COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

INTRODUCTION

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de
The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de
Please send comments and questions to isismtt@teletrust.de

Editors:

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

The following people have contributed to the ISIS-MTT Specification:

Petra Barzin, Fritz Bauspieß, Andreas Berger, Hans-Joachim Bickenbach, Jobst Biester, Jürgen Brauckmann, Holger Ebel, Dirk Fox, Alfred Giessler, Volker Hammer, Tamás Horváth, Karl-Adolf Höwel, Hans-Joachim Knobloch, Ulrike Korte, Rolf Lindemann, Dieter Pfeuffer, Olaf Schlüter, Peter Schmidt, Wolfgang Schneider, Josef Peter Winand and Klaus-Dieter Wirth

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

Table of Contents

1 Objectives 5

1.1 ISIS-MTT Profiles International Standards 5

1.2 The Scope of ISIS-MTT 5

1.3 Some History 6

2 Structure 8

3 Terminology and Notation 9

3.1 Support Requirements 9

3.2 ISIS-MTT Conformance 10

3.3 Notation 10

References 12

1 Objectives

1.1 ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT

This ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT specification intends to promote wide interoperability among client applications and CA services, irrespective of the required security level; a characteristics 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 ISIS-MTT Specification, testing facilities have been specified that can be used to assess the conformity of components with the interoperability specification. This *ISIS-MTT 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 current version of MTT is MTT v2 [MTTv2], which is 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.

Lately, TeleTrusT and T7 have 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 the future.

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 ISIS-MTT contains specifications that provide international compatibility in the technical realization. In particular, the ISIS-MTT 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 ISIS-MTT, implements specific requirements on signatures raised by the German Signature Act and is intended for use only in this specific context.

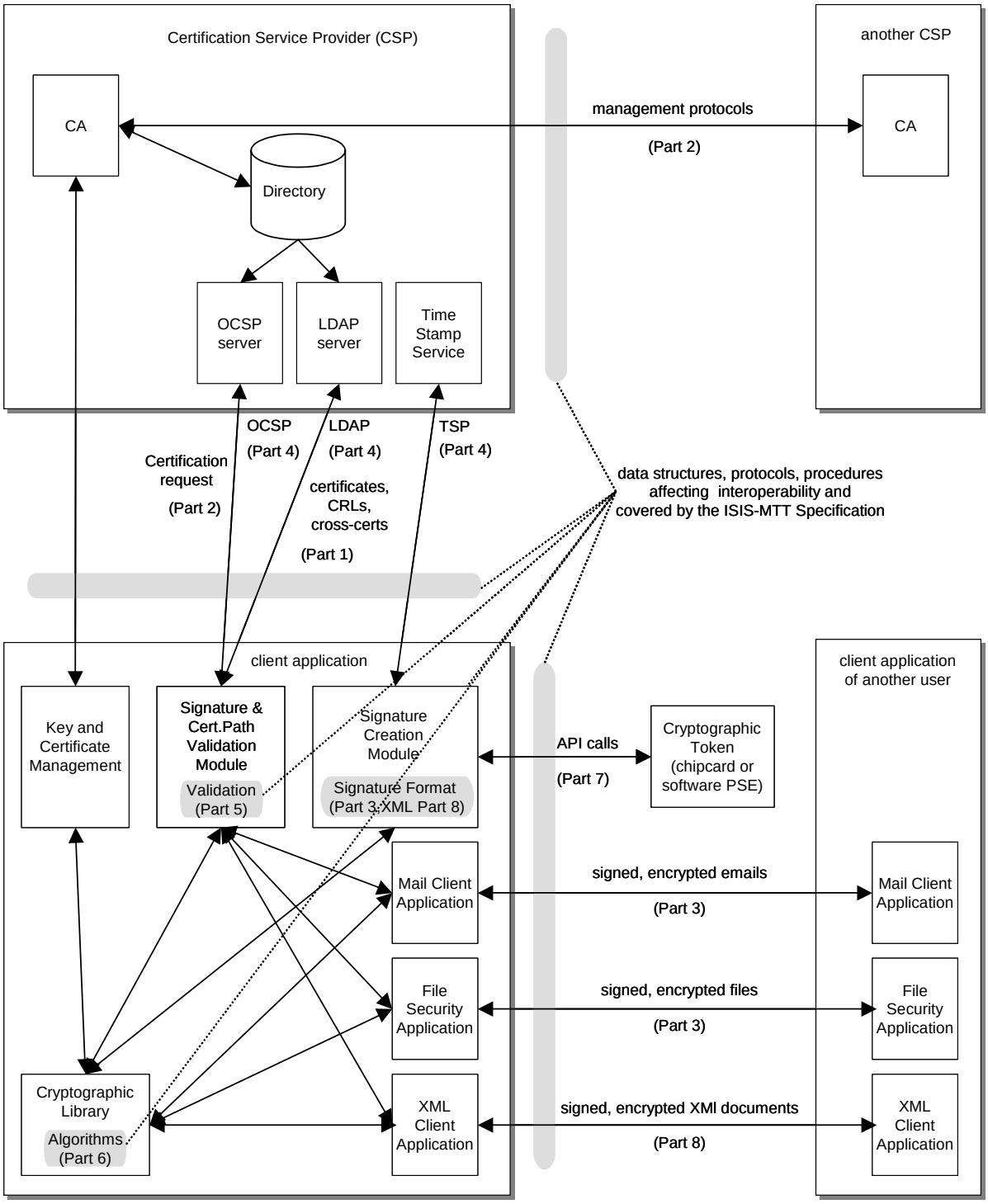


Figure 1: Overview of ISIS-MTT and its relationship to interfaces among PKI components

2 Structure

The ISIS-MTT document is structured into eight *Core Parts* and further *Optional Profiles*. Note that conformance with the ISIS-MTT specification requires to meet the requirements laid down in the Core Parts. The Optional Profiles, on the hand, are enhancements to the core documents and contain requirements for special application areas or user groups. These requirements may, but need not be fulfilled by ISIS-MTT compliant components in general. Local policies may rely on such Optional Profiles in order to assure interoperability within a closed group of users or applications. The Optional Profiles are consistent with the more general ISIS-MTT Specification, i.e. components conforming to an Optional Profile are necessarily and automatically compliant with the ISIS-MTT Core. Currently, the only application area concerned within the ISIS-MTT Specification is that of the German Signature Act (SigG), raising some specific requirements on qualified signatures and services.

In addition to the Core Parts and Optional Profiles, a couple of *Appendices* are planned to be published. These *Appendices* should be regarded only as guidelines that accompany the specification and intend to provide useful additional information to help engineers in implementing compliant components or migrating from legacy systems to this new standard.

The current version of the ISIS-MTT Specification comprises the following *Core Parts*:

- Part 1: Certificate and CRL Profiles,
- Part 2: PKI Management,
- Part 3: Message Formats,
- Part 4: Operational Protocols,
- Part 5: Certificate Path Validation,
- Part 6: Cryptographic Algorithms,
- Part 7: Cryptographic Token Interface, and
- Part 8: XML Signature and Encryption.

The current version of the ISIS-MTT Specification contains the following *Optional Profiles*:

- Optional Profile: SigG-Profile, and
- Optional Profile: Optional Enhancements to the SigG-Profile.

3 Terminology and Notation

3.1 Support Requirements

The ISIS-MTT Specification raises requirements on PKI-components for supporting a variety of objects, such as functions of an API, messages of some communication protocol, specific fields in some data structure. As a basic approach, the ISIS-MTT 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, that 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 ISIS-MTT 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 ISIS-MTT Conformance

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

3.3 Notation

The ISIS-MTT 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, ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT Specification

The SigG-Profile will be referred to as *SigP* (instead of P*x*) in the above short notation.

As readily mentioned in Section 1.2, this ISIS-MTT 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 ‘**ISIS-MTT 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

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

[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

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 1

CERTIFICATE AND CRL PROFILES

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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 obligatory, *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].

Table of Contents

1 Preface 6

2 Public Key Certificate Format 7

2.1 Distinguished Names 11

2.2 GeneralNames 15

2.3 Public Key Certificate Extensions 16

2.3.1 Standard Certificate Extensions 19

2.3.2 PKIX Private Certificate Extensions 32

3 Attribute Certificate Format 36

3.1 Attribute Certificate Attributes 39

3.2 Attribute Certificate Extensions 46

4 CRL Format 47

4.1 CRL Extensions 49

4.2 CRL Entry Extensions 52

5 Cross Certificates 55

6 ISIS-MTT Object Identifiers 56

References 57

1 Preface

This part of the ISIS-MTT 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] and to the PKIX-profile for public key certificates and CRLs [RFC3280]. 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 ISIS-MTT Specification. The full compatibility with the PKIX qualified certificate profile [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 format and [RFC3281] have 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.)

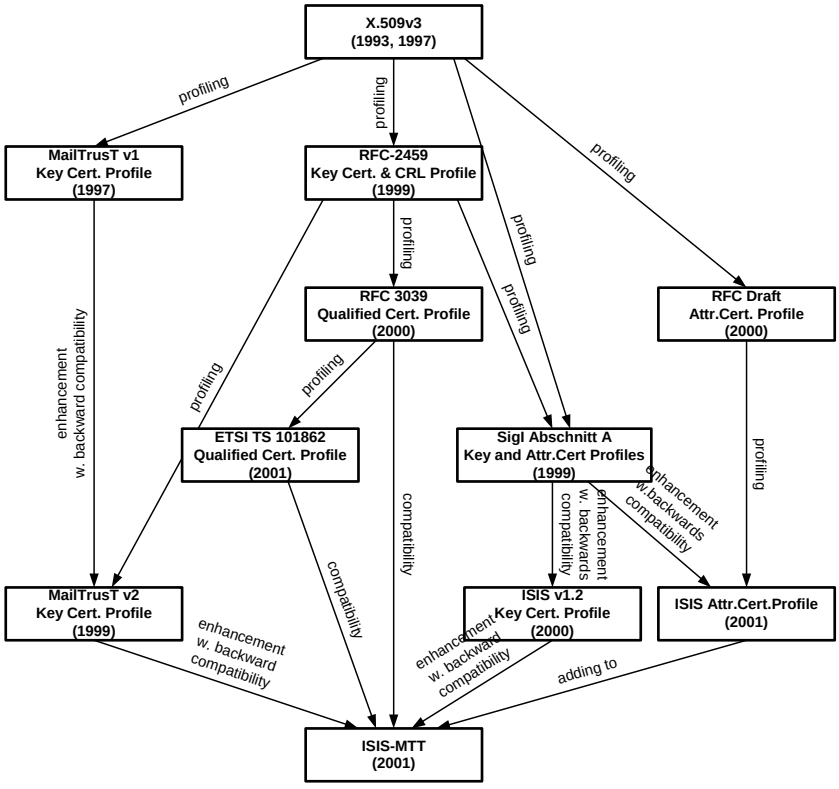


Figure 1: An overview of different standards and profiles on certificate formats

2 Public Key Certificate Format

Table 1: Certificate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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	RFC3280	ISISMTT	
1	**TBSCertificate** ::= SEQUENCE {				4.1.1.1		
2	version [0] EXPLICIT Version DEFAULT v1,	Version number of the 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	
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]	[RFC3280] 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]	[RFC3280]: 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. **ISIS-MTT PROFILE**: the above requirements on length apply.						
[3]	[RFC3280]: The *issuer* name and the *serialNumber* of public key certificates (PKCs) MUST identify a unique certificate. **ISIS-MTT 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 CAs to issue PKCs and ACs at the same time, which is not the case in ISIS-MTT.						
[4]	[RFC3280]: The content must be the same as that of *signatureAlgorithm* in T1.#3.						
[5]	[RFC3280]: 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*. [RFC3039]: 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. **ISIS-MTT 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]	[RFC3280]: 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. [RFC3039]: the *subject* DName MUST contain an appropriate subset of the following attributes: *countryName, commonName, surname, givenName, pseudonym, serialNumber, organizationName, organizationalUnitName, stateOrProvincename, localityName and postalAddress* 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. [RFC3039bis]: The difference with [RFC3039] is that *title* has been added to the list of allowed attributes and *postalAddress* has been removed from this list. [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 of the following attributes: *countryName*, *stateOrProvinceName*, *organizationName* and *commonName*. The *organizationName* and *commonName* SHALL be present. **ISIS-MTT PROFILE:** the subject name of an end entity MUST at least contain the attribute *commonName*. In an ISIS-MTT-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 [RFC3039], 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 ISIS-MTT. Including an *emailAddress* attribute in EE DNames is tolerated by ISIS-MTT for practical compatibility reasons (Netscape).
[7]	[RFC3280]: 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*. **ISIS-MTT 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 *Certificate*S*erialNumber* 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.

Table 3: Validity, Time

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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]	[RFC3280]: 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) **ISIS-MTT 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	RFC3280	ISISMTT	
1	**AlgorithmIdentifier** ::= SEQUENCE {				4.1.1.2		
2	algorithm OBJECT IDENTIFIER,				7.2	P6	[1]
3	parameters ANY DEFINED BY algorithm OPTIONAL }				7.2	P6	
[1]	For permitted algorithm identifiers and parameters refer to Part 6 (Cryptographic Algorithms) of this ISIS-MTT Specification.						

2.1 Distinguished Names

Table 5: Name

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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	RFC3280	ISISMTT	
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] [RFC3280]: the UTF8 encoding method, defined in [RFC2279], is the preferred one among all *CHOICE* options and must be used in all certificates issued after December 31, 2003. Exceptions to the December 31, 2003 UTF8 encoding requirements are as follows: (a) CAs MAY issue "name rollover" certificates to support an orderly migration to UTF8String encoding. Such certificates would include the CA's UTF8String encoded name as issuer and and the old name encoding as subject, or vice-versa. (b) [...] the subject field MUST be populated with a non-empty distinguished name matching the contents of the issuer field in all certificates issued by the subject CA regardless of encoding.
ISIS-MTT PROFILE: Strings MAY be encoded as *PrintableString* even after December 31, 2003, 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”).
This transitional deviation from [RFC3280] shall be reviewed after June 30, 2005 and will be cancelled as soon as application support for UTF8 is considered wide enough.

[2] **ISIS-MTT PROFILE:** Following [MTTv2], ISIS-MTT 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] [RFC3280]: 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*. According to [RFC3280], 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 RFC3280	Support RFC3039 bis	Support ISIS-MTT				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	+	++	+-	++	+-	++	[7]
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 [RFC2247]	++	++	+-	++	+-	++	
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], [8]
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-isismtt-at 14}	DirectoryString	64	n.a.	n.a.	+-	++	+-	++	[6]

[1]	**ISIS-MTT PROFILE:** Following the common practice, the pseudonym MUST be put in the *commonName* attribute and marked with suffix ":PN". To conform with [RFC3039], 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]	**ISIS-MTT PROFILE:** This ISIS-MTT 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]	**ISIS-MTT PROFILE:** *businessCategory* is not listed in any PKIX documents among the obligatory attributes. Hence, this ISIS-MTT specification discourages from its use. For backward compatibility, processing components SHOULD still be able to interpret the attribute.
[4]	**ISIS-MTT PROFILE:** *streetAddress* and *postalCode* are not listed in any PKIX documents among the obligatory attributes. Hence, this ISIS-MTT 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. In conformance with [RFC3039bis], new certificates SHOULD NOT use the *postalAddress* attribute. 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]	**ISIS-MTT PROFILE:** Including an *emailAddress* attribute in DNames is tolerated by ISIS-MTT for practical compatibility reasons (Netscape).
[6]	[RFC3039]: 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). RFC3039 defines a couple of new DName attributes for this purpose (*title*, *dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence*). According to RFC3039, these attributes are to be stored in the *SubjectDirectoryAttributes* extension of the key certificate. RFC3039 explicitly states that new attribute types MAY be included according to local definitions. **ISIS-MTT 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]	[RFC3039bis]: *Title* has been added to the list of allowed attributes in subject Dnames.
[8]	**ISIS-MTT PROFILE:** Including a *gender* attribute in EE subject names of natural persons is permitted.

2.2 GeneralNames

Table 8: GeneralNames

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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 [RFC822]	+-	+			
5	dNSName [2] IMPLICIT IA5String,	Internet domain name as in [RFC1034]	+-	+			
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] and [RFC2255].	+-	+			
10	iPAddress [7] IMPLICIT OCTET STRING,	IP address in IPv4 [RFC791] or in IPv6 [RFC1883] 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 [RFC3280]…. 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*. **ISIS-MTT 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]	[RFC3280] defines type *ORAddress* in appendix A.1 following [X.509]. **ISIS-MTT PROFILE:** As *ORAddress* is considered to be obsolete. Making use of the *ANY* type, the rather elaborate definition in [X.509] 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.

2.3 Public Key Certificate Extensions

Table 9: Extensions

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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 [RFC3280].

Table 10: An overview of key certificate extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT			REFERENCES		NOTES
					GEN CA CERT.	GEN EE CERT.	PROC	RFC	ISISMTT	
	X.509 BASIC EXTENSIONS							**RFC3280**		
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 private key corresponding to the public key contained in the certificate	++ (RFC +)	++ (RFC +-)	++ (RFC +-)	++ (RFC n.a.)	4.2.1.3	T12	
4	PrivateKeyUsagePeriod	{2 5 29 16}	Allows giving a validity period of using the private key different from that of the certificate	--	-	-	- (RFC n.a.)	4.2.1.4	T13	
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.	+-	+-	+-	++	4.2.1.5	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.	--	+-	--	+-	4.2.1.6	T15	
7	SubjectAltNames	{2 5 29 17}	Alternative technical names of the subject: OtherName, e-mail, DNS name, IP address, URI or other	- (RFC +-)	+-	+-	+	4.2.1.7	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	-	+-	+-	+	4.2.1.8	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 3039 n.a.)	+-	+ (RFC 3039 n.a.)	4.2.1.9	T17	
10	BasicConstraints	{2 5 29 19}	Indicates a CA certificate and defines how deep a certificate may exists below that CA.	++	++	+-	++	4.2.1.10	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.	++	+-	--	++	4.2.1.11	T19	

10b	PolicyConstraints	{2 5 29 36}	May be used in CA certificates to constrain path validation.	+-	+-	--	++	4.2.1.12	T20	
11	ExtendedKeyUsage	{2 5 29 37}	Indicates purposes for which the public key can be used, additional to or in place of those in the *KeyUsage* extension.	+-	+-	+-	++ (RFC n.a.)	4.2.1.13	T21	
12	CRLDistributionPoints	{2 5 29 31}	Identifies how CRL information to this certificate can be obtained.	-	+	+/++ dir/ind. CRL	+	4.2.1.14	T22	
	RFC3280 PRIVATE EXTENSIONS							**RFC3280**		
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	
	RFC3039 QC PRIVATE EXTENSIONS							**RFC3039**		
14	BiometricInfo	{id-pe 2}	Stores biometric information for authentication purposes.	--	+-	+-	+	3.2.4	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 3039 +-)	+-	+-	+	3.2.5	T25	[1]
	RFC2560 PRIVATE EXTENSIONS							**RFC2560**		
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:
ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT-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 ISIS-MTT-compliant software.
This flagging and verification policy contributes to achieving interoperability among different security levels, one of the major objectives of this ISIS-MTT Specification.

2.3.1 Standard Certificate Extensions

Table 11: AuthorityKeyIdentifier and SubjectKeyIdentifier

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC3280	ISISMTT	
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] [RFC3280]: *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 identify the certificate rather than the public key.
Both identification methods MAY be used in the same certificate.
ISIS-MTT 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]	[RFC3280]: 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, **ISIS-MTT 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	RFC3280	ISISMTT	
1	`KeyUsage ::= BIT STRING {`	Defines the purpose of 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	`nonRepudiation (1),`	signature verification corresponding to non-repudiation service	+-	++			
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]	[RFC3280]: 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. [RFC3039bis]: The exclusive use of the non-repudiation bit has been removed from [RFC3039]. [ETSI-CPN]: Key usage in end entity certificates for natural persons is restricted to one of the following settings: nonrepudiation bit set (A), nonrepudiation 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. **ISIS-MTT 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 *nonRepudiation* 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 *nonRepudiation* bit set and including the *ExtendedKeyUsage* extension with only the *id-kp-timeStamping* option (see Table 21). • End entity user certificates: 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 non-repudiation (only *nonRepudiation* set), authentication (only *signature* bit set) and encryption (only *dataEncipherment* and *keyEncipherment* set). • End entity qualified signature certificates: the *nonRepudiation* 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. The *nonRepudiation* and *digitalSignature* bits MAY be combined, if these certificates are to be used for other purposes. Compliant CAs MUST issue certificates that are assigned to exactly one of the above purposes. 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 *dataEncypherment* OR *keyEncipherment* to recognize encryption certificates.

