INTERNATIONAL STANDARD

ISO/IEC 18014-1

Second edition
2008-09-01

Information technology — Security techniques — Time-stamping services —

Part 1:
Framework

Technologies de l'information — Techniques de sécurité — Services d'estampillage de temps —

Partie 1: Cadre général

Reference number
ISO/IEC 18014-1:2008(E)

Contents

Page

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 18014-1 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 27, *IT Security techniques*.

This second edition cancels and replaces the first edition (ISO/IEC 18014-1:2002), which has been technically revised.

ISO/IEC 18014 consists of the following parts, under the general title *Information technology — Security techniques — Time-stamping services*:

— *Part 1: Framework*

— *Part 2: Mechanisms producing independent tokens*

— *Part 3: Mechanisms producing linked tokens*

Introduction

The International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) draw attention to the fact that it is claimed that compliance with this International Standard may involve the use of patents.

ISO and IEC take no position concerning the evidence, validity and scope of these patent rights.

The holders of these patent rights have assured ISO and IEC that they are willing to negotiate licences under reasonable and non-discriminatory terms and conditions with applicants throughout the world. In this respect, the statements of the holders of these patent rights are registered with ISO and IEC. Information may be obtained from:

ISO/IEC JTC 1/SC 27 Standing Document 8 (SD 8) "*Patent Information*"

SD 8 is publicly available at: http://www.din.de/ni/sc27

Attention is drawn to the possibility that some of the elements of this International Standard may be the subject of patent rights other than those identified above. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Information technology — Security techniques — Time-stamping services —

Part 1: Framework

1 Scope

This part of ISO/IEC 18014:

— identifies the objective of a time-stamping authority;

— describes a general model on which time-stamping services are based;

— defines time-stamping services;

— defines the basic protocols between the involved entities.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*

ISO/IEC 10118 (all parts), *Information technology — Security techniques — Hash-functions*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1
certification authority
CA
centre trusted to create and assign public key certificates

NOTE Optionally, the certification authority can create and assign keys to the entities.

[ISO/IEC 11770-1:1996]

3.2
collision-resistant hash-function
hash-function satisfying the following property: it is computationally infeasible to find any two distinct inputs which map to the same output

NOTE Computational feasibility depends on the specific security requirements and environment.

[ISO/IEC 10118-1:2000]

3.3
data items' representation
data item or some representation thereof such as a cryptographic hash value

3.4
digital signature
data appended to, or a cryptographic transformation of, a data unit that allows a recipient of the data unit to prove the origin and integrity of the data unit and protect the sender and the recipient of the data unit against forgery by third parties and sender against forgery by the recipient

[ISO/IEC 11770-3:1999]

3.5
entity authentication
corroboration that an entity is the one claimed

[ISO/IEC 9798-1:1997]

3.6
hash-function
function which maps strings of bits to fixed-length strings of bits, satisfying the following two properties:

— it is computationally infeasible to find for a given output, an input which maps to this output;

— it is computationally infeasible to find for a given input, a second input which maps to the same output

NOTE Computational feasibility depends on the specific security requirements and environment.

[ISO/IEC 10118-1:2000]

3.7
hash value
string of bits which is the output of a hash-function

NOTE Identical to the definition of hash-code in ISO/IEC 10118-1:2000.

3.8
private key
that key of an entity's asymmetric key pair which should only be used by that entity

[ISO/IEC 11770-1:1996]

3.9
public key
that key of an entity's asymmetric key pair which can be made public

[ISO/IEC 11770-1:1996]

3.10
public key certificate
public key information of an entity signed by the certification authority and thereby rendered unforgeable

[ISO/IEC 11770-1:1996]

3.11
sequence number
time variant parameter whose value is taken from a specified sequence which is nonrepeating within a certain time period

[ISO/IEC 11770-1:1996]

3.12
time stamp
time variant parameter which denotes a point in time with respect to a common time reference

[ISO/IEC 11770-1:1996]

3.13
time-stamp renewal
process of issuing a new time-stamp token to extend the validity period of an earlier time-stamp token

3.14
time-stamp requester
entity which possesses data it wants to be time-stamped

NOTE A requester can also be a trusted third party including a time-stamping authority.

3.15
time-stamp token
TST
data structure containing a verifiable binding between a data items' representation and a time-value

NOTE A time-stamp token can also include additional data items in the binding.

3.16
time-stamp verifier
entity which possesses data and wants to verify that it has a valid time stamp bound to it

NOTE The verification process can be performed by the verifier itself or by a trusted third party.

