

Protocols and data formats for time-stamping service

17th September 2002

Contents

1	Introduction	3
1.1	Parties of a Time-Stamping System	4
1.2	The Structure of a Time-Stamping System	6
1.3	Components of the Time-Stamping Service	7
1.3.1	Aggregation	7
1.3.2	Adding the Time Value	9
1.3.3	Linking	11
1.3.4	Publishing	12
1.3.5	Extending	13
2	Terms and definitions	13
3	Representation of Data Structures	14
4	Requirements to the Time-Stamping Service	14
5	Time-Stamping Protocol	15
5.1	Hash Chain	15
5.2	Data Formats	15
5.3	The Computation Algorithm	17
5.4	ASN.1 Representation	18
5.5	Time-Stamping Request	18
5.6	Time Stamp	19

6	Time Stamp Extension Protocol	23
6.1	Extension Request	23
6.2	Extension Reply	24
7	The Format of the Published Message Digest	24
8	Algorithms for Managing the Time Stamps	25
8.1	Finding Out the Time Value	25
8.1.1	Inputs	25
8.1.2	Output	27
8.1.3	State Variables	27
8.1.4	Time Stamp Verification	27
8.1.5	Extracting the Time Value	28
8.2	Comparing the Time Stamps	28
8.2.1	Inputs	29
8.2.2	Outputs	30
8.2.3	State Variables	30
8.2.4	Direct Comparison of Time Stamps	31
8.2.5	Indirect Comparison of Time Stamps	31
8.3	Extending a Time stamp Using Another Time Stamp	31
8.3.1	Inputs	32
8.3.2	Outputs	32
8.3.3	State variables	32
8.3.4	The Algorithm	33
9	Transport Protocol	35
10	Supported cryptographic algorithms	36
A	ASN.1 module	36
B	Base32 encoding	38
C	Examples of Time Stamps	39
C.1	Private Key and Certificate of the Time-Stamping Service Provider	40
C.1.1	Private Key of the Time-Stamping Service Provider	40
C.1.2	Certificate of the Time-Stamping Service Provider	41
C.2	Time-stamped Messages	42
C.2.1	Time-stamped Message 1	42

C.2.2	Time-stamped Message 2	43
C.2.3	Time-stamped Message 3	43
C.3	Time Stamps without Aggregation	44
C.3.1	Time Stamp for Message 1	44
C.3.2	Time Stamp for Message 2	47
C.3.3	Time Stamp for Message 3	51
C.3.4	Comparing Time Stamps	54
C.4	Time Stamp Extension	55
C.4.1	Extending the Time Stamp C.3.1 to Time Stamp C.3.3	55
C.4.2	Extending the time stamp C.3.2 to a String	61
C.4.3	Comparing Time Stamps	70
C.4.4	Extending One Time Stamp with the Help of Another	70
C.5	Time Stamps with Aggregation	73
C.6	The Above Messages in PEM-format	77
C.6.1	Private Key of the Time-stamping Service Provider C.1.1	77
C.6.2	Certificate of the Time-stamping Service Provider C.1.2	78
C.6.3	Time-stamped Message C.2.1	78
C.6.4	Time-stamping request Corresponding to Message C.2.1	78
C.6.5	Time-stamped Message C.2.2	78
C.6.6	Time-stamping request Corresponding to Message C.2.2	78
C.6.7	Time-stamped Message C.2.3	78
C.6.8	Time-stamping request Corresponding to Message C.2.3	78
C.6.9	Time Stamp C.3.1	79
C.6.10	Time Stamp C.3.2	79
C.6.11	Time Stamp C.3.3	79
C.6.12	Time Stamping Request C.4.1	80
C.6.13	Time Stamp C.4.1	80
C.6.14	Time Stamping Request C.4.2	81
C.6.15	Time Stamp C.4.2	81
C.6.16	Time Stamp C.4.4	82
C.6.17	Time Stamp C.5	82

1 Introduction

Time stamp is a collection of digital data allowing to prove or to establish temporal relationships between different events. As an example we may be interested in comparing the events of forming a digital signature and of the time stamp itself.

The time stamp can also certify that some data existed at some moment of time shown in the time stamp.

If the time stamp proves or certifies the existence of a document before some time moment, this time stamp is called the *time stamp of the document*. In the time stamp, the document is represented by its digest, computed from the document using a *cryptographic hash function*.

The temporal order of two events can be compared either directly or indirectly:

- Direct comparison is possible if causal relationship exists between two time stamps. These relationships can be created by *linking* using cryptographic hash functions; as a result later time stamps will be undeniably dependent on some or all earlier ones.
- It is also possible to add physical time values to time stamps and later compare these times. Such a comparison is indirect by its nature as there is no way of establishing whether the physical times were attached correctly or not.

Thus we see that physical time values do not *prove* the temporal relationships, but rather give a certain level of assurance. The reliability of such an assurance depends heavily on the level of trust placed into the source of time value. Direct comparison, on the contrary, allows us to prove temporal relationships without any extra trust assumptions towards third parties.

1.1 Parties of a Time-Stamping System

In every system utilising time-stamping, the following parties are involved:

- **Client** – an entity forming the time-stamping request and sending it to the service provider using some communication protocol;
- **Service provider** – an entity accepting time-stamping requests and issuing time stamps;
- **Verifier** – an entity verifying the correctness and relationships of time stamps.

