PKITS
Public Key Infrastructure with Time-Stamping Authority
(a.k.a. PITA)

ETS PROJECT: 23.192

Deliverable D4
Time-Stamping Service Functional Specification and Protocols for Unstructured Data

Produced by: ***FNMT***
Date of issue: 12*th January 1999*
Revision Number: ***17***

TABLE OF CONTENTS

HISTORY

version	date	author	comment
1	1998-may-18	José A. Mañas	1st index proposal First contents for Executive Summary
2	1998-may-28	Juan & Juan Luis	First contents for chapter 3
3	1998-june-1	José A. Mañas	First contents for chapter 8
4	1998-june-2	Manel & Eduard Juan & Juan Luis	First contents for chapter 4 First contents for chapter 5
5	1998-june-15	Antonio Muñoz	First contents for chapter 7
6	1998-june-22	Manel & Eduard Antonio Muñoz Manuel Zaragoza	New contents for chapter 4 Chapter 7 completion New contents for Executive Summary
7	1998-june-23	José A. Mañas	New contents for chapter 5 Chapter 8 completion
8	1998-june-24	Juan & Juan Luis	Chapter 3 completion
9	1998-june-29	Manel & Eduard	New contents for chapter 4
10	1998-june-30	José A. Mañas	Chapter 5 completion
11	1998-july-5	Manel & Eduard	New contents for chapter 4
12	1998-july-6	José A. Mañas	Integration
13	1998-july-9	Juan & Juan Luis	Integration
14	1998-july-9	Manel & Eduard	New contents for chapter 4
15	1998-july-9	Juan & Juan Luis	New contents & Integration
16	1998-july-10	J.A.M.	Overall review
17	1999-jan-12	Manuel Heras	OIDs are proposed for all protocol data structs; occurences of GeneralizedTime have been replaced by UTCTime

GLOSSARY OF TERMS

Specifications:	
SHALL	Essential requirement. A requirement must be fulfilled or a feature implemented wherever this term occurs. The designer is requested, however, to indicate if one or more “shall requirements” would increase the cost or time unreasonably in relation to the total cost or design cost, in which case the specification may have to be revised.
SHOULD	Important requirement. Shall be implemented without or with minimum extra cost. Valid reasons in particular circumstances may allow ignoring such requirements.
MAY	Optional requirement. From case to case, it should be decided whether implementing it or not, in any case without exceeding the budget planned for the related activity.

Technical:	
ASN.1	Abstract Syntax Notation, One
CA	Certification Authority
CRL	Certificate Revocation List
DS	Digital Signature
EDI	Electronic data Interchange

ETS	European Trusted Systems
MTBF	Mean Time Between Fails
MTTF	Mean Time to Fail
NA	Notary Authority
PKI	Public Key Infrastructure
RA	Registration Authority
TS	Time-Stamping
TSA	Time-Stamping Authority
TSS	Time-Stamping Service
TTP	Trusted Third Party

1 EXECUTIVE SUMMARY

This document covers the provision of a time-stamping service when the time-stamping authority has no knowledge of the internal structure of the documents to be time-stamped. Documents are processed as raw data.

The document is organised as follows. Section 3 covers service requirements from the points of view of the service users (requester and verifier) and the providing TTP. Section 4 formalises the protocols described in Deliverable D3 [PKITS D3], going into the ASN.1 details of formats and encoding. Section 5 presents four implementations of the service over Internet. Lastly, section 5 briefly describes real scenarios of use.

Appendixes are included to contain details of ASN.1 encoding: a reminding of PKCS-7 fundamentals, a reminding of CMP from PKIX, and a detailed example of time-stamping exchanges.

1.1 WHAT ARE "UNSTRUCTURED DOCUMENTS"?

The term "unstructured documents", which is used frequently in this document, may be confusing. It is obvious that any document has a minimum structure, no matter how simple it is. So, what does it mean "unstructured"? When speaking about unstructured documents in a TSS context, we make reference to documents or hash of documents which internal structure is unknown to (or intentionally ignored by) the TSA. In opposition to unstructured documents that will be handled as raw data, we identify some kind of documents whose structure needs to be known by the TSA in order to perform specific actions over the structure and embed the time stamp token generated within that structure. Such time-stamping services are covered in subsequent deliverables of this project..

Note that this notion of document is closely related to the TSA knowledge of the data submitted to it, and also connects with the type and structure of users' requests; it is perfectly feasible that a user sends an EDI message embodied in an e-mail and time-stamp it as raw data. It is in fact a recognisable document by an "EDI-enabled" TSA, but the request forces an "append" service rather than an "embed" service.

It is also important to notice that the distinction made between structured and unstructured documents is forced by completely different functional specifications of TSS.

1.2 PROJECT ROADMAP

PKITS structures its technical contributions into a collection of deliverables:

D3. Architecture of Time-Stamping Service and Scenarios of Use: Services and Features
D4. Time-Stamping Service Functional Specification and Protocols for Unstructured Data
D5. TSS Functional Specification and Protocols for EDI Messages and Interchanges
D6. Time-Stamping Service Functional Specification and Protocols for Multimedia Information

D3 studied the definition of the service, identified applicable protocols, and established a working framework. D4, this deliverable, focuses on unstructured documents, that is those documents whose internal structure is unknown to the TSA, and it cannot benefit from any knowledge, nor deal with separate fields in any sensible manner. D5 and D6 will consider those cases where there is a knowledge of the syntactic structure of the documents: D5 studies the case of EDI messages, that are strictly formalised, and where there are fields specifically designed for holding time-stamping information; D6 studies multimedia information where there is knowledge of the audio and video stream structure. D5 and D6 propose mechanisms for embedding time-stamping information into already standardised information structures. D4,

dealing with unstructured data, proposes an appendix to wrap time-stamping information to raw data.

2 REFERENCES

[Adams98] Adams, Cain, Pinkas, Zuccherato, "Internet Public-Key Infrastructure, Part V: Time-Stamp Protocols", PKIX Working Group *Draft*, June 1998. ftp://ftp.ietf.org/internet-drafts/draft-adams-time-stamp-02.txt

[CMP] C. Adams, S. Farrell, "Internet Public Key Infrastructure: Certificate Management Protocols", PKIX Working Group *Draft*, May 1998 ftp://ftp.ietf.org/internet-drafts/draft-ietf-pkix-ipki3cmp-08.txt

[DEC89] Digital Time Service Functional Specification Version T.1.0.5. Digital Equipment Corporation, 1989.

[PKCS #1] RSA Laboratories, "PKCS #1 - RSA Encryption Standard", version 1.5, Nov. 1993. http://www.rsa.com/rsalabs/pubs/PKCS

[PKCS #10] RSA Laboratories, "PKCS #10 – Certification Request Syntax Standard", version 1.0, Nov. 1993. http://www.rsa.com/rsalabs/pubs/PKCS

[PKCS #12] RSA Laboratories, "PKCS #12 – Public Key User Information Syntax Standard", version 1.0, Apr. 1995. http://www.rsa.com/rsalabs/pubs/PKCS

[PKCS #6] RSA Laboratories, "PKCS #6 – Extended-Certificate Syntax Standard", version 1.5, Nov. 1993. http://www.rsa.com/rsalabs/pubs/PKCS

[PKCS #7] RSA Laboratories, "PKCS #7 – Cryptographic Message Syntax Standard", version 1.5, Nov. 1993. http://www.rsa.com/rsalabs/pubs/PKCS

[PKIX] PKIX Working Group, "Internet Public Key Infrastructure - Part IX.509 Certificate and CRL Profile". http://www.ietf.org/html.charters/pkix-charter.html

[PKITS D3] Architecture of Time-Stamping Service and Scenarios of Use: Service and Features, Deliverable D3 of ETS Project 23.192 Public Key Infrastructure with Time-Stamping Authority, May, 1998.

[Ray98] Ray, J.R., The IGP/BIPM time transfer project, in 1998 IGS Analysis Center Workshop Proceedings, European Space Operations Centre, Darmstadt, Germany, in press 1998.

[RFC 1422] S. Kent, "Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management". February 1993. ftp://ds.internic.net/rfc/rfc1422.txt

[RFC 1847] Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted. J. Galvin, S. Murphy, S. Crocker & N. Freed. October 1995.

[RFC 2068] Hypertext Transfer Protocol – HTTP/1.1. R. Fielding et al. January 1997.

[RFC 2311] S/MIME Version 2 Message Specification. S. Dusse, P. Hoffman, B. Ramsdell, L. Lundblade, L. Repka. March 1998.

[RFC 2312] S/MIME Version 2 Certificate Handling:. S. Dusse, P. Hoffman, B. Ramsdell, J. Weinstein. March 1998.

[RFC 781] Su, Z. A specification of the Internet protocol (IP) timestamp option. DARPA Network Working Group Report RFC-781. SRI International, May 1981. ftp://ds.internic.net/rfc/rfc781.txt

[RFC 792] Defense Advanced Research Projects Agency. Internet Control Message Protocol. DARPA Network Working Group Report RFC-792, USC Information Sciences Institute, September 1981. ftp://ds.internic.net/rfc/rfc792.txt

[RFC 867] Postel, J. Daytime protocol. DARPA Network Working Group Report RFC-

867, USC Information Sciences Institute, May 1983.
ftp://ds.internic.net/rfc/rfc867.txt

[RFC 868] Postel, J. "Time protocol. DARPA Network Working Group Report, USC Information Sciences Institute, May 1983.
ftp://ds.internic.net/rfc/rfc868.txt

[RFC 959] File Transfer Protocol. J. Postel, J.K. Reynolds. Oct-01-1985.

[RIPE 95] RIPE Project. Ripe Integrity Primitives: Final Report on Race Integrity Primitives Evaluation; LNCS 1007, Springer-Verlag, 1995.

[Schneier96] B. Schneier, "Applied Cryptography", 2nd ed. John Wiley & Sons, 1996.

[SPKI] Internet Engineering Task Force SPKI Working Group. SPKI Specifications.
http://www.ietf.org/html.charters/spki-charter.html

[X.509 v3] ITU Recommendation X.509, "The Directory – Authentication Framework", version 3, Geneve 1996.
http://www.itu.ch/itudoc/itu-t/rec/x/x500up.html

3 REQUIREMENTS

This section covers the specific requirements of a Time Stamping Service for raw data, where no syntactic structure is to be known by the TSA. Several protocols are proposed in D3 in their general format that are applicable to stamp unstructured documents. Over this protocols and means, requirements will be imposed from actors involved in the scenarios of use: The Users and providers. On the other hand, a given protocol will impose requirements over the software and hardware to be used while following the protocol.

3.1 USERS' REQUIREMENTS

3.1.1 The requester

The requester imposes basic requirements over TSS provision that have to be met in order to satisfy his needs:

1 Reliability.

The essential requirement over TSS is reliability, this is the provision of a correct service, accordant to its specifications. A reliable TSS is an accurate service in absence of errors. In order to achieve this, MTTF shall be kept high, by means of service security and specific countermeasures to detect and correct fails before they reach the user certificates.

UR_R1	The TSA **shall** provide means to prevent from an incorrect or deficient service provision, these regards every element involved in the provision of the service.

2 Availability.

The user of TSS demands a 7-24 service. Therefore, it is expected a strong requirement on availability. The system must be kept the most time in working order. MTBF should be held in a high rate. Security practices are again the most valuable resource to keep system availability in high levels.

UR_R2	The TSPS **may** state an availability rate of the system providing TS service.
UR_R3	The TSA **shall** provide means to avoid service interruption and keep its availability rate within limits established in its TSPS (if stated).

3 Accuracy.

The accuracy of time assigned to documents may be of vital importance to the user. Accuracy is measured in terms of time grain. The smaller the grain of time (guaranteed precision of time tokens), the best. The accuracy of time stamps regards STS and time stamping protocol.

UR_R4	The TSA **shall** specify the origin and nature of its time source/s in its TSPS.
UR_R5	The TSA **shall** state the precision (or precision options in case there are more than one alternative) that can be expected in time stamping service according to the level of service agreed with the TSA. This **shall** be defined in the TSPS.
UR_R6	The TSA **shall** periodically publicise information regarding the accuracy of its time source and inform of deviations along with corrective measures taken.
UR_R7	The TSA **should** provide means in order to check the status of time source and detect deviations.

4 Speed.

The speed of the system may be an important requirement imposed by some requesters using interactive means to obtain time certificates. Speed will be measured in terms of response time for time-stamp provision.

UR_R8	The TSA **shall** provide means (processing power) to keep its response times within the limits stated in its TSPS.
UR_R9	The TSA **should** periodically publicise the distribution of its response times.

5 Security, integrity and authenticity.

The user may require security and integrity over the information exchanged with the TSA. User may also want to protect the communication against "Man-in-the-middle" attacks by requiring Authentication over TSA identity. To meet this requirements public key cryptography and digital signature schemes are demanded.

UR_R10	Each member of the UoU **shall** be in possession of an identity certificate issued by a CA, and **shall** use it to prove identity to the TSA.
UR_R11	The TSA **shall** use a secure digital signature scheme (and key size) to produce the time-stamping certificates.
UR_R12	The TSA **shall** use a secure digital signature scheme (and key size) to communicate with users regarding matters apart from time stamps production.
UR_R13	The TSA **shall** provide means to secure and publicise information regarding its service.
UR_R14	The TSA **shall** not disclose to unauthorised third parties any private information related to the users of its time stamping service.

6 Cryptographic quality.

Users trust will be strongly balanced towards the strength of cryptographic primitives used in the time stamping protocol. The TSA shall state hash functions which shall be accepted in the request's hash field, and will encourage the submission of several hashes of the same document in order to increase the certificate cryptographic quality/security.

UR_R15	The TSA **shall** select and state in the TSPS a set of acceptable hashing algorithms that users may use to produce the hashes of their documents when following the time-stamping protocol.
UR_R16	Each member of the UoU **shall** use an accepted hashing algorithm when following the time-stamping protocol.
UR_R17	Each member of the UoU **may** be able to submit not only one but a set of pairs <Hash algorithm formal identifier, message digest> to the TSA.

7 Legal validity.

The user may need his/her time certificate to be an acceptable evidence for its use in court. TSA must provide means to build evidences that support veracity and integrity of certificates.

UR_R18	The TSA **should** make reference to legal authorisations and permissions to operate and limit its liability and legal validity of certificates within its TSPS.

8 Universality.

An important requirement over time stamping certificates may be its validity beyond

the domain covered by the emitting TSA. In order to met this requirements TSA may establish links either via legal (cross certification) or via technological (synchronisation) towards other TSA's.