3.17
time-stamping authority
TSA
trusted third party trusted to provide a time-stamping service

3.18
time-stamping service
TSS
service providing evidence that a data item existed before a certain point in time

3.19
time variant parameter
data item used by an entity to verify that a message is not a replay, such as a random number, a sequence number, or a time stamp

[ISO/IEC 11770-1:1996]

3.20
trusted third party
TTP
security authority, or its agent, trusted by other entities with respect to security-related activities

[ISO/IEC 11770-3:1999]

3.21
time referencing scheme
concepts for describing temporal characteristics of geographic information, about the use of an atomic clock, the clock of the GPS signal, etc.

NOTE See ISO 19108:2002.

3.22
time-signal emission
standard time signals are emitted with reference to UTC according to standard schemes

[ITU-R TF.460-6]

3.23
time-stamping policy
set of rules that indicates the applicability of a time-stamp token to a particular community and/or class of application with common security requirements

4 Symbols and abbreviated terms

TS (x_1, x_2, …, x_n)	generation of time-stamp token for the data x_1, x_2, …, x_n
D	data to be time-stamped
other info	information used to generate the time-stamp token, and which equals "TSTInfo" less the hash value of the data to be time-stamped
$T_{0,}$ $T_{1,}$ …$T_{n,}$	the point in time to be time-stamped
$t_{0,}$ t_1, t_2, … t_n,	the point in time to be time-stamped
S	the point in time at which the end entity's digital signature is generated

5 General

5.1 Background and Summary

The use of digital data that may be provided on easily modifiable media raises the issue of how to certify when this data was created or last changed. Digital time-stamping shall provide evidence of timeliness. Digital time-stamping shall fulfill the following requirements:

— A time variant parameter shall be bound to the data in a non-forgeable way to provide evidence that the data existed prior to a certain point in time.

— Data shall be provided in a way that it is not disclosed.

The time-stamping methods specified in this international standard solve these requirements by time-stamping the hash value of data, which allows for the control of integrity and nondisclosure. The data themselves are not exposed. The hash of the data will be bound to the current time value by the TSA. This binding demonstrates the integrity and authenticity of the time-stamp. A time-stamp token providing these elements will be sent to the requester of the time-stamp.

Time-stamp tokens may also include information relating to previously generated tokens. Here the data's representation and additional information from data time-stamped prior to that time-stamp request are input parameters to the time-stamping process. The TSA may in addition publish various data items relating to the time-stamping process, to provide evidence that the data was available in a timely manner after the other included data hash. The publication of consecutive hashes gives evidence that the related data existed prior to the second published hash. This approach allows the verifier to verify a time-stamp without involving another authority.

5.2 Services involved in Time-stamping

There are two basic operations involved with time-stamping:

— a time-stamping process, which binds time values to data values, and

— a time-stamp verification process, which evaluates the correctness of those cryptographic bindings.

A Time-Stamping Authority (TSA) provides the time-stamping services, whereas the time-stamp verification process may involve other trusted authorities.

The time provided shall fulfill the general requirement of being accurate; the service providing the time for the TSA is outside the scope of this document.

NOTE Time sources are typically dependent on standard time-signal emissions based on standard time referencing schemes.

5.3 Entities of the Time-Stamping Process

The following entities may be involved when a time-stamp is requested:

An *entity* possesses data it wants to be time-stamped; e.g. to have evidence of the data's existence at a certain point in time. In this case it acts as the requester of a time-stamp. An entity may also request evidence that the time-stamped data received has a valid time-stamp, and may act as the *verifier* of a time-stamp.

The *time-stamping authority* (TSA) offers a time-stamping service. The nature of this service is highly sensitive for it helps to identify the validity of data and especially the validity of cryptographic elements related to these data. The TSA offers evidence that data existed at a certain point in time and guarantees the correctness of the time parameter.

All the entities introduced communicate using a two-way handshake protocol. That is, an entity sends a request to the TSA and receives in return a time-stamp (see details in clause 5.1 and clause 5.2). The token contains sufficient information to allow the entity to verify the token at a later point in time.

A time-stamping service may operate online and offline (e.g. using a store-and-forward protocol); the distinction is made at the transport level of the communication protocols between the involved entities.

5.4 Use of Time-Stamps

A time-stamp does not present the exact time when an electronic document was generated, altered or even signed. The entity providing a document for time-stamping may sign the document independently from the TSA, while the TSA binds a time value to the hash of the signed document.

The only evidence available is that a document existed prior to the included time-stamp.

Time-stamps also play an important role for the validity of signed documents. There exist three different possibilities for the time at which time-stamping and signing of data may occur. Data may be time-stamped before the requester of the time-stamp signs it, after the provision of the signature of the document's sender, and before and after the signature. This leads to different results when examining the timely validity of the signature. Table 1 describes these possibilities.

Table 1 —Timely arrangement of signatures and time-stamps

Case 1	t_1	TSA generates a time-stamp
	S	Requester signs data together with the provided time-stamp
Case 2	S	Requester signs data
	t_2	TSA time-stamps signed data
Case 3	t_1	TSA generates a time-stamp
	S	Requester signs data together with the provided time-stamp
	t_2	TSA time-stamps signed data

Technically:

Case 1: (Signature includes time-stamp) does not exactly define the point in time when data was signed. It states that the signature was provided after data was time-stamped.