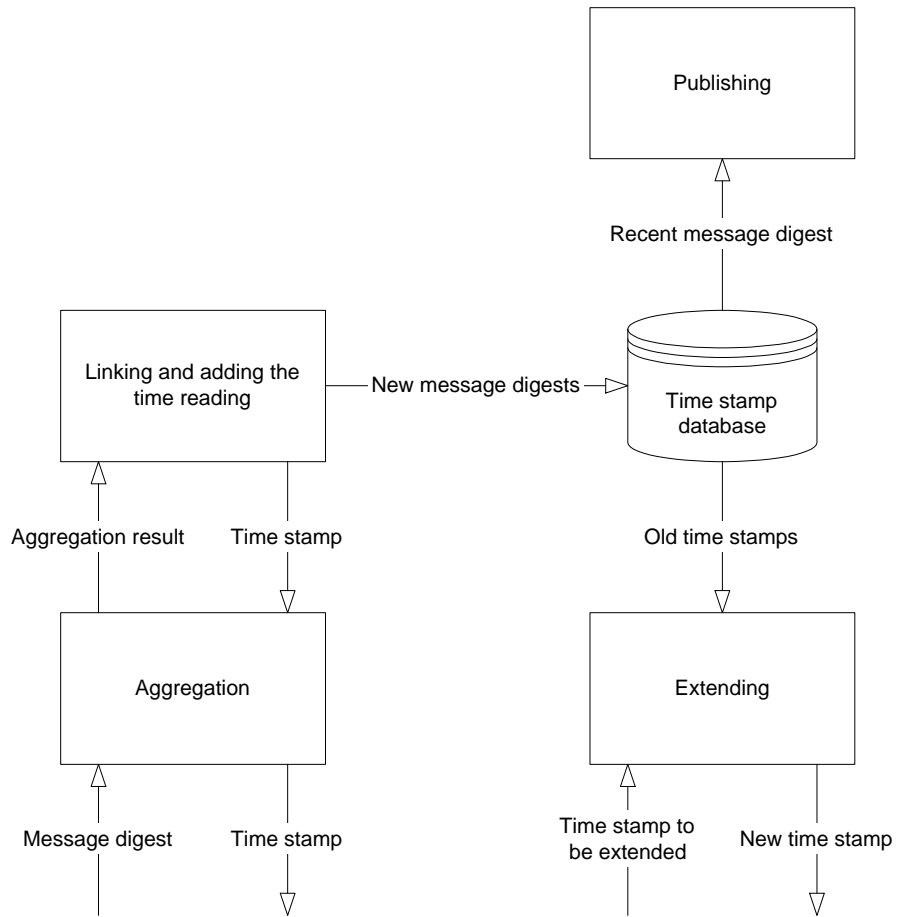


Figure 1: General structure of time-stamping service

1.2 The Structure of a Time-Stamping System

Time-stamping service generally consists of five components with different functionalities (see Figure 1).

- **Aggregation** – a component accepting a sequence of input time-stamping requests and forming one output time-stamping request by means of *authentication trees* (see Subsection 1.3.1).
The purpose of aggregation is improving the system's performance by reducing the server's load as the number of requests sent directly to the server is decreased. Aggregation layer is not compulsory if the server is powerful enough to handle all the requests by itself. On the other hand, this component does not introduce any security risks apart from availability (e.g. higher risk of loss of requests). Hence, it is not necessary to authenticate the aggregation component separately.
- **Adding the time value** – a component that adds the time value to the time stamp to be formed.
- **Linking** – a component creating one-way dependencies between the new time stamp and all or some older time stamps, enabling their direct comparison and auditing of the time-stamping service. The means of deciding which time stamps should be directly dependent is called a *linking scheme*. The direct dependencies are created by computing message digests using cryptographic hash functions. Freshly issued time stamps are also signed by the server assuring their authenticity and instant verifiability.
- **Publishing** – this component is used to make a lately computed message digest available by some public and auditable media, e.g. by printing it in a newspaper. Publishing is one of the central means ensuring the auditability of the service and retaining the long-term proof value of the time stamps.
- **Extending** – this component is used to renew the information necessary for the time stamp verification. When extending a time stamp, the server builds a *hash chain* proving that the time stamp to be extended existed before another time stamp requested by the client. The most useful kind of extending is extending to a lately published time stamp. This enables time stamp verification without any help (or disturbance) of the time-stamping server.

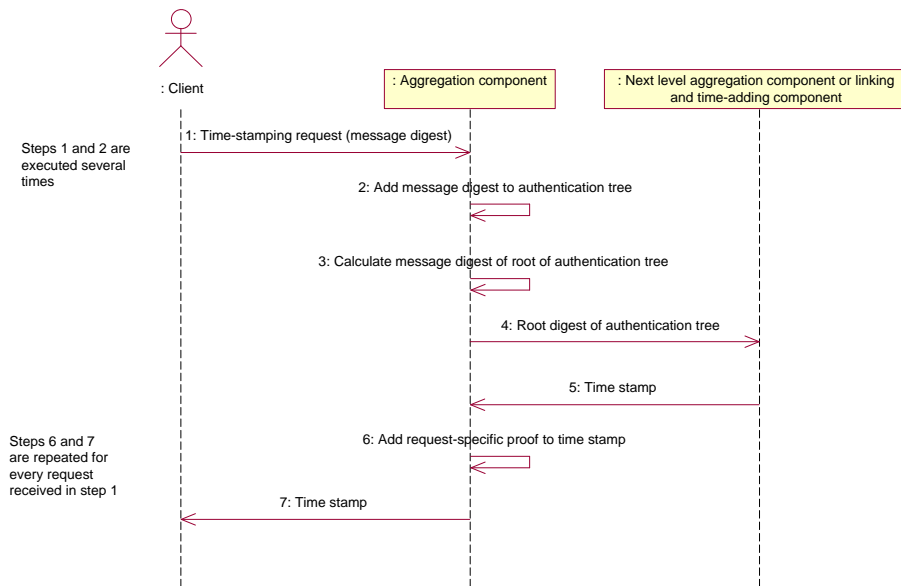


Figure 2: Work of the aggregation component

1.3 Components of the Time-Stamping Service

1.3.1 Aggregation

Aggregation process's work (see Figures 2 and 3) is divided into short time intervals (lasting e.g. one second) called *rounds*. During one round the component

- accepts the time-stamping requests arriving during that round,
- extracts the message digests to be time-stamped from the requests,
- forms a special data structure (authentication tree) from these message digests,
- computes the message digest of the root of the authentication tree,
- forwards this digest to a higher level aggregation component or to the linking and time-adding components.

When the aggregation component gets back a time stamp to the forwarded message digest, it sends the time stamp back to the clients, adding the proofs