Table 13: PrivateKeyUsagePeriod

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**PrivateKeyUsagePeriod** ::= SEQUENCE {	Allows giving a validity period of using the private key different from that of the certificate	-	- (RFC n.a.)	4.2.1.4		
2	notBefore [0] IMPLICIT GeneralizedTime OPTIONAL,				4.1.2.5.2		
3	notAfter [1] IMPLICIT GeneralizedTime OPTIONAL }				4.1.2.5.2		

Table 14: CertificatePolicies

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**CertificatePolicies** ::= SEQUENCE SIZE (1..MAX) OF PolicyInformation	a non-empty list of policy terms	+-	++	4.2.1.5	#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	
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 {`						
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]	Notes on criticality: [RFC3280]: By setting the critical flag on this extension, the CA can enforce its policy on the usage: relying components trying to verify a signature MUST reject a certificate if they cannot recognize a policy OID. The number of policy terms in the list is not limited. **ISIS-MTT PROFILE:** For the sake of *vertical interoperability*, especially for the successful verification of signed documents and certificates outside the ISIS-MTT application group, the extension SHOULD NOT be marked critical. As ISIS-MTT 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-ISIS-MTT-compliant client software may recognize those indicators and ignore this extension, without loosing information on the applying policy.						
[2]	[RFC3280]: *PolicyInformation* SHOULD only contain an OID. Where an OID alone is insufficient, [RFC3280] strongly recommends using the identifiers defined above.						

Table 15: PolicyMappings

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN CA/EE CERT	PROC	RFC3280	ISISMTT	
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*.	+-/--	+-	4.2.1.6		
2	`issuerDomainPolicy CertPolicyId,`				4.2.1.6	T14.#5	
3	`subjectDomainPolicy CertPolicyId }`				4.1.2.6	T14.#5	

Table 16: SubjectAltNames and IssuerAltNames

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**SubjectAltNames** ::= GeneralNames	Alternative technical names of the subject	+-	+	4.2.1.7	T8	[1] [2]
2	**IssuerAltNames** ::= GeneralNames	Alternative technical names of the issuing CA	+-	+	4.2.1.8	T8	[1] [3]
[1]	[RFC3280]: 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]	[RFC3280]: 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. **ISIS-MTT 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 [RFC2255], 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]	**ISIS-MTT 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 [RFC2255], 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	RFC3280	ISISMTT	
1	**SubjectDirectoryAttributes ::=** Attributes	This extension may contain further X.500 attributes of the subject	-/+-	+ (RFC 3039 bis ++)	4.2.1.9	#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]	[RFC3039bis]: 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). RFC3039bis defines a couple of new DName attributes for this purpose (*dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence*). According to RFC3039bis, these attributes are to be stored in the *SubjectDirectoryAttributes* extension of the key certificate. RFC3039bis explicitly states that new attribute types MAY be included according to local definitions. [RFC3281] does not mention, where to record data of this kind. **ISIS-MTT 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) RFC3039 attributes: *dateOfBirth, placeOfBirth, gender, countryOfCitizenship, countryOfResidence*, ISIS-MTT 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	RFC3280	ISISMTT	
1	**BasicConstraints** ::= SEQUENCE {	Indicates a CA certificate and defines how deep a certificate may exists below that CA.	++/+-	++	4.2.1.10		[1]
2	ca BOOLEAN DEFAULT FALSE,	ca=TRUE indicates a CA certificate ca=FALSE indicates an end entity					
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]	[RFC3280] 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. **ISIS-MTT 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	RFC3280	ISISMTT	
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.	+-/--	++	4.2.1.11		[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. [RFC3280]: This extension MUST be included only in CA certificates. Note that RFC3280-compliant client software MUST check naming constraints as described in RFC3280, 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]	[RFC3280]: 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 [RFC3280] 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	RFC3280	ISISMTT	
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.	+-/--	++	4.2.1.12		[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]	[RFC3280]: If the extension is present, at least one optional field MUST be given. Note that RFC3280-compliant client software MUST check *PolicyConstraints* as described in RFC3280, if this extension is present and is marked critical.						

Table 21: ExtendedKeyUsage

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**ExtendedKeyUsage** ::= SEQUENCE SIZE (1..MAX) OF KeyPurposeId	Indicates purposes for which the public key can be used, additional to or in place of those in the *KeyUsage* extension.	+-	++ (RFC n.a.)	4.2.1.13	#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*, *keyEncipherment* or *keyAgreement*	+-	+-			
5	**id-kp-clientAuth** OBJECT IDENTIFIER ::= {id-kp 2}	TLS Web client authentication Consistent only with *KeyUsage* bits: *digitalSignature* and/or *keyAgreement*	+-	+-			
6	**id-kp-codeSigning** OBJECT IDENTIFIER ::= {id-kp 3}	Signing downloadable code Consistent only with *KeyUsage* bits: *digitalSignature*	+-	+-			
7	**id-kp-emailProtection** OBJECT IDENTIFIER ::= {id-kp 4}	E-mail protection Consistent only with *KeyUsage* bits: *digitalSignature*, *nonRepudiation* and/or (*keyEncipherment* or *keyAgreement*)	+-	+-			
8	**id-kp-timeStamping** OBJECT IDENTIFIER ::= {id-kp 8}	Time stamping Consistent only with *KeyUsage* bits: *nonRepudiation*	+-	++			[2b]
9	**id-kp-OCSPSigning** OBJECT IDENTIFIER ::= {id-kp 9}	Signing OCSP responses	+-	++			[3]

[1]	[RFC3280]: 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]	[RFC3280]: 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. **ISIS-MTT PROFILE:** Other key purposes than those listed in this table MAY be included in the *ExtendedKeyUsage* extension.

[2a]	[RFC3280]: 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*. **ISIS-MTT 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	RFC3280	ISISMTT	
1	**CRLDistPointSyntax** ::= SEQUENCE SIZE (1..MAX) OF CRLDistributionPoint	Identifies how CRL information to this certificate can be obtained.	+/++ (RFC+)	+	4.2.1.14	#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: **ISIS-MTT 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 [RFC3280].						
[2]	Notes on support: [RFC3280]: 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. **ISIS-MTT 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 [RFC3280], 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).
[3]	*CHOICE* objects are always *EXPLICIT*ly tagged, independent of the default tagging modus.
[4]	[RFC3280]: If no reasons are specified or only one CRL appears in this extension, the CRL MUST include revocations for all reasons
[5]	[RFC3280]: If this field contains an URL, it MUST be a pointer to the current CRL. Accepted URL formats are described in [RFC3280] Section 4.2.1.7. **ISIS-MTT 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 [RFC2255]. 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.

2.3.2 PKIX Private Certificate Extensions

Table 23: AuthorityInfoAccess

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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	+-	++	A.2		[1]
9	**id-ad-caIssuer** 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]	**ISIS-MTT 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]	**ISIS-MTT PROFILE:** ISIS-MTT 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 24: BiometricData

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3039	ISISMTT	
1	**BiometricSyntax** ::= SEQUENCE OF BiometricData		+-	+	3.2.4	#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]	[RFC3039]: 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	RFC3280	ISISMTT	
1	**QCStatements** ::= SEQUENCE OF QSStatement	A non-empty list of statements	+-	+	[RFC 3039] 3.2.5	#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.					
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] 4.2.1	#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] 4.2.2	#10	
13	**QcEuLimitValue** ::= MonetaryValue	This statement limits the value of transactions, for which the certificate can be used.	+-	+	[ETSI-QC] 4.2.2	#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] 4.2.3	#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] 4.2.4	#10	
[1]	**ISIS-MTT 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]	[RFC3039]: At least one of *semanticsIdentifier* and *nameRegistrationAuthorities* must be present.						

Table 26: OCSPNocheck

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC2560	ISISMTT	
1	**id-ad** OBJECT IDENTIFIER ::= { id-pkix 48 }	Arc for access descriptors			RFC3280		
2	**id-ad-ocsp** OBJECT IDENTIFIER ::= { id-ad 1 }				RFC3280	#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. **ISIS-MTT 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 X.509 standard [X.509]. The draft of the PKIX attribute certificate profile [RFC3281], based on attribute certificate format v2 of X.509, has also been considered here. The attributes and extensions defined there are not yet subject of this version of ISIS-MTT.

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	RFC3281	ISISMTT	
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	RFC3281	ISISMTT	
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)* **ISIS-MTT 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 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. **ISIS-MTT 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 [RFC2255], 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: it 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. **ISIS-MTT 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]	**ISIS-MTT PROFILE:** Both *GeneralizedTime* fields must be encoded according to the format YYYYMMDDHHMMSSZ.
[9]	**ISIS-MTT 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]	**ISIS-MTT PROFILE:** *issuerUniqueID* is supposed to contain *subjectUniqueID* of the CA's certificate. Since ISIS-MTT-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. **ISIS-MTT 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	ISISMTT	
	RFC3281 ATTRIBUTES (NOT YET PART OF ISIS-MTT)						**RFC3281**		
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]
	ISIS-MTT PRIVATE ATTRIBUTES								[2]
7	Procuration	{id-isismtt-at 2}	Procuration information	Y	+-	+-	n.a.	T29a	
8	Admission	{id-isismtt-at 3}	Professional information	N	+-	+-	n.a.	T29b	
9	MonetaryLimit	{id-isismtt-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-isismtt-at 5}	A declaration of majority	N	+-	+-	n.a.	T29d	
11	Restriction	{id-isismtt-at 8}	Some other restriction regarding the usage of this certificate.	Y	+-	+-	n.a.	T29e	[3]
12	AdditionalInformation	{id-isismtt-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]. **ISIS-MTT PROFILE:** These extensions are NOT YET PART of the current version of ISIS-MTT.								

[2]	These attributes were originally defined in the optional SigG-Profile of ISIS-MTT. Applications conforming to the ISIS-MTT core specification MAY include them in attribute certificates.
[3]	**ISIS-MTT PROFILE:** Attribute certificates with restrictive content MUST always be included in the signed document.
[4]	**ISIS-MTT 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-isismtt-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]	**ISIS-MTT 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	ISISMTT	
1	**id-isismtt-at-procuration** OBJECT IDENTIFIER ::= {id-isismtt-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]	**ISIS-MTT 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]	**ISIS-MTT PROFILE:** The *GeneralName* MUST be of type *directoryName* and MAY only contain: - RFC3039 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	ISISMTT	
1	**id-isismtt-at-admission** OBJECT IDENTIFIER ::= {id-isismtt-at 3}	OID for extension/attribute *Admission*			n.a.	T43	
2	**id-isismtt-at-namingAuthorities** OBJECT IDENTIFIER ::= {id-isismtt-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 }						

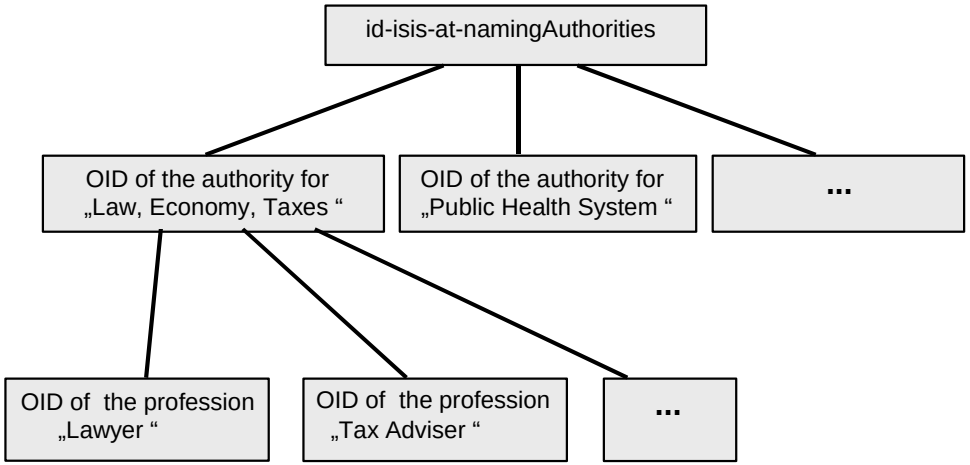
[1] **ISIS-MTT PROFILE:** The relatively complex structure of *AdmissionSyntax* supports the following concepts and requirements:

- 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* are grouped under the OID-branch *id-isis-at-namingAuthorities* and must be applied for.
  See http://www.teletrust.de/anwend.asp?Id=30200&Sprache=E_&HomePG=0 for an application form and http://www.teletrust.de/links.asp?id=30220,11 for an

overview of registered naming authorities.

- 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 in the figure below. The issuing admission authority should always be indicated (field *admissionAuthority*), whenever a registration number is presented. Still, information on admissions can 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	ISISMTT	
1	**id-isismtt-at-monetaryLimit** OBJECT IDENTIFIER ::= {id-isismtt-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	ISISMTT	
1	**id-isismtt-at-declarationOfMajority** OBJECT IDENTIFIER ::= {id-isismtt-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 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]	**ISIS-MTT 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	ISISMTT	
1	`id-isismtt-at-restriction OBJECT IDENTIFIER ::= {id-isismtt-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	ISISMTT	
1	`id-isismtt-at-additionalInformation OBJECT IDENTIFIER ::= {id-isismtt-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	RFC3281	ISISMTT	
	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]
	RFC3280 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]
	RFC3039 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 revokation 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." **ISIS-MTT 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]	**ISIS-MTT 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	RFC3280	ISISMTT	
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 4.1.1.2	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	RFC3280	ISISMTT	
1	`TBSCertList ::= SEQUENCE {`						
2	`  version  Version OPTIONAL,`	Version number of the CRL format	++	++	5.1.2.1	T2.#12	[1]
3	`  signature  AlgorithmIdentifier,`	an identifier of the signature algorithm used by the CA to sign this CRL.			5.1.2.2 4.1.1.2	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]
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	
11	`  } OPTIONAL,`						
12	`  crlExtensions  [0] EXPLICIT Extensions OPTIONAL }`	A non-empty list of CRL extensions	++	++	5.2	T33	
[1]	[RFC3280]: *version* MUST be v2(1), if any extensions present in *crlEntryExtensions* or in *crlExtensions*. Since RFC3280 enforces the presence of extension *CRLNumber*, this is always the case. ISIS-MTT PROFILING: conforming to RFC3280,only *v2(1)* CRLs MUST be issued.						
[2]	Content must be the same as *signatureAlgorithm* in Table 31.3.						
[3]	[RFC3280]: The same constraints apply as for the *issuer* field of key certificates. See Table 2.[4]						
[4]	[RFC3280]: 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]	[RFC3280]: The optional field *nextUpdate* MUST be included in all CRLs. The time period between *thisUpdate* and *nextUpdate* indicates thus the validity of the CRL. 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						

4.1 CRL Extensions

Table 33: An overview of CRL extensions

#	EXTENSION	OID	SEMANTICS	CRITICAL	SUPPORT		REFERENCES		NOTES
					GEN "DIRECT"/ INDIR.CRL	PROC	RFC3280	ISISMTT	
	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.)	++/++ (RFC n.a.)	+	5.2.1 4.2.2.1	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 4.2.1.8	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	

[1] **ISIS-MTT 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] **ISIS-MTT 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 CA 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 CA certificate and the *cRLIssuer* field holds a DName (different from the *subject* of the CA certificate).

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	RFC3280	ISISMTT	
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]	[RFC3280]: CRLs MUST be assigned numbers of an monotonically increasing sequence. This extension allows easily determining whether a particular CRL supersedes another one. **[ISIS-MTT PROFILE]**: [RFC3280] 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	RFC3280	ISISMTT	
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]	[RFC3280]: 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*.						

Table 36: IssuingDistributionPoint

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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]	[RFC3280]: 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]	[RFC3280]: If an URL is given, it MUST point to the most current CRL issued by this CA. The URL schemes *ftp, http, mailto* [RFC1630] and *ldap* [RFC2255] are defined for this purpose. The URI MUST be an absolute, not relative, pathname and MUST specify the host.						

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	RFC3280	ISISMTT	
	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.	--	+-	+-	5.3.2	T39	
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.	--	+-	+-	5.3.4	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.	++	-/++	++	4.3.5	T41	[2]
[1]	[RFC3280]: Conforming CA's SHOULD include these extensions if such information is available.								
[2]	[RFC3280]: Indirect CRLs MUST include the *CertificateIssuer* extension in CRL entries. "Direct" CRLs SHOULD NOT include this extension.								

Table 38: ReasonCode

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
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]	[RFC3280]: 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).						

Table 39: HoldInstructionCode

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**id-ce-holdInstructionCode** OBJECT IDENTIFIER ::= {id-ce 23}	OID of the *HoldInstructionCode* extension			5.3.2		
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.						

Table 40: InvalidityDate

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC3280	ISISMTT	
1	**id-ce-invalidityDate** OBJECT IDENTIFIER ::= { id-ce 24 }	OID of the *InvalidityDate* extension			5.3.3		
2	**InvalidityDate** ::= GeneralizedTime	Syntax of the *InvalidityDate* extension			5.3.3		[1]
[1]	[RFC3280]: The same constraints apply as for the validity field of PKCs. See Table 3.[1]						

Table 41: CertificateIssuer

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN "DIRECT"/ INDIR.CRL	PROC	RFC3280	ISISMTT	
1	**id-ce-certificateIssuer** OBJECT IDENTIFIER ::= { id-ce 29 }	OID of the CertificateIssuer extension			5.3.4		
2	**CertificateIssuer** ::= GeneralNames	Syntax of the CertificateIssuer extension			5.3.4	T8	[1]
[1]	[RFC3280]: 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. **ISIS-MTT 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	ISISMTT	
1	**CertificatePair** ::= SEQUENCE {		+-	++	Chapt.8		[1]
2	forward [0] EXPLICIT Certificate OPTIONAL,		++	++		T1	
3	backward [1] EXPLICIT Certificate OPTIONAL }		+-	+		T1	
[1]	[RFC2587]: The *forward* elements of the *crossCertificatePair* attribute of a CA's directory entry MUST store all CA- as well as cross-certificates, except self-issued certificates issued to this CA. Optionally, the *reverse* 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 *forward* and the *reverse* 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 reverse element is present, the *forward* element value and the *reverse* 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. In the case of V3 certificates, none of the above CA certificates shall include a *BasicConstraints* extension with the *cA* value set to *FALSE*. **ISIS-MTT PROFILE:** Those *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 ISIS-MTT Object Identifiers

The following table lists all ASN.1 object identifiers introduced in the ISIS-MTT Specification.

Table 43: ISIS-MTT Object Identifiers

#	ASN.1 DEFINITION		SEMANTICS	SUPPORT		REFERENCES		NOTES
				GEN	PROC	RFC	ISISMTT	
1	**id-isismtt**	OBJECT IDENTIFIER ::= {1 3 36 8 }		+-	+-	n.a.		
2	**id-isismtt-cp**	OBJECT IDENTIFIER ::= {id-isismtt 1}	Branch for certificate policies	+-	+-	n.a.		
3	**id-isismtt-at**	OBJECT IDENTIFIER ::= {id-isismtt 3}	Branch for attributes and extensions	+-	+-	n.a.		
4	**id-isismtt-at-certHash**	OBJECT IDENTIFIER ::= {id-isismtt-at 13}	OID of an OCSP extension	+-	+-	n.a.	P4.T15	
5	**id-isismtt-at-nameAtBirth**	OBJECT IDENTIFIER ::= {id-isismtt-at 14}	OID of a DName attribute	+-	+-	n.a.	P1.T7	
6	**id-isismtt-at-procuration**	OBJECT IDENTIFIER ::= {id-isismtt-at 2}		+-	+	n.a.	P1.T29a	
7	**id-isismtt-at-admission**	OBJECT IDENTIFIER ::= {id-isismtt-at 3}		+-	+	n.a.	P1.T29b	
8	**id-isismtt-at-monetaryLimit**	OBJECT IDENTIFIER ::= {id-isismtt-at 4}		+-	+	n.a.	P1.T29c	
9	**id-isismtt-at-declarationOfMajority**	OBJECT IDENTIFIER ::= {id-isismtt-at 5}		+-	+	n.a.	P1.T29d	
10	**id-isismtt-at-restriction**	OBJECT IDENTIFIER ::= {id-isismtt-at 8}		+-	+	n.a.	P1.T29e	
11	**id-isismtt-at-namingAuthorities**	OBJECT IDENTIFIER ::= {id-isismtt-at 11}	Branch for registering naming authorities of *Admission* attributes	+-	+-	n.a.	P1.T29b	[1]
12	**id-isismtt-at-additionalInformation**	OBJECT IDENTIFIER ::= {id-isismtt-at 15}		+-	+	n.a.	P1.T29f	
[1]	See http://www.teletrust.de/anwend.asp?Id=30200&Sprache=E_&HomePG=0 for an application form and http://www.teletrust.de/links.asp?id=30220,11 for an overview of registered naming authorities.							