Case 2: expresses that data was signed prior to the stated point in time.

Case 3: defines an interval during which the document was signed.

5.5 Generation of a Time-Stamp Token

When generating a time-stamp token, first the requester computes the hash value for the data to be time-stamped and sends it to TSA within a time-stamp request message. TSA binds the hash value and the time-stamp request message to the current time value as a Time-Stamp Token and sends it back to the requester.

5.6 Verification of a Time-Stamp Token

When verifying a time-stamp token, the validity of the Time-Stamp Token containing the time parameter is verified. Alternatively, the evaluation of the correctness of the Time-Stamp Token may be delegated to a trusted third party (TTP).

5.7 Time-Stamp renewal

Time-stamped data may be time-stamped again at a later time. This process is called time-stamp renewal and may optionally be implemented by the TSA. This may be necessary for example for the following reasons:

— The mechanism used to bind the time value to the data is near the end of its operational life cycle (e.g.: when using a digital signature and the public key certificate is about to expire).

— The cryptographic function used to bind the time value to the data is still trusted; however, there is strong evidence that it will become vulnerable in the near future (e.g. when a hash function is close to being broken by new attacks or available computing power).

— The issuing TSA is about to terminate operations as a service provider.

Renewal is a variant of the basic time-stamping process where an existing time-stamp over previously time-stamped digital data is explicitly incorporated as data to be bound to a newly generated time-stamp over the same digital data with a new (current) time value. By incorporating the pre-existing time-stamp in the generation of the new time-stamp, and assuming that security considerations are satisfied appropriately, the validity period of the earlier time-stamp over the time-stamped digital data is extended by the coverage of the new time-stamp.

Let data D be time-stamped at time T_0

    TS(D, (other info), T_0)

At time T_1, while the time-stamp is trusted, a renewal such as

    TS(D, (TS(D, (other info), T_0), other info), T_1)

still proves the existence of D at T_0, given that the first time-stamp is valid at T_1.

Renewal has to be performed before any of the above-mentioned conditions make the original time-stamp void.

When verifying a renewed time-stamp token:

— the outermost time-stamp token issued at T_1 is verified at the current time

— the enclosed time-stamp token issued at T_0 is verified at the time of issuance T_1 of the enclosing time-stamp token.

In the event of multiple nested renewals, each nested time-stamp token is verified at the time of issuance of the subsequent renewal, until the innermost time-stamp token is verified.

6 Communications between entities involved

The entities involved in the time-stamping process are an entity either requesting a time-stamp or verifying a time-stamp, and one or more TSAs on the other side. The transactions between these entities will be introduced in the following clauses.

6.1 Time-Stamp Request Transaction

The communication between an entity (requester) and the TSA when requesting a time-stamp consists of the following steps:

The requester generates the hash value for the data to be time-stamped. The hash generation mechanism shall be one of the hash-functions specified in ISO/IEC 10118.

a) A time-stamp request message is sent to the TSA including the following data:

  — Hash value,

  — Hash algorithm used, and

  — (optional) a nonce.

b) The TSA checks the completeness of the received request.

c) The TSA generates a time-stamp (Time-Stamp Token). The time-stamp itself is a data structure containing

— The time-parameter generated or received from a reliable source,

— The data delivered by the requester, and

— Data generated by the TSA to bind the time value to the hash value, hash algorithm, and optionally, the nonce.

d) If the cryptographic binding uses digital signatures then the TSA may use cryptographic algorithms as provided in ISO/IEC 14888. The TSA returns the Time-Stamp Token to the requesting entity.

e) The entity may immediately check the completeness and correctness of the received Time-Stamp Token, or allow the eventual relying party to do so.

Figure 1 shows the communications between the requester and the TSA; the lettering refers to the text.

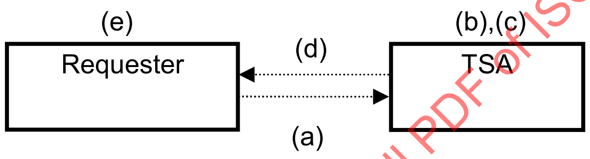

Figure 1 — Communications between Requester and TSA

6.2 Time-Stamp Verification Transaction

Verification of tokens produced using independent token mechanisms makes use of information contained in a single time-stamp token. The verifier may be required to obtain additional information required by the mechanism in order to complete the verification operation; this may be done by the requesting entity or on behalf of the entity by another TTP.

Verification of tokens produced using linked token mechanisms makes use of information contained in a single time-stamp token and possibly other tokens produced by the TSA. The verifier may be required to obtain additional information required by the mechanism in order to complete the verification operation; this may be done by the requesting entity or on behalf of the entity by another TTP.