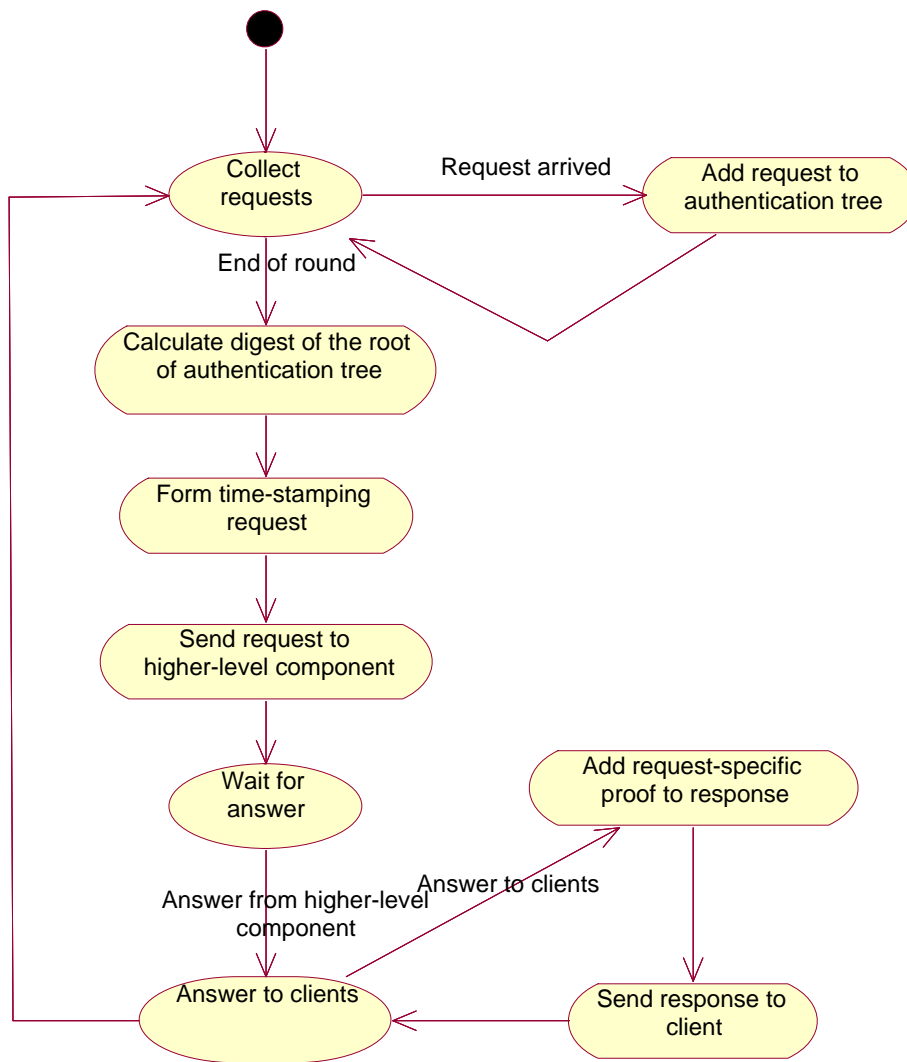


Figure 3: State diagram representing the work of aggregation component

that the time-stamping request sent by the client took part in forming the round's digest.

The aggregation process is considerably less resource-consuming than linking and signing the time stamps. It allows to reduce the workload of the time-stamping server with the price of increasing the size of time stamps and loss of precision determined by the length of the round (e.g. one second which is generally acceptable imprecision in practice). This price is not too high as it guarantees that the workload of linking and signing components does not depend on the number of time-stamping requests.

The aggregation component uses Merkle's authentication trees for aggregation [Mer80, Mer89]. The security of aggregation depends only on collision-resistance of the hash function used to build the authentication tree. As the same condition is needed for the security of digital signatures, we do not need to introduce any extra security requirements.

Merkle's authentication tree is a method of providing short proofs (logarithmic in the number of inputs) proofs that a bitstring x belongs to the set of bitstrings $\{x_1, x_2, \dots, x_n\}$. During a round a binary tree is constructed as follows (see Figure 4):

- the leaves are labeled with message digests extracted from the time-stamping requests obtained during the round,
- every non-leaf is labeled with a message digest computed using a hash function H over a concatenation of the labels of its children.

Hence, the label of the root of the tree depends on all the leaves, i.e. aggregated bitstrings. For every leaf it is possible to prove this dependence by exhibiting some more vertices of the tree; the minimal collection of such extra nodes is called an *authentication path*.

In Figure 5 we see an example of a Merkle's authentication tree where in order to prove that the root value d is dependent on the input x_1 it is enough to add vertices x_2 and h_{34} and compute that $h_{12} = H(x_1, x_2)$ and $h_{1234} = H(h_{12}, h_{34})$.

1.3.2 Adding the Time Value

In order to add the current physical time, a new data structure is formed from the data to be time-stamped and the time value. If the time-stamping system uses

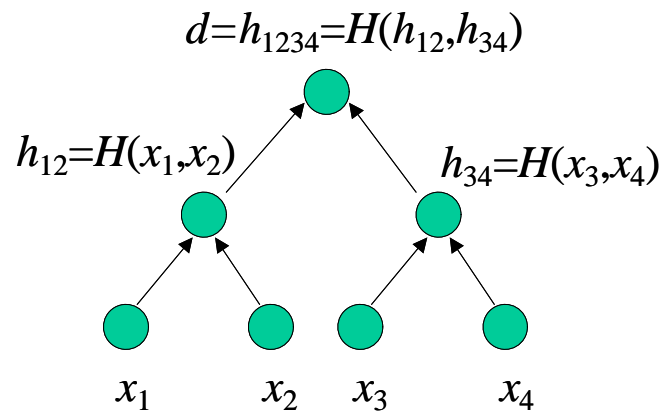


Figure 4: Merkle's authentication tree

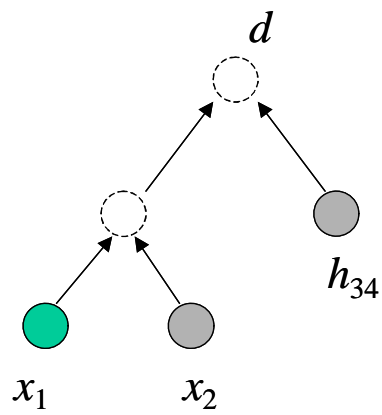


Figure 5: Authentication path for bitstring x_1

aggregation then the data to be time-stamped consists of output digest of the aggregation phase, otherwise the message digest contained in the time-stamping request is used.

The physical time value added to the data comes from the *time source*. It outputs universal coordinated time (UTC) with some error depending on its properties. This error is included in the time stamp together with the time value and it can consist of two components:

- error of the time source together with the error made when reading the time,
- error caused by aggregation (the times of receiving the time-stamping request and the moment of reading the time can differ by the length of one aggregation period).

If these components are known, the time-stamping service provider must supply them together with issued time stamps. This error information is taken into account by the user who compares the time stamps based on the physical time value.

1.3.3 Linking

The purpose of linking is to make the time stamp dependent on some or all of all the previous issued time stamps. It is not reasonable to create direct dependencies between every pair of time stamps as the resources (both time and space) needed to prove these dependencies would grow linearly in the number of issued time stamps. To save these resources, time stamps are made dependent following *linking schemes*. Linking schemes are collections of rules specifying which older time stamps need to be added as extra inputs to a new time stamp so that it would depend on all the previous ones either directly or via some intermediate time stamps. Linking schemes also specify what information needs to be published in order to ensure efficient comparison and extension procedures of timestamps.