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]	Draft ETSI TS 102 280 v0.1.1 (2004-01) : X.509 V.3 Certificate Profile for Certificates Issued to Natural Persons
[ETSI-QC]	Draft ETSI TS 101 862 v1.3.0 (2004-01): Qualified Certificate Profile, Technical Specification
[ETSI-TSP]	ETSI TS 101 861 v1.2.1 (2003-03): 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
[RFC1034]	Domain Names – Concepts and facilities, November 1987
[RFC1630]	Universal Resource Identifiers in WWW, June 1994
[RFC1738]	Uniform Resource Locators (URL), December 1994
[RFC1883]	Internet Protocol, Version 6 (IPv6) Specification, December 1995
[RFC2247]	Using Domains in LDAP/X.500 Distinguished Names, January 1998
[RFC2255]	An LDAP URL Format, June 1996
[RFC2279]	UTF-8, a transformation format of ISO 10646, January 1998
[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
[RFC3039bis]	Internet X.509 Public Key Infrastructure Qualified Certificates Profile, <draft-ietf-pkix-sonof3039-04.txt>, January 2004
[RFC3161]	Internet X.509 Public Key Infrastructure - Time Stamp Protocol (TSP), August 2001
[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
[RFC791]	Internet Protocol – DARPA Internet Program Protocol Specification (v4), September 1981
[RFC822]	Standard for the format of ARPA Internet Messages, August 13, 1982
[X.509]	ITU-T X.509: Information Technology - Open Systems Interconnection – The Directory: Authentication Framework, 1997

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 2

PKI MANAGEMENT

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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.

Table of Contents

1 Preface 5

2 Simple Enrollment Protocol 6

2.1 Protocol Elements 6

2.1.1 PKCS#10 Certification Request Data Object 6

2.1.2 PKCS#7 Certification Response Data Object 6

2.2 PKI Messages 10

2.2.1 PKCS#10 Messages 10

2.2.2 PKCS#7 Messages 10

2.3 Transport 10

2.3.1 Simple Enrollment Requests 10

2.3.2 Simple Enrollment Responses 10

Annexes 11

Annex A: ASN.1 Definitions 11

Annex B: Abbreviations 12

References 13

1 Preface

This part of the ISIS-MTT specification addresses online communication between PKI components. It defines a profile for ISIS-MTT components that is mainly based on the Internet document "Certificate Management Messages over CMS (CMC)" [RFC 2797], and on the following standards:

- "Cryptographic Message Syntax" [RFC 2630],
- "Internet X.509 Certificate Request Message Format" [RFC 2511],
- "PKCS#10: Certification Request Syntax" [RFC 2314],
- "PKCS#7: Cryptographic Message Syntax" [RFC 2315], and
- "S/MIME Version 3 Message Specification" [RFC 2633].

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 ISIS-MTT specification does only consider conformance requirements for the simple enrollment protocol that MUST be supported by compliant ISIS-MTT 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 ISIS-MTT compliant components MUST satisfy, are specified in the following chapter.

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 [RFC 2314] certification request data object, and a PKCS#7 [RFC 2315] certification response (degenerated *signedData* CMS [RFC 2630]) 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				ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	TABLE	GEN	PROC	VALUES	
1	*certificationRequestInfo*	DER encoded certification request information to be signed	RFC 2314 RFC 2797	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 2797	6.1 3.3.1	+-		+-EE	++CA		
1.4.1	*ExtensionReq*	Attribute that allows to incorporate one or more standard X.509v3 extensions within the PKCS#10 request	RFC 2797	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 ISIS-MTT specification.

[RFC 2797]: This field MAY contain the value NULL, but MUST be present.

ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT specification.

[4] For permitted algorithm identifiers refer to Part 6 (Cryptographic Algorithms) of this ISIS-MTT specification.

Table 2: Fields of *ContentInfo* for Certification Responses

FIELDS			REFERENCES				ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	TABLE	GEN	PROC	VALUES	
1	*contentType*	Indication of the type of content	RFC 2630 RFC 2315	3 7	++		++CA	++EE	OID: 1.2.840.113549.1.7.2	[1]
2	*content*	Content of signed-data	RFC 2630 RFC 2315	5.1 9.1	++		++CA	++EE		
2.1	*version*	Version number of CMS syntax	RFC 2630 RFC 2315	5.1 9.1	++		++CA	++EE	1 3	[2]
2.2	*digestAlgorithms*	Collection (including zero) of message digest algorithm identifiers	RFC 2630 RFC 2315	5.1 9.1	++		++CA	++EE	OID: 1.3.14.3.2.26	[3]
2.3	*encapContentInfo* *contentInfo*	Data to be protected	RFC 2630 RFC 2315 RFC 2797	5.2 9.1 4.3	 --		 --CA	 ---EE		[4]
2.4	*certificates*	Collection of certificates	RFC 2630 RFC 2315	5.1 9.1	-+		+-CA	+-EE		[5]
2.5	*crls*	Collection of CRLs	RFC 2630	5.1	-+		-+CA	-+EE		

			RFC 2315	9.1						
2.6	*signerInfos*	Collection of per-signer information	RFC 2630 RFC 2315 RFC 2797	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] This OID identifies the SHA-1 hash algorithm, which shall be supported by compliant components. The support for other hash algorithms is OPTIONAL. For permitted hash algorithm identifiers refer to P6.S2.1 (Cryptographic Algorithms) of this ISIS-MTT 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 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.

2.3.2 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 ISIS-MTT specification.

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

Attributes ::=                      SET OF Attribute

AttributeValue ::                   =ANY

CertificateChoices ::=              CHOICE {
     certificate                    Certificate,
     extendedCertificate            [0] IMPLICIT ExtendedCertificate,
     attrCert                       [1] IMPLICIT AttributeCertificate }

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

[RFC 2314] B. Kaliski: PKCS#10: Certification Request Syntax; October 1997

[RFC 2315] B. Kaliski: PKCS#7: Cryptographic Message Syntax; October 1997

[RFC 2511] M. Myers, C. Adams, D. Solo, D. Kemp: Internet X.509 Certificate Request Message Format; March 1999

[RFC 2630] R. Housley: Cryptographic Message Syntax; June 1999

[RFC 2633] B. Ramsdell: S/MIME Version 3 Message Specification; June 1999

[RFC 2797] M. Myers, X. Liu, J. Weinstein: Certificate Management Messages over CMS, <draft-ietf-pkix-rfc2797-bis-01.txt>; July 2001

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 3

MESSAGE FORMATS

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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.

Table of Contents

1 Preface 5

2 Message Types Based on S/MIME 6

2.1 S/MIME Message Types 6

2.1.1 Message Type for Enveloped Data 7

2.1.2 Message Type for Signed Data 7

2.1.3 Message Type for Certificates-Only Messages 8

2.1.4 Message Type for Signed Data With Multipart Encoding 8

2.2 S/MIME Message Transformations 9

3 Data Structures in S/MIME Messages 10

3.1 Summary of Conformance Requirements 10

3.1 General CMS Syntax 11

3.2 Signed-data Content Type 12

3.3 Enveloped-data Content Type 17

4 File Signature and Encryption 21

4.1 File Signature 21

4.2 File Encryption 22

Annex A: ASN.1 Definitions 23

References 27

1 Preface

This part of the ISIS-MTT specification addresses message formats to be used during the exchange of data between PKI components. It defines a profile for ISIS-MTT message formats that is mainly based on the Internet documents for S/MIME [RFC 2633], MIME [RFC 2045, RFC 2046], and CMS [RFC 3369].

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 gives references to the standards on which this part of ISIS-MTT 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 ISIS-MTT 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 requirements 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.

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 2633, 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 an 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 2633, 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 an 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 2633, 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 an 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.

2.2 S/MIME Message Transformations

Compliant components SHALL support the MIME transformations defined in [RFC 2633, chapter 3] that are required to create an S/MIME message with the following exception during the preparation of MIME objects.

ISIS-MTT 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, 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 [RFC3369] 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				ISIS-MTT			
#	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 3369	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 3369	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 ISIS-MTT.										

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				ISIS-MTT			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of CMS syntax	RFC 3369	5.1	++		++	++	1 2 3 4	[1] [5] [2] [5]
2	*digestAlgorithms*	Collection (including zero) of message digest algorithm identifiers	RFC 3369	5.1	++		++	++	OID: 1.3.14.3.2.26	[3]
3	*encap-ContentInfo*	Data to be protected	RFC 3369	5.1	++	Table 3	++	++		
4	*certificates*	Collection of certificates	RFC 3369	5.1	+-		+-	+-		[4]
5	*crls*	Collection of CRLs	RFC 3369	5.1	+-		+-	+-		
6	*signerInfos*	Collection of per-signer information	RFC 3369	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]	This OID identifies the SHA-1 hash algorithm, which shall be supported by compliant components. The support for other hash algorithms is optional.
[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.
[5]	These versions are currently not considered in ISIS-MTT.

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				ISIS-MTT			
#	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 3369	5.2	++		++	++	OID: 1.2.840.113549.1.7.1	[1]
2	*eContent*	Associated and protected content	RFC 3369	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				ISIS-MTT			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	RFC 3369	5.3	++		++	++	1	[1]
2	*sid*	Identification of the signers certificate	RFC 3369	5.3	++		++	++		[2]
3	*digestAlgorithm*	Identification of the signers hash algorithm	RFC 3369	5.3	++		++	++	OID: 1.3.14.3.2.26	[3]
4	*signedAttrs*	Collection of signed attributes	RFC 3369	5.3	+-	Table 5	+- ++	++ ++		[4] [5]
5	*signatureAlgorithm*	Identification of the signers signature algorithm	RFC 3369	5.3	++		++	++		[6]
6	*signature*	Digital signature of the signer	RFC 3369	5.3	++		++	++		
7	*unsignedAttrs*	Collection of unsigned attributes	RFC 3369	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]	This OID identifies the SHA-1 hash algorithm, which shall be supported by compliant components. The support for other hash algorithms is optional. The value provided in this field SHALL be contained in the *SignedData.digestAlgorithms* field (see T2.#2).
[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 ISIS-MTT 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				ISIS-MTT			
#	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 3369	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 3369	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 3369	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	ETSI TS 101 733	8.8.2	+-		+-	+-		[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.	ETSI TS 101 733	9.2.1	+-		+-	+-		[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	ETSI TS 101 733	9.2.2	+-		+-	+-		[4]

		validation of the signer and CA certificates in an electronic signature.								
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	ETSI TS 101 733	9.3.3	+-		+-	+-		[4]
8	*signingCertificate* *id-aa-signingCertificate* {1 2 840 113549 1 9 16 212}	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]

[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*. **ISIS-MTT 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]	**ISIS-MTT 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.

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				ISIS-MTT			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	RFC 3369	6.1	++		++	++	0	[1]
2	*originatorInfo*	Signer information including certificates and CRLs	RFC 3369	6.1	+-		--	--		[1]
3	*recipientInfos*	Collection of per-recipient information	RFC 3369	6.1	++	Table 7	++	++		
4	*encryptedContentInfo*	Encrypted data	RFC 3369	6.1	++	Table 9	++	++		
5	*unprotectedAttrs*	Collection of non-encrypted attributes	RFC 3369	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				ISIS-MTT			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*ktri*	per-recipient information using key transport	RFC 3369	6.2.1	++	Table 8	++	++		[1]
2	*kari*	recipient information using key agreement	RFC 3369	6.2.2	++		--	--		[1]
3	*kekri*	recipient information using previously distributed symmetric key-encryption keys	RFC 3369	6.2.3	++		--	--		[1]
4	*pwri*	recipient information using a password or shared secret value	RFC 3369	6.2.4	++		--	--		[1]
5	*ori*	recipient information for additional key management techniques	RFC 3369	6.2.5	++		--	--		[1]

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

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				ISIS-MTT			
#	NAME	SEMANTICS	DOCUMENT	CHAP.	STATUS	TABLE	SUPPORT			NOTES
							GEN	PROC	VALUES	
1	*version*	Version number of syntax	RFC 3369	6.2.1	++		++	++	0	[1]
2	*rid*	Identification of the recipients certificate	RFC 3369	6.2.1	++		++	++		[2]
3	*keyEncryptionAlgorithm*	Identification of the key-encryption algorithm	RFC 3369	6.2.1	++		++	++		
4	*encryptedKey*	Encrypted content-encryption key	RFC 3369	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				ISIS-MTT			
#	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 3369	6. 1	++		++	++	OID: 1.2.840.113549.1.7.1	[1]
2	*contentEncryption Algorithm*	Identification of the content-encryption algorithm	RFC 3369	6. 1	++		++	++		
3	*encryptedContent*	Encrypted content-encryption key	RFC 3369	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 3369].

The following CMS container types MUST be supported by ISIS-MTT-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* using the *SigningCertificate* attribute, which is defined in [RFC 2634]. Additionally, the *certs* field SHOULD contain all certificates in the certificate path up to the certificate of root or top-level CA.

[RFC 2630] 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 ISIS-MTT-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. ISIS-MTT-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 ISIS-MTT specification in alphabetical order.

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

```
AttributeValue ::= ANY
```

```
CertificateChoices ::= CHOICE {
  certificate Certificate,
  extendedCertificate [0] IMPLICIT ExtendedCertificate,
  attrCert [1] IMPLICIT AttributeCertificate }
```

```
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 OriginatorInfo OPTIONAL,
  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 CertificateRevocationLists OPTIONAL }
```

```
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
```

```
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 CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos }
```