UR_R19	The TSPS **shall** define links with other TSS and domain where its certificates are valid (and verifiable).
UR_R20	The TSA **may** provide synchronisation means to establish links to other TSS enlarging the domain where its time certificates are regarded as valid.

9 Lasting certificates.

The user may require a long term for his/her certificates. Renewal process may be regarded as uncomfortable to the user.

UR_R21	The TSA **shall** assist its users in the periodical renewal of its certificates.
UR_R22	The TSA **shall** publicise among its users information on cryptographic advances that could affect the validity of their time stamping certificates.
UR_R23	On cessation of its activities, the TSA **shall** transfer to another TSA all the information that could be needed to verify or renew the time-stamping certificates issued during its activity period.

10 Ease of use

UR_R24	No special abilities **should** be required to the user and software (if needed) **should** be provided.

11 Minimum equipment requirements.

UR_R25	Users **should** have no requirement on extra equipment, and software.

3.1.2 The verifier

Verifier constitutes another user profile whose demands on TSS not differ much from the requester. In this section it can be found similar requirements than the previous one and new specific ones:

1 Reliability.

The verifier also demands a reliable service. MTBF in verification process should be kept high. Therefore, requirements are identical to requester's.

UR_V1	The TSA **shall** provide means to prevent from an incorrect or deficient service provision, these regards every element involved in the provision of the service.

2 Availability.

The verifier may demand a 7-24 service. Therefore, it is expected a strong requirement on availability. The verification subsystem must be kept working a high time rate.

UR_V2	The TSA **shall** assist in the verification of **any** time certificate issued by it.
UR_V3	The TSPS **may** state an availability rate of the system providing TS verification service.
UR_V4	The TSA **shall** provide means to avoid service interruption and keep its

	availability rate within limits established in its TSPS (if there stated).

3 Speed.

The speed of the system may be an important requirement in stamps verification. Speed will be measured in terms of response time for time-stamp verification.

UR_V8	The TSA **shall** provide means (processing power) to keep its response times within the limits stated in its TSPS.
UR_V9	The TSA **should** periodically publicise the distribution of its response times for verification service.

4 Authenticity, security and integrity.

The verifier may also require security and integrity over the information received from the TSA.

UR_V10	Each member of the UoU **shall** be in possession of an identity certificate issued by a CA, and **shall** use it to prove identity to the TSA.
UR_V11	The TSA **shall** use a secure digital signature scheme (and key size) to communicate with verifiers.
UR_V12	The TSA **shall** provide means to secure and make reachable its public information.

5 Evidence.

The requirements regarding storage of evidence are imposed from the verifier and included in TSPS. Verifier may also want to check temporal relationship among documents time stamped by different TSA's. So s/he may require evidences of synchronisation from a given TSA.

UR_V13	The TSPS **shall** state the period in which the time-stamped evidences shall be maintained, published and finally removed.
UR_V14	The TSA **shall** keep information (evidences) for verification purposes as stated in TSPS.
UR_V15	Each member of the UoU **should** store the time-stamped documents in a safe place, since they will be needed for verification and certificate renewal.

6 Ease of use.

UR_V16	No special abilities **should** be required to the verifier of a time certificate and software (if needed) **should** be provided.

7 Minimum equipment requirements.

UR_V17	Verifiers **should** have no requirement on extra equipment, and software.

3.1.3 The provider (TSA)

We identify requirements imposed from the TSA either to protect its assets or to be able to provide a given service quality as specified. In this sense, requirements will be imposed on users and on itself.

1 Service provision and level of service

UR_P1	The TSA **shall** publicise a TSPS defining its service and therefore limiting its liability.
UR_P2	The TSPS **shall** state the duration of the pair of keys used to time-stamp. The key used to time-stamp messages must be changed every specific period or every specific number of messages time-stamped. This period of time and number of messages **shall** be stated in the TSPS.
UR_P3	The TSPS **shall** specify the protocol used in the TSS provision.
UR_P4	The TSPS **shall** specify the term in which the time-stamped evidences **shall** be maintained in the TSA.
UR_P5	The TSPS **shall** state the precision (or precision options in case there are more than one alternative) that can be expected in the time stamping service according to the level of services agreed with the TSA.
UR_P6	The TSPS **shall** state the source from which it obtains the time reference and the procedures to avoid deviations from this time base.
UR_P7	The TSPS **shall** specify hash algorithms which shall be accepted in the corresponding request field, as well as the formal identifiers of them.
UR_P8	The TSPS **shall** specify the origin and nature of unpredictable events within the link chain, as well as the number of unrelated bits used to encode these events.
UR_P9	The TSPS **should** make reference to international standards applicable to the Time Stamping Service Provision followed by the TSA. This item is important for the interoperability and synchronisation with other TSA's.
UR_P10	The TSPS **shall** state the communication channel to be used for the communications between the TSA and users.
UR_P11	The TSA **may** be provided with redundant access channels. The use of several accesses will protect the availability of the TSS.
UR_P12	The TSPS **shall** state how it shall proceed in case of going out of the Time Stamping Service Provision activity.
UR_P13	The TSA **may** encourage the use of more than one hash for the same document.
UR_P14	The TSPS **shall** state how all deviation from the normal operation shall be communicated to the customers and published to made them known to any interested party. Deviations includes loss of synchronisation, deviations from the time base, compromised keys, etc.

2 Security practice.

UR_P15	TSA **shall** use technical security controls to deliver its services in a secure way. Security control of the equipment, of the life cycle of the software, of the networks and of the cryptographic modules are included.
UR_P16	Each member of the UoU **shall** be in possession of an identity certificate issued by the CA, and **shall** use it to prove identity to the TSA.
UR_P17	The TSA **should** be provided with a special security length key for time stamp tokens signing purposes. The length of the key should be secure "enough" regarding the state of art for signature forgery.
UR_P18	The TSA signing key lifetime **should** be restricted to a given "short" period of time.
UR_P19	The TSA signing key lifetime **should** be restricted to a maximum number of digital signature operations.
UR_P20	Synchronisation procedure **should** be launched periodically and its frequency must be high "enough" to detect and correct possible deviations.
UR_P21	Access to time source receptor **shall** be strictly restricted and placed in a secure location.
UR_P22	Receptor location **shall** provide an stable signal reception and **shall** avoid shade or dark reception zones.

UR_P23	The TSPS **shall** state how and when audits are carried out and who must do it (external or internal).
UR_P24	The TSPS **shall** state what events shall be recorded, the type of log files, the procedure to do it, who is the responsible, and how long the log files shall be maintained.
UR_P25	The TSA **should** insert the hash of an unpredictable event every N documents.
UR_P26	The TSA **should** insert unpredictable events periodically within its linking information.
UR_P27	The TSA **shall** state its internal and external procedures to recover after a disaster of any type has occurred.
UR_P28	The TSA **shall** use non technical security controls to deliver its services in a secure way. Physical, personnel and procedural controls are included.

3.2 SOFTWARE REQUIREMENTS

This section refers to the specific software requirements that are needed for the scheme TSA-user to work properly. Formally speaking, software is not only algorithm and programs but everything conceptual implemented over hardware such as communication protocols, data bases, cryptographic functions and schemes...

3.2.1 Cryptography

- ✓ Hash function.
  Users shall use one or more hash functions to calculate message digests that shall be sent to TSA.

- ✓ Digital signature scheme.
  Requests and tokens shall be signed by the sender, therefore a digital signature scheme is needed.

3.2.2 Communications

- ✓ TCP/IP support.
  Actual communications are basically based on TCP/IP so support for these protocols is essential.

- ✓ Communication applications: Web browsers, E-mail clients, ftp apps,...
  Users may use some client applications for accessing to the Time Stamping Service.

- ✓ HTTP server, E-mail server, ftp server,...
  TSA shall provide server application in order to respond to client service requests.

- ✓ Access to corresponding Authorities, CA's.
  Both the user and the TSA shall be provided with communication means to access services delivered by other authorities.

3.2.3 Data bases

- ✓ Requesters information storage.
  The TSA shall store this information for verification purposes.

- ✓ Evidences storage.
  The TSA shall store this information for verification purposes.

- ✓ Audit trails.
  The TSA shall store this information for audit purposes.

3.2.4 Security

- ✓ Software firewalls.
  The TSA shall use firewalls to restrict access to its infrastructures.

- ✓ Monitoring of network traffic.
  The TSA may use specific software for monitoring the use of its services.

3.2.5 Other

- ✓ Trusted Operating System.
  The TSA shall use high security operating systems.

- ✓ NTP software.
  If the TSA is to use NTP as secure time source, it shall be provided with software to do so.

3.3 HARDWARE REQUIREMENTS

3.3.1 Processing power

- ✓ CPU processing power.
  The TSA shall arrange processing power enough to deal with Time Stamping requests.

- ✓ Cryptographic processing power.
  The TSA may be provided with specific cryptographic hardware units.

3.3.2 Storage requirements

- ✓ Hard disks.

- ✓ Streamers.

- ✓ Redundant storage equipment.

3.3.3 Communications

- ✓ LAN.
  TSA subsystems may be connected by means of Local Area Networks.

- ✓ Modem.

- ✓ Radio receiver.

- ✓ GPS receiver.

- ✓ Routers.

- ✓ Communication lines.

- ✓ Communications management equipment.

3.3.4 Security

- ✓ Hardware firewall.
  The TSA shall use firewalls to restrict access to its infrastructures.

- ✓ UPS (Uninterrupted Power Supply).

3.3.5 Other

✓ Internal management and invoicing equipment.

4 PROTOCOLS

We present time stamping, time stamping renewal, time stamping verification and TSAs synchronisation protocol data formats in ASN.1 for the Time Stamping Service.

The syntax provided below is closely related to the syntax proposed in [Adams 98]. It is the intention to look for convergence as the protocols stabilise in the future.

4.1 TIME-STAMPING

4.1.1 General data structures

The protocol to be used in time-stamping service is a two-way handshake protocol. A user must send a **TimeStampRequest** protocol data unit to the TSA and it will answer with a **TimeStampToken** protocol data unit. These two protocol data units are also used in the time stamping renewal and TSA's synchronisation protocols.

TimeStampRequest

The **TimeStampRequest** protocol data unit provides basic information in a time stamp service request is encapsulated as a PKCS#7 **SignedData** construct that should append specific protocol information through authenticated attributes to the **TimeStampRequest** protocol data unit. The content type of the **contentInfo** field is the OID:

```
TimeStampRequestId ::= OBJECT IDENTIFIER { pkcs-7 2003 }
```

The content of the **contentInfo** field is the following **TimeStampRequest** ASN.1 data structure:

```
MessageImprint ::= SEQUENCE {
      hashAlgorithm         AlgorithmIdentifier,
      hashedMessage         OCTET STRING
}
```

Data field	Description
`hashAlgorithm`	Hash function OID (e.g. md-5, sha-1)
`hashedMessage`	The corresponding hash value of the message to be time-stamped calculated with the hash function specified in **hashAlgorithm** data field.

```
TimeStampRequest ::= SEQUENCE {
      version               Integer  { v1(0) },
      policy                PolicyInformation OPTIONAL,
      nonce                 Integer,
      messageImprints       SEQUENCE OF MessageImprint
}
```

Data field	Description
`version`	It is the syntax version number. It shall be 1 for this version of the specification
`policy`	Service policy requested to the TSA under the **TimeStampToken** shall be issued
`nonce`	TSS transaction identification number which relates a **TimeStampRequest** protocol data unit to a **TimeStampToken** protocol data unit. The **TimeStampRequest** and the **TimeStampToken** the TSA issued as a response must have the same nonce value
`messageImprints`	Hashes calculated with some digest algorithms (e.g sha-1,md5,md2..) over the document to be time-stamped. This sequence must be sorted by lexicographic order of the field **hashAlgorithm** (function OID)

Depending on the protocol required different authenticated attributes would be included into the **SignedData** structure.

The **SignedData** structure that encapsulates a **TimeStampRequest** shall be signed by the requester of the time stamp service, so the **SignerInfos** field of the PKCS#7 data structure that encapsulates the **TimeStampRequest** structure shall include a **signerInfo** data structure with all the required information related to the requester. This **SignedData** structure will be a time certification request.

TimeStampToken

The **TimeStampToken** protocol data unit provides basic time stamp service information. This protocol data unit is encapsulated as a PKCS#7 **SignedData** construct that shall append specific protocol related information. This **SignedData** structure will be a time certificate.
The content type of the **contentInfo** field is the OID:

```
TimeStampTokenId ::= OBJECT IDENTIFIER { pkcs-7 2004 }
```

The content of the **contentInfo** field is the following **TimeStampToken** ASN.1 data structure:

PKIStatusInfo is defined in Section 3.2.3 of [CMP]. More codes have been added to the **PKIFailureInfo** structure in order to supply more detailed service status information. We only present the **PKIFailureInfo** values related to Time Stamping service. The complete values list can be found at [CMP]. Values in parenthesis () are defined in [CMP] and values in brackets {} are defined in this document

```
PKIFailureInfo ::= BIT STRING {
      BadAlg           (0),  -- unrecognized or unsupported Algorithm Identifier
      BadMessageCheck  (1),  -- integrity check failed (e.g., signature did not verify)
      BadRequest       (2),  -- transaction not permitted or supported
      BadCertId        (4),  -- no certificate could be found
      BadDataFormat    (5),  -- the data submitted has the wrong format
      CertRevoked      (10), -- the certificate is revoked
      CertExpired      (11), -- the certificate has expired
      CertNotActive    (13), -- the certificate was not active at the given time
      TSANotAvailable  {14}, -- the TSAs service is not available
      NotSoManyLinks   {15}  -- Linking information chain end reached in
                             -- verification protocol, or not so many links
```

```
      VerificationFails      {16}  -- TimeStampToken verification fails
      UnknownRequester       {17}  -- The requester was not specified
      BadTimeStampProtocol   {18}  -- TimeStampProtocol not defined
      TSLinkFound            { 19 }  -- TimeStampLink found
      TSLinkLost             { 20 }  -- TimeStampLink lost
}


TimeStampToken ::=SEQUENCE {
      version                Integer  { v1(0) },
      policy                 PolicyInformation,
      status                 PKIStatusInfo,
      stampedTime            UTCTime,
      nonce                  Integer,
      messageImprints        SEQUENCE OF MessageImprint,
}
```

Data field	Description
`version`	It is the syntax version number. It shall be 1 for this version of the specification
`policy`	Policy used by the TSA in the service
`status`	Give information about the service
`stampedTime`	The time the TSA included in the **TimeStampToken**
`nonce`	Must be the same value included in the request related to this token
`messageImprints`	The same message imprints the requester submitted to the TSA

Several authenticated attributes shall be included in both **TimeStampRequest** and **TimeStampToken** related **SignedData** structures depending on the protocols requested. The relation between protocols and authenticated attributes will be detailed later on each protocol description.