Several linking schemes for several needs have been proposed, but time-stamp issuance and verification procedures are the same for all of them. For further information about various linking schemes see [ABSW01, BLLV98, BL98, BLS00, HS91, Lip99, Wil02].

Two types of linking information is kept in a time stamp:

1. **Historic message digests** that enable temporal comparison between this time stamp and earlier time stamps. This piece of information does not change over time.

2. **Extension information** enabling temporal comparison between the time stamp and later time stamps. This information is also used if it is necessary to prove that some published digest depends on the time stamp. As this proof can be carried out without any interaction with the time-stamping server, long-term proof value of time stamps can be achieved. The extension information can be changed during the extension process.

Linking also protects the physical time information included to time stamps, ensures its integrity and supports auditing the time stamping service.

1.3.4 Publishing

When issuing a time stamp, the server signs it with its private key in order to ensure the time stamp's integrity during the communication process and to allow instant verification.

The security level provided by signature depends on several circumstances: security of the signature scheme, security of the private key etc. In short term, potential vulnerabilities caused by these security assumptions can be regarded as negligible. But on the other hand time stamps are meant to provide long-term proof value (even if the underlying cryptographic primitives are broken), additional measures must be taken to guarantee this value.

Sufficient level of security can be achieved by publishing. As the name of this component reveals, the service provider publishes regularly the latest message digests (depending on all the issued time stamps) produced by its linking scheme in some widely accessible media. Printed media is very suitable for this purpose as all the physical copies of a newspaper are independent from each other and modifying all of them after printing is not realistic.

In order to verify the time stamp's integrity based on the published values, the verifier needs a hash chain connecting the time stamp and a published value. Such chains are provided by the extension service.

If a time stamp has been extended to a published value, its validity can not be denied or changed any more, even by the time-stamping authority itself; the only prerequisite is the integrity of the published information. Long-term security of the server's private key and the signature scheme are not required.

Publishing enables also auditing the time-stamping authority's actions. As the published values depend on all the previous time stamps, it must be possible to extend every time stamp to any later published digest. If the time-stamping service provider is unable to do so, it can be accused in cheating.

1.3.5 Extending

The main purpose of extending is to renew the information proving the validity of the time stamp. For extending a time stamp, the party interested in retaining the proof value, sends the time stamp back to the server. The server updates the validity proofs and sends the new time stamp back to the client.

In the process of extending the server

1. constructs a new hash chain based on the linking scheme; and
2. signs the extended time stamp.

The hash chain can be extended to a time stamp requested by the client or (if there is no specific request) to the last published digest. Extending to another time stamp allows comparing the two time stamps. Extending to the latest published digest makes the time stamp independent of the server's private key's security. Validity of a time stamp extended to a published value can be verified independently of any actions taken (or not taken) by the time-stamping server. E.g. the server can publish its private key without damaging the time stamps. This way there is no need to trust the time-stamping server in a long run.

The process of extending a time stamp does not create a new time stamp. The message digest of the data that was time-stamped does not (and can not) change. Only the information proving the time stamp's validity is updated.

2 Terms and definitions

Time stamp – collection of digital data created with a purpose to prove temporal relationship (before, after) between different events (e.g. the creation of some other digital data (document)).

Trust anchor – data that is used to verify time stamps but that can not be verified itself. Trust anchor must be distributed by some channel external to the system (where the system consists of the time-stamping service and its clients).

Self-signed certificate – a certificate that is signed with a private key corresponding to the public key contained in the certificate and the names of the owner and the issuer of the self-signed certificate coincide. Such a certificate does not prove that the owner really is the person referred in the certificate, but only that someone really knows the private key corresponding to the given public key. Self-signed certificates are used for preserving and transporting public

keys; the authenticity of such certificates must be verified before usage by some out-of-system means.

Time source – appliance, program or system outputting time value that is correct in some sense, e.g. local time of some geographic location.

Time value – time output of the time source.

Signature value – byte string obtained from a document and a signer's private key by applying a signature scheme (e.g. RSA).

Authentication tree – tree-like data structure where the label of every non-leaf is computed as a hash over its children's labels. The labels of the leaves are given and need not be computed.

Authentication path – (minimal) set of message digests aiming to prove that the label of the root of an authentication tree depends on the label of a leaf.

Linking scheme – a prescript determining how new digests must be created from input values and older digests in order to enable secure and efficient time stamp verification and comparison.

Hash chain – a sequence of digests created when comparing two time stamps.

3 Representation of Data Structures

This specification uses the language ASN.1 (*Abstract Syntax Notation One*, see the standard X.860 [X.680]) to represent data structures. The data described using ASN.1 is transformed into byte stream according to DER-encoding rules (standard X.690 [X.690]).

4 Requirements to the Time-Stamping Service

The following requirements are set to the time-stamping service.

1. If the time between issuance of two time stamps exceeds 1.5 maximum aggregation period durations, the earlier time stamp must be extensible to the later one.
2. If the time between issuance of two time stamps exceeds 1.5 maximum aggregation period durations, these time stamps must be directly comparable, provided the earlier one has been properly extended.

3. For all the time stamps issued before some published message digest it must be possible to extend them to this digest so that after the extension one can verify these time stamps without using the service provider's public key (see Subsection 8.1.4).

5 Time-Stamping Protocol

5.1 Hash Chain

Hash chain is an acyclic data structure representing cryptographic computations. The input of every computation step is a pair of message digests, one of which is the output of the previous computation step and the other one is given as a constant argument. (I.e. for every hash step there is an additional message digest given by the hash chain data structure. These digests form in fact a cryptographic proof that the output of the hash chain computation depends on its input).

Every element of the hash chain describes one computation step. During the hash step a cryptographic hash function is applied to the input pair and a new message digest is produced. The output of the step is $alg_1 || h_1 || alg_2 || h_2$, where h_i denotes a message digest, alg_i denotes the corresponding hash algorithm identifier and $||$ denotes concatenation. One of the values h_i is the hash computation output and another one is the constant digest given by the hash chain. Which one of the digests is h_1 and which one h_2 may vary from step to step.

Hash chain is used to represent the authentication path proving the dependence of the output of the aggregation process on some of its inputs. The output values of hash chain computations coincide with values in the nodes of authentication tree occurring on the path from some leaf to the root. If computed correctly, the last output value of the hash chain must coincide with the value in the authentication tree's root.