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

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

```
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

[ETSI TS 101 733] ETSI TS 101 733 v1.4.0: Electronic Signature Formats, September 2002

[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 2633] B. Ramsdell: S/MIME Version 3 Message Specification; June 1999

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

[RFC 3369] R. Housley: Cryptographic Message Syntax; August 2002

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 4

OPERATIONAL PROTOCOLS

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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.

Table of Contents

1 Preface 5

1.1 Compatibility Aspects 6

2 Directory Access via LDAP 7

2.1 The ISIS-MTT LDAP Schema 8

2.2 Access Protocol 18

2.3 Using Delta CRLs 19

3 Directory Access via OCSP 20

3.1 Protocol Elements 20

3.1.1 Standard OCSP Extensions 26

3.1.2 ISIS-MTT Private OCSP Extensions 29

3.2 Certificate Contents 30

3.2.1 Queried certificates 30

3.2.2 Responder's certificates 30

3.3 Transport over HTTP 31

4 Directory Access via FTP and HTTP 32

5 Time Stamp Protocol (TSP) 33

References 34

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 ISIS-MTT Specification to select a “necessary minimum” of possible repository functions and access methods, which shall be supported by all ISIS-MTT-compliant repositories and client systems. In this way, interoperability within the ISIS-MTT 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 ISIS-MTT 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. ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT Specification, such definitions are always commented and the corresponding note is marked with the words '**ISIS-MTT 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 [RFC2251]. Nevertheless, only protocol elements specified in LDAP v2 [RFC1777] are and SHOULD be used in a PKIX- and respectively ISIS-MTT-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, that must be supported by all conforming certification authorities, is defined in [RFC2560] and will be profiled in this ISIS-MTT specification.

When offering or accessing time stamp services, ISIS-MTT–compliant systems MUST apply the protocol defined in [RFC3161] and profiled in [ETSI-TSP]. 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 [RFC2551]. Basically, only protocol elements specified in LDAP v2 [RFC1777] are and MAY be used in a PKIX- and respectively ISIS-MTT-compliant PKI. Nevertheless, ISIS-MTT-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. To go around this problem, Section 8 of [RFC2587] proposes to employ the encoding specified for the *Undefined* attribute syntax: the DER-encoding of the PKC or CRL will be encoded similarly to the *OCTET STRING* syntax. To change the representation of PKCs/CRLs from the string form to the *Undefined* syntax, the standard definition of attribute syntaxes might need to be altered in the LDAP v2 directory.

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 [RFC2256], 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.

A further reason for using LDAP v3 is related to the representation of string attribute values in requests and responses. In LDAP v2, those strings may contain only printable ASCII characters. First [RFC2253] introduces UTF8 strings for the representation of DNames in the LDAP v3 access protocol. As DNames often contain country-specific, non-printable characters, the ISIS-MTT specification relies exclusively on LDAP v3.

Current RFCs on PKI applications of LDAP concern only PKCs and CRLs. Standardization work on attribute certificates (ACs) is just at the beginning at IETF and there exist only a draft paper [DraftSchema] on how to include ACs and CRLs on ACs in an LDAP directory schema. Considering that this paper has not been finalized yet, 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 ISIS-MTT specification proposes to handle ACs and CRLs of ACs within the LDAP/OCSP-infrastructure as if they were PKCs and respectively RFCs of PKCs. This is also the approach followed by current systems and simply 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. ISIS-MTT-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.

ISIS-MTT PROFILE: Note that handling PKCs and ACs in the same way is a different approach than that followed in [RFC3281] and [DraftSchema]. In those document, ACs are forced to be kept separated from PKCs: ACs and CRLs of ACs are kept in different directory attributes (*attributeDescriptorCertificate* and *attributeCertificaterevocationList*). Furthermore, those PKIX documents forbid the same CA to issue the same CRL to keep information about PKCs and ACs at same time. In contrast to that, ISIS-MTT 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 ISIS-MTT 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.

ISIS-MTT 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.

ISIS-MTT PROFILE: Note that the only requirement for a directory to be ISIS-MTT-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.

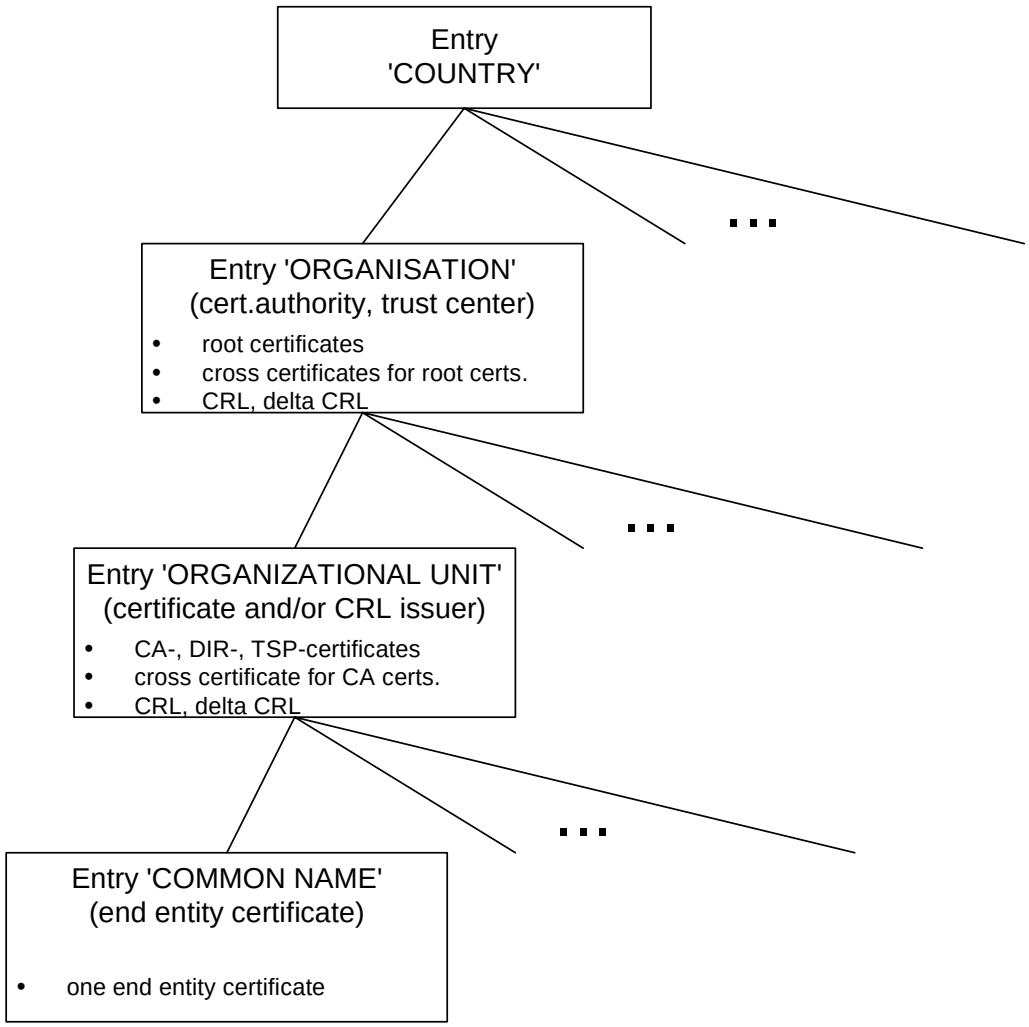


Figure 1: An exemplary DIT structure for ISIS-MTT-compliant directories

Table 1: Attribute Types and Attribute Sets

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT LDAP SERVER	REFERENCES		NOTES
				RFC	TABLE	
	STANDARD X.520 DNAME ATTRIBUTES					
1	name ATTRIBUTE ::={ WITH SYNTAX DirectoryString {32768}, EQUALITY MATCHING RULE caseIgnoreMatch, SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) name(41) }	an abstract class used to derive other DName attribute types below	no relevance	RFC2256 5.42		
2	commonName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) commonName(3) }		++	RFC2256 5.4		
3	surname ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) surname(4) }		+	RFC2256 5.5		
4	givenName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) givenName(42) }		+	RFC2256 5.43		
5	serialNumber ATTRIBUTE ::={ WITH SYNTAX DirectoryString {1..64}, EQUALITY MATCHING RULE caseIgnoreMatch, SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) serialNumber(5) }		++	RFC2256 5.6		[1]
6	countryName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX PrintablString (SIZE(2)) SINGLE VALUE TRUE ID joint-iso-ccitt(2) ds(5) attributeType(4) countryName(6) }		++	RFC2256 5.7		
7	localityName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {128} ID joint-iso-ccitt(2) ds(5) attributeType(4) localityName(7) }		+	RFC2256 5.8		

8	stateOrProvinceName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {128} ID joint-iso-ccitt(2) ds(5) attributeType(4) stateOrProvinceName(8) }		+	RFC2256 5.9		
9	streetAddress ATTRIBUTE ::={ WITH SYNTAX DirectoryString {128}, EQUALITY MATCHING RULE caseIgnoreMatch, SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) streetAddress(9) }		+	RFC2256 5.10		
10	organizationName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) organizationName(10) }		++	RFC2256 5.11		
11	organizationalUnitName ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) organizationalUnitName(11) }		++	RFC2256 5.12		
12	title ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) title(12) }		+-	RFC2256 5.13		
13	businessCategory ATTRIBUTE ::={ WITH SYNTAX DirectoryString {128}, EQUALITY MATCHING RULE caseIgnoreMatch, SUBSTRINGS MATCHING RULE caseIgnoreSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) businessCategory(15) }	occupation of a person	-	RFC2256 5.16		[2]
14	postalAddress ATTRIBUTE ::={ WITH SYNTAX SEQUENCE SIZE(1..6) OF DirectoryString(30), EQUALITY MATCHING RULE caseIgnoreListMatch, SUBSTRINGS MATCHING RULE caseIgnoreListSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) postalAddress(16) }		+	RFC2256 5.17		
15	postalCode ATTRIBUTE ::={ WITH SYNTAX DirectoryString {40}, EQUALITY MATCHING RULE caseIgnoreListMatch, SUBSTRINGS MATCHING RULE caseIgnoreListSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) postalCode(17) }		+	RFC2256 5.18		

16	initials ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) initials(43) }		+-	RFC2256 5.44		
17	generationQualifier ATTRIBUTE::={ SUBTYPE OF name WITH SYNTAX DirectoryString {64} ID joint-iso-ccitt(2) ds(5) attributeType(4) generationQualifier(44) }		+-	RFC2256 5.45		
18	dnQualifier ATTRIBUTE ::={ WITH SYNTAX PrintableString, EQUALITY MATCHING RULE caseIgnoreListMatch, SUBSTRINGS MATCHING RULE caseIgnoreListSubstringsMatch, ID joint-iso-ccitt(2) ds(5) attributeType(4) dnQualifier(46) }	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	+-	RFC2256 5.47		
	PKI-SPECIFIC ATTRIBUTES					
19	userCertificate ATTRIBUTE ::= { WITH SYNTAX Certificate EQUALITY MATCHING RULE certificateExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) userCertificate(36) }		++	RFC2256 5.37 RFC2587 3.1		
20	cACertificate ATTRIBUTE ::= { WITH SYNTAX Certificate EQUALITY MATCHING RULE certificateExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) cACertificate(37) }		++	RFC2256 5.38 RFC2587 3.2		
21	crossCertificatePair ATTRIBUTE::={ WITH SYNTAX CertificatePair EQUALITY MATCHING RULE certificatePairExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) crossCertificatePair(40)}		++	RFC2256 5.41 RFC2587 3.2		
22	authorityRevocationList ATTRIBUTE::={ WITH SYNTAX CertificateList EQUALITY MATCHING RULE certificateListExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) certificateRevocationList(38)}		++	RFC2256 5.39 RFC2587 3.2		
23	certificateRevocationList ATTRIBUTE::={ WITH SYNTAX CertificateList EQUALITY MATCHING RULE certificateListExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) certificateRevocationList(39)}		++	RFC2256 5.40 RFC2587 3.2		
24	deltaRevocationList ATTRIBUTE ::= { WITH SYNTAX CertificateList EQUALITY MATCHING RULE certificateListExactMatch ID joint-iso-ccitt(2) ds(5) attributeType(4) deltaRevocationList(53) }		+-	RFC2256 5.54 RFC2587 3.2.1		

25	domainComponent ATTRIBUTE ::= { WITH SYNTAX IA5String EQUALITY MATCHING RULE caseIgnoreIA5Match ID 0 9 2342 19200300 100 1 25 }	Syntax definition for a *domainComponent* attribute of type *IA5String* in X.500 style. This is a single-valued attribute !	+-	RFC2247 Section 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], [RFC2256]: serial number of a device [RFC2456]: not mentioning *serialNumber* [RFC3039]: 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 **ISIS-MTT 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 [RFC2256] 5.16: This attribute describes the kind of business performed by an organization. **ISIS-MTT PROFILE:** the interpretation of this attribute is as in [X520], i.e. occupation of a person or organization
[3]	**ISIS-MTT 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	top OBJECT CLASS ::= { MUST CONTAIN {objectClass } ID joint-iso-ccitt(2) ds(5) objectClass(6) top(0) }	abstract class to derive other classes below	no relevance	RFC2256 7.1		
2	country OBJECT CLASS ::= { SUBCLASS OF {top} MUST CONTAIN {countryName } MAY CONTAIN { description | searchGuide | } ID joint-iso-ccitt(2) ds(5) objectClass(6) country(2) }	class to define country entries in the DIT	++	RFC2256 7.3		
3	organization OBJECT CLASS ::= { SUBCLASS OF {top} MUST CONTAIN {organizationName } MAY CONTAIN OrganizationalAttributeSet ID joint-iso-ccitt(2) ds(5) objectClass(6) organization(4) }		++	RFC2256 7.5		
4	organizationalUnit OBJECT CLASS ::= { SUBCLASS OF {top} MUST CONTAIN {organizationalUnitName } MAY CONTAIN OrganizationalAttributeSet ID joint-iso-ccitt(2) ds(5) objectClass(6) organizationalUnit(5) }		++	RFC2256 7.6		
5	person OBJECT CLASS ::= { SUBCLASS OF {top} MUST CONTAIN {commonName | surname } MAY CONTAIN {...} ID joint-iso-ccitt(2) ds(5) objectClass(6) person(6) }		++	RFC2256 7.7		[1]
	PKIX-SPECIFIC OBJECT CLASSES					

6	strongAuthenticationUser OBJECT CLASS ::= {
	SUBCLASS OF {top}
	KIND auxiliary
	MUST CONTAIN {userCertificate}
	ID joint-iso-ccitt(2) ds(5) objectClass(6) strongAuthenticationUser(15)
	}
			6 or 7: ++	RFC2256 7.16		[2]

7	pkiUser OBJECT-CLASS ::= { SUBCLASS OF {top} KIND auxiliary MAY CONTAIN{userCertificate} ID joint-iso-ccitt(2) ds(5) objectClass(6) pkiUser(21) }		6 or 7: ++	RFC2587 3.1		[2]
8	certificationAuthority OBJECT-CLASS ::= { SUBCLASS OF { top} KIND auxiliary MUST CONTAIN {cACertificate | certificateRevocationList | authorityRevocationList } MAY CONTAIN crossCertificatePair ID joint-iso-ccitt(2) ds(5) objectClass(6) pkiCA(16)}		8 or 9: ++	RFC2256 7.17		[3]
9	pkiCA OBJECT-CLASS ::= { SUBCLASS OF {top} KIND auxiliary MAY CONTAIN {cACertificate | certificateRevocationList | authorityRevocationList | crossCertificatePair } ID joint-iso-ccitt(2) ds(5) objectClass(6) pkiCA(22)}		8 or 9: ++	RFC2587 3.2		[3]
10	dcObject OBJECT-CLASS ::= { SUBCLASS OF({top} KIND auxiliary MUST CONTAIN { domainComponent } ID 1 3 6 1 4 1 1466 344 }	An auxiliary class defined in X.500 style to contain a *domainComponent* attribute	+-	RFC2247 5.1		
	ISIS-MTT-SPECIFIC OBJECT CLASSES					
11	pkiUserData OBJECT-CLASS ::= { SUBCLASS OF {top} KIND auxiliary MAY CONTAIN {countryName | serialNumber | givenName | title | postalAttributeSet | organizationName | organizationalUnitName | organizationalAttributeSet } ID joint-iso-ccitt(2) bmpt(262) telekom(1) security(10) objectClass(3) pkiUserData(6)}		++			

12	pkiCaData OBJECT-CLASS ::= { SUBCLASS OF {top} KIND auxiliary MUST CONTAIN commonName MAY CONTAINdeltaRevocationList ID joint-iso-ccitt(2) bmpt(262) telekom(1) security(10) objectClass(3) pkiCaData(7)}		++			
[1]	**ISIS-MTT PROFILE:** These object class definitions are not identical with those in RFC2256 and X.521. Optional attributes, not listed in Table 1, have been left out.					
[2]	RFC2256 and respectively RFC2587 give slightly different class definitions for representing a certificate holder. Any of them MAY be used in ISIS-MTT-compliant directories.					
[3]	RFC2256 and respectively RFC2587 give slightly different class definitions for representing a certificate authority. Any of them MAY be used in ISIS-MTT-compliant directories.					

Table 3: Entries of the Proposed Directory Schema

#	ENTRY NAME	ENTRY STRUCTURE	SEMANTICS	REFEREN-CES	NO TES
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]	**ISIS-MTT PROFILE:** When using this schema, *organizationName* MUST be unique among all certification authorities of the PKI.
[2]	**ISIS-MTT 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 ISIS-MTT-conforming clients. The following operations MUST be supported by all ISIS-MTT-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 an 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.	RFC2251 4.2 RFC2559 5.1 and 6.1	
2	unbind	Closes or aborts an LDAP session.	RFC2251 4.3 RFC2559 5.3 and 6.3	
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>	RFC2251 4.5 RFC2559 5.2	
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 >	RFC2251 4.5 RFC2559 5.2	
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.	RFC2251 4.5 RFC2559 5.2	

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.	RFC2251 4.5 RFC2559 5.2	
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.	RFC2251 4.5 RFC2559 5.2	
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.	RFC2251 4.5 RFC2559 6.3	

2.3 Using Delta CRLs

In this profile, using delta CRL is optional for directory servers as well as for clients. In conformance with [RFC 2459], the CA MUST always issue a complete CRL whenever a delta CRL is issued, so that clients not supporting the delta mechanism, are still able to obtain up-to-date status information.

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	RFC2560	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	+-	+-	RFC2459 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	+-	++	RFC2459 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			RFC2459 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. **ISIS-MTT 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]	**ISIS-MTT PROFILE:** the list MUST contain at least one single request.

Table 6: (Single) Request

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC2560	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	+-	++	RFC2459 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			RFC2459 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			RFC2459 4.1.2.2	P1.T2	
[1]	**ISIS-MTT PROFILE:** The hash values in *certID* MUST be built using SHA-1. Processing components (typically the responder) MUST support SHA-1 and SHOULD support RIPEMD160 and MD5.						
[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	RFC2560	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	RFC2560	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			RFC2459 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	+-	+	RFC2459 4.1.1	P1.T1	[3]
8	**ResponseData** ::= SEQUENCE {				4.2.1		
9	version [0] EXPLICIT Version DEFAULT v1,	Version of BasicOCSPResponse			RFC2459 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	+-	++	RFC2459 4.1	T9, P1.T9	
14	**ResponderID** ::= CHOICE {				4.2.1		
15	byName [1] EXPLICIT Name,	DName of the responder	+-	++	RFC2459 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	+-	++	RFC2459 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	+-	+-	RFC2459 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)
ISIS-MTT 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 ISIS-MTT-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.
ISIS-MTT 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] **ISIS-MTT 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] **ISIS-MTT PROFILE:** Time instances MUST be specified using the format YYYYMMDDhhmmssZ.

[5] **ISIS-MTT 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] **ISIS-MTT 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.
ISIS-MTT PROFILE: This ISIS-MTT-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	RFC2560	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	
	ISIS-MTT 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	RFC2560	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. **ISIS-MTT 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 ISIS-MTT-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	RFC2560	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	RFC2560	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. **ISIS-MTT 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	RFC2560	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. **ISIS-MTT 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. ISIS-MTT-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]	**ISIS-MTT PROFILE:** ArchiveCutoff MUST have the format YYYYMMDD000000Z.						

Table 14: ServiceLocator

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC2560	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,				RFC2459 4.1.2.4	P1.T5	
4	locator AuthorityInfoAccess OPTIONAL }				RFC2459 4.2.2.1	P1.T23	
[1]	**ISIS-MTT 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 ISIS-MTT Private OCSP Extensions

Table 15: CertHash (Positive Statement)

#	ASN.1 DEFINITION	SEMANTICS	SUPPORT		REFERENCES		NOTES
			GEN	PROC	RFC2560	TABLE	
1	**id-isismtt-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.			RFC2459 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] **ISIS-MTT 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 [RFC2459], 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).

ISIS-MTT 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.

ISIS-MTT 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 *extKeyUsage* 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 [RFC2459].
(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.

ISIS-MTT 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. ISIS-MTT compliant systems MUST employ the Hypertext Transfer Protocol (HTTP) [RFC 2068] 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 ISIS-MTT-compliant directories is LDAP v3, which provides access to certificates and CRLs including search and matching facilities. This ISIS-MTT 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 ISIS-MTT-compliant directory services MAY, but need not make certificates and CRLs available for download via FTP and/or HTTP and respectively that ISIS-MTT-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 ISIS-MTT directory schema.

5 Time Stamp Protocol (TSP)

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

ISIS-MTT 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 ISIS-MTT 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

[DraftSchema] Internet X.509 Public Key Infrastructure - Additional Schema for PKIs and PMIs, <draft-ietf-pkix-ldap-schema-01.txt>, September 2000

[ETSI-TSP] ETSI TS 101 861 v1.2.1: Time Stamping Profile, March 2003

[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

[RFC1778] The String Representation of Standard Attribute Syntaxes, RFC 1778, March 1995

[RFC2068] Hypertext Transfer Protocol -- HTTP/1.1, January 1997

[RFC2251] Lightweight Directory Access Protocol (v3), RFC 2551, December 1997

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

[RFC2253] Lightweight Directory Access Protocol (v3): UTF-8 String Representation of Distinguished Names, RFC 2253, December 1997

[RFC2256] A Summary of the X.500(96) User Schema for use with RFC2251, RFC 2256, December 1997

[RFC2559] Internet X.509 Public Key Infrastructure - Operational Protocols - LDAPv2, RFC 2559

[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

[RFC2587] Internet X.509 Public Key Infrastructure - LDAPv2 Schema, RFC 2587, 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

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

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 5

CERTIFICATE PATH VALIDATION

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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.

Table of Contents

1 Preface 5

2 Certificate Path Validation Procedure 7

2.1 Building the Certificate Path 11

2.2 Validating the Certificate Path 17

2.3 Checking the Revocation Status 22

References 30

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), that 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 of a communication system, 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 ISIS-MTT 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, ISIS-MTT-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*, *CerificationPolicy*, *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, ISIS-MTT-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 ISIS-MTT 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 ISIS-MTT 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 certification path validation algorithm presented here is laid out to comply with the one described in [RFC2459]. The algorithm described in the RFC is based on the simplifying assumption that all valid paths start at a single "most trusted" root CA. Furthermore, the RFC does not specify how to obtain certificates, how to build a certificate path and how to obtain CRLs or certificate status information.

The ISIS-MTT 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.

[RFC2459] deals only with PKCs. The certificate path building and validation algorithm has been extended here 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 beyond the requirements of the PKIX profile.

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.

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 structures borrow the name of the corresponding ASN.1 data type (e.g. *Certificate*, *Name*) and are assumed to be implemented as object classes that offer methods for accessing embedded data fields (e.g. *GetIssuer*()), as usual in the object-oriented programming discipline. Some new data types are introduced in Table 1. Whenever possible, variable names follow the notation of RFC 2459 too.

Table 1: Common Data Types

#	DATA TYPE	DESCRIPTION	NOTES

1	`typedef enum {`
	`  RootCACert;`
	`  CACert;`
	`  CrossCert;`
	`  EndEntityPKC;`
	`  EndEntityAC;`
	`} CertType;`
		The *CertType* enumeration type is used to classify certificates.	

2	`class CertInfo {`
	`  CertType  certType;`
	`  bool  revoked;`
	`  Time  revocTime;`
	`  CRLReason  revocReason;`
	`  Time  statusInfoNextUpdate;`
	`  Certificate  cert;`
	`  AttributeCertificate  acert;`
	`};`
		This data structure can be seen as the basic item of the local certificate repository. It is used to 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 these 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 ISIS-MTT 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	typedef IA5String LdapUrl;	An URL for accessing a directory over LDAP. As described in RFC 2255, the URL format does not only contain a server address, but parameters for the LDAP-read or search operation.	
7	typedef IA5String OcspUrl;	An URL for accessing the OCSP-service of a directory. The standard transport mechanism for OCSP-messages is HTTP.	

8	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.	

9	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, are 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 ISIS-MTT Specification.

Table 2: ValidateCertificate()

#	PSEUDO-CODE	COMMENTS	NOTES

1	`bool ValidateCertificate(`
	`  Certificate  in  tbvCert,`
	`  CertInfoList in  tbvCerts,`
	`  Time  in  refTime,`
	`  KeyPurpose  in  intendedKeyUsage,`
	`  PolicyList  in  initialPolicySet,`
	`  CertInfoList inout trustedCerts,`
	`  CrlInfoList  inout trustedCrls )`
	`{`
		This is the main entry point of the certificate path validation algorithm.
		The 'to be verified' certificate or attribute certificate of the EE is passed in *tbvCert*. Additionally, *tbvCerts* may contain zero or more certificates 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 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).
		The required usage of the certified key is to be used is indicated in *intendedKeyUsage*. In case of an attribute certificate, this parameter is ignored by the procedure.
		initialPolicySet contains a set of initial policy identifiers (each comprising a sequence of policy element identifiers), which identifies one or more certificate policies, any one of which would be accepted by the relying party or application for the purposes of certification path processing. An empty policy set indicates that *any policy* is acceptable.
		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	`Certificate &tbvCert = tbvCerts.getLastItem();`	*tbvCert* is a reference (or alias) to the 'to be verified' certificate that is the last element of *tbvCerts*.	

3	`if( tbvCert.certType != EndEntityAC )`
	`{`
	`  if( tbvCert.KeyUsagePresent()==false )`
	`  return false;`
	`  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*. CA certificates (i.e. CA-, root-CA- and cross-certificates) MUST furthermore contain the *BasicConstraints* extension and MUST have the *cA*-flag set. If the intended usage is not permitted, *ValidatCertificate*() returns *false*.
		ISIS-MTT PROFILE: Note that the *KeyUsage* extension MUST be present in all PKCs and is always critical (P1.T12.[1]).	

4	`CertInfoList tbvPath, trustedPath;`
	`tbvPath.Clear();`
	`if( BuildAndValidateCertPath( tbvCert,`
	`tbvCerts,`
	`refTime,`
	`initialPolicySet,`
	`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.	

5	`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.	
6	`  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 consist 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 $A_{i+1}\neq A_i$, 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 build and verified, even if the local database is empty (up to one trusted root certificate).

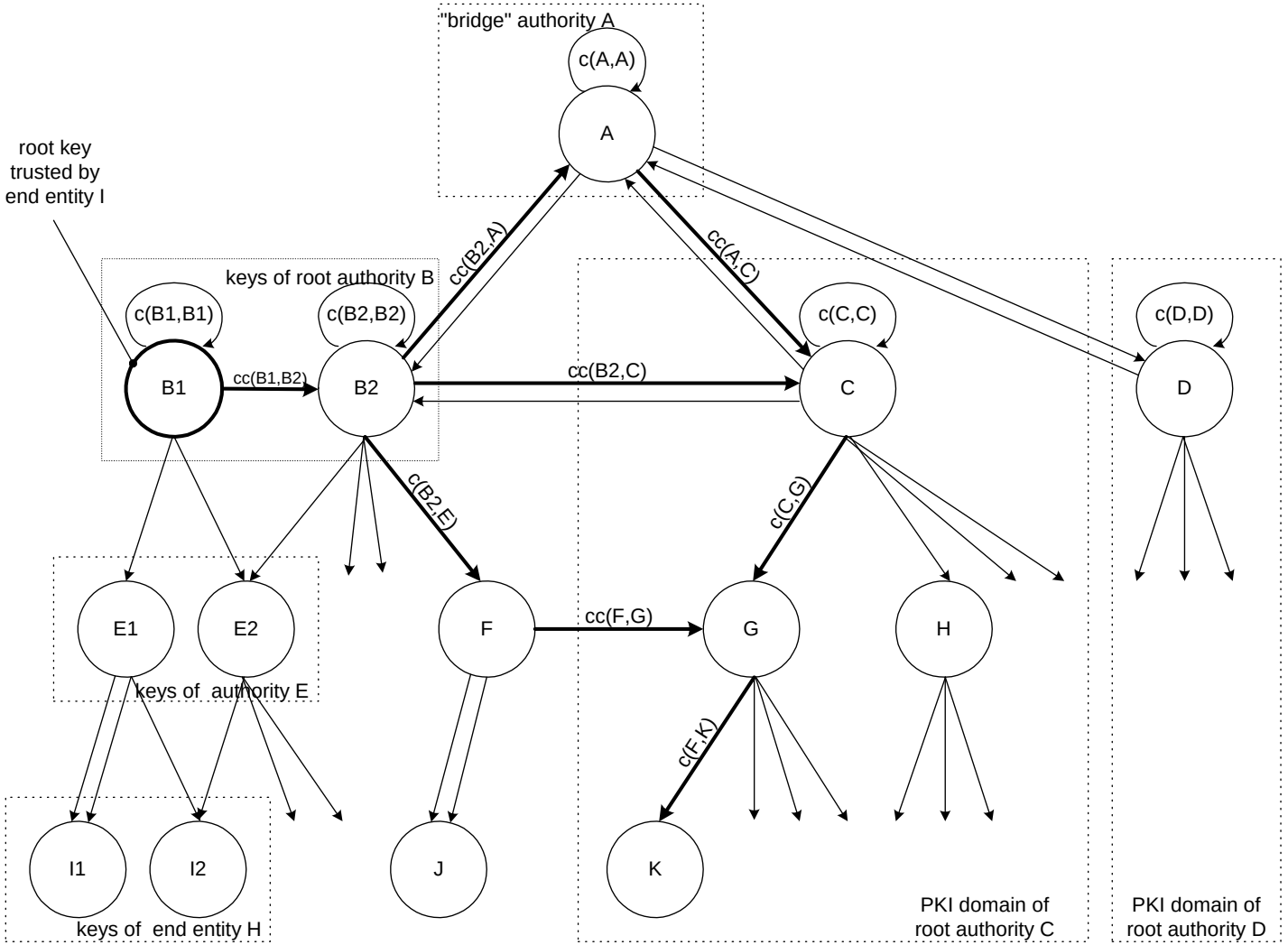


Figure 1: An example of a PKI with multiple root CAs and cross-certification

Table 3: BuildAndValidateCertPath( )

#	PSEUDO-CODE	COMMENTS	REF. TO DPBA	NOTES

1	`bool BuildAndValidateCertPath(`
	`  Certificate in tbvCert,`
	`  CertInfoList in tbvCerts,`
	`  Time in refTime,`
	`  PolicyList in initialPolicySet,`
	`  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 identical to the one used in RFC 2459:
		• 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' EE 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.certType == rootCACert )` `  {`	If the certificate just reached is a self-signed root certificate, the search terminates.	2)	

5	`  if( trustedCert.findCert( tbvCert )`
	`  {`
	`  if( ValidateCertPath( tbvCertPath`
	`  tbvCerts,`
	`  refTime,`
	`  intendedKeyUsage,`
	`  initialPolicySet,`
	`  trustedCerts,`
	`  trustedCrls )==false )`
	`  return false;`
	`  trustedCertPath = tbvCertPath;`
	`  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;`
		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)	

	`OCTET_STRING authorityKeyId;` `authorityKeyId = tbvCert.GetKeyIdFromAuthKeyId();`	key identifier will be retrieved from the *keyIdentifier* field of the *AuthorityKeyIdentifier* extension of *tbvCert.* **ISIS-MTT 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, initialPolicySet,`
	`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.
		ISIS-MTT 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, initialPolicySet,`
	`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.certType != 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.
		ISIS-MTT 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 [RFC 2587], cross-certificates MUST occur in the *forward* 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 *backward* fields of *CrossCertificatePair* attributes in the directory entry of the trusted, issuing CA. Applications may achieve better performance, if collecting all '*backward*' 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, initialPolicySet,`
	`  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 2459. Some minor modifications (corrections and enhancements) have been applied to BPVA, which are conspicuously indicated by the words '**ISIS-MTT PROFILE'.** The algorithm assumes that certificates do not use subject or unique identifier fields or private critical extensions, as recommended in RFC2459 and as strictly enforced by ISIS-MTT. 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	REF. TO BPVA	NOTES

1	`bool ValidateCertPath( CertInfoList in tbvCertPath,`
	`CertInfoList in tbvCerts,`
	`Time in refTime,`
	`KeyPurpose in intendedKeyUsage,`
	`PolicyList in initialPolicySet,`
	`CertInfoList inout trustedCerts,`
	`CrlInfoList inout trustedCrls )`
	`{`
	`int n = tbvCertPath.size();`
		This function performs certificate path validation and is compatible, with minor changes, with the BPVA of RFC2459. *tbvCertPath* 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' EE certificate
		The other function parameters have the same meaning and constraints as those of *BuildAndValidatePath*() in Table 3. *tbvCerts* may contain further certificates that may be useful in validating ternary signed objects, like CRLs.
		This function returns *true* if the certificate can be successfully validated, including mathematical verification and checking constraints; respectively *false,* if mathematical verification failed or some constraint cannot be met.		

2	`GeneralNames permittedSubtrees;`	*permittedSubtrees* (called '*constrained subtrees*' in BPVA) contains a set of root names defining a set of sub-trees within which all subject names in subsequent certificates in the certification path shall fall. Initially, the list is empty, which indicates that no constraints apply on subject names. (Compared to BPVA, no dedicated '*unbounded*' value is used here.) **ISIS-MTT PROFILE:** only the following name forms, corresponding to technical address spaces, are supported: - Distinguished name (represented by a *directoryName*). - URI - Email address (represented by an *rfc822Name*) - DNS name - IP address For the syntax and semantics of name values refer to Section 4.2.1.11 of [RFC2459].	state var. (b)	
3	`GeneralNames excludedSubtrees;`	*excludedSubtrees* contains a set of root names defining a set of sub-trees within which no subject name in subsequent certificates in the certification path may fall. Initially, the list is empty. **ISIS-MTT PROFILE:** only the name forms listed under #2 need to be supported. For the syntax and semantics of name values refer to Section 4.2.1.11 of [RFC2459].	state var. (c)	
4	`PolicyList acceptablePolicySet;`	A set of certificate policy identifiers comprising the policies recognized by the public key user together with policies deemed equivalent through policy mapping. Initially, *acceptablePolicySet* is empty, indicating that any policy is acceptable. (Compared to BPVA, no dedicated *any-policy* identifier used here.)	state var. (a)	
5	`int explicitPolicy = n+1;`	*explicitPolicy* is an integer which indicates if an explicit policy identifier is required. The integer indicates the last certificate in the path before an explicit policy is required. Once set, this variable may be decreased, but may not be increased. (That is, if a certificate in the path requires explicit policy identifiers, a later certificate can not remove this requirement.) The initial value is n+1. **ISIS-MTT PROFILE:** Departing from BPVA, the integer indicates the last certificate in the path before an explicit policy is required. (In BPVA the integer indicates the first certificate in the path where an explicit policy identifier is required. However, we suspect a logical error in the further usage under #20 and #24, which can be corrected by proposed slight modification of the semantics.	state var. (d)	
6	`int policyMapping = n+1;`	*policyMapping* is an integer which indicates if policy mapping is permitted. The integer indicates the last certificate on which policy mapping may be applied. Once set, this variable may be decreased, but may not be increased. (That is, if a certificate in the path specifies policy mapping is not permitted, it can not be overridden by a later certificate.) The initial value is n+1.	state var. (e)	

7	`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.		
8	`Certificate &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.		

9	`Certificate &issCert;`
	`if( i>1 )`
	`issCert = certPath.GetItem(i-1);`
	`else`
	`issCert = certPath.GetItem(i);`
		issCert is just a reference (or alias) to the *i-1*th item of the path, which is a certificate of the issuer of *tbvCert*. It can be a CA-, a root-CA- or a cross-certificate.		

10	`if( i>1 )`
	`{`
	`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. The signature over the self-signed root certificate does not contribute to security. Hence, verifying this signature is omitted.	step (a)(1)	

11	`if( (refTime < tbvCert.GetValidityNotBefore()) or`
	`(refTime > tbvCert.GetValidityNotAfter()) )`
	`return false;`
		Check whether *refTime* lies within the validity period.	step (a)(2)	

12	`if( CheckRevocationStatus( tbvCert,`
	`tbvCerts,`
	`refTime,`
	`initialPolicySet,`
	`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.	step (a)(3)	

13	`if( tbvCert.GetIssuer() != issCert.GetSubject() )` `return false;`	Verify that certificates correctly chain, i.e. that the issuer of *tbvCert* is the subject of *issCert*.	step (a)(4)	

14	`if( i<n )`
	`{`
	`if( tbvCert.IsCaCertificate()==false )`
	`return false;`
	`if( tbvCert.KeyUsageExtIsCritical() )`
	`{`
	`if( tbvCert.GetKeyCertSignKeyUsageBit() != true )`
	`return false;`
	`}`
	`}`
		All certificates of the path, where *i<n*, are 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.
		If *KeyUsage* extension is present and marked critical, ensure the *keyCertSign* bit is set.
		ISIS-MTT PROFILE: Note that the *KeyUsage* extension MUST be present and MUST be marked critical (P1.T12.[1]).
			steps (i) & (m)	

15	`if( tbvCert.SubjectAltNamesContainsDirectoryName()==false )`
	`{`
	`  if( SubtreesContainDName( permittedSubtrees,`
	`  tbvCert.GetSubject() )==false )`
	`  return false;`
	`}`
	`if( SubtreesContainAllGeneralNames( permittedSubtrees,`
	`  tbvCert.GetSubjectAltNames())==false )`
	`  return false;`
		Verify that the subject name or the *subjectAltNames* extension (critical or non-critical) is consistent with the *permittedSubtrees* variable.
		ISIS-MTT PROFILE: Restrictions apply only when the specified name form is present in *tbvCert*. If no name of the type is in the certificate, the certificate is acceptable.
		Only name forms listed under #2 need to be supported according to the rules in Section 4.2.1.11 of [RFC2459]. If a *directoryName* is defined in the *SubjectAltNames* extension, this name shall be checked against *permittedSubtrees*, where the subject DName should may be ignored. On the other hand, if no *directoryName* option is defined in *SubjectAltNames*, the subject DName is to be checked against *permittedSubtrees*. URI name forms shall be checked with no regard to the transport protocol (HTTP, FTP etc.). In particular, all name forms, except *directoryName*, shall be compared case-ignorant.
			step (b)	

16	`if( tbvCert.SubjectAltNamesContainsDirectoryName()==false )`
	`{`
	`  if( SubtreesContainDName( excludedSubtrees,`
	`  tbvCert.GetSubject() )==true )`
	`  return false;`
	`}`
	`if( SubtreesContainAnyOfGeneralNames( excludedSubtrees,`
	`  tbvCert.GetSubjectAltNames())==true )`
	`  return false;`
		Verify that the subject name or the *subjectAltNames* extension (critical or non-critical) is consistent with the *excludedSubtrees* variable.
		ISIS-MTT PROFILE: the same remarks apply for *excludedSubtrees* as for *permittedSubtrees* above.
			step (c)	

17	`if( tbvCert.NameConstraintsIsPresent() )`
	`{`
	`  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.
			step (j)	

18	`PolicyList explicitPolicies = tbvCert.GetCertPolicyOIDs();`
	`if( explicitPolicy > i )`
	`{`
	`  if( IntersectionIsEmpty( explicitPolicies,`
	`  initialPolicySet )`
	`  return false;`
	`}`
		Verify that policy information is consistent with *initialPolicySet*:
		if *explicitPolicy>i*, an explicit policy identifier in *tbvCert* MUST be present in *initialPolicySet*.
		ISIS-MTT PROFILE: The condition *explicitPolicy<=i* is supposed to be wrong in BPVA, as *explicitPolicy* contains the index of the last certificate in the path before an explicit policy is required. According to this and the slightly modified semantics of #5, the condition is corrected here to *explicitPolicy>i.*
		GetCertPolicyOIDs() MUST consider policy OIDs not only in the *CertificatePolicies* extension, but in *QualifiedCertificateStatements* as well.
			step (d)(1)	

19	`if( policyMapping > i )`
	`{`
	`  PolicyList mappedPolicies =`
	`  tbvCert.GetCertMappedPolicyOIDs( acceptablePolicySet );`
	`  if( mappedPolicies.IsEmpty()==false )`
	`  return false;`
	`}`
		If the *policyMapping>i*, no policy identifier may be mapped.
		ISIS-MTT PROFILE: The condition *policyMapping <=i* is supposed to be wrong in BPVA, as *policyMapping* contains the index of the last certificate of the path where mapping is prohibited. Accordingly, the condition is corrected here to *policyMapping>i.*