These authenticated attributes shall be included in the **SignerInfos** data structure encapsulated into the **SignedData**:

- In order to identify the protocol requested by the user to the TSA, the **TimeStampProtocol** authenticated attribute shall be included in both **TimeStampRequest** and **TimeStampToken** related **SignedData** structures. The **TimeStampProtocol** authenticated attribute is identified by the following OID

  ```
  TimeStampProtocolId ::= { pkcs-9 32}
  ```

  The **TimeStampProtocol** attribute type has the following ASN.1 data structure:

  ```
  TimeStampProtocol ::= ENUMERATED {simpleProtocol (0),
  linkingProtocol(1)}
  ```

  In case the **TimeStampProtocol** authenticated attribute is omitted, the TSA shall respond with a **TimeStampToken** structure encapsulated in a PKCS#7 **SignedData** with the value **BadTimeStampProtocol** in the **PKIFailureInfo** field

- In order to limit the validity period of the **TimeStampToken** returned by the TSA, the **TimeStampValidity** authenticated attribute should be included in the **SignedData**

structure. The **TimeStampValidity** authenticated attribute is identified by the following OID:

```
TimeStampValidityId ::= { pkcs-9 34}
```

The **TimeStampValidity** attribute type has the following ASN.1 data structure:

```
TimeStampValidity ::= UTCTime
```

If this authenticated attribute is omitted, the default validity period indicated in the policy requested must be assumed.

- The **SignerInfos** field of the **SignedData** structure related to the **TimeStampToken** shall include the **TimeStampRequester** authenticated attribute that identifies the requester of the **TimeStampToken** (the owner).

  This authenticated attribute is identified by the following OID:

  ```
  TimeStampRequesterId ::= { pkcs-9 33}
  ```

  The **TimeStampRequesterId** attribute type has the following ASN.1 data structure:

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

Data field	Description
`issuer`	Certificate authority name
`serialNumber`	Value that uniquely identifies the certificates issued by the certificate authority specified in the issuer field

  ```
  TimeStampRequester ::=  IssuerAndSerialNumber
  ```

  The TSA will retrieve this information from the **issuerAndSerialNumber** field of the **SignedData SignerInfo** field related to the **TimeStampReq** service request. Because of this, the value of this field shall be the certificate authority name and the certificate serial number of the certificate the user used to sign PKCS#7 **SignedData** structures.

  In case the **signerInfos** field of the Signed Data that encapsulates a **TimeStampReq** is void, this authenticated attribute shall be omitted and status field of the **TimeStampToken** structure shall have the value **PKIFailureInfo** set to **UnknownRequester**.

- In order to identify a **TimeStampToken**, serial number shall be included in the **SignedData** structure as an authenticated attribute. This authenticated attribute has the following OID:

  ```
  TimeStampSNId ::= { pkcs-9 35}
  ```

  The **TimeStampSNId** attribute has the following ASN.1 data structure:

  ```
  TimeStampSN ::= Integer
  ```

  This number identifies the **TimeStampToken** issued by the TSA that signed the PKCS#7 **SignedData** structure this authenticated attribute appears in.

- In order to identify temporal data tokens, that may be introduced by the TSA according to criteria described in [PKITS D3], an authenticated attribute is specified which has the following OID:

  ```
  TemporalDataId ::= { pkcs-9 36}
  ```

  The **TemporalDataId** attribute has the following ASN.1 data structure:

  ```
  TemporalData ::= SEQUENCE {
       source     Name,
       info       BIT STRING
  }
  ```

  Source identifies the temporal data provider, while the info is the unpredictable information itself.

  The same attribute shall be used both in the request and in the response.

- When data are submitted to synchronisation by another TSA, according to the criteria described in [PKITS D3], an authenticated attribute is specified which has the following OID:

  ```
  SynchronisedId ::= { pkcs-9 37 }
  ```

  The **SynchronisedId** attribute has the following ASN.1 data structure:

  ```
  SynchronisationData ::= IssuerAndSerialNumber
  ```

  Name identifies the TSA used to synchronise with.

4.1.2 Simple protocol

The protocol to be used in time-stamping service is a two-way handshake protocol. A user must send a **TimeStampRequest** protocol data unit encapsulated in a PKCS#7 **SignedData** structure (time certification request) to the TSA and it will answer with a **TimeStampToken** protocol data unit encapsulated in a PKCS#7 **SignedData** structure (time certificate).

TimeStampRequest

A user must send a **TimeStampRequest** protocol data encapsulated in a PKCS#7 **SignedData** structure. This **SignedData structured** shall be signed by the user, so it shall include a **SignerInfo** field with all the required information related to the end user. As described above, if this **SignerInfo** field is omitted TSA shall return an error by setting **PKIFailureinfo** to **UnknownRequester**.

In a simple protocol time stamp service request, the following authenticated attributes shall be present in the **SignerInfo** field related to the requester:

- A **TimeStampProtocol** authenticated attribute with the value **simpleProtocol** should be included in the **SignedData** structure.

TimeStampToken

The TSA will receive this request and will respond with a **TimeStampToken** encapsulated in a PKCS#7 **SignedData** structure. Because the TSA shall sign this structure, a **SignerInfo** field with all the required information related to the TSA shall be present.

Also, the following authenticated attributes shall be included into the **SignerInfo** field related to the TSA:

- A **TimeStampProtocol** authenticated attribute with value **simpleProtocol**.
- A **TimeStampRequester** authenticated attribute which identifies the requester (the owner) of the **TimeStampToken**.
- A **TimeStampSN** authenticated attribute which is the serial number of the **TimeStampToken**.
- A **TimeStampValidity** authenticated attribute. However, the default validity indicated in the request policy shall be assumed if this authenticated attribute is omitted.

4.1.3 Linking protocol

The protocol to be used in PKI time-stamping service is also a two-way handshake protocol. As we described in previous deliverables (PKITS D3), in linking protocol each time certificate issued by the TSA is linked with the previous and the following time certificates issued (or that will be issued). We will refer to previous time certificates as backward links, and following time certificates as forward links.

TimeStampRequest

A user must send a **TimeStampRequest** protocol data unit to the TSA encapsulated in a PKCS#7 **SignedData**. As simple protocol, this is an authenticated request, so a **SignerInfo** field shall be included with all the required information related to requester user into the **SignedData** structure.

This **SignerInfo** structure shall also include the following authenticated attributes:

- An authenticated attribute **TimeStampProtocol** with value **linkingProtocol**.

TimeStampToken

The TSA will respond with a **TimeStampToken** protocol data unit encapsulated in a PKCS#7 structure that will be a time certificate. This structure shall be signed by the TSA, so again all required information related to the TSA shall be included in a **SignerInfo** field and the following authenticated attributes shall be present in it:

- A **TimeStampProtocol** authenticated attribute with value **linkingProtocol**
- A **TimeStampRequester** authenticated attribute which identifies the requester (the owner) of the **TimeStampToken**
- A **TimeStampSN** authenticated attribute which is the serial number of the **TimeStampToken**
- A **TimeStampValidity** authenticated attribute. However, the default validity indicated in the requested policy shall be assumed if this authenticated attributed is omitted.
- A **TimeStampLinks** authenticated attribute, which is linking information to the previous and the following **TimeStampToken** issued by the TSA. This attribute is identified by the following OID:

```
TimeStampLinksId ::= { pkcs-9 38 }
```

And has the following ASN.1 specification:

```
TimeStampLinks ::= SEQUENCE {
   version               Integer  { v1(0) }
   backwardLink          TimeStampBwdLink,
   forwardLink           TimeStampFwdLink}
```

Data field	Description
`version`	It is the syntax version number. It shall be 1 for this version of the specification.
`backwardLink`	A link to the previous TimeStampTokens issued by the TSA
`forwardLink`	A link to the following TimeStampTokens the TSA will issue

TimeStampBwdLink and **TimeStampFwdLink** are specified as follows:

```
TimeStampBwdLink ::= SEQUENCE {
      serialNumber       Integer,
      requester          IssuerAndSerialNumber,
      stampedTime        UTCTime,
      messageImprints    SEQUENCE OF MessageImprint,
      previous           SEQUENCE OF MessageImprint,
}
```

Data field	Description
`serialNumber`	Identifies the previous token
`requester`	User who was the requester of the previous timestamp issued by the TSA
`stampedTime`	Time stamped to the previous token issued by the TSA. If it is present, the user can compare it to the time included in his time stamp token when he receives the link token. If it is omitted, the user shall interact with the requester in order to compare times and to verify its token.
`messageImprints`	Hashes of the message timestamped in the previous timestamptoken issued by the TSA
`previous`	Hashes of the TimeStampBwdLink data structure included in the previous timestamptoken issued by the TSA.

```
TimeStampFwdLink ::=SEQUENCE {
   serialNumber          Integer,
}
```

Where `serialNumber` is the unique identification of the next time-stamp certificate to be issued by the TSA.

4.1.4 Distributed protocol

Distributed time stamping makes sense when there is no clear central authority to provide the service. When a user wants to time stamp a document with the distributed protocol he shall issue a **TimeStampRequest** encapsulated in a PKCS#7 **SignedData** to the TSAs selected through a pseudo-random function as described in D3. As a result, the requester will receive *n*

TimeStampToken structures encapsulated on PKCS#7 **SignedData** structures from the selected TSA's that the requester shall encapsulate as a single time certificate.

Because the requester-TSA interaction is independent from the interactions the requester do with the others *n-1* TSAs, the same ASN.1 data structures defined for simple and linking protocol can be used. This is, the requester will interact with the TSAs with simple or linking protocol and for each interaction he will receive a **SignedData** structure that encapsulates a **TimeStampToken**. The time certificate then is a **SignedData** structure that encapsulates all the **TimeStampToken** structures received.

However, an additional authenticated attribute shall be included in the TSA and the requester **signerInfo** field of the PKCS#7 **SignedData** structure that encapsulates TimeStampRequests TimeStampTokens the requester and the TSAs interchange.

This additional authenticated attribute is:

- An authenticated attribute in order to specify which pseudo-random number generator algorithm the user uses to select the time-stamping authorities. This attribute is identified by the following OID

  ```
  TimeStampRandomId ::= { pkcs-9 39 }
  ```

  The **TimeStampRandom** attribute type has the following ASN.1 data structure:

  ```
  TimeStampRandom ::= OBJECT IDENTIFIER
  ```

  This authenticated attribute also distinguishes between simple and linking protocol, and the distributed scheme using simple or linking protocol. So, if this authenticated attribute appears in a time certification request it indicates that the user will interact with more than one TSAs. The authenticated attribute **TimeStampProtocol** then specifies the interaction the user wants with the selected TSAs.

In the distributed protocol, the requester will receive n **TimeStampToken** (one from every TSA involved in the distributed protocol) that shall be considered as a single time certificate. Because of this, the requester shall encapsulate these *n* **TimeStampToken** in a single PKCS#7 **SignedData** construct as follows:

The content type of the **contentInfo** field is the OID:

```
TimeStampDistrbTokenId ::= OBJECT IDENTIFIER { pkcs-7 2005 }
```

The content of the **contentInfo** field is the following **TimeStampDistrbToken** ASN.1 data structure:

```
TimeStampDistrbToken ::=SEQUENCE OF SignedData
```

The user shall sign this structure in order to link securely all the **SignedData** (TimeStampTokens) encapsulated. Because of this, a **SignerInfo** field with all the required information related to the requester of the distributed **TimeStamp** protocol shall be present, and it must contain only the signature of the requester.

4.2 RENEWAL

Time stamps may be valid for a limited period of time, that may be explicit in the policy statement, or in the certificates that supports the signing key of the TSA, or implicit if

cryptography advances introduce reasonable doubts about the soundness of the elements (either a hash functions is subject to attacks, or the signing protocol, or the keys, or …). For one or other reason, a time-stamp certificate may become void, and need to go under a renewal process to extend the non-repudiation period.

We describe how simple, linking and distributed time certificates can be renewed and the protocols to be used for this purpose.

4.2.1 Simple Protocol TimeStampToken renewal

When a user requests a time certificate using the simple protocol he gets a PKCS#7 Signed Data structure that encapsulates a **TimeStampToken** data structure and some authenticated attributes as described in chapter 4.1.2. This PKCS#7 **SignedData** structure is the time certificate the user shall store with the document he wanted to time stamp.

Time-stamp certificate renewal shall be performed as a normal new time stamping process by submitting to the TSA the identifier of the time certificate to be renewed or the complete time certificate, and the resulting receipt of rehashing the original document together with the existing time certificate.

Because the renewal process is a normal document time stamp process, the same protocol described for simple, linking and distributed time stamping protocols can be used to renewal a time stamp token that was obtained through a simple time stamp protocol. This is, a simple protocol **TimeStampToken** can be renewed throw the simple, linking or distributed protocol, and the receipt issued by the TSA(s) will be a new **TimeStampToken** (if simple or linking protocol is requested) or a new **TimeStampDistrbToken** (if distributed protocol is requested).

In case distributed protocol is requested, the user shall encapsulated the n-**TimeStampToken** received from TSAs as a single time certificate as described in the distributed protocol chapter 4.1.4

4.2.2 Linking Protocol TimeStampToken renewal

The same process as in simple protocol **TimeStampToken** renewal shall be applied. When a user requests a time certificate using the linking protocol, he gets a PKCS#7 **SignedData** structure that also encapsulates a **TimeStampToken** and some authenticated attributes. In this case, one of these authenticated attributes stores some linking information to other time certificates.

The same information as in simple protocol **TimeStampToken** renewal shall be submitted to the TSA in a linking protocol **TimeStampToken** renewal. The user shall submit to the TSA the identifier of the time certificate to be renewed or the complete time certificate, and the resulting receipt of rehashing the original document together with the existing time certificate

Because the renewal process of a time certificate issued with linking protocol is also a normal document time stamp process, it can be renewed using the same protocols described for simple, linking and distributed time stamping protocols. The response issued by the TSA(s) will be a new **TimeStampToken** (if simple or linking protocol is requested) or a new **TimeStampDistrbToken** (if distributed protocol is requested in this renewal process).

In case distributed protocol is requested, the user shall encapsulated the *n*-**TimeStampToken** received from TSAs as a single time certificate as described in the distributed protocol chapter 4.1.4

4.2.3 Distributed Protocol TimeStampToken renewal

When a user requests a time certificate using the distributed protocol, he gets n PKCS#7 **SignedData** structure that also encapsulates a **TimeStampToken**. In this case, the user shall encapsulate all the **TimeStampTokens** received in a single **TimeStampDistrbToken** time certificate.