Hash chains are also used to ensure direct comparability of time stamps in linking process.

5.2 Data Formats

Hash chain's binary representation is constructed by concatenation of the byte strings representing the hash steps (see Figure 6).

Binary representation of a hash step consists of a concatenation of the following components (see Figure 7):



Figure 6: Byte string representation of hash chain

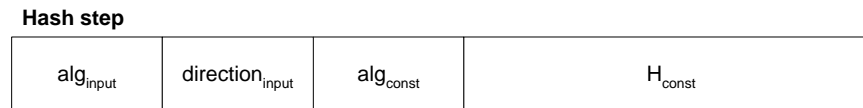


Figure 7: Byte string representation of hash step

- alg_{input} (length: 1 byte) – identifier of the algorithm used to hash the hash step input. This protocol specifies only one possible algorithm:
 - Identifier 0 – SHA1 (RFC3174, [SHA1])
- $direction_{input}$ (length: 1 byte) – shows whether the result of hashing is concatenated right or left from the constant message digest when forming the result of a hash step. If the value of this byte is 0 then the input is concatenated right from the input ((alg_2, h_2) is the digest of the input, (alg_1, h_1) is the given constant); if this byte is 1, then the input is concatenated left from the input ((alg_1, h_1) is the digest of the input, (alg_2, h_2) is the given constant).
- alg_{const} (length: 1 byte) – identifier of the algorithm used to compute the digest given as a constant. The possible values of this byte coincide with the values of alg_{input} .
- H_{const} (the length is determined as the output length of the algorithm specified by alg_{const} ; e.g. 160 bits in case of SHA-1) – the digest given as a constant.

When computing the output of the hash step, the two digests (the digest of the input and the digest given as a constant) are concatenated as shown in Figure 8 where

- alg_1 and alg_2 (both 1 byte) – the algorithms used to compute the digests (i.e. alg_{input} and alg_{const} in some order),

Output of hash step

alg ₁	H ₁	alg ₂	H ₂
------------------	----------------	------------------	----------------

Figure 8: Byte string representation of hash step output

- H_1 and H_2 (the lengths are determined by the output lengths of the hash algorithms) – message digests.

5.3 The Computation Algorithm

The computations specified by the hash chain are done by the following algorithm. The algorithm's input is the byte string `input_sequence`, the output is another byte string. The following state variables are used:

- `step_result` – the resulting value of the last hash step.
- `alg_input` – the field alg_{input} of the current hash step.
- `H_input` – the hash of the input of the current hash step.
- `direction_input` – the field $direction_{input}$ of the current hash step.
- `alg_const` – the field alg_{const} of the current hash step.
- `H_const` – the field H_{const} of the current hash step.

The computation algorithm consists of the following steps.

1. `step_result := input_sequence`
2. Start processing from the beginning of the hash chain.
3. If the hash chain is empty (i.e. has length 0 bytes) or the processing has come to the end of hash chain, proceed with step 8.
4. Decode next hash step and assign values to the variables `alg_input`, `direction_input`, `alg_const` and `H_const`.

5. Compute the input's message digest as
 $H_input := hash(alg_input, step_result)$, where *hash()* applies the hash function specified by the first argument to the second argument.
6. Compute the output of the hash step.
 - (a) If $direction_input=1$ then
 $step_result := alg_input || H_input || alg_const || H_const$
 - (b) If $direction_input=0$ then
 $step_result := alg_const || H_const || alg_input || H_input$
7. Continue with step 3.
8. Return $step_result$.

5.4 ASN.1 Representation

In the data structures used in time-stamping protocol the hash chain is represented by an ASN.1 type OCTET STRING. Its content is the byte string described in Subsection 5.2.

5.5 Time-Stamping Request

The time-stamping request is given by the TimestampRequest ASN.1 type.

```
TimestampRequest ::= SEQUENCE {
    version INTEGER { v1(1) },
    messageImprint MessageImprint,
    extensions Extensions OPTIONAL
}
```

The meanings of the fields contained in this structure are the following.

- *version* – version of the time-stamping protocol in use. The current specification describes version 1.
- *messageImprint* – message digest of the byte string to be time-stamped.

- `extensions` – reserved for later extensions of the protocol. No extensions are specified in the current protocol version.

In order to represent the message digest together with the identifier of the hash algorithm `MessageImprint` ASN.1 structure is used.

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

The fields of the last structure have the following meanings.

- `hashAlgorithm` – identifier of the algorithm used to create the message digest. The `AlgorithmIdentifier` ASN.1 type is defined in X.509 standard ([X.509]).
- `hashedMessage` – the message digest.

5.6 Time Stamp

The time stamp issued by the time-stamping service is represented as `LinkedTimestampToken` ASN.1 structure.

```
LinkedTimestampToken ::= SEQUENCE {
    version INTEGER { v1(1) },
    aggregations HashChain,
    timestampInfo TimestampInfo,
    linkingInfo HashChain,
    signatureInfo SignatureInfo,
    extensions Extensions OPTIONAL
}
```

The meanings of the fields contained in this structure are the following.

- `version` – version of the time-stamping protocol in use. The current specification describes version 1.

- `aggregations` – hash chain connecting the time-stamped data and the output of the aggregation component contained in the time stamp. If the input of the computation specified by the hash chain is the time-stamped bit string, the output must correspond to the field `timestampInfo.messageImprint`.

Remark. The aggregation component forms an authentication tree from the message digests of the time-stamped byte strings (which are obtained from the time-stamping requests). At the same time input of the hash chain representing the aggregation is the byte string itself. Hence, the first step in aggregation hash chain computation must be computing message digests of the byte strings forming the leaves of the authentication tree. For achieving this it is enough to guarantee that the parameter alg_{input} of the first step of the hash chain corresponds to the field `messageImprint.hashAlgorithm` of the time-stamping request. As a result, from the byte string to be time-stamped a message digest is computed and the result is concatenated with another digest given by the hash step. Altogether we get the first step in authentication path computation.

In case aggregation is not used, the length of this field is 0 and the field `timestampInfo.messageImprint` coincides with the `messageImprint` field from the request (as the output of the hash chain of zero length coincides with its input, only the message digest of the time-stamped byte string will be included into the time stamp).