			step (d)(2)	

20	`if( tbvCert.CertPoliciesExtIsCritical() )`
	`{`
	`  if( IntersectionIsEmpty( explicitPolicies,`
	`  acceptablePolicySet )`
	`  return false;`
	`}`
	`acceptablePolicySet = Intersection (`
	`  explicitPolicies,`
	`  acceptablePolicySet );`
		Verify that policy information is consistent with the *acceptablePolicySet*:
		(1) if the certificate policies extension is present and marked critical, the intersection of the policies extension and the acceptable policy set shall be non-null;
		(2) the acceptable policy set is assigned the resulting intersection as its new value.
			step (e)(1) & (e)(2)	

21	`PolicyList mappedPolicies =`
	`  tbvCert.GetCertMappedPolicyOIDs( acceptablePolicySet );`
	`acceptablePolicySet = Union(`
	`  mappedPolicies,`
	`  acceptablePolicySet );`
		ISIS-MTT PROFILE: add to *acceptablePolicies* all policy identifiers deemed equivalent with any acceptable policy.
		This step is missing from BPVA (should this have been the missing step (f) ?) and is especially relevant for cross-certification.
			–	

22	`if( IntersectionIsEmpty( initialPolicySet,`
	`  acceptablePolicySet ) )`
	`  return false;`
		Verify that the intersection of the *acceptablePolicySet* and the *initialPolicySet* is non-null.	step (g)	

23	`if( tbvCert.PolicyConstraintsIsPresent() )`
	`{`
	`  if( tbvCert.requireExplicitPolicyIsPresent() )`
	`  {`
	`  int r = tbvCert.GetRequireExplicitPolicy();`
	`  explicitPolicy = min( r+i, explicitPolicy );`
	`  }`
	`  if( tbvCert.inhibitPolicyMapping() )`
	`  {`
	`  int q = tbvCert.GetInhibitPolicyMapping();`
	`  policyMapping = min( q+i, policyMapping );`
	`  }`
	`}`
		If a policy constraints extension is included in the certificate, modify the *explicitPolicy* and *policyMapping* state variables as follows:
		(1) If *requireExplicitPolicy* is present and has value *r*, the *explicitPolicy* state variable is set to the minimum of its current value and the sum of *r* and *i*.
		(2) If *inhibitPolicyMapping* is present and has value *q*, the *policyMapping* state variable is set to the minimum of its current value and the sum of *q* and *i*.
		Note that the *PolicyConstraints* extension may only occur in CA certificates.
			step (l)(1) & (l)(2)	

24	`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.
		ISIS-MTT PROFILE: Departing from the BPVA, this step is performed for the EE certificate too.
			step (h)	

25	`}`	End of for-cycle on code line #6.		
26	`return true;` `}`	All checks 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,`
	`PolicyList in initialPolicySet,`
	`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,`
	`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 furthermore be advantageous, to check first for an appropriate, locally available CRL, before using an on-line service.	

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

6	else
	  return false;
	}
		Status could not be checked, because no directory access information was available.	

Table 6: CheckStatusUsingCRL()

#	PSEUDO-CODE	COMMENTS	NOTES

1	bool CheckStatusUsingCRL(CertInfo inout tbvCert,
	  CertInfo in issuerCert,
	  CertInfoList in tbvCerts,
	  Time in refTime,
	  PolicyList in initialPolicySet,
	  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. **ISIS-MTT 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.	

4	`Name crlIssuerDName;`
	`if( crlIsIndirect )`
	`  crlIssuerDName = cdp.crlIssuer.GetDirectoryName();`
	`else`
	`  crlIssuerDName = tbvCert.GetIssuerDName();`
		The DName of the CRL-issuer is determined.
		ISIS-MTT 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.
		[RFC3280]: Note that the 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. **ISIS-MTT 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, initialPolicySet,`
	`  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 V*alidateCertificate*(). 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, initialPolicySet,`
	`  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;`
		In this step, URLs will be determined for directory access.
		#ISIS-MTT 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]).	

	`if( crlIsIndirect )`
	`{`
	`  if(RequestCertsViaLdap(crlIssCertUrl,downloadedCerts)==false)`
	`  return false;`
	`}`
	`else`
	`{`
	`  if( RequestCaAndCrossCertsViaLdap(crlIssCertUrl,`
	`  downloadedCerts)==false )`
	`  return false;`
	`}`
	`}`
	`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, initialPolicySet,`
	`  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,`
	`PolicyList in initialPolicySet,`
	`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).
		ISIS-MTT PROFILE: Note that *certID* (P4.T6.#4) MUST be build using SHA1 and respectively the OID '*sha_1*'.	

3	`Certificate respCert = response.GetResponderCert();`
	`if( VerifySignature( response.GetToBeSignedData(),`
	`responderCert.GetPublicKeyInfo())==false)`
	`return false;`
	`if( ValidateCertificate( respCert,`
	`response.RetrieveCerts(),`
	`response.GetProducedAtTime();`
	`ocspSigning,`
	`initialPolicySet,`
	`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.
		ISIS-MTT PROFILE: Note that ISIS-MTT 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( AnyOfPoliciesEnforcesPositiveStatement( initialPolicySet ) )`
	`{`
	`  if( singleResp.CertHashIsPresent()==false )`
	`  return false;`
	`  CertHash certHash = singleResp.GetCertHash();`
	`  if( Hash( tbvCert.DerEncode(), certHash.hashAlgorithm ) !=`
	`  certHash.certificateHash )`
	`  return false;`
	`}`
		Some policies may demand that the responder delivers an evidence that the certificate has 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*.
		ISIS-MTT PROFILE: Note that ISIS-MTT conforming responders may return status 'good' only if they possess definite knowledge about the requested certificate's status.	

References

[RFC2255] Network Working Group - The LDAP URL Format, RFC 2255, December 1997

[RFC2459] Internet X.509 Public Key Infrastructure - Certificate and CRL Profile, RFC 2459, January 1999

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

[RFC2587] Network Working Group – PKIX LDAPv2 Schema, RFC 2587, June 1999

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 6

CRYPTOGRAPHIC ALGORITHMS

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

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.

Table of Contents

1 Preface 5

2 Algorithm Support 6

2.1 One-Way Hash Functions 6

2.2 Signature Algorithms 6

2.3 Content Encryption Algorithms 7

2.4 Symmetric Key Wrap 7

2.5 Key Encryption Algorithms 7

2.6 Key AgreementAlgorithms 7

2.7 Subject Public Key Algorithms 7

2.8 Message Authentication Algorithms 7

References 18

1 Preface

This part of the ISIS-MTT 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 ISIS-MTT specification.

It is mainly based on the PKIX documents [RFC2459] and [RFC3279], the W3C documents [XML_SIG] and [XML_ENC], and the OSCI profile [OSCI]. It contains all supplementary specifications, recommendations and restrictions the ISIS-MTT document has defined in addition to the corresponding base documents.

The S/MIME standard version 3 documents [RFC 2632] and [RFC 2633] 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 ISIS-MTT 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 ISIS-MTT specification.

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

- the PKIX documents [RFC 3280] 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).

ISIS-MTT 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.

ISIS-MTT 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.

ISIS-MTT compliant components SHALL satisfy the conformance requirements for data encryption algorithms as specified in Table 3.

2.4 Symmetric Key Wrap

ISIS-MTT 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.

ISIS-MTT 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 ISIS-MTT for components that support XML.

ISIS-MTT 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

ISIS-MTT 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.

ISIS-MTT 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			ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	SHA-1	one-way hash function	[RFC3279] [RFC2633] [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	[XML_ENC] [FIPS-180-2]		n. a.	+-	+	http://www.w3.org/2001/04/xmlenc#sha256	[3, 4]
3	SHA-512	one-way hash function	[XML_ENC] [FIPS-180-2]		n. a.	+-	+	http://www.w3.org/2001/04/xmlenc#sha512	[3,4]
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	[RFC3279] [RFC 1319]	2.1.1	-	--	--	OID: 1.2.840.113549.2.2	
6	MD5	one-way hash function	[RFC3279] [RFC 2633] [RFC1321]	2.1.2 2.1	-	--	+-	OID: 1.2.840.113549.2.5	[6]

[1] 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 180-1] and [ISO/IEC 10118-3].

[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 and SHA-512 are referenced in XML_ENC, but not in XML_DSIG.

[5] **ISIS-MTT PROFILE:** The support of the RIPEMD-160 hash function on the processing side is recommended. This algorithm is published in [RegTP 2003] 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.

[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			ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
	sha-1WithRSAEncryption	RSA signature algorithm	[RFC3279] [RFC 2633] [FIPS 180-1] [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] [7]
	rsaSignatureWithripemd160	RSA signature algorithm	[RIPEMD-160] [ISO/IEC 10118-3] [OSCI]		n. a.	-	+	OID: 1.3.36.3.3.1.2 http://www.w3.org/2001/04/xmlenc#ripemd160	[2,3,6] [7]
	md2-WithRSAEncryption	RSA signature algorithm	[RFC3279] [RFC 2633]	2.2.1 2.2	+-	--	--	OID: 1.2.840.113549.1.1.2	[1,3,4,6]
	md5WithRSAEncryption	RSA signature algorithm	[RFC3279] [RFC 2633]	2.2.1 2.2	+-	--	+-	OID: 1.2.840.113549.1.1.4	[1,3,4,6]
	dsa-with-sha1	DSA signature algorithm	[RFC3279] [RFC2633] [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]
	ecdsa-with-SHA1	ECDSA signature algorithm	[RFC3279] [X9.62]	2.2.3	+-	+-	+-	OID: 1.2.840.10045.4.1	

[1]	The PKIX documents do not make any recommendation which of the RSA signature algorithms (md2withRSAEncryption, md5withRSAEncryption, sha-1WithRSAEncryption) should be preferred. **ISIS-MTT PROFILE:** sha-1WithRSAEncryption is the preferred signature algorithm.
[2]	**ISIS-MTT PROFILE:** The support of the RIPEMD-160 hash function on the processing side is recommended. This algorithm is published in [RegTP 2003] 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 padding and encoding conventions described in PKSC#1 [RFC2437]. The parameter component of this algorithm identifier shall be the ASN.1 type NULL.
[4]	S/MIMEv3 requires that receiving agents should be capable of verifying signatures on certificates and CRLs made with md2withRSAEncryption, md5withRSAEncryption, and sha-1WithRSAEncryption with key sizes from 512 bits to 2048 bits.
[5]	S/MIMEv3 requires that sending and receiving agents MUST support id-dsa. Receiving agents must be capable of verifying signatures on certificates and CRLs made with id-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 2630] 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].

Table 3: Content Encryption Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS	DOCUMENT	CHAPTER	STATUS	GEN	PROC	VALUES	
1	des-cbc	content encryption algorithm	[RFC2633]	2.7	++	++	++	OID: 1.3.14.3.2.7	[1], [6]
2	des-ede3-cbc	content encryption algorithm	[RFC2633, X9.52] [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	[RFC2633]	2.7	+-	--	+-	OID: 1.2.840.113549.3.2	[4]
5	aes128-cbc aes192-cbc aes256-cbc	content encryption algorithm	[FIPS-197] [SMIME-AES] [XML_ENC]	5	+-	+-	+-	OID: 2.16.840.1.101.3.4.1.2 http://www.w3.org/2001/04/xmlenc#aes128-cbc OID: 2.16.840.1.101.3.4.1.22 http://www.w3.org/2001/04/xmlenc#aes192-cbc OID: 2.16.840.1.101.3.4.1.42 http://www.w3.org/2001/04/xmlenc#aes256-cbc	[5] [7] [7] [7]

[1]	The DES algorithm is defined in [FIPS 46-2]; 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]. S/MIME v3 requires that DES MUST be supported by sending and receiving agents.
[2]	S/MIMEv3 [RFC2633] requires that sending and receiving agents MUST support encryption and decryption with dES-EDE3-CBC [X9.52] in 3-key mode of operation.
[3]	Triple-DES is standardized in [X9.17] in 2-key mode of operation. **ISIS-MTT 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 [RFC2633] requires that receiving agents SHOULD support encryption and decryption using the RC2 [RFC2268] 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 [SMIME-AES].
[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			ISIS-MTT 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			ISIS-MTT 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]		+	++	++	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 [RFC2633] requires that sending and receiving agents MUST support Diffie-Hellman defined in [RFC 2631], and SHOULD support rsaEncryption.
ISIS-MTT PROFILE: Diffie-Hellman key agreement is only considered in ISIS-MTT for components that support XML.
RSAES-PKCS1-v1_5 is included in [PKCS#1] only for compatibility with existing applications, and is not recommended for new applications.

[2] [PKCS#1] recommends the use of RSAES-OAEP for new applications, e.g. for wrapping of AES content encryption keys.
ISIS-MTT 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 2630] and its successor [RFC 3370]. Therefore it is OPTIONAL.

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

Table 6: Key Agreement Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			ISIS-MTT 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] This is only a requirement for compliant components that support XML. Diffie-Hellman key agreement is only considered in ISIS-MTT for components that support XML.									

Table 7: Subject Public Key Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			ISIS-MTT 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] **ISIS-MTT PROFILE:** Diffie-Hellman key agreement [X9.42] is not considered in ISIS-MTT.

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

[3] **ISIS-MTT PROFILE:** KEA key exchange is not considered in ISIS-MTT.

[4] This OID is used in public key certificates for both ECDSA signature keys and ECDH encryption keys.
ISIS-MTT 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 ISIS-MTT

Table 8: Message Authentication Algorithms

CRYPTOGRAPHIC ALGORITHMS			REFERENCES			ISIS-MTT 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] [XML_DSIG]		+-	+	+	OID: 1.3.6.1.5.5.8.1.2 http://www.w3.org/2000/09/xmldsig#hmac-sha1	[2] [3,4]

[1] The DES-MAC uses DES as defined in [FIPS 46-2] and data authentication as defined in [X9.9] (binary option) and [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 ISIS-MTT and thus HMAC may return. So leaving it here will perhaps then make things a little easier.

References

[FIPS 113]	Federal Information Processing Standards (FIPS PUB) 113: Computer Data Authentication; May 1995
[FIPS 180-1]	Federal Information Processing Standards (FIPS PUB) 180-1: Secure Hash Standard; April 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 197]	Federal Information Processing Standards (FIPS PUB) 197: Advanced Encryption Standard; November 2001
[FIPS 46-2]	National Institute of Standards and Technology (formerly National Bureau of Standards): Data Encryption Standard. December 30, 1993
[FIPS 81]	National Institute of Standards and Technology (formerly National Bureau of Standards): DES Modes of Operation. December 1980
[ISO/IEC 10118-3]	International Organization for Standardization: IT-Security Techniques Hash Functions Part 3: Dedicated Hash-Functions, 1998
[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#1.5]	RSA Laboratories, “PKCS #1 v1.5: Password-Based Encryption Standard”, November 1993
[PKCS#1]	RSA Laboratories, “PKCS #1 v2.0: RSA Cryptography Standard”, October 1998
[PKCS#5]	RSA Laboratories, “PKCS #1 v1.5: Password-Based Encryption Standard,” November 1993
[RegTP 2003]	Regulatory authority for telecommunications and post: Declaration on appropriate algorithms for electronic signatures (Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung), published in German federal law gazette (Bundesanzeiger) from 14.02.9811. March 2003
[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 2437]	B. Kaliski, J. Staddon: PKCS #1: RSA Cryptography Specifications, Version 2.0, October 1998
[RFC 2630]	B. Ramsdell: Cryptographic Message Syntax, June 1999

[RFC 2631]	E. Rescorla, „Diffie-Hellman Key Agreement Method“, RFC2631, 1999
[RFC 2631]	Rescorla, E., "Diffie-Hellman Key Agreement Method", June 1999
[RFC 2632]	B. Ramsdell: S/MIME Version 3 Certificate Handling, June 1999
[RFC 2633]	B. Ramsdell: S/MIME Version 3 Message Specification, June 1999
[RFC 3279]	Bassham L., Hously R., Polk W.: Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and CRL Profile, <draft-ietf-pkix-ipki-pkalgs.05.txt>, October 2001
[RFC 3280]	Housley, R., Ford, W., Polk, W. and D. Solo: Internet X.509 Public Key Infrastructure – Certificate and CRL Profile, April 2002
[RFC 3370]	R. Housley: Cryptographic Message Syntax (CMS) Algorithms, August 2002
[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 on the Conditions for Electronic Signatures (Gesetz über Rahmenbedingungen für elektronische Signaturen und zur Änderung weiterer Vorschriften), Bundesgesetzblatt No. 22, 2001, p. 876
[SMIME-AES]	J. Schaad, and R. Housley: Use of the AES Encryption Algorithm and RSA-OAEP Key Transport in CMS; <draft-ietf-smime-aes-alg-04.txt>; January 2002
[X9.17]	American National Standard X9.17: Financial Institution Key Management (Wholesale). 1985
[X9.42]	"Public Key Cryptography For The Financial Services Industry: Agreement of Symmetric Keys Using Discrete Logarithm Cryptography", December 1999
[X9.52]	American National Standard X9.52: Triple Data Encryption Algorithm Modes of Operation. 1998
[X9.52]	Triple Data Encryption Algorithm Modes of Operation, ANSI X9.52, 1998
[X9.62]	"Public Key Cryptography For The Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", January 1998
[X9.9]	American National Standard X9.9: Financial Institution Message Authentication Code. 1982
[XML_DSIG]	W3C: XML-Signature Syntax and Processing, 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
[XML_ENC]	W3C: „XML Encryption Syntax and Processing“, 10 December 2002, http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 7

CRYPTOGRAPHIC TOKEN INTERFACE

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

Jürgen Brauckmann

Alfred Giessler

Tamás Horváth

Hans-Joachim Knobloch

Document History

VERSION DATE	CHANGES
1.0.1 November 15th 2001	First public edition
1.0.2 July 19th 2002	Editorial and stylistic changes, 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) A new paragraph has been added after the first paragraph in chapter 4 indicating that the set of functions to be supported depends on the certification policies of CAs and the token personalization status.

Table of Contents

1 Preface 6

2 General Data Types 7

2.1 General Information 7

2.2 Token and Slots 7

2.3 Sessions 8

2.4 Objects 9

2.5 Mechanisms 10

2.6 Functions 11

2.7 Locking 11

3 Objects 12

3.1 Creation 12

3.2 Modification 12

3.3 Copying 12

3.4 Common Attributes 12

3.5 Data Objects 12

3.6 Certificate Objects 13

3.7 Key Objects 13

3.8 Public Key Objects 13

3.9 Private Keys 13

3.10 Secret Keys 14

4 Functions 15

4.1 General Purpose 15

4.2 Slot and Token Management 15

4.3 Session Management 15

4.4 Object Management 16

4.5 Encryption 16

4.6 Decryption 16

4.7 Message Digesting 16

4.8 Signing and MACing 16

4.9 Verification of Signatures and MACs 16

4.10 Multiple Cryptographic Operations 17

4.11 Key Management 17

4.12 Random Number Generation 17

4.13 Parallel Function Management 17

5 Mechanisms 18

6 PKCS#11 Definitions 20

References 27

1 Preface

This part of the ISIS-MTT specification lists conformance requirements for components that support a cryptographic token interface (Cryptoki), which is mainly based on the RSA document “Cryptographic Token Interface Standard” [PKCS#11].

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

Compliant components are not required to provide simultaneous access to several tokens [PKCS#11, 5.2].

ISIS-MTT does not consider multi-threading [PKCS#11, 5.2.2].

This document contains the following chapters:

- Chapter 2 specifies requirements for general data formats.
- Chapter 3 contains requirements for objects.
- Chapter 4 lists requirements for functions.
- Chapter 5 contains requirements for mechanisms.
- Chapter 6 provides PKCS#11 type definitions in alphabetic order.
- Chapter 7 gives references to the standards on which this part of ISIS-MTT is based.

2 General Data Types

2.1 General Information

Compliant components SHALL support the following data types for general information as listed in the following table.

Table 1: Data Types for General Information

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUP-PORT	VALUES	NOTES
1	*CK_VERSION*	Version of Cryptoki	8.1	++	version 2.1	
2	*CK_VERSION_PTR*	Pointer to *CK_VERSION*	8.1	++		
3	*CK_INFO*	General information about Cryptoki	8.1	++		
4	*CK_INFO_PTR*	Pointer to CK_INFO	8.1	++		
5	*CK_NOTIFICATION*	Cryptoki notification to an application	8.1	+-		

2.2 Token and Slots

Compliant components SHALL support the following data types for token and slots as listed in the following table.

Table 2: Data Types for Token and Slots

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUPPORT	VALUES	NOTES
1	*CK_SLOT_ID*	Slot identification	8.2	++		
2	*CK_SLOT_ID_PTR*	Pointer to *CK_SLOT_ID*	8.2	++		
3	*CK_SLOT_INFO*	Information about Slot	8.2	++		
4	*CK_SLOT_INFO_PTR*	Pointer to *CK_SLOT_INFO*	8.2	++		
5	*CK_TOKEN_INFO*	Information about Token	8.2	++		
6	*CK_TOKEN_INFO_PTR*	Pointer to *CK_TOKEN_INFO*	8.2	++		

2.3 Sessions

Compliant components SHALL support the following data types for sessions as listed in the following table.

Table 3: Data Types for Sessions

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUPPORT	VALUES	NOTES
1	*CK_SESSION_HANDLE*	Session identification	8.3	++		
2	*CK_SESSION_HANDLE_PTR*	Pointer to *CK_SESSION_HANDLE*	8.3	++		
3	*CK_USER_TYPE*	Identification of user type	8.3	++		
4	*CK_STATE*	Status information about session	8.3	++		
5	*CK_SESSION_INFO*	Information about session	8.3	++		
6	*CK_SESSION_INFO_PTR*	Pointer to *CK_SESSION_INFO*	8.3	++		

2.4 Objects

Compliant components SHALL support the following data types for objects as listed in the following table.

Table 4: Data Types for Objects

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUPPORT	VALUES	NOTES
1	*CK_OBJECT_HANDLE*	Object identification	8.4	++		
2	*CK_OBJECT_HANDLE_PTR*	Pointer to *CK_OBJECT_HANDLE*	8.4	++		
3	*CK_OBJECT_CLASS*	Identification of object class	8.4	++		[1]
4	*CK_OBJECT_CLASS_PTR*	Pointer to *CK_OBJECT_CLASS*	8.4	++		
5	*CK_KEY_TYPE*	Identification of key type	8.4	++		[2]
6	*CK_CERTIFICATE_TYPE*	Identification of certificate type	8.4	++		[3]
7	*CK_ATTRIBUTE_TYPE*	Identification of attribute type	8.4	++		[4]
8	*CK_ATTRIBUTE*	Structure that contains the type, length and value of an attribute	8.4	++		
9	*CK_ATTRIBUTE_PTR*	Pointer to *CK_ATTRIBUTE*	8.4	++		
10	*CK_DATE*	Structure that defines a date	8.4	++		

[1] The following object classes SHALL be supported: CKO_DATA, CKO_CERTIFICATE, CKO_PUBLIC_KEY, CKO_PRIVATE_KEY, and CKO_SECRET_KEY

[2] The following key types SHALL be supported: CKK_RSA, CKK_DES, and CKK_DES2. The support for CKK_IDEA is recommended.

[3] The certificate type CKC_X_509 SHALL be supported.

[4] The attribute types to be supported are listed in section 3.4..

2.5 Mechanisms

Compliant components SHALL support the following data types for mechanisms as listed in the following table.

Table 5: Data Types for Mechanisms

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUPPORT	VALUES	NOTES
1	*CK_MECHANISM_TYPE*	Identification of mechanism	8.5	++		[1]
2	*CK_MECHANISM_TYPE_PTR*	Pointer to *CK_MECHANISM_TYPE*	8.5	++		
3	*CK_MECHANISM*	Structure that contains the type a mechanism and its parameters	8.5	++		
4	*CK_MECHANISM_PTR*	Pointer to *CK_MECHANISM*	8.5	++		
5	*CK_MECHANISM_INFO*	Structure that contains the information about a mechanism	8.5	++		[2]
6	*CK_MECHANISM_INFO_PTR*	Identification of certificate type	8.5	++		

[1] The following mechanism types SHALL be supported:
CKM_RSA_PKCS_KEY_PAIR_GEN, CKM_RSA_PKCS, CKM_SHA1_RSA_PKCS, CKM_DES_KEY_GEN, CKM_DES_MAC, CKM_DES_CBC_PAD, CKM_DES3_KEY_GEN, CKM_DES3_CBC_PAD, and CKM_SHA_1.
The support of CKM_RSA_9796, CKM_DES3_MAC, CKM_SHA_1_HMAC, CKM_IDEA_KEY_GEN, and CKM_IDEA_CBC is recommended.

[2] Information about a mechanism includes minimum and maximum key length, and the properties of the mechanism.

2.6 Functions

Compliant components SHALL support the following data types for functions as listed in the following table.

Table 6: Data Types for Functions

TYPES			REFS TO PKCS #11	ISIS-MTT		
#	NAME	SEMANTICS		SUPPORT	VALUES	NOTES
1	*CK_RV*	Return value of Cryptoki function	8.6	++		
2	*CK_NOTIFY*	Pointer to a function to perform notification callbacks	8.6	n.a.		
3	*CK_C_XXX*	Pointer to function C_XXX	8.6	++		
4	*CK_FUNCTION_LIST*	Structure that contains the Cryptoki version and a pointer to each function	8.6	++		[1]
5	*CK_FUNCTION_LIST_PTR*	Pointer to *CK_FUNCTION_LIST*	8.6	++		
6	*CK_FUNCTION_LIST_PTR_PTR*	Pointer to *CK_FUNCTION_LIST_PTR*	8.6	++		

[1] The list of all Cryptoki functions can be obtained by calling the function C_GETFunctionList. Compliant components SHALL provide a stub for Cryptoki functions that are not supported in the structure CK_FUNCTION_LIST.

2.7 Locking

Compliant components are not required to support locking-related types, since simultaneous access via multithreading is not mandated.

3 Objects

3.1 Creation

Compliant components SHALL support the functions *C_CreateObject*, *C_ GenerateObject*, *C_GenerateKeyPair*, and *C_UnwrapKey* for the creation of objects. They are not required to support the function *C_DeriveKey*, since derived keys are not considered.

Compliant components SHALL cope with the error cases as described in [PKCS#11, 9.1.1] and shall support the error codes *CKR_ATTRIBUTE_TYPE_INVALID*, *CKR_ATTRIBUTE_VALUE_INVALID, CKR_TEMPLATE_INCOMPLETE*, *CKR_TEMPLATE_INCONSISTENT*, and *CKR_ATTRIBUTE_READ_ONLY*. However, templates that specify a value for a read-only attributes SHALL NOT be rejected with the return code *CKR_ ATTRIBUTE_READ_ONLY*, if the attribute has been defined as modifiable.

3.2 Modification

Compliant components SHALL support the function *C_SetAttributeValue* for the modification of objects as described in [PKCS#11, 9.1.2].

Compliant components SHALL cope with the error cases as described in [PKCS#11, 9.1.2] and shall support all error codes listed in the previous section with the exception of *CKR_ TEMPLATE_INCONSISTENT*.

3.3 Copying

Compliant components SHALL support the function *C_CopyObject* for the copying of objects as described in [PKCS#11, 9.1.3].

Compliant components SHALL cope with the error cases as described in [PKCS#11, 9.1.2] and shall support all error codes listed in the previous section with the exception of *CKR_ TEMPLATE_INCONSISTENT*.

3.4 Common Attributes

Compliant components SHALL support the attributes *CKA_CLASS, CKA_TOKEN, CKA_ PRIVATE, CKA_MODIFIABLE,* and *CKA_LABEL* that are common to all objects as described in [PKCS#11, 9.2].

3.5 Data Objects

Compliant components MAY optionally support the attributes *CKA_APPLICATION*, and *CKA_VALUE* for data objects that hold information defined by an application as described in [PKCS#11, 9.3].

3.6 Certificate Objects

Compliant components SHALL support the common certificate object attribute *CKA_ CERTIFICATE_TYPE*, and the special X.509 certificate object attributes *CKA_SUBJECT*, *CKA_ ID*, *CKA_ISSUER*, *CKA_SERIAL_NUMBER*, and *CKA_VALUE* as described in [PKCS#11, 9.4].

3.7 Key Objects

Compliant components SHALL support the common key attributes *CKA_ KEY_TYPE CKA_ ID*, *CKA_ START_DATE*, *CKA_END_DATE*, and *CKA_LOCAL* as described in [PKCS#11, 9.5].

Compliant components are not required to support the attribute *CKA_DERIVE*, since derived keys are not considered. Its value shall always be *FALSE*.

3.8 Public Key Objects

Compliant components SHALL support the common public key attributes *CKA_ SUBJECT*, *CKA_ ENCRYPT*, *CKA_ VERIFY*, and *CKA_WRAP* as described in [PKCS#11, 9.6].

Compliant components are not required to support the attribute *CKA_VERIFY_RECOVER* within the object class *CKO_PUBLIC_KEY*, since the related mechanism is not considered in ISIS-MTT. The value of this attribute shall always be *FALSE*.

Compliant components SHALL only support the public key type *CKK_RSA* within the object class *CKO_PUBLIC_KEY*.

Compliant components SHALL support the RSA public key attributes *CKA_ MODULUS*, *CKA_ MODULUS_BITS*, and *CKA_PUBLIC_EXPONENT* as described in [PKCS#11, 9.6.1].

3.9 Private Keys

Compliant components SHALL support the common private key attributes *CKA_ SUBJECT*, *CKA_ SENSITIVE*, *CKA_ DECRYPT*, *CKA_ SIGN*, *CKA_ UNWRAP*, *CKA_EXTRACTABLE*, *CKA_ ALWAYS_SENSITIVE*, and *CKA_NEVER_EXTRACTABLE* as described in [PKCS#11, 9.7].

Compliant components are not required to support the attribute *CKA_SIGN_RECOVER* within the object class *CKO_PRIVATE_KEY*, since the related mechanism is not considered. Its value SHALL always be FALSE.

The values for the attributes *CKA_SENSITIVE*, *CKA_DECRYPT*, *CKA_EXTRACTABLE*, *CKA_ALWAYS_SENSITIVE*, and *CKA_NEVER_EXTRACTABLE* SHALL also always be set

to *TRUE*.

Compliant components SHALL only support the key type *CKK_RSA* within the object class *CKO_PRIVATE_KEY*.

Compliant components SHALL support the RSA private key attributes *CKA_ MODULUS*, *CKA_PUBLIC_EXPONENT*, *CKA_ PRIVATE_EXPONENT*, *CKA_ PRIME_1*, *CKA_ PRIME_2*, *CKA_ EXPONENT_1*, *CKA_ EXPONENT_2*, and *CKA_COEFFICIENT* as described in [PKCS#11, 9.7.1].

3.10 Secret Keys

Compliant components SHALL support the common secret key attributes *CKA_SENSITIVE*, *CKA_ENCRYPT*, *CKA_ DECRYPT*, *CKA_ SIGN*, *CKA_VERIFY*, *CKA_WRAP*, *CKA_UNWRAP*, *CKA_EXTRACTABLE*, *CKA_ALWAYS_SENSITIVE*, and *CKA_NEVER_ EXTRACTABLE* within the object class *CKO_SECRET_KEY* as described in [PKCS#11, 9.8].

The value for the attribute *CKA_EXTRACTABLE* SHALL also always be set to *TRUE*. The values for the attributes *CKA_SENSITIVE*, *CKA_WRAP*, *CKA_UNWRAP*, *CKA_ALWAYS_ SENSITIVE*, and *CKA_NEVER_ EXTRACTABLE* SHALL also always be set to *FALSE*.

Compliant components SHALL only support the key types *CKK_DES* and *CKK_DES2* within the object class *CKO_SECRET_KEY*.

Compliant components SHALL support the *DES* and *DES2* secrete key attribute *CKA_ VALUE* as described in [PKCS#11, 9.8.5 and 9.8.6]. **ISIS-MTT PROFILE:** recommends not to check the parity of DES-keys by the recipient. Therefore, it cannot be assumed that an error is returned after decryption of a DES-key with invalid parity.

The support for *IDEA* as described in [PKCS#11, 9.8.11] is recommended.

4 Functions

Compliant components are not required to support the complete set of functions as defined in [PKCS#11, 10]. However, compliant components SHALL at least provide a stub for every unsupported function which returns the value *CKR_FUNCTION_NOT_SUPPORTED*. The functions entry in the structure *CK_FUNCTION_LIST* SHALL point to this stub. Compliant components SHALL support the functions that are listed in the following sections.

NOTE: Some of these functions MUST only be supported for use by the security officer during the token personalization phase (i.e. *C_InitToken*, *C_InitPIN)*. The certification policies of CAs MAY further restrict the set of allowed functions for normal users. In the case of completely personalized SmartCards as write-protected tokens, the use of the functions *C_OpenSession* requesting write permissions, *C_CreateObject*, *C_CopyObject*, *C_DestroyObject*, *C_SetAttributeValue* and even *C_SetPIN* MAY also be prohibited.

4.1 General Purpose

Compliant components SHALL support the general-purpose functions *C_Initialize*, *C_Finalize*, *C_GetInfo*, and *C_GetFunctionList* as defined in [PKCS#11, 10.4] with the only exception that the *pInitArgs* parameter of the function *C_Initialize* SHALL always have the value *NULL_PTR*, since simultaneous access via multithreading is not considered in ISIS-MTT.

4.2 Slot and Token Management

Compliant components SHALL support the slot and token management functions *C_GetSlotList*, *C_GetSlotInfo*, *C_WaitForSlotEvent*, *C_GetMechanismList*, *C_GetMechanismInfo*, *C_InitToken*, *C_InitPIN*, *and C_SetPIN* as defined in [PKCS#11, 10.5].

Compliant components SHALL NOT support the token-dependent use of the functions *C_InitToken*, *C_InitPin* and *C_SetPin* in the case of protected authentication path other than *PINpad*.

4.3 Session Management

Compliant components SHALL support the session management functions *C_OpenSession*, *C_CloseSession*, *C_CloseAllSessions*, *C_GetSessionInfo*, *C_Login*, and *C_Logout* as defined in [PKCS#11, 10.6] with the following exceptions.

Token-dependent ejection of the token from the reader cannot be assumed for compliant components after closing of the last session in *C_CloseSession* and *C_CloseAllSessions*.

Compliant components SHALL NOT support the token-dependent use of the function *C_Login* in the case of protected authentication path other than *PIN*.

Compliant components are not required to support the functions *C_GetOperationState* and *C_SetOperationState*.

4.4 Object Management

Compliant components SHALL support the object management functions *C_CreateObject*, *C_CopyObject*, *C_DestroyObject*, *C_GetObjectSize*, *C_GetAttributeValue*, *C_SetAttributeValue*, *C_FindObjectsInit*, *C_FindObjects*, and *C_FindObjectsFinal* as defined in [PKCS#11, 10.7].

4.5 Encryption

Compliant components SHALL support the encryption functions *C_Encrypt*, *C_EncryptInit*, *C_EncryptUpdate*, and *C_EncryptFinal* as defined in [PKCS#11, 10.8].

4.6 Decryption

Compliant components SHALL support the decryption functions *C_DecryptInit*, *C_Decrypt*, *C_DecryptUpdate*, and *C_DecryptFinal* as defined in [PKCS#11, 10.9].

4.7 Message Digesting

Compliant components SHALL support the message digesting functions *C_DigestInit*, *C_Digest*, *C_DigestUpdate*, *C_DigestKey*, and *C_DigestFinal* as defined in [PKCS#11, 10.10].

4.8 Signing and MACing

Compliant components SHALL support the signing and message digesting functions *C_SignInit* and *C_Sign* as defined in [PKCS#11, 10.11].

Compliant components are not required to support the functions *C_SignUpdate*, *C_SignFinal*, *C_SignRecoverInit*, and *C_SignRecover*.

4.9 Verification of Signatures and MACs

Compliant components SHALL support the signature verification functions *C_VerifyInit* and *C_Verify* as defined in [PKCS#11, 10.12].

Compliant components are not required to support the functions *C_VerifyUpdate*, *C_VerifyFinal*, *C_VerifyRecoverInit*, and *C_VerifiyRecover*.

4.10 Multiple Cryptographic Operations

Compliant components SHALL support the combined functions *C_DigestEncryptUpdate* and *C_DecryptDigestUpdate* as defined in [PKCS#11, 10.13].

Compliant components are not required to support the functions *C_SignEncryptUpdate* and *C_DecryptVerifyUpdate*.

4.11 Key Management

Compliant components SHALL support the key management functions *C_GenerateKey*, *C_GenerateKeyPair*, *C_WrapKey*, and *C_UnwrapKey* as defined in [PKCS#11, 10.14].

Compliant components are not required to support the key management function *C_DeriveKey*.

4.12 Random Number Generation

Compliant components SHALL support the random number functions *C_SeedRandom* and *C_GenerateRandom* as defined in [PKCS#11, 10.15].

4.13 Parallel Function Management

Compliant components are not required to support the functions *C_GetFunctionStatus* and *C_CancelFunction* as defined in [PKCS#11, 10.16].

5 Mechanisms

Mechanisms precisely specify how cryptographic operations have to be performed. The following table contains a mapping of mechanisms to cryptographic operations. It contains a subset of the table provided in [PKCS#11, 11], which is extended with *RIPEMD-160*, *RSA_OEAP*, and *AES*. Compliant components SHALL comply with the conformance requirements indicated in the support column in this table. For further details concerning cryptographic algorithms see ISIS-MTT Part 6 “Cryptographic Algorithms”.

Table 7: Mapping of Mechanisms to Functions

SUPPORT OF MECHANISMS			REFS TO	ISIS-MTT	
#	MECHANISM	FUNCTIONS	PKCS#11	SUPPORT	NOTES
1	*CKM_RSA_PKCS_ KEY_PAIR_GEN*	Generate Key / Key Pair	11.1.1	++	[1]
2	*CKM_RSA_PKCS*	Sign and Verify Wrap and Unwrap	11.1.2	++ ++	[2]
3	*CKM_RSA_OEAP_PKCS*	Wrap and Unwrap		+	
4	*CKM_RSA_9796*	Sign and Verify	11.1.5	+	
5	*CKM_SHA_1_RSA_PKCS*	Sign and Verify	11.1.5	++	
6	*CKM_DES_KEY_GEN*	Generate Key / Key Pair	11.18.1	++	[3]
7	*CKM_DES_CBC_PAD*	Encrypt and Decrypt	11.18.4	++	
8	*CKM_DES_MAC*	Sign and Verify	11.18.6	++	[4]
9	*CKM_DES2_KEY_GEN*	Generate Key / Key Pair	11.19.1	++	[3]
10	*CKM_DES3_CBC_PAD*	Encrypt and Decrypt	11.18.4	++	
11	*CKM_DES3_MAC*	Sign and Verify	11.18.6	+	[4]
12	*CKM_SHA_1*	Digest	11.26.1	++	[6]
13	*CKM_IDEA_CBC*	Encrypt and Decrypt		+	
14	*CKM_SHA_1_HMAC*	Sign and Verify	11.26.3	+	[6]
15	*CKM_AES_128_CBC*	Encrypt and Decrypt		+	[7]
16	*CKM_AES_192_CBC*	Encrypt and Decrypt		+	[7]
17	*CKM_AES_256_CBC*	Encrypt and Decrypt		+	[7]

[1] This mechanism is based on [PKCS#1]. The generated RSA keys can also be used with the mechanisms *CKM_RSA_PKCS, CKM_MD2_RSA_PKCS, CKM_MD5_RSA_ PKCS, CKM_SHA_1_RSA_PKCS, CKM_RSA_9796,* and *CKM_RSA_ OEAP_PKCS*.

[2]	This mechanism is based on [PKCS#1]. Compliant components SHALL support the mechanism *CKM_RSA_PKCS* for the encryption and decryption of session keys and the generation and verification of digital signatures, which are used in the functions *C_Sign*, *C_Verify*, *C_WrapKey*, and *C_UnwrapKey*.
[3]	The correct setting of parity bits for DES keys is specified in [FIPS PUB 46-2]. It is recommended not to check the parity by the recipient. Therefore, it cannot be assumed that the decryption of a DES key with invalid parity will result in the indication of an error.
[4]	These mechanisms serve for authentication, which is specified in [FIPS PUB 113].
[6]	The related hash algorithm is specified in [FIPS PUB 180-1].
[7]	The related content encryption algorithm is specified in [FIPS PUB 197].

6 PKCS#11 Definitions

This chapter contains a list of PKCS#11 definitions that are used in this part of the ISIS-MTT specification in alphabetic order.

Note

The key mechanisms *CKM_AES_128_CBC*, *CKM_AES_192_CBC*, and *CKM_AES_259_CBC* are announced for the next version. They still need to be defined.