The renewal process of a distributed time certificate is exactly as in simple protocol time certificate or linking protocol time certificate renewal. However, it is applied over this **SignedData TimeStampDistrbToken** which encapsulates the *n* TimeStampTokens certificates received from the TSAs selected in the distributed time stamp protocol, instead of being applied over a **SignedData** structure that encapsulates a single **TimeStampToken** resulting from a simple or linking protocol time stamping.

The same information as in simple and linking time certificate renewals shall be submitted to the TSA in a distributed protocol time certificate renewal. The user shall submit to the TSA the complete "distributed" time certificate, and the resulting receipt of rehashing the original document together with the existing "distributed" time certificate.

Because the renewal process of a time certificate issued with distributed protocol is also a normal document time stamp process, it can be renewed using the same protocols described for simple, linking and distributed time stamping protocols. The response issued by the TSA(s) will be a new **TimeStampToken** (if simple or linking protocol is requested) or a new **TimeStampDistrbToken** (if distributed protocol is requested in this renewal process).

In case distributed protocol is requested, the user shall encapsulated the n-**TimeStampToken** received from TSAs as a single time certificate **TimeStampDistrbToken** as described in the distributed protocol chapter 4.1.4

4.2.4 ASN.1 Renewal Protocol specification

As we described above, the same simple, linking and distributed protocols can be used to renewal time certificates. However, there is a small difference in the data the user submits to the TSA if we compare it to a normal document time stamp process.

The user must fill the **messageImprint** field in **TimeStampRequest** structures with the resulting hashes of the time certificate to be renewed appended to the original document **stampedTime**. Because of this, the **messageImprint** field in **TimeStampRequest** shall be filled with the resulting hashes of the following ASN.1 structure:

```
DataForRenewal ::=SEQUENCE {
    tstoken          SignedData,
    info             BIT STRING
}
```

Data field	Description
`tstoken`	The **SignedData** structure which encapsulates the **TimeStampToken** the user wants to renew. In case the distributed protocol was used to get this time certificate, this Signed Data will encapsulate *n* TimeStampTokens considered as a single time certificate.
`info`	The document related to the **tstoken** time stamp token

Moreover, the following additional authenticated attributes shall be included in the **TimeStampReq** and **TimeStampToken** PKCS#7 **SignedData** structures in order to identify

this kind of service and to identify the **TimeStampToken** that is being renewed. These authenticated attributes shall be included in the TSA and requester **SignerInfo** structure of the PKCS#7 **SignedData** structure.

- A **TimeStampRenewal** authenticated attribute identified by the following OID:

```
TimeStampRenewalId ::= { pkcs-9 40 }
```

The **TimeStampRenewal** authenticated attribute has the following ASN.1 specification:

```
TimeStampRenewal ::= CHOICE {
      tokenId          Integer,
      timeStamp        SignedData
}
```

Data field	Description
timeStamp	This **SignedData** structure is the time certificate to be renewed. Depending on the protocol used in the past to get the time certificate that is going to be renewed, it will encapsulate a **TimeStampToken** and some authenticated attributes or a **TimeStampDistrbToken**. If this field is used, the resulting renewed time certificate (**TimeStampToken** or **TimeStampDistrbToken**) will encapsulated the original time certificate. Again, if this new **TimeStampToken** is also renewed using the **SignedData** field, the resulting one will encapsulate both old time certificates, so the complete renovation chain is stored with last token renewed.
tokenId	Instead of storing the complete **TimeStampToken** renewed, only the **TimeStamp** serial number is indicated. The renovation chain can also be retrieved by following this field on the renewed TimeStampTokens. This option is not applicable if the time certificate to renewal is a **SignedData** which encapsulates a **TimeStampDistrbToken** because there is no serial number that identifies uniquely this kind of time certificate.

In order to break the renovation chain and to distribute it into several renewed TimeStampTokens, both strategies can be applied. For instance, several TimeStampTokens can store different fragments of the renovation chain (using the **timeStamp** field of the authenticated attribute), and these TimeStampTokens were renewed using the **tokenId** field.

This is:

Suppose a **SignedData** (**TimeStampToken**) T which is renewed by using **timeStamp** field of the **TimeStampRenewal** authenticated attribute.

T renewed -> T1 (T)

As a result, a T1 **TimeStampToken** encapsulated in a **SignedData** is returned by the TSA. T1 encapsulates the **SignedData** (**TimeStampToken**) T through the **TimeStampRenewal** authenticated attribute (**timeStamp** field)

T1 is also renewed using **timeStamp** field

T1 renewed-> T2(T1(T))

As a result, a T2 **TimeStampToken** encapsulated in a **SignedData** is returned by the TSA. T2 encapsulates the **SignedData**(**TimeStampToken**) T1 through the **TimeStampRenewal** authenticated attribute (**timeStamp** field), so it also encapsulates the **SignedData** (**TimeStampToken**) T through the **TimeStampRenewal** authenticated attribute included in the **SignedData** that encapsulates Time Stamp T1.

The **SignedData**(**TimeStampToken**) T2 stores all the renovation chain of the **SignedData**(**TimeStampToken**) T.

Then, T2 is renewed using the **tokenId** field.

T2 renewed -> T3 (refers to T2)

As a result a T3 **TimeStampToken** encapsulated in a **SignedData** is returned by the TSA. T3 only includes a reference to **TimeStampToken** T2 through the **TimeStampRenewal** authenticated attribute included in the **SignedData**, so it is necessary to retrieve T2 in order to complete the renovation chain.

4.3 VERIFICATION

Verification phase covers the activities related to the assurance that the time certificate is valid. It may be requested by the owner of the certificate to assure himself of the correctness of its object. It may be requested by another party that is concerned, either to prove it is a correct certificate, or looking after proving it is not a valid one. Verification may lastly be carried out by a third party on behalf of the disputing parties.

Verification is not a simple task, not even an easy to formalise one. It may be as simple as verifying the signature of the TSA (contents, dates, rights to sign, liabilities, etc.) or may go through the chain of links (in the linked protocol) or explore the collection of singular certificates (in the distributed protocol), or a mixture of each the previous ones.

Verification may include the analysis of synchronisation stages between different TSA to prevent some operation risks and collusion attacks (see [PKITS D3] for further details).

If temporal data tokens are used (e.g. in the linked protocol), the correctness of the data provided has to be verified as well.

When the linking protocol is used, the verifier traverses the chain of links, performing singular verification checks on every link, or on a selected subset of links. A verifier may prefer to check thoroughly the immediate previous and posteriors time-certificates, or may prefer a random selection, either close or remote. The only objective is to assure that the chain of links is genuine, and that the TSA behaves correctly.

Lastly, verification may imply auditing the TSA operation: installation, procedures, stored data, and so on, as well as extend the analysis to those TSA used for mutual synchronisation.

In this complicated scenario, three coarse procedures may be foreseen:

1. verify the correctness of a single certificate
2. ask the TSA, that is trusted, to emit a verdict according to some procedure previously agreed and publicised
3. retrieve evidence (collections of certificates direct or indirectly related) to carry out a specific verification analysis

First procedure is a traditional service provided by a notary authority that checks the formal correctness of the signed data, and certifies that the identity certificates are valid, and each entity is duly entitled for its work.

The second procedure asks the TSA to play a role closed to that of a notary, being specialised on time-stamping. The benefit for users is that standardised procedures help to resolve disputes, and the TSA has the needed machinery to perform a verification in house, and to contact synchronising TSA for needed evidence.

The third procedure may be expected from a notary authority applying its own criteria to issue a verdict based on enough evidence. Involved parties are requested to provide needed evidence (that is, stored time-certificates).

Two protocols are identified to support these foreseen scenarios:

V1: It is a two-way handshake protocol between a user and a TSA where the user asks the TSA to issue a verdict for a time-certificate, according to some procedure stated in its TSPS, registered, and identified by its OID. The TSA responds with a verdict.

V2: It is a retrieval protocol between a user and a TSA where the user requests one or more certificates. Certificates are identified by their unique serial number. If the TSA is using a linking protocol, a collection of certificates (e.g. N before, N after) may be requested simultaneously. The TSA response is the requested collection of time-certificates, if available, or a reason to explain why a certificate cannot be retrieved.

4.3.1 TSA Verification Protocol

This protocol is a two-way handshake protocol. The user sends a **TimeStampVerReq** data structure encapsulated in a PKCS#7 **SignedData** to the TSA and the time stamp authority responds with a **TimeStampVerToken** encapsulated in a PKCS#7 **SignedData** structure.

TimeStampVerReq

The **TimeStampVerReq** protocol data unit is encapsulated as a PKCS#7 **SignedData** construct. The content type of the **contentInfo** field is the OID:

```
TimeStampVerReqId::= OBJECT IDENTIFIER { pkcs-7 2006 }
```

The content of the **contentInfo** field is the following **TimeStampVerReq** ASN.1 data structure:

```
TimeStampVerReq ::=SEQUENCE {
      versionID        Integer {v1(0)},
      nonce            Integer,
      tsa              IssuerAndSerialNumber,
      serialNumber     Integer,
      messageImprints  SEQUENCE OF MessageImprint,
      procedureID      OBJECT IDENTIFIER
}
```

Data field	Description
versionID	To identify this syntax
nonce	To uniquely identify this request
tsa	Identification of the TSA that issued the certificate, that must be the same one that is addressed now for verification
serialNumber	To uniquely identify the certificate subject to verification
messageImprints	Those of the object corresponding to the certificate subject to verification
procedureID	one of those stated in the TSPS, describing the verification procedure to be carried out by the TSA

The **SignedData** structure shall be signed by the user, so it shall include a **SignerInfo** field with all the required information related to the end user.

TimeStampVerToken

The **TimeStampVerToken** protocol data unit is encapsulated as a PKCS#7 **SignedData** construct. The content type of the **contentInfo** field is the OID:

```
TimeStampVerTokenId::= OBJECT IDENTIFIER { pkcs-7 2007 }
```

The content of the **contentInfo** field is the following **TimeStampVerToken** ASN.1 data structure:

```
TimeStampVerToken ::=SEQUENCE {
      versionID        Integer {v1(0)},
      requester        IssuerAndSerialNumber,
      nonce            Integer,
      serialNumber     Integer,
      messageImprint   SEQUENCE OF MessageImprint,
      procedureID      OBJECT IDENTIFIER,
      status           PKIStatusInfo
}
```

Data field	Description
versionID	To identify this syntax
requester	of the verification service
nonce	That matches the requesting nonce
serialNumber	of the time-certificate subject to verification
messageImprints	of the object subject to verification
procedureID	Applied by the TSA to carry out the verification
status	Verification outcome, either OK, or a reason for failure

The TSA must sign this token so all information related to TSA shall be included in a **SignerInfo** field of the PKCS#7 **SignedData** structure.

4.3.2 Certificate Retrieval Protocol

This protocol is also a two-way handshake protocol. The user sends a **TimeStampRetrvReq** data structure encapsulated in a PKCS#7 **SignedData** requesting one or more certificates to the TSA, and the time stamp authority responds with a **TimeStampRetrvToken** encapsulated in a PKCS#7 **SignedData** data structure.

TimeStampRetrvReq

The **TimeStampRetrvReq** protocol data unit is encapsulated as a PKCS#7 **SignedData** construct. The content type of the **contentInfo** field is the OID:

```
TimeStampRetrvReqId::= OBJECT IDENTIFIER { pkcs-7 2008 }
```

The content of the **contentInfo** field is the following **TimeStampRetrvReq** ASN.1 data structure:

```
TimeStampRetrvReq ::=SEQUENCE {
      versionID        Integer {v1(0)},
      nonce            Integer,
      tsa              IssuerAndSerialNumber,
      serialNumbers    SEQUENCE OF Integer,
      numberOfLinks    Integer OPTIONAL
}
```

Data field	Description
versionID	To identify this syntax
nonce	To uniquely identify this request
tsa	Identification of the TSA that issued the certificate, that must be the same one that is addressed now for verification
serialNumber	One or more, identifying requested certificates
numberOfLinks	[optional] for a linking TSA to retrieve *N* certificates before, and *N* certificates after, around each one of the identified certificates

TimeStampRetrvToken

The **TimeStampRetrvToken** protocol data unit is encapsulated as a PKCS#7 **SignedData** construct. The content type of the **contentInfo** field is the OID:

```
TimeStampRetrvTokenId::= OBJECT IDENTIFIER { pkcs-7 2009 }
```

The content of the **contentInfo** field is the following **TimeStampRetrvToken** ASN.1 data structure:

```
RetrvCertificates ::=SEQUENCE {
      status           PKIStatusInfo,
      certificate      SignedData OPTIONAL
}
```

Data field	Description
`status`	The status may be FOUND, or give a reason for the certificate not being available (bad TSA, bad serial number, not so many links, lost, ...)
`certificate`	Time certificate (if available) encapsulated in a PKCS#7 **SignedData** structure

```
TimeStampRetrvToken ::=SEQUENCE {
      versionID        Integer {v1(0)},
      requester        IssuerAndSerialNumber,
      nonce            Integer,
      certificates     SEQUENCE OF RetrvCertificates,
}
```

Data field	Description
`versionID`	to identify this syntax
`requester`	of the verification service
`nonce`	that matches the requesting nonce
`certificates`	as many as requested, each including both a request status and the requested certificate if available

The TSA must sign this token so all information related to TSA shall be included in a **SignerInfo** field of the PKCS#7 **SignedData** structure.

4.4 SYNCHRONISATION

The TSA may synchronise itself with other TSA to extend trust beyond its own universe of users.

To strengthen the system security, and for auditing purposes, the different TSAs in a network of trust will periodically time-stamp each other's linking values or activity reports. This administrative and maintenance procedure permits the comparison of time-stamps issued by different TSA limiting the risk of TSA misbehaviour (e.g. collusion, or clock deviations). Moreover, this synchronisation procedure also permits to the TSA to introduce additional external time references into its time certificate chains.

The synchronisation process has two phases:

In the first phase, the TSA requests the synchronisation process by submitting a time certification request to the other TSA. The document to certificate is historical information since last synchronisation time (for example, a hash calculated over the linking chain since the last synchronisation token (see later), or an activity report). The protocols used in this internal TSA maintenance process to synchronise with different TSA are the same protocols described in section 4.1 (Time Stamping). Each TSA issues a **TimeStampReq** data structure encapsulated in a PKCS#7 **SignedData** to others TSAs, and these respond with a **TimeStampToken** also encapsulated in a PKCS#7 **SignedData**.