Additional information is provided in ISO/IEC 18014-2 and ISO/IEC 18014-3.

7 Message Formats

There are two types of messages that are needed to generate the transactions introduced in Clause 5: messages between time-stamping requester/verifier and TSA, and messages between TSA and requester/verifier. All messages will be described in the ASN.1 notation. A complete ASN.1 module is provided in annex A. Messages will be distinguished according to the service they represent.

7.1 Time-stamp request

TimeStampReq messages are used by entities to request time-stamping services from time-stamping authorities. A TimeStampReq message is formed as follows:

```
TimeStampReq ::= SEQUENCE {

    version          Version,

    messageImprint   MessageImprint,

    reqPolicy        TSAPolicyId OPTIONAL,

    nonce            INTEGER OPTIONAL,

    certReq          BOOLEAN DEFAULT FALSE,

    extensions       [0]Extensions OPTIONAL

}
```

The following table explains the variables and their values.

Data field	Description
version	The syntax version number
messageImprint	The MessageImprint to which the service provider is to bind a time value
reqPolicy	Service policy requested from the TSA issuing the Time-Stamp Token
nonce	Identifies the specific request; the purpose of this value is to tie a specific request to the respective reply.
certReq	Signals the TSA to provide certificate information, if present.
extensions	Contains any extensions required to properly fulfill the requested time-stamping operation.

Type *MessageImprint* is used to encapsulate the message imprint data along with an indicator of the algorithm used to generate the message imprint.

```
MessageImprint ::= SEQUENCE {

    hashAlgorithm    DigestAlgorithmIdentifier,

    hashedMessage    OCTET STRING

}
```

Data field	Description
hashAlgorithm	Hash Algorithm Identifier and parameter value
hashedMessage	The corresponding hash value of a message to be time-stamped, as calculated with the hash-function specified in the **hashAlgorithm** data field.

The hash-function shall be a collision-resistant hash-function.

`*TSAPolicyId* is defined as follows:`

```
TSAPolicyId ::= POLICY.&id({TSAPolicies})
```

7.2 Time-stamp response

The answer to a time-stamp request is a ***TimeStampResp*** data structure. It has the following form:

```
TimeStampResp ::= SEQUENCE {

    status           PKIStatusInfo,
    timeStampToken   TimeStampToken OPTIONAL
}
```

The TimeStampToken structure is defined as follows:

```
TimeStampToken ::= SEQUENCE {

    contentType CONTENT.&id({Contents}),
    content [0]
    EXPLICIT CONTENT.&Type ({Contents}{@contentType})
}
```

This structure is used to encapsulate a TSTInfo structure. The TSTInfo structure is defined as follows:

```
TSTInfo ::= SEQUENCE {

    version          Version,
    policy           TSAPolicyId,
    messageImprint   MessageImprint,
    serialNumber     SerialNumber,
    genTime          GeneralizedTime,
    accuracy         Accuracy OPTIONAL,
    ordering         BOOLEAN DEFAULT FALSE,
    nonce            Nonce OPTIONAL,
    tsa              [0]EXPLICIT GeneralName OPTIONAL,
    extensions       [1]Extensions OPTIONAL
}
```

The following table describes those data fields that are yet not defined.

Data field	Description
accuracy	The accuracy of the genTime field as compared with UTC. TSA guarantees the time difference between UTC and its internal clock is limited within the accuracy.
genTime	The time the TSA included in the Time-Stamp Token
serialNumber	serialNumber may be set to zero. If serialNumber is not zero, then this field is an integer assigned by the TSA to each TimeStampToken and shall be unique for each TimeStampToken issued by a given TSA.

The *TSTInfo* structure is encapsulated in the *TimeStampToken* structure using the *ContentInfo* encapsulation method appropriate to the TSA implementation. When encapsulated in a *ContentInfo* structure, which uses the *EncapsulatedContentInfo* structure, the eContentType field contains the following object identifier:

```
id-ct-TSTInfoOBJECT IDENTIFIER ::= {

    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs-9(9)

    smime(16) ct(1)4

}
```

In addition, the eContent field contains the TSTInfo structure, DER-encoded as an octet string.

GeneralizedTime is a combination of the basic format for calendar dates in complete representation and the basic format for the coordinated universal time according to ISO 8601, *Data elements and interchange formats — Information interchange — Representation of dates and times*. This format has the following form:

Y Y Y Y M M D D [T] h h m m s s [, f f f]Z

Where each of the characters with the exception of the last is a substitute for a single digit:

YYYY represents the years (0000-9999),

MM represents the actual month (01-12),

DD represents the actual day of the month (01-31),

[T] represents time designator to indicate the start of the representation of the time of day component or this field itself may be ignored.

hh represents the actual hour of the day (00-23),

mm stands for the minutes of the hour (00-59), and

ss represents the seconds of the minute (00-59).

fff is an abbreviation for fractions of a second without trailing 0.