```
#define CKC_X_509 0x00000000
#define CKK_DES 0x00000013
#define CKK_DES2 0x00000014
#define CKK_IDEA 0x0000001A
#define CKK_RSA 0x00000000
#define CKM_DES_CBC_PAD 0x00000125
#define CKM_DES_KEY_GEN 0x00000120
#define CKM_DES_MAC 0x00000123
#define CKM_DES3_CBC_PAD 0x00000136
#define CKM_DES3_KEY_GEN 0x00000131
#define CKM_DES3_MAC 0x00000134
#define CKM_IDEA_CBC 0x00000342
#define CKM_IDEA_KEY_GEN 0x00000340
#define CKM_RSA_9796 0x00000002
#define CKM_RSA_PKCS 0x00000001
#define CKM_RSA_PKCS_KEY_PAIR_GEN 0x00000000
#define CKM_SHA_1 0x00000220
#define CKM_SHA_1_HMAC 0x00000221
#define CKM_SHA1_RSA_PKCS 0x00000006
#define CKO_CERTIFICATE 0x00000001
#define CKO_DATA 0x00000000
#define CKO_PRIVATE_KEY 0x00000003
#define CKO_PUBLIC_KEY 0x00000002
#define CKO_SECRET_KEY 0x00000004
#define CKR_ATTRIBUTE_READ_ONLY 0x00000010
#define CKR_ATTRIBUTE_TYPE_INVALID 0x00000012
#define CKR_ATTRIBUTE_VALUE_INVALID 0x00000013
#define CKR_FUNCTION_NOT_SUPPORTED 0x00000054
#define CKR_TEMPLATE_INCOMPLETE 0x000000D0
#define CKR_TEMPLATE_INCONSISTENT 0x000000D1
CK_DEFINE_FUNCTION(CK_RV,
C_GetSlotInfo)(
  CK_SLOT_ID slotID,
  CK_SLOT_INFO_PTR pInfo );
CK_DEFINE_FUNCTION(CK_RV,
C_GetTokenInfo)(
  CK_SLOT_ID slotID,
  CK_TOKEN_INFO_PTR pInfo );
CK_DEFINE_FUNCTION(CK_RV,
C_WaitForSlotEvent)(
  CK_FLAGS flags,
  CK_SLOT_ID_PTR pSlot,
  CK_VOID_PTR preserved );
CK_DEFINE_FUNCTION(CK_RV,
C_GetMechanismList)(
  CK_SLOT_ID slotID,
  CK_MECHANISM_TYPE_PTR
                  pMechanismList,
  CK_ULONG_PTR pulCount );
CK_DEFINE_FUNCTION(CK_RV,
C_GetMechanismInfo)(
  CK_SLOT_ID slotID,
  CK_MECHANISM_TYPE type,
  CK_MECHANISM_INFO_PTR pInfo );
CK_DEFINE_FUNCTION(CK_RV,
C_InitToken)(
  CK_SLOT_ID slotID,
  CK_CHAR_PTR pPin,
  CK_ULONG ulPinLen,
```