- `timestampInfo` – this field contains data (aggregation result, historical message digests, time value) that is confirmed by the time-stamping service provider when issuing the time stamp, and that the service provider is responsible for. Message digest of the DER-encoding of the `timestampInfo` field is used as the input for linking.
- `linkingInfo` – hash chain representing extension information joining the linking input and some protected message digest. This digest can be verified using some trusted source (public key of the service provider or published data). In case when the message digest of the `timestampInfo` field is used as input to this hash chain computation, the output must coincide with the `signatureInfo.tbsSignatureInfo.dataHash` field of the time stamp. If a time stamp containing the `linkingInfo`

field is extended to published data, the output of the hash chain computation must coincide with the message digest contained in the published data.

- `signatureInfo` – time stamping service provider’s signature over the extension information.
- `extensions` – reserved for later extensions of the protocol. No extensions are specified in the current protocol version.

The `TimestampInfo` ASN.1 structure contains information that the time stamping service provider binds itself to when issuing the time stamp.

```
TimestampInfo ::= SEQUENCE {
    stampIdentifier OCTET STRING,
    messageImprint MessageImprint,
    genTime GeneralizedTime,
    accuracy [0] Accuracy OPTIONAL,
    historicalDigests SEQUENCE OF MessageImprint,
    extensions Extensions OPTIONAL
}
```

The meanings of the fields contained in this structure are the following.

- `stampIdentifier` – identifier of the time stamp. This identifier must be unique for every time stamp issued by the service provider. This property must also hold if the service is temporarily down (e.g. due to server crash). The users of the time-stamping service are not supposed to interpret this field in any way, for them this field contains meaningless byte string.
- `messageImprint` – result of the aggregation (see the explanation of the field aggregations of the structure `LinkedTimestampToken`).
- `genTime` – the time value expressing the time of time stamp creation.
- `accuracy` – maximal allowed error for the time value in the `genTime` field. The real time of time stamp creation is $accuracy \pm genTime$.
- `historicalDigests` – some of the previously computed digests. The user is not supposed to make any assumptions about the order and relationships between these digests.

- `extensions` – reserved for later extensions of the protocol. No extensions are specified in the current protocol.

In order to represent the maximum allowed error in the time value, the structure `Accuracy` is used.

```
Accuracy ::= SEQUENCE {
    seconds INTEGER OPTIONAL,
    millis [0] INTEGER (1..999) OPTIONAL,
    micros [1] INTEGER (1..999) OPTIONAL
}
```

The maximum allowed error for the time given in the time stamp can be computed by the formula

$$\text{seconds} + \text{millis} \cdot 10^3 + \text{micros} \cdot 10^6.$$

If either one of the fields `millis` or `micros` is missing, its value must be assumed to be 0.

In order to represent the time stamping service provider's signature the `SignatureInfo` ASN.1 structure is used.

```
SignatureInfo ::= SEQUENCE {
    tbsInformation TBSInformation,
    signatureAlgorithm AlgorithmIdentifier,
    -- signatureValue is calculated over
    -- DER-encoding of the tbsInformation field.
    signatureValue OCTET STRING
}
```

The meanings of the fields contained in this structure are the following.

- `tbsInformation` – the data going to be signed.
- `signatureAlgorithm` – identifier of the signing algorithm. ASN.1 type `AlgorithmIdentifier` is defined in the X.509 standard [X.509].
- `signatureValue` – the value of signature computed over the DER-encoding of the `tbsInformation` field with the algorithm specified by the `signatureAlgorithm` field.

The data to be signed is represented by `TBSInformation` ASN.1 structure.

```
TBSInformation ::= SEQUENCE {
    dataHash MessageImprint,
    certHash MessageImprint
}
```

The meanings of the fields contained in this structure are the following.

- `dataHash` – the result of computing the extension information.
- `certHash` – message digest of certificate corresponding to the key used to create the signature by the time-stamping service provider (the format of the certificate is specified in the standard X.509 [X.509]). This field connects the signature with the certificate of the service provider (the certificate can contain restrictions on the certificate's usage) and uniquely identifies the certificate that must be used to verify the time stamp. This message digest is computed over the DER-encoding of `Certificate` ASN.1 structure representing the certificate.

6 Time Stamp Extension Protocol

The client who wants to extend his/her time stamp, sends this time stamp to the time-stamping service provider together with the identifier referring to some published byte string or some other time stamp (that has been issued later than the client's one). As a reply, the time-stamping server returns another time stamp that has the `aggregations` and `timestampInfo` fields identical to the corresponding fields in the time stamp to be extended. The extension information of the reply contains authentication path from the `timestampInfo` field to the protected message digest referred to by the identifier in the extension request. In case of a published byte string this message digest is contained in the field `messageImprint` of the data structure `PublishedData`; in case of a time stamp this message digest is contained in the time stamp field `signatureInfo.tbsSignatureInfo.dataHash`.

6.1 Extension Request

Extension request is represented by `ExtensionRequest` ASN.1 structure.

```

ExtensionRequest ::= SEQUENCE {
    oldStamp LinkedTimestampToken,
    newStampIdentifier OCTET STRING OPTIONAL,
    extensions Extensions OPTIONAL
}

```

The meanings of the fields contained in this structure are the following.

- `oldStamp` – the time stamp to be extended.
- `newStampIdentifier` – identifies the message digest to which the given time stamp is extended. This digest can be either a published byte string or an identifier of some other time stamp issued by the same service provider. If this field has no value attached, the time stamp is extended to the latest issued time stamp. This approach can be used to renew time stamp's signature of the service provider in case the signing key has been changed.

Remark. The latest issued time stamp does not have to be published.

- `extensions` – reserved for later extensions of the protocol. No extensions are specified in the current protocol.

6.2 Extension Reply

The reply to the time stamp extension request is an extended time stamp represented by the `ExtensionResponse` ASN.1 type.

```

ExtensionResponse ::= LinkedTimestampToken

```

7 The Format of the Published Message Digest

Message digest is published as Base-32-encoded (see Appendix B) DER-encoding of `PublishedData` data structure.

```

PublishedData ::= SEQUENCE {
    messageImprint MessageImprint,
    stampIdentifier OCTET STRING
}

```

The meanings of the fields contained in this structure are the following.

- `messageImprint` – the digest to be published.
- `stampIdentifier` – byte string identifying the message digest to be published. This identifier must be unique for every time stamp issued by the service provider. This property must also hold if the service is temporarily down (e.g. due to server crash). The users of the time-stamping service are not supposed to interpret this field in any way, for them this field contains meaningless byte string.

8 Algorithms for Managing the Time Stamps

8.1 Finding Out the Time Value

This algorithm verifies the given time stamp and extracts the time value. The procedure consists of two steps:

- verifying the time stamp;
- extracting the time value.