Two TSA are involved in a synchronisation exchange. Let TSA1 be the requesting TSA, that submits a document to be time-stamped by TSA2.

If TSA1 uses the simple protocol, the document to time-stamp by TSA2 is the activity report of TSA1 since last synchronisation request. The request token shall include a mark that identifies the token as subject to synchronisation. This mark contains the identification of TSA2. This information will be used during auditing.

If TSA1 uses the linking protocol, the document to time-stamp by TSA2 is the current value of the link. The link value will be attached to a time-stamp token returned to some TSA1 requester (it might be a temporal data token inserted by TSA1 into its own chain of links). The containing token shall include a mark that identifies the link as subject to synchronisation. This mark shall contain the identification of TSA2. This information will be used during auditing, as well as when verifying a time-stamp certificate issued by TSA1.

In either case, the time-stamp token returned by TSA2 shall be time-stamped by TSA1 upon reception and included in its activity report and, if the linking protocol is used, into its chain of links.

TimeStampRequest

In a simple protocol and linking protocol a time stamp service synchronisation request is encapsulated in a PKCS#7 **SignedData** construct. This **SignedData** structure shall include a **SignerInfo** field with all the required information related to the requester TSA, and shall also include the following authenticated attributes:

- A **TimeStampProtocol** authenticated attribute with the value **simpleProtocol** or **linkingProtocol** should be included in the **SignedData** structure depending on the protocol the requester TSA wants to use.

- A **SynchronisedId** authenticated attribute shall be introduced, containing as value the identification of TSA2.

TimeStampToken

The target TSAs will receive this request and will respond with a **TimeStampToken** encapsulated in a PKCS#7 **SignedData** structure. All information related to TSA which issues this **TimeStampToken** shall be included in a **SignerInfo** field into the PKCS#7 **SignedData** structure.

Also, the following authenticated attributes shall be included:

- A **TimeStampProtocol** authenticated attribute with value **simpleProtocol** or **linkingProtocol** depending on the protocol specified in the synchronisation request.

- A **TimeStampLinks** authenticated attribute shall be present if linking Protocol is requested

- A **TimeStampRequester** authenticated attribute which identifies the requester TSA which asked for synchronisation

- A **TimeStampSN** authenticated attribute which is the serial number of the resulting **TimeStampToken** PKCS#7 **SignedData**.

- A **TimeStampValidity** authenticated attribute. However, the default validity indicated in the requested policy shall be assumed if this authenticated attributed is omitted.

- A **SynchronisedId** authenticated attribute shall be introduced, containing as value the identification of TSA2. It is the same attribute as used in the request phase.

Using this synchronisation process, the TSA acts as a time stamping service client and as a time stamping authority. In the first phase, it acts as a client that requests a time certificate over historical information. As a result, it gets a time certificate from other TSA involved in the PKI. In the second phase, it acts as a time stamping authority because it generates a new time certificate that encapsulates the received time certificate and that it will introduce into its own links chain.

4.5 AUTHENTICATED ATTRIBUTES AND PROTOCOLS OF SERVICE

The following tables summarise which authenticated attributes shall be used in the different protocols presented above.

Rows: authenticated attributes
Columns: Protocols data structures

	Time Stamping					
	Simple Protocol		Linking Protocol		Distributed Protocol	
	Time Stamp Request	Time Stamp Token	Time Stamp Request	Time Stamp Token	Time Stamp Request	Time Stamp Token
Time Stamp Protocol	Shall (value simple protocol)	Shall (value simple Protocol)	Shall (value linking Protocol.)	Shall (value linking Protocol.)	(2) Shall. Values simple or linking protocol	(2) Shall. Values simple or linking protocol
Time Stamp Requester	NA	Shall	NA	Shall	NA	Shall
Time Stamp SN	NA	Shall	NA	Shall	NA	Shall
Time Stamp Validity	NA	Optional (1)	NA	Optional (1)	NA	Optional (1)
Time Stamp Links	NA	NA	NA	Shall	NA	Shall if linking protocol is used. NA if simple protocol is used
Time Stamp Random	NA	NA	NA	NA	Shall	Shall

TimeStampRenewal , TimeStampSynchronisation and TimeStampSyncToken authenticated attributes are not applicable in TimeStamping requests.

	Time Stamping Renewal		Time Stamping Synchronisation		
	Time Stamp Request	Time Stamp Token	Time Stamp Request	Time Stamp Token (returned by others TSAs)	Time Stamp Token (Synchronisation point included by the TSA into its link chain)
Time Stamp Protocol	(3)	(3)	Only simple and linking protocol are applicable	Only simple and linking protocol are applicable	Shall. Value linking protocol
Time Stamp Requester	NA	Shall	NA	Shall	NA
Time Stamp SN	NA	Shall	NA	Shall	Shall
Time Stamp Validity	NA	Optional (1)	NA	Optional (1)	Optional (1)
Time Stamp Links	NA	Shall if linking or link.dist. protocol is used. NA if simple or distr. simple protocol is used	NA	Shall if linking protocol is used. NA if simple protocol is used	Shall
Time Stamp Random	Shall if distributed protocol is used	Shall if distributed protocol is used	NA	NA	NA
Time Stamp Renewal	Shall	Shall	NA	NA	NA
Time Stamp Synchronisation	NA	NA	Shall	Shall	Shall
Time Stamp Sync Token	NA	NA	NA	NA	Shall Encapsulates the TimeStampToken returned by the TSAs that issued the syncronisation token in the synchronisation process when received the synchronisation request.

See the following notes:

(1) Default validity period specified in the policy requested shall be assumed if it is omitted
(2) The same TimeStampProtocol authenticated attribute values in simple and linking protocol can be used, depending on the protocol the requester uses to interact with the TSAs involved in the distributed protocol
(3) The same TimeStampProtocol authenticated attribute values in simple, linking and distributed protocol can be used. An existing TimeStampToken or TimeStampDistrbToken can be renewed with simple, linking or distributed protocol.

5 ACCESS MEANS

The previously described formats for provision of the time-stamping service may be carried out over a number of transfer protocols to achieve a client-server application, where the TSA plays the server role, and service users are the clients. Four basic means are identified below:

	Web	**E-mail**	**TCP/IP**	**FT**
frequency of use	medium	low	high	low
user	human	human	application	any
interactive	interactive	batch	interactive	batch
volume	light	medium	heavy	heavy
number of certificates	one by one	one or more	one or more	one or more

Whatever the access means is, the client is expected to submit properly packaged requests, and to be able to understand properly formatted responses, as well as to provide the means to store evidence as needed for future use. This statement has a serious impact on the need to provide adequate client user interfaces to

- collect needed information (e.g. the list of hash values)
- pack the information and sign it
- request the service and interpret the response
- store evidence locally

5.1 HUMAN ORIENTED WEB SERVICE

This subsection specifies a possible procedure to access a Time Stamping Service via the HyperText Transfer Protocol (HTTP) [RFC 2068].

User must access the TSA web site and a form will be downloaded to his/her browser. This form will be paired with a local application that may be an applet, a plug-in, or whatever procedure is available to allow for the previously mentioned tasks to perform. For the time being, for seldom use, a Java applet may be a fine option, but others may appear in the future, or client applications may be widely distributed, or even embedded within popular web browsers.

The client interface will allow the user to fill in the required information:

- identification of the document to be signed
- election of hash functions
- election of TSA, and time-stamping policy

The applet will collect the information, calculate the needed hash values, construct and sign the PKCS-7 request, and send it over HTTP as a POST method.

The TSA responds immediately over the same HTTP session, and the client application must be able to

- interpret the response
- present it to the user
- permit the user to perform its verifications
- allow for local storage of evidence, altogether with document identification, and every parameter in the request

The client application should generate enough log information as to track activity.

The response from the TSA has two components: there is an HTTP level response that may include a PKCS-7 response in the body. There are a large number of HTTP/1.1 response codes

that will be interpreted as usual by the web browser (see [RFC 2068] for details). Only if the response is "200 OK" we can proceed to analyse the PKCS-7 response in the body that will provide detailed results concerning the time-stamping activity.

Time-stamping service requests, renewal, and verification requests may be handled as described. Synchronisation makes not sense over HTTP.

Besides the core time-stamping service, TSA may provide a storage service that keeps time-stamping certificates on-line for users retrieval. A useful application would return a time-stamp certificate upon user request, altogether with hyperlinks to relevant authenticity certificates that back TSA and user's identities. In the linking protocol, hyperlinks to previous and posterior time-stamp certificates may be added as well. Lastly, when there are synchronisation points, a link to the other(s) time-stamping authority can be provided to follow the chain. The result of a web session following certificates will permit to download into local storage of as many additional certificates as needed for local (off-line) verification. The technology to provide this additional service is based on dynamic HTML generation. It is not a mandatory service for a TSA to provide, nor there is a need for a standardisation, but it may prove a valuable service for service users.

5.2 INTERACTIVE TCP/IP BASED SERVICE

The following simple socket-based protocol is suitable for cases where the time-stamping service is initiated by an application: the service request and response analysis is embedded within a bigger application. Still the service is interactive.

The protocol basically assumes a listener process on a TSA which can accept time-stamping requests on a well-defined port. The transport socket is established upon client request, establishing a two-way reliable transmission channel that, once established may be used for more than one transaction. This architecture permits batch operation: the client application prepares a bunch of requests, and processes them in a row. The architecture is oriented towards unattended services without human intervention.

The protocol goes through three stages:

- connection establishment
- time-stamping exchanges
- disconnection

Connection establishment shall identify the requester, and this identification will be the only one that may be used along the individual transactions to be carried on later on the established channel.

Time-stamping requests and responses shall be PKCS-7 packets as described above. An error response mentioning inadequate identification of the requester shall be the response when the identity of the signer of one individual request is not the same one as the one that established the channel. Each request shall be valid by itself, without regard to the preceding or following ones, in particular, each request shall be independently signed.

The cycle request-response may be carried out many times during a session. There is no parallel flow of activity, and the same bidireccional channel is used both for requesting and responding. Therefore, requests are implicitly serialised along a single session, although there is no fundamental restriction to have more than one session simultaneously open between a client and a server.

Disconnection may be initiated by the client or the server for a number of reasons:

- client: no further service is required

- client or server: time out
- server: service cannot be further provided

The client application shall provide the standard means to collect local information, to keep records of activity, to store needed evidence, and to handle errors (e.g. requesting user help before retrying).

Connection establishment requires identification of the requester. Since this identification must match the identification provided when the service is requested, the user shall provide its certificate(s) as part of the connection protocol. A traditional protocol may be used to establish mutual authentication between the parties; this is not required by the time-stamping service that has its own security mechanisms, but the establishment of a secure channel from the very beginning simplifies the situations to be handled along the time-stamping phase.

The following exchanges establish a channel:

1. the client says HELLO, and provides a nonce X, and its certificate(s)
2. the server responds with another nonce Y, and its certificated(s) of identity
3. the client digitally signs X•Y, and sends it to the server
4. the server digitally signs X•Y, and sends it to the client

A clear channel is established with the guarantees that the client and the server are the intended ones, and that no third party has spoofed any identity; both parties have the certificates of the other one for the necessary checks about service provision constraints that may force an immediate channel disconnection if constraints are not fulfilled.

Certificates concerning the parties are exchanged once upon connection establishment, and are not needed later on along the time-stamping cycle.

Either party may establish an upper bound on the maximum number of documents to be time-stamped during a TCP session (in the order of hundreds), and on the maximum allowed period for a single session to remain (in the order of hours).

A established session may be used to exchange any kind of time-stamping requests and responses: basic time-stamping, renewal, verification, and synchronisation.

5.3 E-MAIL BASED SERVICE

This section specifies a means for time-stamping documents using electronic mail. E-mail may be used to time-stamp a single document, or a larger number of them. Users may interact manually with the mail agent, or the mail agent may be exercised by another application as transport mechanism. In the first situation, manual interaction, the user is expected to have external tools to prepare requests, and to receive responses.

The core idea is to provide a MIME envelope to PKCS-7 objects as identified above. The rules to produce secure MIME objects [RFC 1847] will follow S/MIME [RFC 2311, RFC 2312] recommendations. Notice that E-mail is used as a transport agent, and therefore S/MIME is not interfering with the inners of PKCS-7 objects exchanged.

A MIME envelope will be defined that permits

- to exchange data transparently
- to pack several requests and responses in a single bundle
- to provide the needed certificates to identify the requester once in a bundle encompassing several individual requests
- to provide the needed certificates to authenticate the time-stamping authority once ina bundle encompassing several individual responses

A client may send a TSA a large e-mail with a large number of individual requests, each one properly identified by its nonce. The user is required to keep track of sent requests. The TSA responds by sending another bunch of responses that the user shall take under consideration to match pending requests. It is not mandatory that the TSA bundles responses in the same pack as the client does. Therefore, the client application is expected to keep track of the state of pending requests. Tracking may be simple for humans issuing individual requests, or require database support for applications sending large packs.

A sending agent **should** include at least one chain of certificates up to, but not including, a Certificate Authority (CA) that it believes that the recipient may trust as authoritative. A receiving agent **should** be able to handle an arbitrarily large number of certificates and chains. The appropriate certificates **shall** be included once in the S/MIME package, using a degenerate signed message (an empty information with a PKCS-7 envelope). These certificates shall be checked once by the receiver to decide whether the other party is properly identified and service may be acknowledged. The other pieces of the message are signed requests and signed responses, respectively signed by the service requester and the service provider.

A sample message would be:

```
Content-Type: multipart/mixed; boundary=***

--***
Content-Type: application/pkcs7-mime;
      smime-type=certs-only; name=USERcert.p7c
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=USERcert.p7c

... ... ...

--***

Content-Type: application/pkcs-mime;
      smime-type=time-stamp-request; name=reqXXX.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=reqXXX.p7m

... ... ...

--***--
```

A sample response message would be:

```
Content-Type: multipart/mixed; boundary=***

--***
Content-Type: application/pkcs7-mime;
      smime-type=certs-only; name=TSAcert.p7c
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=TSAcert.p7c

... ... ...

--***
```

```
Content-Type: application/pkcs-mime;
     smime-type=time-stamp-response; name=ansXXX.p7m
Content-Transfer-Encoding: base64
Content-Disposition: attachment; filename=ansXXX.p7m

... ... ...

--***--
```

A renewal request follows the very same pattern, although the contents of the PKCS-7 will show the aim of the request.