```
  CK_CHAR_PTR pLabel );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_InitPIN)(
  CK_SESSION_HANDLE hSession,
  CK_CHAR_PTR pPin,
  CK_ULONG ulPinLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_SetPIN)(
  CK_SESSION_HANDLE hSession,
  CK_CHAR_PTR pOldPin,
  CK_ULONG ulOldLen,
  CK_CHAR_PTR pNewPin,
  CK_ULONG ulNewLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_OpenSession)(
  CK_SLOT_ID slotID,
  CK_FLAGS flags,
  CK_VOID_PTR pApplication,
  CK_NOTIFY Notify,
  CK_SESSION_HANDLE_PTR
                       PhSession );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_CloseSession)(
  CK_SESSION_HANDLE hSession);
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_CloseAllSessions)(
  CK_SLOT_ID slotID );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetSessionInfo)(
  CK_SESSION_HANDLE hSession,
  CK_SESSION_INFO_PTR pInfo );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetOperationState)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pOperationState,
  CK_ULONG_PTR
           PulOperationStateLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_SetOperationState)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pOperationState,
  CK_ULONG ulOperationStateLen,
  CK_OBJECT_HANDLE hEncryptionKey,
  CK_OBJECT_HANDLE
             hAuthenticationKey );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_EncryptInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Encrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pEncryptedData,
  CK_ULONG_PTR
             pulEncryptedDataLen);
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_EncryptUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pPart,
  CK_ULONG ulPartLen,
  CK_BYTE_PTR pEncryptedPart,
  CK_ULONG_PTR
             pulEncryptedPartLen);
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_EncryptFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pLastEncryptedPart,
  CK_ULONG_PTR
        PulLastEncryptedPartLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_DecryptInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Decrypt)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pEncryptedData,
  CK_ULONG ulEncryptedDataLen,
  CK_BYTE_PTR pData,
  CK_ULONG_PTR pulDataLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_DecryptUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pEncryptedPart,
  CK_ULONG ulEncryptedPartLen,
  CK_BYTE_PTR pPart,
```