A time stamp can be considered correct only if all the checks in the algorithm succeed. At every step one has to check that all the signature and hash algorithms are secure and reliable for the user. If any of the used cryptographic functions is insecure (from the verifier's viewpoint) the algorithm must be terminated and the time stamp must be considered incorrect.

The general structure of the algorithm is shown in Figure 9.

8.1.1 Inputs

The algorithm for extracting the creation time from a time stamp uses the following inputs.

- `input` – the time-stamped byte string.
- `stamp` – the time stamp under consideration.
- The trust anchor which is one of the following:

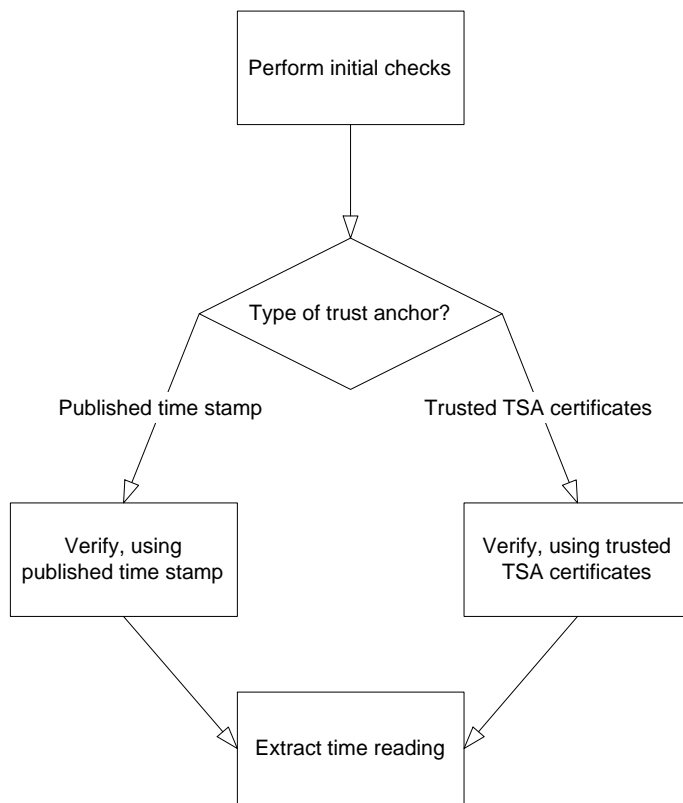


Figure 9: Time value extraction component

- `tsacert` – the list of certificates of trusted time stamping service providers. This list must originate from an authentic source and can contain self-signed certificates.
- `pubdata` – published message digest, in `PublishedData` format.

In order to use both the list of trusted certificates and the published data to verify time stamps, one needs to run the verification procedure several times, each time using different trust anchors. If one of the checks is successful, the time stamp is considered to be correct.

8.1.2 Output

This algorithm outputs the time value contained in the time stamp, or an error message.

8.1.3 State Variables

The procedure of time value extraction uses the following state variables.

- `aggrreq` – aggregation result before hashing.
- `linkresult` – data whose message digest is protected by publishing or the signature of time-stamping service provider.
- `sigcert` – certificate of the time-stamping service provider used to sign the time stamp.

8.1.4 Time Stamp Verification

1. Check that `stamp.version` is `v1`.
2. Apply the hash chain computation described by the field `stamp.aggregations` to the byte string input. Let the result be `aggrreq`.
3. Check that `stamp.timestampInfo.messageImprint` contains the message digest of `aggrreq`.
4. Apply the hash chain computation described by the field `stamp.linkingInfo` to the DER-encoding of the field `stamp.timestampInfo`. Let the result be `linkresult`.

5. If the trust anchor is the published string `pubdata`, check that `pubdata.messageImprint` contains the message digest of `linkresult`.
6. If the trust anchor is the list of certificates of trusted time stamping service providers `tsacert`, then:
 - (a) Check that `stamp.signatureInfo.tbsInformation.dataHash` contains the message digest of `linkresult`.
 - (b) Find such a certificate from the list `tsacert` whose DER-encoding's message digest coincides with the field `stamp.signatureInfo.tbsInformation.certHash`. Let this certificate be `sigcert`. If such a certificate can not be found, regard this time stamp as invalid.
 - (c) Check that the field `stamp.signatureInfo.tbsInformation.signatureValue` is computed using the secret key corresponding to the public key contained in `sigcert`. If this verification fails, regard this time stamp as invalid.

8.1.5 Extracting the Time Value

If all the verifications in the previous subsection were successful, output the value `stamp.timestampInfo.genTime`. If the field `stamp.timestampInfo.accuracy` has a value, it contains the accuracy of the time stamp, otherwise the accuracy remains unknown.

8.2 Comparing the Time Stamps

The time stamp comparison algorithm finds out the temporal relationship of two time stamps, answering the question “is the first time stamp earlier than the second one?”. For this purpose, either direct or indirect comparison can be used.

Two time stamps τ_1 and τ_2 are directly comparable under the following conditions.

- τ_1 is strictly earlier than τ_2 , i.e. they are not issued during the same aggregation period.
- τ_1 is extended to some time stamp τ that is strictly later than τ_2 .

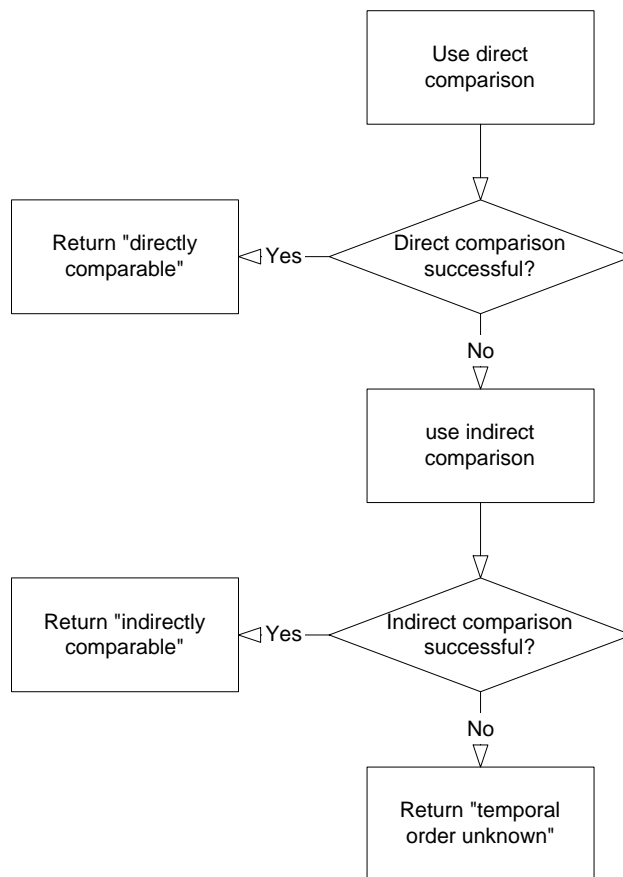


Figure 10: Time stamp comparison procedure

The time stamp can be considered correct only if all the verifications in the algorithm succeed. At every step one has to check that that all the signature and hash algorithms are secure and reliable for the user. If any of the used cryptographic functions is insecure (from the verifier's viewpoint) the comparison algorithm must be terminated and the time stamp must be considered incorrect.

