` `if( CheckRevocationStatus( tbvCert,` `tbvCerts,` `eeCertSigningTime,` `pathConstraints,` `trustedCerts,` `trustedCrls )==false )` `return false;` `}` `else` `return false;` `}`	Step #14 of P5.T4 MAY be replaced by the one here, if the certificate path *tbvCertPath* should be validated according to the SigG-model. If *CheckRevocationStatus*() returns *false*, this indicates that either the certificate was revoked before *refTime* or no status information could be obtained. Instead of ceasing path validation immediately, as the basic path validation algorithm of [RFC 5280] does, this algorithm variant checks, whether: - the certificate is a CA certificate or a cross certificate and - it was revoked and - the revocation reason was not *keyCompromise* nor *cACompromise* If these conditions are met, the algorithm takes the "escape route" by calling *CheckRevocationStatus*() again with the time instance parameter changed from *refTime* to the signing time of the EE certificate, which is the last element of *tbvCertPath*. If any of the above conditions is not met, the function returns *false*, as the original algorithm. **COMMON PKI PROFILE:** If during the revocation of a certificate a key compromise cannot be excluded with sufficient probability, the CA SHALL set the reason code to *keyCompromise* or *cACompromise*. Hence the reason code *unspecified* MAY be treated as "unknown, but no key compromise".	P4.T5.#12	

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

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.

This provision is an extension of the algorithm specified in Section 2.3 of Part 5, in particular step #4 of the *CheckStatusUsingCRL*() function in P5.T6. The modification of this step is given in Table 17, using the same tabular form and notation as in Part 5.

Table 17: CheckStatusUsingCRL()

#	PSEUDO-CODE	COMMENTS	NOTES
1	`Name crlIssuerDName;` `if( crlIsIndirect )` `  crlIssuerDName = cdp.crlIssuer.GetDirectoryName();` `else` `  crlIssuerDName = tbvCert.GetIssuerDName();`	The DName of the CRL-issuer is determined. **COMMON PKI PROFILE:** Note that the CDP MUST contain the DName of the issuer of each indirect CRL (P1.T22.#5 & [5]). For indirect CRLs, other CRL-issuer DNames SHOULD also be acceptable, provided there is a matching CRL-signing certificate that can be validated to the same root CA as *tbvCert*.	

Other applications MAY adopt this behaviour when evaluating indirect CRLs.

7 Algorithms

This RIPEMD-160 hash algorithm is published in [BNetzA08] as an algorithm appropriate and allowed for signing according to the German law on digital signatures [SigG01]. It has also been used in certificates of the Federal Network Agency for Electricity, Gas, Telecommunications, Post and Railway (BNetzA). Hence it is urgently RECOMMENDED that components compliant with this profile accept data elements signed using RIPEMD-160 as a hash function.

References

[BNetzA08] Federal Network Agency for Electricity, Gas, Telecommunications, Post and Railway: Notification in Accordance with the Electronic Signatures Act and the Electronic Signatures Ordinance (Overview of Suitable Algorithms), published in German Federal Gazette (Bundesanzeiger) No 19, pp 376 of 5 February 2008 (in German)

[DraftOCSPv2] Online Certificate Status Protocol, version 2, draft-ietf-pkix-ocspv2-02.txt, March 2001

[ECDIR] Directive 1999/93/EC of the European Parliament and of the Council of 13 December 1999 on a Community Framework for Electronic Signatures

[ETSI-CPN] ETSI TS 102 280 v1.1.1 (2004-03) : X.509 V.3 Certificate Profile for Certificates Issued to Natural Persons

[ETSI-POL] ETSI TS 101 456 v1.4.3 (2007-05): Policy Requirements for Certification Authorities Issuing Qualified Certificates, Technical Specification

[ETSI-QC] ETSI TS 101 862 v1.3.3 (2006-01): Qualified Certificate profile

[ETSI-SIG] ETSI ES 201 733 v1.1.3 (2000-05): Electronic Signature Format

[ISIS] Industrial Signature Interoperability Specification ISIS, Version 1.2, December 1999, T7 i.Gr., www.t7-isis.de

[RFC2560] X.509 Internet Public Key Infrastructure Online Certificate Status Protocol -OCSP, June 1999

[RFC3039] Internet X.509 Public Key Infrastructure Qualified Certificates Profile, January 2001

[RFC3161] Internet X.509 Public Key Infrastructure - Time Stamp Protocol (TSP), RFC 3161, August 2001

[RFC3281] An Internet Attribute Certificate Profile for Authorization, April 2002

[RFC3739] Internet X.509 Public Key Infrastructure: Qualified Certificates Profile, March 2004

[RFC5280] Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, May 2008

[SigG01] Law on the Conditions for Electronic Signatures (Gesetz über Rahmenbedingungen für elektornische Signaturen und zur Änderung weiterer Vorschriften), Bundesgesetzblatt Nr. 22, 2001, S.876.

[SigV01] Ordinance on Digital Signatures (Verordnung zur digitalen Signatur – SigV), 2001