```
  CK_ULONG_PTR pulPartLen );
CK_DEFINE_FUNCTION(CK_RV,
C_DecryptFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pLastPart,
  CK_ULONG_PTR pulLastPartLen );
CK_DEFINE_FUNCTION(CK_RV,
C_DigestInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism );
CK_DEFINE_FUNCTION(CK_RV,
C_Digest)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pDigest,
  CK_ULONG_PTR pulDigestLen );
CK_DEFINE_FUNCTION(CK_RV,
C_DigestUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pPart,
  CK_ULONG ulPartLen );
CK_DEFINE_FUNCTION(CK_RV,
C_DigestKey)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hKey );
CK_DEFINE_FUNCTION(CK_RV,
C_DigestFinal)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pDigest,
  CK_ULONG_PTR pulDigestLen );
CK_DEFINE_FUNCTION(CK_RV,
C_SignInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey );
CK_DEFINE_FUNCTION(CK_RV,
C_Sign)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG_PTR pulSignatureLen );
CK_DEFINE_FUNCTION(CK_RV,
C_VerifyInit)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hKey );
CK_DEFINE_FUNCTION(CK_RV,
C_Verify)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pData,
  CK_ULONG ulDataLen,
  CK_BYTE_PTR pSignature,
  CK_ULONG ulSignatureLen );
CK_DEFINE_FUNCTION(CK_RV,
C_DigestEncryptUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pPart,
  CK_ULONG ulPartLen,
  CK_BYTE_PTR pEncryptedPart,
  CK_ULONG_PTR
              pulEncryptedPartLen);
CK_DEFINE_FUNCTION(CK_RV,
C_DecryptDigestUpdate)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pEncryptedPart,
  CK_ULONG ulEncryptedPartLen,
  CK_BYTE_PTR pPart,
  CK_ULONG_PTR pulPartLen);
CK_DEFINE_FUNCTION(CK_RV,
C_GenerateKey)(
  CK_SESSION_HANDLE hSession
  CK_MECHANISM_PTR pMechanism,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR phKey );
CK_DEFINE_FUNCTION(CK_RV,
C_GenerateKeyPair)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_ATTRIBUTE_PTR
               pPublicKeyTemplate,
  CK_ULONG
       ulPublicKeyAttributeCount,
  CK_ATTRIBUTE_PTR
             pPrivateKeyTemplate,
  CK_ULONG
      ulPrivateKeyAttributeCount,
  CK_OBJECT_HANDLE_PTR
```

```
                        phPublicKey,
  CK_OBJECT_HANDLE_PTR
                    PhPrivateKey );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_WrapKey)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hWrappingKey,
  CK_OBJECT_HANDLE hKey,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG_PTR pulWrappedKeyLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_UnwrapKey)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hUnwrappingKey,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG ulWrappedKeyLen,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulAttributeCount,
  CK_OBJECT_HANDLE_PTR phKey );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_SeedRandom)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pSeed,
  CK_ULONG ulSeedLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GenerateRandom)(
  CK_SESSION_HANDLE hSession,
  CK_BYTE_PTR pRandomData,
  CK_ULONG ulRandomLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_CopyObject)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR
                    phNewObject);
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_CreateObject)(
  CK_SESSION_HANDLE hSession,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount,
  CK_OBJECT_HANDLE_PTR phObject);
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_DestroyObject)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Finalize) (
  CK_VOID_PTR preserved );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_FindObjects)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE_PTR phObject,
  CK_ULONG ulMaxObjectCount,
  CK_ULONG_PTR pulObjectCount );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_FindObjectsFinal)(
  CK_SESSION_HANDLE hSession );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_FindObjectsInit)(
  CK_SESSION_HANDLE hSession,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetAttributeValue)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetFunctionList)(
  CK_FUNCTION_LIST_PTR_PTR
  PpFunctionList );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetInfo) (  CK_INFO_PTR pInfo );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetObjectSize)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ULONG_PTR pulSize );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_GetSlotList)(
  CK_BBOOL tokenPresent,
  CK_SLOT_ID_PTR pSlotList,
  CK_ULONG_PTR pulCount );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Initialize)(
  CK_VOID_PTR pinitArgs );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Login)(
```

```
  CK_SESSION_HANDLE hSession,
  CK_USER_TYPE userType,
  CK_CHAR_PTR pPin,
  CK_ULONG ulPinLen );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_Logout)(
  CK_SESSION_HANDLE hSession );
```

```
CK_DEFINE_FUNCTION(CK_RV,
C_SetAttributeValue)(
  CK_SESSION_HANDLE hSession,
  CK_OBJECT_HANDLE hObject,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulCount );
```

```
typedef CK_ULONG
CK_ATTRIBUTE_TYPE;
```

```
typedef CK_ULONG
CK_CERTIFICATE_TYPE;
```

```
typedef CK_ULONG CK_KEY_TYPE;
```

```
typedef CK_ULONG
CK_MECHANISM_TYPE;
```

```
typedef CK_ULONG CK_NOTIFICATION;
```

```
typedef CK_ULONG CK_OBJECT_CLASS;
```

```
typedef CK_ULONG CK_OBJECT_HANDLE;
```

```
typedef CK_ULONG CK_RV;
```

```
typedef CK_ULONG
CK_SESSION_HANDLE;
```

```
typedef CK_ULONG CK_SLOT_ID;
```

```
typedef CK_ULONG CK_STATE;
```

```
typedef CK_ULONG CK_USER_TYPE;
```

```
typedef struct CK_ATTRIBUTE {
  CK_ATTRIBUTE_TYPE type;
  CK_VOID_PTR pValue;
  CK_ULONG ulValueLen;
} CK_ATTRIBUTE;
```

```
typedef struct CK_DATE {
  CK_CHAR year[4];
  CK_CHAR month[2];
  CK_CHAR day[2];
} CK_DATE;
```

```
typedef struct CK_FUNCTION_LIST {
  CK_VERSION version;
  CK_C_Initialize C_Initialize;
  CK_C_Finalize C_Finalize;
  CK_C_GetInfo C_GetInfo;
  CK_C_GetFunctionList
               C_GetFunctionList;
  CK_C_GetSlotList C_GetSlotList;
  CK_C_GetSlotInfo C_GetSlotInfo;
  CK_C_GetTokenInfo
                 C_GetTokenInfo;
  CK_C_GetMechanismList
              C_GetMechanismList;
  CK_C_GetMechanismInfo
              C_GetMechanismInfo;
  CK_C_InitToken C_InitToken;
  CK_C_InitPIN C_InitPIN;
  CK_C_SetPIN C_SetPIN;
  CK_C_OpenSession C_OpenSession;
  CK_C_CloseSession
                 C_CloseSession;
  CK_C_CloseAllSessions
              C_CloseAllSessions;
  CK_C_GetSessionInfo
                C_GetSessionInfo;
  CK_C_GetOperationState
             C_GetOperationState;
  CK_C_SetOperationState
             C_SetOperationState;
  CK_C_Login C_Login;
  CK_C_Logout C_Logout;
  CK_C_CreateObject
                 C_CreateObject;
  CK_C_CopyObject C_CopyObject;
  CK_C_DestroyObject
                C_DestroyObject;
  CK_C_GetObjectSize
                C_GetObjectSize;
  CK_C_GetAttributeValue
             C_GetAttributeValue;
  CK_C_SetAttributeValue
             C_SetAttributeValue;
  CK_C_FindObjectsInit
               C_FindObjectsInit;
  CK_C_FindObjects C_FindObjects;
  CK_C_FindObjectsFinal
              C_FindObjectsFinal;
  CK_C_EncryptInit C_EncryptInit;
  CK_C_Encrypt C_Encrypt;
  CK_C_EncryptUpdate
                C_EncryptUpdate;
  CK_C_EncryptFinal
```

```
                    C_EncryptFinal;
  CK_C_DecryptInit C_DecryptInit;
  CK_C_Decrypt C_Decrypt;
  CK_C_DecryptUpdate
                  C_DecryptUpdate;
  CK_C_DecryptFinal
                    C_DecryptFinal;
  CK_C_DigestInit C_DigestInit;
  CK_C_Digest C_Digest;
  CK_C_DigestUpdate
                    C_DigestUpdate;
  CK_C_DigestKey C_DigestKey;
  CK_C_DigestFinal C_DigestFinal;
  CK_C_SignInit C_SignInit;
  CK_C_Sign C_Sign;
  CK_C_SignUpdate C_SignUpdate;
  CK_C_SignFinal C_SignFinal;
  CK_C_SignRecoverInit
                C_SignRecoverInit;
  CK_C_SignRecover C_SignRecover;
  CK_C_VerifyInit C_VerifyInit;
  CK_C_Verify C_Verify;
  CK_C_VerifyUpdate
C_VerifyUpdate;
  CK_C_VerifyFinal C_VerifyFinal;
  CK_C_VerifyRecoverInit
              C_VerifyRecoverInit;
  CK_C_VerifyRecover
                  C_VerifyRecover;
  CK_C_DigestEncryptUpdate
            C_DigestEncryptUpdate;
  CK_C_DecryptDigestUpdate
            C_DecryptDigestUpdate;
  CK_C_SignEncryptUpdate
              C_SignEncryptUpdate;
  CK_C_DecryptVerifyUpdate
            C_DecryptVerifyUpdate;
  CK_C_GenerateKey C_GenerateKey;
  CK_C_GenerateKeyPair
                C_GenerateKeyPair;
  CK_C_WrapKey C_WrapKey;
  CK_C_UnwrapKey C_UnwrapKey;
  CK_C_DeriveKey C_DeriveKey;
  CK_C_SeedRandom C_SeedRandom;
  CK_C_GenerateRandom
```

```
                    C_GenerateRandom;
  CK_C_GetFunctionStatus
               C_GetFunctionStatus;
  CK_C_CancelFunction
                  C_CancelFunction;
  CK_C_WaitForSlotEvent
                C_WaitForSlotEvent;
} CK_FUNCTION_LIST;
```

```
typedef struct CK_INFO {
  CK_VERSION cryptokiVersion;
  CK_CHAR manufacturerID[32],
  CK_FLAGS flags;
  CK_VERSION libraryVersion
} CK_INFO;
```

```
typedef struct CK_MECHANISM {
  CK_MECHANISM_TYPE mechanism;
  CK_VOID_PTR pParameter;
  CK_ULONG ulParameterLen;
} CK_MECHANISM;
```

```
typedef struct CK_MECHANISM_INFO {
  CK_ULONG ulMinKeySize;
  CK_ULONG ulMaxKeySize;
  CK_FLAGS flags;
} CK_MECHANISM_INFO;
```

```
typedef struct CK_SESSION_INFO {
  CK_SLOT_ID slotID;
  CK_STATE state;
  CK_FLAGS flags;
  CK_ULONG ulDeviceError;
} CK_SESSION_INFO;
```

```
typedef struct CK_SLOT_INFO{
  CK_CHAR slotDescription[64];
  CK_CHAR manufacturerID[32];
  CK_FLAGS flags;
  CK_VERSION hardwareVersion;
  CK_VERSION firmwareVersion;
} CK_SLOT_INFO;
```

```
typedef struct CK_TOKEN_INFO{
  CK_CHAR label[32];
  CK_CHAR manufacturerID[32];
  CK_CHAR model[16];
  CK_CHAR serialNumber[16];
  CK_FLAGS flags;
  CK_ULONG ulMaxSessionCount;
  CK_ULONG ulSessionCount;
```

```
  CK_ULONG ulMaxRwSessionCount;
  CK_ULONG ulRwSessionCount;
  CK_ULONG ulMaxPinLen;
  CK_ULONG ulMinPinLen;
  CK_ULONG ulTotalPublicMemory;
  CK_ULONG ulFreePublicMemory;
  CK_ULONG ulTotalPrivateMemory;
  CK_ULONG ulFreePrivateMemory;
  CK_VERSION hardwareVersion;
  CK_VERSION firmwareVersion;
  CK_CHAR utcTime[16];
} CK_TOKEN_INFO;
```

```
typedef struct CK_VERSION {
  CK_BYTE major;
  CK_BYTE minor,
} CK_VERSION;
```

References

[FIPS PUB 113] National Institute of Standards and Technology: FIPS PUB 113: Computer Data Authentication, May 1985

[FIPS PUB 180-1] National Institute of Standards and Technology: Secure Hash Standard, April 1995

[FIPS PUB 197] Federal Information Processing Standards (FIPS PUB) 197: Advanced Encryption Standard; November 2001

[FIPS PUB 46-2] National Institute of Standards and Technology: FIPS PUB 46-2: Data Encryption Standard, December 1993

[PKCS#1] RSA Laboratories: RSA Encryption Standard; Version 2.0; October 1998

[PKCS#11] RSA Laboratories: Cryptographic Token Interface Standard; Version 2.01; December 1997

[RFC 1319] B. Kaliski: The MD2 Message-Digest_algorithm, April 1992

[RFC 1321] R. Rivest: The MD5 Message-Digest_algorithm, April 1992

[RIPEMD-160] H.Dobertin, A. Bosselaers, and B. Preneel: RIPEMD-160: A strengthened version of RIPEMD; Tast Software Encryption - Cambridge Workshop 1996, LNCS, Vol. 1039, Springer; April 1996

COMMON ISIS-MTT SPECIFICATIONS
FOR INTEROPERABLE PKI APPLICATIONS

FROM T7 & TELETRUST

SPECIFICATION

PART 8

XML SIGNATURE AND ENCRYPTION MESSAGE FORMATS

VERSION 1.1 – 16 MARCH 2004

Contact Information

ISIS-MTT Working Group of the TeleTrusT Deutschland e.V.: www.teletrust.de

The up-to-date version of ISIS-MTT can be downloaded from the above web site, from www.isis-mtt.org or from www.isis-mtt.de

Please send comments and questions to isismtt@teletrust.de

Editors:

Hans-Joachim Bickenbach

Jürgen Brauckmann

Alfred Giessler

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.

Table of Contents

1 Preface 5

2 XML Signature Format 7

2.1 Signature Element 8

2.2 SignatureValue Element 8

2.3 SignedInfo Element 8

2.3.1 CanonicalizationMethod Element 8

2.3.2 SignatureMethod Element 10

2.3.3 Reference Element 11

2.4 KeyInfo Element 13

2.4.1 RetrievalMethod Element 14

2.4.2 X509Data Element 15

2.5 Object Element 16

3 XML Encryption format 17

3.1 EncryptedKey Element 17

3.2 EncryptedDataType 19

3.3 EncryptionMethodType 21

4 Algorithm Support 23

4.1 Cryptographic Algorithms 23

4.2 Canonicalization 23

4.3 Transforms 24

4.4 Decoding 25

5 XML Schema Redefines 26

5.1 XML_DSIG Redefine 26

5.2 XML_ENC Redefine 31

References 34

1 Preface

This part of the ISIS-MTT specification provides the ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT and OSCI. OSCI can now be redefined as a special signature profile based on this ISIS-MTT general XML signature profile without any essential changes.

This ISIS-MTT 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 ISIS-MTT 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 whith more experience and when requirements will become clearer.

A few notes on Web Service Security 1.0 [WS_SEC]. 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”) 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 new draft specification of WS Security – X.509 Token Profile [WS_SEC_DR2] includes more data types to be contained in wsse:SecurityTokenReference.

3 [WS_SEC] recommends Exclusive XML Canonicalization and XML Decryption Transform.

4 A special STR Dereference Transform in WS Security – SOAP Message Specification [WS_SEC_DR1] 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

- During the work on this specification we could have a look at the draft specification for PDF 1.5. There digital signatures are covered but to our knowledge by no means compatible to XML_DSIG. Based on that information we would rate PDF 1.5 not compatible to this profile.
- 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 ISIS-MTT Specification) as in the other parts of the ISIS-MTT 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.

An area of concern is the usage of the RIPEMD algorithm. The ISIS-MTT 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 ISIS-MTT specification and OSCI. It is one of the goals of this document to harmonize ISIS-MTT and OSCI in a way that OSCI can (almost) without changes become a profile of this ISIS-MTT 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 ISIS-MTT.					

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		<xsd:enumeration value="http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments"/>					[2]
5		<xsd:enumeration value="http://www.w3.org/TR/2001/REC-xml-exc-c14n-20010315"/>	++	++		--	[2] [3]
6		<xsd:enumeration value="http://www.w3.org/TR/2001/REC-xml-exc-c14n-20010315#WithComments"/>					[2]

[1]	This restriction is suitable, because only [XML_CAN] and [XML_EXCAN] are allowed.
[2]	Canonicalization is the standard serialization method of XML. For definitions and usage of canonicalization in XML see [XML_CAN] and [XML_EXCAN] or follow the links noted in #3 and #4. Both algorithms MUST be supported. Other canonicalization algorithms MUST NOT be used in conformance with ISIS-MTT. This delimits usage to the most common types and specifically rules out any proprietary algorithms. **Note:** Although [XML_CAN] 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_CAN] 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.

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]	**ISIS-MTT 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]. It may happen that in the future XKMS will become important for ISIS-MTT 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]	**ISIS-MTT PROFILE:** Although RIPEMD160 has proven to be a valuable hash algorithm in the next version of ISIS-MTT it will no longer be included as a mandatory algorithm neither on the generating nor on the processing side. This is due to the fact that a great number of applications is practically excluded from conformance because RIPEMD160 is not implemented. So we discourage 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 the meaning of both so Clients SHOULD interprete 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. **ISIS-MTT PROFILE:** URIs of types HTTP, HTTPS and LDAP are RECOMMENDED. For LDAP see also ISIS-MTT 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]	**ISIS-MTT 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 the meaning of both, so Clients SHOULD interprete 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 ISIS-MTT 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			4.4	
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	--	--		
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]	**ISIS-MTT PROFILE:** The only way of storing KeyInfo data are X509Data for coherence with the rest of the ISIS-MTT specification.
[3]	OSCI conformance; to be clarified by OSCI
[4]	XML_ENC: To support Diffie-Hellman key agreement for encrypting data, see Part6: XML_ENC chapter **5.5 noch zu aktualieren**

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]	**ISIS-MTT 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 ISIS-MTT Part 3, T2.#4.
[4]	For coherence with the rest of the ISIS-MTT 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 ISIS-MTT 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 ISIS-MTT references [XAdES]. ISIS-MTT 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 [RFC 2630]. 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 ISIS-MTT 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 ISIS-MTT conforming clients that support XAdES MUST be able to process signatures without any of the XAdES elements present. Also ISIS-MTT 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]	**ISIS-MTT 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]	**ISIS-MTT 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]	**ISIS-MTT 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 ISIS-MTT specification are listed in Part 6 “Cryptographic Algorithms” of the ISIS-MTT 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	ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS		GEN	PROC	VALUES	
1.1	Canonical XML	Canonicalization algorithm	[XML_DSIG] [XML_CAN]	+	++	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	
1.2	Canonical XML with Comments	Canonicalization algorithm	[XML_DSIG] [XML_CAN]	+-	+	http://www.w3.org/TR/2001/REC-xml-c14n-20010315#WithComments	
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] Not specified in [XML_DSIG]							

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	ISIS-MTT SUPPORT			NOTES
#	NAME	SEMANTICS		GEN	PROC	VALUES	
1.1	Canonical XML	Canonicalization algorithm	[XML_DSIG] [XML_CAN]	++	++	http://www.w3.org/TR/2001/REC-xml-c14n-20010315	
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] 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.

4.4 Decoding

Table 14: Decoding Algorithms

ALGORITHMS			REFERENCES	ISIS-MTT 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">
                        ISIS-MTT – Restrictions for XML DSIG
                        Based on OSCI 1.2
                </xsd:documentation>
        </xsd:annotation>
        <!-- ### redefinitions ### -->
        <xsd:redefine schemaLocation="http://www.w3.org/TR/2001/CR-xmldsig-core-20010419/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/TR/2001/REC-xml-exc-c14n-20010315"/>
                                                <xsd:enumeration
                                                        value="http://www.w3.org/TR/2001/REC-xml-exc-c14n-
                                                20010315#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: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.osci.de/2002/04/osci#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.osci.de/2002/04/osci#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">
                        ISIS-MTT – 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

[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
[RFC2630]	R. Housley, „Cryptograhic Message Syntax“, RFC2630, 1999
[WS_SEC]	IBM, Microsoft, Verisign: Web Services Security, Version 1.0, April 2002,
[WS_SEC_DR1]	OASIS Open: Web Services Security: SOAP Message Security, Working Draft 17, August 2003
[WS_SEC_DR2]	OASIS Open: Web Services Security: X.509 Certificate Token Profile, Working Draft 10, August 2003
[X.509]	ITU-T X.509: Information Technology - Open Systems Interconnection – The Directory: Authentication Framework, 1997
[XAdES]	ETSI TS 101 903 V1.1.1 (2002-02): XML Advanced Electronic Signatures (XAdES), Technical Specification
[XKMS]	W3C: XML Key Management (XKMS 2.0) Requirements, 05 May 2003 http://www.w3.org/TR/2003/NOTE-xkms2-req-20030505
[XML_CAN]	W3C: Canonical XML 1.0, 15 March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315
[XML_DSIG]	W3C: XML-Signature Syntax and Processing, 12 February 2002, http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/
[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.w3.org/TR/2001/REC-xml-c14n-20010315
[XML_SCHEMA]	W3C: XML Schema Part 1: Structures, 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/
[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.html