The Character Z (Zulu Time) stands for Universal Coordinated Time (UTC).

```
Accuracy ::= SEQUENCE {

    seconds         INTEGER OPTIONAL,

    millis          [0]INTEGER (1..999) OPTIONAL,

    micros          [1]INTEGER (1..999) OPTIONAL

}
```

(ALL EXCEPT ({- none; at least one component shall be present - }))

7.3 Time-stamp verification

The verification protocol is similar to the time-stamp request protocol; it also consists of a request message (*VerifyReq*) and the respective reply (*VerifyResp*). The following data structures apply:

```
VerifyReq ::= SEQUENCE {
    version          Version,
    tst              TimeStampToken,
    requestID        [0]OCTET STRING OPTIONAL
}
```

and

```
VerifyResp ::= SEQUENCE {
    version          Version,
    status           PKIStatusInfo,
    tst              TimeStampToken,
    requestID        [0]OCTET STRING OPTIONAL
}
```

The field requestID ties the request to its respective response.

7.4 Extension fields

7.4.1 ExtHash extension

A requester of time-stamping services may wish to submit for time-stamping more than one hash value derived from a single data item.

Submitting multiple hash values derived from a single data item using different hash functions allows the requester to insulate the resulting time-stamp token from the cryptographic failure of any single hash function.

To enable the submission of multiple hash values the following extension is defined:

```
    ExtHash ::= SEQUENCE SIZE (1..MAX)OF MessageImprint
    tsp-ext-hash ::= OBJECT IDENTIFIER { tsp-ext 1 }
    extHash EXTENSION ::= {
      SYNTAX ExtHash IDENTIFIED BY tsp-ext-hash
    }
```

This extension is carried in the "extensions" field of both the TimeStampReq message sent by a requester to a TSA and in the "extensions" field of the resulting TimeStampToken structure formed by the TSA and returned to the requester.

If this extension is present and the TSA is able to process it, then the TSA shall bind both the hash value in the time-stamp request message specified in the messageImprints field and those included in this extension to the time value it assigns to the resulting time-stamp token.

7.4.2 ExtMethod extension

A requester of time-stamping services may wish to indicate to a specific TSA which time-stamping method to use when forming the eventual time-stamp token. To enable a requester to indicate to a specific TSA which time-stamping method to use in forming the resulting time-stamp token, the following extension is defined:

```
Method ::= METHOD.&id ({Methods})

ExtMethod ::= SEQUENCE SIZE (1..MAX)OF Method

tsp-ext-meth ::= OBJECT IDENTIFIER { tsp-ext 2 }

extMethod EXTENSION ::= {

  SYNTAX ExtMethod IDENTIFIED BY tsp-ext-meth

}
```

This extension is carried in the "extensions" field of both the TimeStampReq message sent by a requester to a TSA and in the "extensions" field of the resulting TimeStampToken structure formed by the TSA to the requester.

If this extension is present, then the TSA shall attempt to fulfill the request for the specified method if it is available, otherwise return an error. If the requester specifies more than one possible method, the TSA should select one of the suggested methods for use in forming the time-stamp token. If this extension is not present, the TSA uses its default time-stamping mechanism.

7.4.3 ExtRenewal extension

A requester of time-stamping services may wish to indicate to the TSA that the current time-stamping request is a renewal of a time-stamp on data that were already time-stamped in the past, so that the validity period of the old time-stamp is effectively extended. For this purpose, the following extension is defined:

```
extRenewal EXTENSION ::= { SYNTAX ExtRenewal IDENTIFIED BY tsp-ext-renewal }

ExtRenewal ::= TimeStampToken
```

tsp-ext-renewal OBJECT IDENTIFIER ::= { tsp-ext renewal(3) }

This extension is carried in the “extensions” field of both the TimeStampReq message sent by a requester to a TSA and in the “extensions” field of the resulting TSTInfo structure formed by the TSA and returned to the requester. The TSA reproduces the contents of this extension unmodified.

Annex A
(normative)