A verification request will have its own type:

```
smime-type=time-stamp-verification
```

Synchronisation requests may use this access means transparently.

5.4 OFF-LINE FT SERVICE

This format is intended for batch transactions. S/MIME will be used to make a pack of requests, that is transferred to the TSA for certification. The TSA responds with another pack of responses.

This protocol may use the popular Internet FTP [RFC 959] or any other file transfer means, such as diskettes, streamers, etc.

The client shall have tools to generate PKCS-7 individual requests, to pack bunches of requests with a single certificate, to analyse responses, and to keep track of requests and responses to keep final users informed of the state of the time-stamping service.

6 SCENARIOS OF USE

The protocols and means described in this document will be used in a large number of situations to provide time-stamping services to users. Four means of access have been identified above: web, e-mail, TCP/IP, and file transfer (FT). Web and e-mail are intended for light use by individuals, while TCP/IP is intended for heavy automated use by remote applications. File transfer may play a role in off-line bulk time-stamping. The aforementioned basic use will not preclude any access mean to be used in any circumstance besides its basic intended scenario.

In order to dig further into the understanding of the provision of the service, let us consider the point of view of where is the TSA located: either in the public sector or in the private sector:

- A TSA in the public sector may provide time-stamping services to citizens, to private entities, and to other public entities. Sometimes, an intermediary authority will act as a front end as in the case of Registry Authorities, and Notaries.

- A TSA in the private sector may provide time-stamping services to customers, that may be individuals or other companies (clients and providers).

When an individual addresses a TSA, it may use either an interactive interface or a batch interface. For an interactive service, the use of WWW is the most natural one; while for non-interactive service, the e-mail interface will be preferred. An interactive service will provide a resolution of seconds, while an e-mail service will provide a response time of hours (at most a few days) with a resolution of minutes. The client will be required to identify itself to the authority as entitled to use the service. The situation may be quite similar for citizens that need to belong to the scope of the public TSA, and for customers of a private TSA.

Individuals may use the TSS for non repudiation of origin of their documents. That is an individual action that results in a time-stamp to be used in the future for any purpose.

Time-stamps will be frequently used to support transactions:

- An individual submits a document to the public administration, and requests a time-stamped response (e.g. via a Registry Authority). Either www or e-mail will be the preferred means of request and response, and the time-stamping service will be an add-on protocol associated with the basic transfer protocol. May be the Registry Authority is a TSA by itself, or it may request the service from a subsidiary TSA as an internal protocol. The end user only receives the last, integrated, bunch of signatures.

  Some authorities may work in real time, as may be frequent of RA for routine submissions, while others may work off-line, as may be usual of notaries.

- An individual performs a transaction with a private entity, either to buy or sell something, and request a proof for non-repudiation of the other partner. Similar scenario as before.

- A private company trades with the public administration. In this situation it is unlikely to use interactive procedures, but rather a batch procedure, that may be e-mail based or, more naturally, a FT-based exchange. If there is a frequent relationship between the entities, it makes sense to establish a direct channel using the TCP-based protocol for many atomic time-stamping operations.

- Private trading may again be seldom or frequent, going for the batch (FT) or the interactive, TCP, protocols respectively.

Using a public or a private TSA may have an impact on the value of the time-stamp, depending on the agreements of the involved TSA with third parties: a public TSA may be automatically authoritative within the public administration, while a private entity may need a previous recognition agreement. An agreement is needed as well between public entities to extend the value beyond administrative borders. If the time-stamp is to be used in court, the practice statements and the synchronisation policy of the TSA will play a key role in determining the actual value of the time-stamp.

TSA will need to synchronise eventually to extend its value to third parties, as specified in the practice statement. TSA synchronisation may be frequent or seldom. Frequent (and most usual) synchronisation requires an unattended interactive operation over a single link token, a service that is perfectly provided by the TCP interface. Rare synchronisation frequency may be performed synchronous or asynchronously. It depends on the quality of the service that is promised to the clients.

The linking protocol poses a difficult issue for interactive protocols: knowledge of the identity of the next user, since it is part of the certificate. It may not be acceptable to delay the response until another request is satisfied by the TSA. Notice that some TSA may be quite busy, while others may issue time-stamps from time to time. In order to provide a quick response, the TSA will return an indirect link to the identity of the next certificate (e.g. a URL in the www scenario). This link points nowhere when the current certificate is issued, and will be filled with meaningful content when the next certificate is issued. Since the time window of resolution of a time-stamp is closed by the next event, it may not delay for any time, but an upper bound is to be established in the practice statement. The link may be easily frozen as a side effect of synchronising with a second TSA. In order to keep the system homogeneous, the first TSA will include an unpredictable event (from a TDS), and then run a synchronising step with a second

TSA: the time-stamp of the unpredictable event fills the indirect link, and the second TSA puts an upper time bound.

Indirect links are as well the mechanism for later verification. According the practice statement of the TSA, these links will be kept in-line for a given period, then off-line for another period, and eventually may be removed (if the TSPS allows for this removal). Links must be unique for the activity period of the TSA, and the TSA identity must be part of the link identification.

Following links for provision of evidence may be an interactive work (e.g. using a www browser to navigate along the links) or a batch work (e.g. requesting a jury to provide a chain of evidence in a row). It must be noted that links are initially unidimensional (the chain of links of the linking protocol) but become quick multidimensional when synchronising events establish relations between two chains, and when there are renewal processes that relate largely separated events in the chain. A last source of complexity to the graph is introduced by the use of more than one TSA to time-stamp a document. Going beyond a few steps may turn verification into a complex task, difficult to automate. When robots are used to track chains of links, the TCP protocol becomes a must.

7 APPENDIX A: EXTRACT OF “PKCS#7:CRYPTOGRAPHIC MESSAGE SINTAX STANDARD”

An RSA Laboratories Technical Note
Version 1.5
Revised November 1, 1993

1.Definitions

For the purposes of this standard, the following definitions apply.

AlgorithmIdentifier: A type that identifies an algorithm (by object identifier) and associated parameters. This type is defined in X.509.

ASN.1: Abstract Syntax Notation One, as defined in X.208.

Attribute: A type that contains an attribute type (specified by object identifier) and one or more attribute values. This type is defined in X.501.

BER: Basic Encoding Rules, as defined in X.209.

Certificate: A type that binds an entity's distinguished name to a public key with a digital signature. This type is defined in X.509. This type also contains the distinguished name of the certificate issuer (the signer), an issuer-specific serial number, the issuer's signature algorithm identifier, and a validity period.

CertificateSerialNumber: A type that uniquely identifies a certificate (and thereby an entity and a public key) among those signed by a particular certificate issuer. This type is defined in X.509.

CertificateRevocationList: A type that contains information about certificates whose validity an issuer has prematurely revoked. The information consists of an issuer name, the time of issue, the next scheduled time of issue, and a list of certificate serial numbers and their associated revocation times. The CRL is signed by the issuer. The type intended by this standard is the one defined RFC 1422.

DER: Distinguished Encoding Rules for ASN.1, as defined in X.509, Section 8.7.

DES: Data Encryption Standard, as defined in FIPS PUB 46-1.

desCBC: The object identifier for DES in cipher-block chaining (CBC) mode, as defined in [NIST91].

ExtendedCertificate: A type that consists of an X.509 public- key certificate and a set of attributes, collectively signed by the issuer of the X.509 public-key certificate. This type is defined in PKCS #6.
MD2: RSA Data Security, Inc.'s MD2 message-digest algorithm, as defined in RFC 1319.

md2: The object identifier for MD2, as defined in RFC 1319.

MD5: RSA Data Security, Inc.'s MD5 message-digest algorithm, as defined in RFC 1321.

md5: The object identifier for MD5, as defined in RFC 1321.

Name: A type that uniquely identifies or "distinguishes" objects in an X.500 directory. This type is defined in X.501. In an X.509 certificate, the type identifies the certificate issuer and the entity whose public key is certified.

PEM: Internet Privacy-Enhanced Mail, as defined in RFCs 1421-1424.

RSA: The RSA public-key cryptosystem, as defined in [RSA78].

rsaEncryption: The object identifier for RSA encryption, as defined in PKCS #1.

2.Overview

The syntax is general enough to support many different content types. This standard defines six: data, signed data, enveloped data, signed-and-enveloped data, digested data, and encrypted data. Other content types may be added in the future. The use of content types defined outside this standard is possible, but is subject to bilateral agreement between parties exchanging content.

This standard exports one type, ContentInfo, as well as the various object identifiers.

There are two classes of content types: base and enhanced. Content types in the base class contain "just data," with no cryptographic enhancements. Presently, one content type is in this class, the data content type. Content types in the enhanced class contain content of some type (possibly encrypted), and other cryptographic enhancements. For example, enveloped-data content can contain (encrypted) signed-data content, which can contain data content. The four non-data content types fall into the enhanced class. The content types in the enhanced class thus employ encapsulation, giving rise to the terms "outer" content (the one containing the enhancements) and "inner" content (the one being enhanced).

3.Useful types

This section defines types that are useful in at least two places in the standard.

3.1 CertificateRevocationLists

The CertificateRevocationLists type gives a set of certificate-revocation lists. It is intended that the set contain information sufficient to determine whether the certificates with which the set is associated are "hot listed," but there may be more certificate-revocation lists than necessary, or there may be fewer than necessary.

CertificateRevocationLists ::= SET OF CertificateRevocationList

3.2 ContentEncryptionAlgorithmIdentifier

The ContentEncryptionAlgorithmIdentifier type identifies a content-encryption algorithm such as DES. A content- encryption algorithm supports encryption and decryption operations. The encryption operation maps an octet string (the message) to another octet string (the ciphertext) under control of a content-encryption key. The decryption operation is the inverse of the encryption operation. Context determines which operation is intended.

ContentEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

3.3 DigestAlgorithmIdentifier

The DigestAlgorithmIdentifier type identifies a message- digest algorithm. Examples include MD2 and MD5. A message- digest algorithm maps an octet string (the message) to another octet string (the message digest).

DigestAlgorithmIdentifier ::= AlgorithmIdentifier

3.4 DigestEncryptionAlgorithmIdentifier

The DigestEncryptionAlgorithmIdentifier type identifies a digest-encryption algorithm under which a message digest can be encrypted. One example is PKCS #1's rsaEncryption. A digest-encryption algorithm supports encryption and decryption operations. The encryption operation maps an octet string (the message digest) to another octet string (the encrypted message digest) under control of a digest- encryption key. The decryption operation is the inverse of the encryption operation. Context determines which operation is intended.

DigestEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

3.5 ExtendedCertificateOrCertificate

The ExtendedCertificateOrCertificate type gives either a PKCS #6 extended certificate or an X.509 certificate. This type follows the syntax recommended in Section 6 of PKCS #6:

```
ExtendedCertificateOrCertificate ::= CHOICE {
  certificate Certificate, -- X.509
  extendedCertificate [0] IMPLICIT ExtendedCertificate
}
```

3.6 ExtendedCertificatesAndCertificates

The ExtendedCertificatesAndCertificates type gives a set of extended certificates and X.509 certificates. It is intended that the set be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the signers with which the set is associated, but there may be more certificates than necessary, or there may be fewer than necessary.

ExtendedCertificatesAndCertificates ::= SET OF ExtendedCertificateOrCertificate

Note. The precise meaning of a "chain" is outside the scope of this standard. Some applications of this standard may impose upper limits on the length of a chain; others may enforce certain relationships between the subjects and issuers of certificates in a chain. An example of such relationships has been proposed for Privacy-Enhanced Mail in RFC 1422.

3.7 IssuerAndSerialNumber

The IssuerAndSerialNumber type identifies a certificate (and thereby an entity and a public key) by the distinguished name of the certificate issuer and an issuer-specific certificate serial number.

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

}

3.8 KeyEncryptionAlgorithmIdentifier

The KeyEncryptionAlgorithmIdentifier type identifies a key-encryption algorithm under which a content-encryption key can be encrypted. One example is PKCS #1's rsaEncryption. A key-encryption algorithm supports encryption and decryption operations. The encryption operation maps an octet string (the key) to another octet string (the encrypted key) under control of a key-encryption key. The decryption operation is the inverse of the encryption operation. Context determines which operation is intended.

KeyEncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

3.9 Version

The Version type gives a syntax version number, for compatibility with future revisions of this standard.

Version ::= INTEGER

4. General syntax

The general syntax for content exchanged between entities according to this standard associates a content type with content. The syntax shall have ASN.1 type ContentInfo:

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

ContentType ::= OBJECT IDENTIFIER
```

The fields of type ContentInfo have the following meanings:

- contentType indicates the type of content. It isan object identifier, which means it is a unique string of integers assigned by the authority that defines the content type. This standard defines six content types (see Section 14): data, signedData, envelopedData, signedAndEnvelopedData, digestedData, and encryptedData.

- content is the content. The field is optional, and if the field is not present, its intended value must be supplied by other means. Its type is defined along with the object identifier for contentType.

Notes.

1. The methods below assume that the type of content can be determined uniquely by contentType, so the type defined along with the object identifier should not be a CHOICE type.

2. When a ContentInfo value is the inner content of signed-data, signed-and-enveloped-data, or digested-data content, a message-digest algorithm is applied to the contents octets of the DER encoding of the content field. When a ContentInfo value is the inner

content of enveloped-data or signed-and-enveloped-data content, a content-encryption algorithm is applied to the contents octets of a definite-length BER encoding of the content field.

3. The optional omission of the content field makes it possible to construct "external signatures," for example, without modification to or replication of the content to which the signatures apply. In the case of external signatures, the content being signed would be omitted from the "inner" encapsulated ContentInfo value included in the signed-data content type.

5. Data content type

The data content type is just an octet string. It shall have ASN.1 type Data:

Data ::= OCTET STRING

The data content type is intended to refer to arbitrary octet strings, such as ASCII text files; the interpretation is left to the application. Such strings need not have any internal structure although they may; they could even be DER encodings).

6. Signed-data content type

The signed-data content type consists of content of any type and encrypted message digests of the content for zero or more signers. The encrypted digest for a signer is a "digital signature" on the content for that signer. Any type of content can be signed by any number of signers in parallel. Furthermore, the syntax has a degenerate case in which there are no signers on the content. The degenerate case provides a means for disseminating certificates and certificate-revocation lists.

It is expected that the typical application of the signed-data content type will be to represent one signer's digital signature on content of the data content type. Another typical application will be to disseminate certificates and certificate-revocation lists.

The process by which signed data is constructed involves the following steps:

1. For each signer, a message digest is computed on the content with a signer-specific message-digest algorithm. (If two signers employ the same message-digest algorithm, then the message digest need be computed for only one of them.) If the signer is authenticating any information other than the content (see Section 6.2), the message digest of the content and the other information are digested with the signer's message digest algorithm, and the result becomes the "message digest."

2. For each signer, the message digest and associated information are encrypted with the signer's private key.

3. For each signer, the encrypted message digest and other signer-specific information are collected into a SignerInfo value, defined in Section 6.2. Certificates and certificate-revocation lists for each signer, and those not corresponding to any signer, are collected in this step.

4. The message-digest algorithms for all the signers and the SignerInfo values for all the signers are collected together with the content into a SignedData value, defined in Section 6.1.

A recipient verifies the signatures by decrypting the encrypted message digest for each signer with the signer's public key, then comparing the recovered message digest to an independently computed message digest. The signer's public key is either contained in a certificate included in the signer information, or is referenced by an issuer distinguished name and an issuer-specific serial number that uniquely identify the certificate for the public key.

6.1 SignedData type

The signed-data content type shall have ASN.1 type SignedData:

```
 SignedData ::= SEQUENCE {
  version Version,
  digestAlgorithms DigestAlgorithmIdentifiers,
  contentInfo ContentInfo,
  certificates     [0] IMPLICIT ExtendedCertificatesAndCertificates OPTIONAL,
  crls [1] IMPLICIT CertificateRevocationLists OPTIONAL,
  signerInfos SignerInfos
}

DigestAlgorithmIdentifiers ::=  SET OF DigestAlgorithmIdentifier

SignerInfos ::= SET OF SignerInfo
```

The fields of type SignedData have the following meanings:

- Version is the syntax version number. It shall be 1 for this version of the standard.

- digestAlgorithms is a collection of message-digest algorithm identifiers. There may be any number of elements in the collection, including zero. Each element identifies the message-digest algorithm (and any associated parameters) under which the content is digested for a some signer. The collection is intended to list the message-digest algorithms employed by all of the signers, in any order, to facilitate one-pass signature verification. The message-digesting process is described in Section 6.3.

- contentInfo is the content that is signed. It can have any of the defined content types.

- certificates is a set of PKCS #6 extended certificates and X.509 certificates. It is intended that the set be sufficient to contain chains from a recognized "root" or "top-level certification authority" to all of the signers in the signerInfos field. There may be more certificates than necessary, and there may be certificates sufficient to contain chains from two or more independent top-level certification authorities. There may also be fewer certificates than necessary, if it is expected that those verifying the signatures have an alternate means of obtaining necessary certificates (e.g., from a previous set of certificates).

- crls is a set of certificate-revocation lists. It is intended that the set contain information sufficient to determine whether or not the certificates in the certificates field are "hot listed," but such correspondence is not necessary. There may be more certificate-revocation lists than necessary, and there may also be fewer certificate-revocation lists than necessary.

- signerInfos is a collection of per-signer information. There may be any number of elements in the collection, including zero.

Notes.

1. The fact that the digestAlgorithms field comes before the contentInfo field and the signerInfos field comes after it makes it possible to process a SignedData value in a single pass.

2. The differences between version 1 SignedData and version 0 SignedData (defined in PKCS #7, Version 1.4) are the following:

   - the digestAlgorithms and signerInfos fields may contain zero elements in version 1, but not in version 0

   - the crls field is allowed in version 1, but not in version 0

   Except for the difference in version number, version 0 SignedData values are acceptable asversion 1 values. An implementation can therefore process SignedData values of either version as though they were version 1 values. It is suggested that PKCS implementations generate only version 1 SignedData values, but be prepared to process SignedData values of either version.

3. In the degenerate case where there are no signers on the content, the ContentInfo value being "signed" is irrelevant. It is recommended in that case that the content type of the ContentInfo value being "signed" be data, and the content field of the ContentInfo value be omitted.

6.2 SignerInfo type

Per-signer information is represented in the type SignerInfo:

```
SignerInfo ::= SEQUENCE {
  version Version,
  issuerAndSerialNumber IssuerAndSerialNumber,
  digestAlgorithm DigestAlgorithmIdentifier,
  authenticatedAttributes    [0] IMPLICIT Attributes OPTIONAL,
  digestEncryptionAlgorithm    DigestEncryptionAlgorithmIdentifier,
  encryptedDigest EncryptedDigest,
  unauthenticatedAttributes [1] IMPLICIT Attributes OPTIONAL
}

EncryptedDigest ::= OCTET STRING
```

The fields of type SignerInfo have the following meanings:

- version is the syntax version number. It shall be1 for this version of the standard.

- issuerAndSerialNumber specifies the signer's certificate (and thereby the signer's distinguished name and public key) by issuer distinguished name and issuer-specific serial number.

- digestAlgorithm identifies the message-digest algorithm (and any associated parameters) under which the content and authenticated attributes (if present) are digested. It should be among those in the digestAlgorithms field of the superior SignerInfo value. The message-digesting process is described in Section 6.3.

- authenticatedAttributes is a set of attributes that are signed (i.e., authenticated) by the signer. The field is optional, but it must be present if the content type of the ContentInfo value being signed is not data. If the field is present, it must contain, at a minimum, two

attributes:

1. A PKCS #9 content-type attribute having as its value the content type of the ContentInfo value being signed.

2. A PKCS #9 message-digest attribute, having as its value the message digest of the content (see below).

Other attribute types that might be useful here, such as signing time, are also defined in PKCS #9.

- digestEncryptionAlgorithm identifies the digest-encryption algorithm (and any associatedparameters) under which the message digest andassociated information are encrypted with the signer's private key. The digest-encryption process is described in Section 6.4.

- encryptedDigest is the result of encrypting the message digest and associated information with the signer's private key.

- unauthenticatedAttributes is a set of attributes that are not signed (i.e., authenticated) by thesigner. The field is optional. Attribute types that might be useful here, such as countersignatures, are defined in PKCS #9.

Notes.

1. It is recommended in the interest of PEM compatibility that the authenticatedAttributes field be omitted whenever the content type of the ContentInfo value being signed is data and there are no other authenticated attributes.

2. The difference between version 1 SignerInfo and version 0 SignerInfo (defined in PKCS #7, Version 1.4) is in the message-digest encryption process (see Section 6.4). Only the PEM-compatible processes are different, reflecting changes in Privacy-Enhanced Mail signature methods. There is no difference in the non-PEM-compatible message-digest encryption process.

3. It is suggested that PKCS implementations generate only version 1 SignedData values. Since the PEM signature method with which version 0 is compatible is obsolescent, it is suggested that PKCS implementations be prepared to receive only version 1 SignedData values.

6.3 Message-digesting process

The message-digesting process computes a message digest on either the content being signed or the content together with the signer's authenticated attributes. In either case, the initial input to the message-digesting process is the "value" of the content being signed. Specifically, the initial input is the contents octets of the DER encoding of the content field of the ContentInfo value to which the signing process is applied. Only the contents octets of the DER encoding of that field are digested, not the identifier octets or the length octets.

The result of the message-digesting process (which is called, informally, the "message digest") depends on whether the authenticatedAttributes field is present. When the field is absent, the result is just the message digest of the content. When the field is present, however, the result is the message digest of the complete DER encoding of the Attributes value containted in the authenticatedAttributes field. (For clarity: The IMPLICIT [0] tag in the authenticatedAttributes field is not part of the Attributes value. The Attributes value's tag is SET OF, and the DER

encoding of the SET OF tag, rather than of the IMPLICIT [0] tag, is to be digested along with the length and contents octets of the Attributes value.) Since the Attributes value, when the field is present, must contain as attributes the content type and the message digest of the content, those values are indirectly included in the result.

When the content being signed has content type data and the authenticatedAttributes field is absent, then just the value of the data (e.g., the contents of a file) is digested. This has the advantage that the length of the content being signed need not be known in advance of the encryption process. This method is compatible with Privacy-Enhanced Mail.

Although the identifier octets and the length octets are not digested, they are still protected by other means. The length octets are protected by the nature of the message-digest algorithm since it is by assumption computationally infeasible to find any two distinct messages of any length that have the same message digest. Furthermore, assuming that the content type uniquely determines the identifier octets, the identifier octets are protected implicitly in one of two ways: either by the inclusion of the content type in the authenticated attributes, or by the use of the PEM-compatible alternative in Section 6.4 which implies that the content type is data.

Note. The fact that the message digest is computed on part of a DER encoding does not mean that DER is the required method of representing that part for data transfer. Indeed, it is expected that some implementations of this standard may store objects in other than their DER encodings, but such practices do not affect message-digest computation.

6.4 Digest-encryption process

The input to the digest-encryption process--the value supplied to the signer's digest-encryption algorithm--includes the result of the message-digesting process (informally, the "message digest") and the digest algorithm identifier (or object identifier). The result of the digest-encryption process is the encryption with the signer's private key of the BER encoding of a value of type DigestInfo:

```
DigestInfo ::= SEQUENCE {
  digestAlgorithm DigestAlgorithmIdentifier,
  digest Digest
}

Digest ::= OCTET STRING
```

The fields of type DigestInfo have the following meanings:

- digestAlgorithm identifies the message-digest algorithm (and any associated parameters) under which the content and authenticated attributes are digested. It should be the same as the digestAlgorithm field of the superior SignerInfo value.

- digest is the result of the message-digesting process.

Notes.

1. The only difference between the signature process defined here and the signature algorithms defined in PKCS #1 is that signatures there are represented as bit strings, for consistency with the X.509 SIGNED macro. Here, encrypted message digests are octet strings.

2. The input to the encryption process typically will have 30 or fewer octets. If digestEncryptionAlgorithm is PKCS #1's rsaEncryption, then this means that the input can be encrypted in a single block as long as the length of the RSA modulus is at least 328 bits, which is reasonable and consistent with security recommendations.

3. A message-digest algorithm identifier is included in the DigestInfo value to limit the damage resulting from the compromise of one message- digest algorithm. For instance, suppose an adversary were able to find messages with a given MD2 message digest. That adversary could then forge a signature by finding a message with the same MD2 message digest as one that a signer previously signed, and presenting the previous signature as the signature on the new message. This attack would succeed only if the signer previously used MD2, since the DigestInfo value contains the message-digest algorithm. If a signer never trusted the MD2 algorithm and always used MD5, then the compromise of MD2 would not affect the signer. If the DigestInfo value contained only the message digest, however, the compromise of MD2 would affect signers that use any message-digest algorithm.

4. There is potential for ambiguity due to the fact that the DigestInfo value does not indicate whether the digest field contains just the message digest of the content or the message digest of the complete DER encoding of the authenticatedAttributes field. In other words, it is possible for an adversary to transform a signature on authenticated attributes to one that appears to be just on content by changing the content to be the DER encoding of the authenticatedAttributes field, and then removing the authenticatedAttributes field. (The reverse transformation is possible, but requires that the content be the DER encoding of an authenticated attributes value, which is unlikely.) This ambiguity is not a new problem, nor is it a significant one, as context will generally prevent misuse. Indeed, it is also possible for an adversary to transform a signature on a certificate or certificate-revocation list to one that appears to be just on signed-data content.

7. Object identifiers

This standard defines seven object identifiers: pkcs-7, data, signedData, envelopedData, signedAndEnvelopedData, digestedData, and encryptedData.

The object identifier pkcs-7 identifies this standard.

pkcs-7 OBJECT IDENTIFIER ::={ iso(1) member-body(2) US(840) rsadsi(113549)pkcs(1) 7 }

The object identifiers data, signedData, envelopedData, signedAndEnvelopedData, digestedData, and encryptedData, identify, respectively, the data, signed-data, enveloped-data, signed-and-enveloped-data, digested-data, and encrypted-data content types.

data OBJECT IDENTIFIER ::= { pkcs-7 1 }
signedData OBJECT IDENTIFIER ::= { pkcs-7 2 }
envelopedData OBJECT IDENTIFIER ::= { pkcs-7 3 }
signedAndEnvelopedData OBJECT IDENTIFIER ::= { pkcs-7 4 }
digestedData OBJECT IDENTIFIER ::= { pkcs-7 5 }
encryptedData OBJECT IDENTIFIER ::= { pkcs-7 6 }

These object identifiers are intended to be used in the contentType field of a value of type ContentInfo (see Section 5). The content field of that type, which has the content-type-specific syntax ANY DEFINED BY contentType, would have ASN.1 type Data, SignedData, envelopedData, SignedAndEnvelopedData, DigestedData, and EncryptedData, respectively. These object identifiers are also intended to be used in a PKCS #9 content-type attribute.

8 APPENDIX B: EXTRACT OF “INTERNET PUBLIC KEY INFRASTRUCTURE: CERTIFICATE MANAGEMENT PROTOCOLS” [CMP]: PKISTATUSINFO

3.2.3 Status codes and Failure Information for PKI messages

All response messages will include some status information. The following values are defined.

```
PKIStatus ::= INTEGER {
        granted                 (0),    -- you got exactly what you asked for
        grantedWithMods         (1),    -- you got something like what you asked for; the
                                        -- requester is responsible for ascertaining the
                                        --differences
        rejection               (2),    -- you don't get it, more information
                                        --elsewhere in the message
        waiting                 (3),    -- the request body part has not yet been processed,
                                        -- expect to hear more later
        revocationWarning       (4),    -- this message contains a warning that a revocation is
                                        -- imminent
        revocationNotification  (5),    -- notification that a revocation has occurred
        keyUpdateWarning        (6)     -- update already done for the oldCertId specified in
                                        -- the key update request message
}
```

Responders may use the following syntax to provide more information about failure cases.