ASN.1 Module for time-stamping

This module contains correct ASN.1 based on the current ASN.1 standards that has passed successfully through a reliable syntax checker used by the ITU-T ASN.1 Project.

```
TimeStampProtocol {

    iso(1) standard(0) time-stamp(18014) modules(0) tsp(1)}

    DEFINITIONS IMPLICIT TAGS ::= BEGIN

-- EXPORTS All ; --

IMPORTS

   -- ISO/IEC 9594-8 | ITU-T Rec.  X.509 AuthenticationFramework --

    EXTENSION, Extensions

       FROM AuthenticationFramework  {

           joint-iso-itu-t ds(5) module(1) authenticationFramework(7) 4 }

    -- ISO/IEC 9594-8 | ITU-T Rec. X.509 CertificateExtensions --

    GeneralName

       FROM CertificateExtensions {

           joint-iso-itu-t ds(5) module(1) certificateExtensions(26) 4 }

    AuthenticatedData, SignedData

       FROM CryptographicMessageSyntax {

           iso(1)  member-body(2)  us(840)  rsadsi(113549)  pkcs(1)  pkcs-9(9)  smime(16)
           modules(0) cms(1) };


time-stamping-services OBJECT IDENTIFIER ::=

          { iso(1) standard(0) time-stamp(18014)  }

modules OBJECT IDENTIFIER ::= {  time-stamping-services modules(0) }

tsp-ext OBJECT IDENTIFIER ::= { time-stamping-services ext(1) }


TimeStampReq ::= SEQUENCE  {

    version             Version,

    messageImprint      MessageImprint,
    reqPolicy          TSAPolicyId  OPTIONAL,
    nonce              Nonce OPTIONAL,
    certReq            BOOLEAN DEFAULT FALSE,
    extensions[0]      Extensions OPTIONAL
}

MessageImprint ::= SEQUENCE   {

    hashAlgorithm      DigestAlgorithmIdentifier,
    hashedMessage      OCTET STRING
}

DigestAlgorithmIdentifier ::= AlgorithmIdentifier {{ DigestAlgorithms }}

DigestAlgorithms ALGORITHM :: = {

        { OID sha1 PARMS  NULL },

         --

    ...  -- Expect additional digest algorithms --

}

TSA PolicyId :: =  POLICY.&id({TSAPolicies})

TSAPolicies POLICY ::= {

         --

        ...  -- Any supported TSA policy --

}

TimeStampResp ::=  SEQUENCE {

    status             PKIStatusInfo,
    timeStampToken     TimeStampToken OPTIONAL
}

PKIStatusInfo  ::= SEQUENCE {

    status             PKIStatus,
    statusString       PKIFreeText OPTIONAL,
    failInfo           PKIFailureInfo OPTIONAL
}

PKIStatus ::= INTEGER {

    granted            (0), --  the request is completely granted
    grantWithMods      (1), --  modifications were necessary, the requester is responsi
                                ble for asserting the differences
    rejection          (2), --  the request could not be fulfilled, the failure code
                                delivers additional information
    waiting            (3), --  the request is not processed
    revocationWarning  (4), --  a revocation is imminent
    revocationNotification    (5) -- notification that a revocation has been occurred
}

PKIFreeText ::= SEQUENCE SIZE(1..MAX) OF UTF8String

PKIFailureInfo ::= BIT STRING {

    badAlg             (0), -- unrecognized or unsupported Algorithm Identifier
    badRequest         (2), -- transaction not permitted or supported
    badDataFormat      (5), -- data submitted  has the wrong format
    timeNotAvailable   (14),-- the TSAs service is not  available
    unacceptedPolicy   (15),-- the requested TSA policy is not supported
    unacceptedExtension     (16), -- the requested TSA extension is not supported,
    addInfoNotAvailable     (17), --   the   requested   additional   information   is   not
available,
    systemNotAvailable    (24),-- system is not available
    systemFailure      (25),-- System Failure
    verificationFailure   (27) -- verification of time stamp has failed
}

TimeStampToken ::= SEQUENCE {

    contentType        CONTENT.&id({Contents}),
    content[0]         EXPLICIT CONTENT.&Type({Contents}{@contentType})
}

Contents CONTENT ::= {

    { SignedData IDENTIFIED BY id-signedData } |
    { AuthenticatedData IDENTIFIED BY id-ct-authData } |
    { ETSTInfo IDENTIFIED BY id-data } |
    { DigestedData IDENTIFIED BY id-digestedData },
     --
    ... -- Expect additional time-stamp mechanisms  --
}

TSTInfo ::=  SEQUENCE {

    version            Version,
    policy             TSAPolicyId,
    messageImprint     MessageImprint,
    serialNumber       SerialNumber OPTIONAL,
    genTime            GeneralizedTime,
    accuracy           Accuracy OPTIONAL,
    ordering           BOOLEAN DEFAULT FALSE,
    nonce              Nonce OPTIONAL,
    tsa                [0] EXPLICIT GeneralName OPTIONAL,
    extensions         [1] Extensions OPTIONAL
}
```

```
Version ::= INTEGER { v1(1) }

SerialNumber ::= INTEGER  -- Expect large values

Accuracy ::= SEQUENCE {

    seconds             INTEGER  OPTIONAL,
    millis[0]           INTEGER(1..999)  OPTIONAL,
    micros              [1] INTEGER(1..999)  OPTIONAL
}

 (ALL EXCEPT({ -- no components present -- }))

Ordering ::= BOOLEAN

Nonce ::= INTEGER  -- Expect large values

-- Time-stamping extensions --

TSExtensions EXTENSION ::= {

    extHash |
    extMethod |
    extRenewal,
    --
    ...  -- Expect additional extensions --
}

extHash EXTENSION ::=  {SYNTAX ExtHash IDENTIFIED BY tsp-ext-hash}

ExtHash ::= SEQUENCE SIZE(1..MAX) OF MessageImprint


extMethod EXTENSION ::= {SYNTAX ExtMethod IDENTIFIED BY tsp-ext-meth}

ExtMethod ::= SEQUENCE SIZE(1..MAX) OF Method

Method ::= METHOD.&id({Methods})

Methods  METHOD ::= {

    --
    ...  -- Any time-stamping method  --
}


extRenewal EXTENSION ::= { SYNTAX ExtRenewal IDENTIFIED BY tsp-ext-renewal }

ExtRenewal ::= TimeStampToken


EncapsulatedContentInfo::=  SEQUENCE {

    eContentType        CONTENT.&id({EContents}),
    eContent            [0] EXPLICIT
                            CONTENT.&Type({EContents}
                              {@eContentType})
}
EContents CONTENT ::= {
    { ETSTInfo IDENTIFIED BY id-ct-TSTInfo },
    --
    ...     -- Expect additional content types --
}
-- Supporting definitions
AlgorithmIdentifier { ALGORITHM:IOSet } ::= SEQUENCE {
    algorithm   ALGORITHM.&id({IOSet}),
    parameters  ALGORITHM.&Type({IOSet}{@algorithm})   OPTIONAL
}
ALGORITHM ::= CLASS {
    &id    OBJECT IDENTIFIER  UNIQUE,
    &Type  OPTIONAL
}
WITH SYNTAX { OID &id [PARMS  &Type] }
CONTENT ::= TYPE-IDENTIFIER-- ISO/IEC 8824-2, Annex A
OIDS ::= CLASS {
    &id  OBJECT IDENTIFIER  UNIQUE
}
    WITH SYNTAX { OID &id }
POLICY ::= OIDS  -- Supported TSA policies
METHOD ::= OIDS  -- TSA Methods
    -- Information object identifiers
--
tsp-ext-hash OBJECT IDENTIFIER ::= { tsp-ext hash(1)  }
tsp-ext-meth OBJECT IDENTIFIER ::= { tsp-ext meth(2) }
tsp-ext-renewal OBJECT IDENTIFIER ::= { tsp-ext renewal(3) }
der OBJECT IDENTIFIER ::= {
    joint-iso-itu-t asn1(1) ber-derived(2) distinguished-encoding(1) }
sha1 OBJECT IDENTIFIER ::= {
    iso(1) identified-organization(3) oiw(14) secsig(3) 2 26 }
pkcs7 OBJECT IDENTIFIER ::= {
    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1) pkcs7(7)
}
```

```
id-data OBJECT IDENTIFIER : := {

    pkcs7 data(1) }

id-signedData OBJECT IDENTIFIER : := {

    pkcs7 signedData(2) }

id-ct-authData OBJECT IDENTIFIER ::= {

    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)

    pkcs-9(9) smime(16) ct(1) 2 }

id-ct-TSTInfo OBJECT IDENTIFIER ::= {

    iso(1) member-body(2) us(840) rsadsi(113549) pkcs(1)

    pkcs-9(9) smime(16) ct(1) 4 }

    -- verification of a timestamp token

VerifyReq ::=  SEQUENCE {

    version             Version,
    tst                 TimeStampToken,
    requestID           [0] OCTET STRING OPTIONAL
}

VerifyResp ::=  SEQUENCE {

    version             Version,
    status              PKIStatusInfo,
    tst                 TimeStampToken,
    requestID           [0] OCTET STRING OPTIONAL
}


END  -- TimeStampProtocol --
```