```
 PKIFailureInfo ::= BIT STRING {
        -- since we can fail in more than one way!
        -- More codes may be added in the future if/when required.
        badAlg                  (0),    -- unrecognized or unsupported Algorithm Identifier
        badMessageCheck         (1),    -- integrity check failed (e.g., signature did not verify)
        badRequest              (2),    -- transaction not permitted or supported
        badTime                 (3),    -- messageTime was not sufficiently close to the
                                        -- system time as defined by local policy
        badCertId               (4),    -- no certificate could be found matching the provided
                                         --criteria
        badDataFormat           (5),    -- the data submitted has the wrong format
        wrongAuthority          (6),    -- the authority indicated in the request is different
                                        -- from the one creating the response token
        incorrectData           (7),    -- the requester's data is incorrect (used for notary
                                        -- services)
        missingTimeStamp        (8)     -- when the timestamp is missing but should be there
                                        --(by policy)
}
```

PKIFreeText::= SEQUENCE SIZE (1..MAX) OF UTF8String --text encoded as UTF-8 String (note: each UTF8String SHOULD include an RFC 1766 language tag to indicate the language of the contained text)

```
PKIStatusInfo ::= SEQUENCE {
        status          PKIStatus,
        statusString    PKIFreeText     OPTIONAL,
        failInfo        PKIFailureInfo  OPTIONAL
}
```

9 APPENDIX C: TIME STAMPING WITH LINKING PROTOCOL ASN.1 DATA STRUCTURES EXAMPLE

TimeStamp Linking Protocol example:

The PKCS#7 that encapsulates the TimeStampReq is as follows:

```
SEQUENCE {
   version     INTEGER  1
   digestAlgorithms SET OF {
      SEQUENCE {
            algorithm    OBJECT  IDENTIFIER  -- sha1 (for instance)
            parameters TYPE   with {
            NULL
         }
      }
   }
   contentInfo SEQUENCE {
      contentType OBJECT IDENTIFIER TimeStampRequestId
      content TYPE with {
            -- TimeStampReq detailed later.
            SEQUENCE ... (*)
      }
   }
   certificates   IMPLICIT SET OF {
      -- See PKCS#7 Signed Data documentation on Appendix A
   }
   crls IMPLICIT  SET OF {
      -- See PKCS#7 Signed Data documentation on Appendix A
   }
   signerInfos    SET OF {
      -- Set of SignerInfo data structures. Each SignerInfo data structure provides information about a signer
      -- The SignedData that encapsulates a TimeStampReq shall be signed by the requester, so a requester SignerInfo
      -- shall be included with all the authenticated attributes specified above.
      SEQUENCE {
         version        INTEGER   1
         issuerAndSerialNumber   SEQUENCE  { -- The requester certificate serial number
```

```
   issuer        SEQUENCE OF RDN {
   CN= -- The CA who issues the certificate used to sign the PKCS#7 SignedData
   }
   serialNumber INTEGER  -- The serial Number
}
digestAlgorithm      SEQUENCE  {
   algorithm  OBJECT IDENTIFIER  -- sha1 (for instance)
   parameters TYPE    with {
      NULL
   }
}
authenticatedAttributes IMPLICIT SET OF {
-- Authenticated Attributes (*** Authenticated attributes are required in PKCS#7 SignerInfo structures)
   (***) SEQUENCE {  -- PKCS#9 Content-Type authenticated attribute. PKCS#7 required authenticated
                     -- attribute (See PKCS#7 documentation in Appendix A)
      type OBJECT IDENTIFIER contentType { pkcs-9 3 }
      value TYPE with {
         SET OF {
            OBJECT IDENTIFIER TimeStampTokenId
         }
      }
   }
   (***) SEQUENCE {  -- PKCS#9 Message-digest authenticated attribute.
                     -- PKCS#7 required authenticated attribute
                     -- (See PKCS#7 documentation in Appendix A)
      type OBJECT IDENTIFIER messageDigest { pkcs-9 4 }
      value TYPE with {
         SET OF {
            OCTET STRING -- The message digest of the Content field of the ContentInfo data structure
         }
      }
   }
   SEQUENCE {  -- TimeStampProtocol authenticated attribute
      type OBJECT IDENTIFIER TimeStampProtocolId
      value TYPE with {
         SET OF {
            ENUMERATED LinkingProtocol
```

```
                  }
               }
            }
         }
         digestEncryptionAlgorithm  SEQUENCE  {
            algorithm  OBJECT IDENTIFIER  -- pkcs1-rsaEncryption (for instance)
            parameters TYPE     with {
               NULL
            }
         }
         encryptedDigest      OCTET STRING  -- The encrypted digest
         unauthenticatedAttributes  IMPLICIT SET OF  OPTIONAL NOT PRESENT
      }
   }
}
```

(*) The TimeStampReq data structure encapsulated in the ContentInfo field of the Signed Data is:

```
SEQUENCE  {
   version   INTEGER  0
   policy    SEQUENCE  {
      -- Policy Information ...
      policyIdentifier OBJECT IDENTIFIER -- The policy object identifier OID
      policyQualifiers SEQUENCE OF  {
         SEQUENCE  {
            policyQualifierId OBJECT IDENTIFIER --The policy qualifier id OID
            qualifier     TYPE   with { --for instance
               IA5String  -- http address
            }
         }
      }
   }
   nonce     INTEGER 0  -- The time stamp transaction ID
   messageImprints SEQUENCE OF { -- Sequence of messageImprint calculated over the document to TimeStamp
      SEQUENCE {
         hashAlgorithm SEQUENCE   {
            algorithm    OBJECT  IDENTIFIER sha1 (for instance)
```

```
            parameters TYPE with {
               NULL
               }
         }
         hashedMessage OCTET  STRING   (20)  97BB04A786B1A62851C6292CB0BF83545853890F
      }
      SEQUENCE {
         hashAlgorithm SEQUENCE   {
            algorithm    OBJECT  IDENTIFIER md5 (for instance)
            parameters TYPE with {
               NULL
               }
         }
         hashedMessage OCTET  STRING   (20)  5EAE342786B2ADAEEAFOEAEBC0BF835458534E09
      }
   }
}
```

The PKCS#7 Signed Data that encapsulates the TimeStampToken data structure the TSA will return is as follows.

```
SEQUENCE {
   version     INTEGER  1
   digestAlgorithms SET OF {
      SEQUENCE {
         algorithm    OBJECT  IDENTIFIER  -- sha1 (for instance)
         parameters TYPE   with {
            NULL
         }
      }
   }
   contentInfo SEQUENCE {
      contentType OBJECT IDENTIFIER TimeStampTokenId
      content TYPE with {
            -- TimeStampToken detailed later.
            SEQUENCE ... (****)
      }
   }
```

```
certificates   IMPLICIT SET OF {
   -- See PKCS#7 Signed Data documentation on Appendix A
}
crls IMPLICIT  SET OF {
   -- See PKCS#7 Signed Data documentation on Appendix A
}
signerInfos    SET OF {
   -- Set of SignerInfo data structures. Each SignerInfo data structure provides information about a signer
   -- The SignedData that encapsulates a TimeStampToken shall be signed by the TSA, so a a TSA SignerInfo shall
   -- be included with all the authenticated attributes specified above.
   SEQUENCE {
      version        INTEGER   1
      issuerAndSerialNumber   SEQUENCE  { -- The TSA certificate serial number (**)
         issuer        SEQUENCE OF RDN {
         CN= -- The CA who issues the certificate the TSA uses to sign the PKCS#7 SignedData TimeStampToken
         }
         serialNumber INTEGER  -- The serial Number
      }
      digestAlgorithm      SEQUENCE  {
         algorithm  OBJECT IDENTIFIER  -- sha1 (for instance)
         parameters TYPE    with {
            NULL
         }
      }
      authenticatedAttributes IMPLICIT SET OF {
      -- Authenticated Attributes (*** Authenticated attributes are required in PKCS#7 SignerInfo structures)
         (***) SEQUENCE {  -- PKCS#9 Content-Type authenticated attribute. PKCS#7 required authenticated
                           -- attribute
                           -- (See PKCS#7 documentation in Appendix A)
            type OBJECT IDENTIFIER contentType { pkcs-9 3 }
            value TYPE with {
               SET OF {
                  OBJECT IDENTIFIER TimeStampTokenId
               }
            }
         }
         (***) SEQUENCE {  -- PKCS#9 Message-digest authenticated attribute. PKCS#7 required authenticated
```

```
                -- attribute
                -- (See PKCS#7 documentation in Appendix A)
   type OBJECT IDENTIFIER messageDigest { pkcs-9 4 }
   value TYPE with {
      SET OF {
         OCTET STRING -- The message digest of the Content field of the ContentInfo data structure
      }
   }
}
SEQUENCE {  -- TimeStampProtocol authenticated attribute. The same values as in the TimeStampRequest
   type OBJECT IDENTIFIER TimeStampProtocolId
   value TYPE with {
      SET OF {
         ENUMERATED LinkingProtocol
      }
   }
}
SEQUENCE {  -- TimeStampRequester authenticated attribute.
   type OBJECT IDENTIFIER TimeStampRequesterId
   value TYPE with { --IssuerAndSerialNumber. It shall have the same values as the
                     -- issuerAndSerialNumber field
                     -- of the SignerInfo structure included in the SignedData that encapsulates
                     -- the TimeStampReq(**)
      SET OF {
         SEQUENCE  { -- The requester certificate serial number
            issuer        SEQUENCE OF RDN {
               CN= -- The CA who issues the certificate the requester used to sign the TimeStampReq
                   -- PKCS#7 SignedData
            }
            serialNumber INTEGER  -- The requester certificate serial Number
         }
      }
   }
}
SEQUENCE {  -- TimeStampSN authenticated attribute
type OBJECT IDENTIFIER TimeStampSNId
   value TYPE with { -- TimeStampSN data structure
```

```
      SET OF {
         INTEGER --The TimeStampToken serial number
      }
   }
}
SEQUENCE { -- TimeStampLinks authenticated attribute
   type OBJECT IDENTIFIER TimeStampLinksId
   value TYPE with { -- TimeStampLinks data structure
      SET OF {
         SEQUENCE {
            version INTEGER 0
            backwardLink SEQUENCE { -- TimeStampBwdLink data structure
               serialNumber INTEGER    -- The previous TimeStampToken serial number
               requester SEQUENCE { -- IssuerAndSerialNumber data structure
                  issuer        SEQUENCE OF RDN {
                     CN= -- The CA who issues the certificate the requester used to sign the
                         -- TimeStampReq PKCS#7 SignedData
                         -- related to the previous TimeStampToken issued by the TSA
                  }
                  serialNumber INTEGER  -- The requester certificate serial Number
               }
               stampedTime UTCTime OPTIONAL -- The time included in the previous TimeStampToken
                                                  -- the TSA issued
               messageImprints SEQUENCE OF { --MessageImprint field included in the previous
                                             -- TimeStampToken the TSA issued
                  SEQUENCE {
                     hashAlgorithm SEQUENCE   {
                        algorithm    OBJECT  IDENTIFIER sha1 (for instance)
                        parameters TYPE with {
                           NULL
                        }
                     }
                     hashedMessage OCTET  STRING   (20)  126AD4A786B1A62851C6292CB05490545853890F
                  }
                  SEQUENCE {
                     hashAlgorithm SEQUENCE   {
                        algorithm    OBJECT  IDENTIFIER md5 (for instance)
```

```
                parameters TYPE with {
                   NULL
                   }
                hashedMessage OCTET  STRING   (20)  8275342786B653AFDCFOEAEBC0BF836578534E09
             }
          }
          previous SEQUENCE { -- hashes of the TimeStampBwdLink data structure included in the
                              -- previous TimeStampToken SignedData the TSA issued
             SEQUENCE {
                hashAlgorithm SEQUENCE   {
                   algorithm    OBJECT  IDENTIFIER sha1 (for instance)
                   parameters TYPE with {
                      NULL
                   }
                }
                hashedMessage OCTET  STRING   (20)  54FFD4A786B1A6FF51C6292CB0FF905458FF890F
             }
             SEQUENCE {
                hashAlgorithm SEQUENCE   {
                   algorithm    OBJECT  IDENTIFIER md5 (for instance)
                   parameters TYPE with {
                      NULL
                   }
                hashedMessage OCTET  STRING   (20)  FFF5342786DD53AFDCFOEAEBC0BF83657AA34111
             }
          }
       }
       forwardLink SEQUENCE { -- TimeStampFwdLink data structure
          serialNumber   INTEGER -- The following TimeStampToken serial number issued
                                 -- by the TSA
          requester SEQUENCE OPTIONAL { -- IssuerAndSerialNumber data structure
              issuer        SEQUENCE OF RDN {
                  CN= -- The CA who issues the certificate the requester used to sign the
                      -- TimeStampReq PKCS#7 SignedData
                      -- related to the following TimeStampToken issued by the TSA
              }
              serialNumber INTEGER  -- The requester certificate serial Number of the
```

```
                                                -- following TimeStampToken issued
                                                -- by the TSA
                              }
                              stampedTime UTCTime OPTIONAL -- The time included in the following
                                                           -- TimeStampToken the TSA issued
                           }
                        }
                     }
                  }
               }
            }
         }
         digestEncryptionAlgorithm  SEQUENCE  {
         algorithm  OBJECT IDENTIFIER  -- pkcs1-rsaEncryption (for instance)
            parameters TYPE     with {
               NULL
            }
         }
         encryptedDigest      OCTET STRING  -- The encrypted digest
                                            -- (See encrypted digest process on the PKCS#7 documentation
                                            -- in Appendix A )
         unauthenticatedAttributes  IMPLICIT SET OF  OPTIONAL NOT PRESENT
      }
   }
}
```

(****) The TimeStampToken data structure encapsulated in the ContentInfo field of the Signed Data is:

```
SEQUENCE   {
   version   INTEGER    0
   policy    SEQUENCE  {   -- The same policy specified in the TimeStampRequest (if it was correct)
      -- Policy Information ...
      policyIdentifier OBJECT IDENTIFIER -- The policy object identifier OID
      policyQualifiers SEQUENCE OF  {
      SEQUENCE  {
            policyQualifierId OBJECT IDENTIFIER --The policy qualifier id OID
            qualifier     TYPE  with { --for instance
```

```
               IA5String  -- http address
            }
         }
      }
   }
   status    SEQUENCE  {
      status    INTEGER    0  -- TimeStampToken granted
      statusString CHOICE {
         iA5String      IMPLICIT IA5String   "TSTOKEN OK"   --Textual additional information (See Appendix B)
      }
      failInfo  BIT STRING  OPTIONAL NOT PRESENT
   }
   stampedTime   UTCTime  "19980626000520Z" (for instance)
   nonce     INTEGER 0 -- TimeStampTransaction ID.The same value as in the TimeStampReq
   messageImprints SEQUENCE OF { -- Sequence of messageImprint calculated over the document to TimeStamp.
                                 -- Exactly the same field value as in the TimeStampReq
      SEQUENCE {
         hashAlgorithm SEQUENCE   {
            algorithm    OBJECT  IDENTIFIER sha1 (for instance)
            parameters TYPE with {
               NULL
            }
         }
         hashedMessage OCTET  STRING   (20)  97BB04A786B1A62851C6292CB0BF83545853890F
      }
      SEQUENCE {
         hashAlgorithm SEQUENCE   {
            algorithm    OBJECT  IDENTIFIER md5 (for instance)
            parameters TYPE with {
               NULL
            }
         }
         hashedMessage OCTET  STRING   (20)  5EAE342786B2ADAEEAFOEAEBC0BF835458534E09
      }
   }
}
```