Annex B
(normative)

Excerpt of the Cryptographic Message Syntax

This annex specifies Cryptographic Message Syntax (CMS), which includes the content types required for time-stamping. The ASN.1 syntax makes use of the current standards, which are listed in the bibliography of the document.

NOTE The definitions of CMS and the content types are given in RFC 3852 [2].

B.1 Introduction

The Cryptographic Message Syntax (CMS) is used to digitally sign, digest, or authenticate arbitrary messages.

The CMS is an encapsulation syntax for data protection. It supports digital signatures and encryption. The syntax allows multiple encapsulations one encapsulation envelope can be nested inside another. Likewise, one party can digitally sign some previously encapsulated data. It also allows arbitrary attributes, such as signing time, to be signed along with the message content, and provides for other attributes such as countersignatures to be associated with a signature.

The CMS values are generated using ASN.1, using BER-encoding. Values are typically represented as octet strings. While many systems are capable of transmitting arbitrary octet strings reliably, it is well known that many electronic mail systems are not. This section does not address mechanisms for encoding octet strings for reliable transmission in such environments.

B.2 General Overview

The Cryptographic Message Syntax (CMS) is general enough to support many different content types. This section defines one protection content, ContentInfo. ContentInfo encapsulates a single identified content type, and the identified type may provide further encapsulation.

As a general design philosophy, each content type permits single pass processing using indefinite-length Basic Encoding Rules (BER) encoding. Single-pass operation is especially helpful if content is large, stored on tapes, or is "piped" from another process. Single-pass operation has one significant drawback: it is difficult to perform encode operations using the Distinguished Encoding Rules (DER) encoding in a single pass since the lengths of the various components may not be known in advance. However, signed attributes within the signed-data content type need to be transmitted in DER form to ensure that recipients can verify a content that contains one or more unrecognised attributes. Signed attributes are the only data types used in the CMS that requires DER encoding.

B.3 General Syntax

The following object identifier identifies the content information type:

```
id-ct-contentInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2)

      us(840) rsadsi(113549) pkcs(1) pkcs9(9) smime(16) ct(1) 6 }
```

The CMS associates a content type identifier with content. The syntax shall have ASN.1 type ContentInfo:

```
ContentInfo ::= SEQUENCE {

    ContentType ContentType,
    content [0] EXPLICIT  ANY DEFINED BY contentType }
ContentType ::= OBJECT IDENTIFIER
```

The fields of ContentInfo have the following meanings:

> contentType indicates the type of the associated content. It is an object identifier; it is a unique string of integers assigned by an authority that defines the content type.
>
> content is the associated content. The type of content can be determined uniquely by contentType. Content types for data, and signed-data are defined in this section.

B.4 Data Content Type

The following object identifier identifies the data content type:

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

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 could have their own ASN.1 definition or other structure).

The data content type is generally encapsulated in the signed-data content type.

B.5 Signed-data Content Type

The signed-data content type consists of a content of any type and zero or more signature values. Any number of signers in parallel can sign any type of content.

The typical application of the signed-data content type represents one signer's digital signature on content of the data content type. Another typical application disseminates certificates and certificate revocation lists (CRLs).

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

— For each signer, a message digest, or hash value, is computed on the content with a signer-specific message-digest algorithm. If the signer is signing any information other than the content, the message digest of the content and the other information are digested with the signer's message digest algorithm (see clause B.5.4), and the result becomes the "message digest."

— For each signer, the message digest is digitally signed using the signer's private key.

— For each signer, the signature value and other signer-specific information are collected into a SignerInfo value, as defined in clause B.5.3. Certificates and CRLs for each signer, and those not corresponding to any signer, are collected in this step.

— 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, as defined in clause B.5.1.

A recipient independently computes the message digest. This message digest and the signer's public key are used to verify the signature value. The signer's public key is referenced either by an issuer distinguished name along with an issuer-specific serial number or by a subject key identifier that uniquely identifies the certificate containing the public key. The signer's certificate may be included in the SignedData certificates field.

This clause is divided into six parts. The first part describes the top-level type SignedData, the second part describes EncapsulatedContentInfo, the third part describes the per-signer information type SignerInfo, and the fourth, fifth, and sixth parts describe the message digest calculation, signature generation, and signature verification processes, respectively.

B.5.1 SignedData Type

The following object identifier identifies the signed-data content type:

```
id-signedData OBJECT IDENTIFIER ::= { iso(1) member-body(2)

    us(840) rsadsi(113549) pkcs(1) pkcs7(7) 2 }
```

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

```
SignedData ::= SEQUENCE {

    version          CMSVersion,

    digestAlgorithms DigestAlgorithmIdentifiers,

    encapContentInfo EncapsulatedContentInfo,

    certificates     [0] IMPLICIT CertificateSet OPTIONAL,

    crls             [1] IMPLICITCertificat eRevocationLists OPTIONAL,

    signerInfos SignerInfos }
```

with

```
DigestAlgorithmIdentifiers    ::= SET OF DigestAlgorithmIdentifier
```

and

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

The fields of type SignedData have the following meanings:

version is the syntax version number. If no attribute certificates are present in the certificates field, the encapsulated content type is id-data, and all of the elements of SignerInfos are version 1, then the value of version shall be 1. Alternatively, if attribute certificates are present, the encapsulated content type is other than id-data, or any of the elements of SignerInfos are version 3, then the value of version shall be 3.

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, along with any associated parameters, used by one or more 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. Implementations may fail to validate signatures that use a digest algorithm that is not included in this set. The message digesting process is described in clause B.5.4.

encapContentInfo is the signed content, consisting of a content type identifier and the content itself. Details of the EncapsulatedContentInfo type are discussed in clause B.5.2.

certificates is a collection of certificates. It is intended that the set of certificates be sufficient to contain certification paths 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