The general structure of the algorithm is shown in Figure 10.

8.2.1 Inputs

The time stamp comparison algorithm uses the following inputs.

- The time stamps `stamp1` and `stamp2` to be compared.

- The trust anchor which is one of the following:
 - `tsacert` – the list of certificates of trusted time stamping service providers. This list must originate from an authentic source and can contain self-signed certificates.
 - `pubdata` – message digest published in `PublishedData` format.

In order to use both the list of trusted certificates and the published data to verify time stamps, one needs to run the verification procedure several times, each time using different trust sources. If one of the checks is successful, the time stamp is considered to be correct.

8.2.2 Outputs

The time stamp comparison algorithm gives the following outputs.

- `vresult` – the result of the comparison. It can be either “earlier” (if `stamp1` is earlier than `stamp2`), “error” (if one of the stamps is incorrect) or otherwise “not earlier”. The last result can mean either that `stamp2` is earlier than `stamp1` or that no temporal relationship can be determined.
- `vtype` – the used verification method. It can have value “direct”, “indirect” or “unknown”. The last value is used if `vresult` is either “error” or “not earlier”.

8.2.3 State Variables

The time stamp comparison algorithm uses the following state variables.

- `time1, time2` – time values from the time stamps `stamp1` and `stamp2`.
- `accuracy1, accuracy2` – accuracies found in time stamps `stamp1` and `stamp2`. If no accuracy is specified, the value of these variables can also be “unknown”.
- `t1max, t2min` – respectively the maximal possible value of the time value of `stamp1` and the minimal possible value of the time value of `stamp2`.

8.2.4 Direct Comparison of Time Stamps

1. Verify `stamp2` using the procedure from Subsection 8.1.4 without the steps 2 and 3 (they use the inputs `input` and `agrreq`). If the verification fails, return `vresult="error"` and `vtype="unknown"`.
2. Perform the hash chain computation described by the field `stamp1.linkingInfo`, using the DER-encoding of the field `stamp1.timestampInfo` as input. After computing the value of every hash step, check whether it coincides with some value from the list of message digests `stamp2.timestampInfo.historicalDigests`. If this is the case, return `vresult="earlier"` and `vtype="direct"`.

8.2.5 Indirect Comparison of Time Stamps

1. Verify `stamp2` using the procedure from Subsection 8.1.4 without the steps 2 and 3 (they use the inputs `input` and `agrreq`). If the verification fails, return `vresult="error"` and `vtype="unknown"`.
2. Extract the time value and the accuracy of `stamp1` as described by the procedure of Subsection 8.1.5. Let these be `time1` and `accuracy1`.
3. Extract the time value and the accuracy of `stamp2` as described by the procedure of Subsection 8.1.5. Let these be `time2` and `accuracy2`.
4. If `accuracy1="unknown"` then `t1max=time1`, else `t1max=time1+accuracy1`.
5. If `accuracy2="unknown"` then `t2min=time2`, else `t2min=time2-accuracy2`.
6. If `t1max < t2min` then return `vresult="earlier"` and `vtype="indirect"`, else return `vresult="not earlier"` and `vtype="unknown"`.

8.3 Extending a Time stamp Using Another Time Stamp

This algorithm allows off-line time stamp extension without communicating to the time-stamping service provider. The data required for extension comes

from another time stamp. The resulting time stamp satisfies all the requirements presented in Section 4.

In order to extend the time stamp τ_1 using the time stamp τ_2 (which is itself extended to the time stamp τ) the following preconditions must hold:

- the time stamp τ_1 must be strictly earlier than the stamp τ_2 ,
- the time stamp τ_1 must be extended to a time stamp strictly later than τ_2 .

The algorithm below does not verify these preconditions. If the time stamps were not suitable, the algorithm quits with an error.

8.3.1 Inputs

The extension algorithm uses the following inputs.

- `ext` – the time stamp to be extended.
- `source` – the time stamp to which `ext` is going to be extended.

8.3.2 Outputs

The algorithm returns the extended time stamp or an error.

8.3.3 State variables

The extension algorithm uses the following state variables.

- `length` – the length of the hash chain to be included in the resulting time stamp.
- `ext_length` – the number of hash steps in the extension information of `ext`.
- `source_length` – the number of hash steps in the extension information of `source`.
- `step_result` – array of `ext_length` elements containing the results of hash step computations of the extension information of `ext`.

- `step_const` – array of `source_length` elements containing the message digest values given in the extension information of `source` as constants.
- `ext_index` – the number of hash steps from `ext` to be used in the result.
- `source_index` – the number of hash steps from `source` to be used in the result.
- `stamp_result` – the resulting extended time stamp.
- `chain_result` – the hash chain forming the extension information of `stamp_result`.

8.3.4 The Algorithm

1. Find the number of hash steps in the hash chain `ext.linkingInfo` and let it be `ext_length`. If `ext_length=0` then exit with an error.
2. Find the number of hash steps in the hash chain `source.linkingInfo` and let it be `source_length`. If `source_length=0` then exit with an error.
3. Perform the hash chain computation described by the field `ext.timestampInfo`, using the DER-encoding of the field `ext.timestampInfo` as input. Let the results of the hash steps form the array `step_result`.
4. Let the message digests given in the hash chain computation contained in the field `source.linkingInfo` form the array `step_const`.
5. Find a common point of the hash chains `ext.linkingInfo` and `source.linkingInfo` using the following procedure.
 - (a) `length=2`
 - (b) `ext_index=max(1,length – source_length)`
 - (c) `source_index=length–ext_index`
 - (d) If `step_result[ext_index]=step_const[source_length–source_index+1]` then the common point has been found; exit the procedure.

- (e) $\text{ext_index} = \text{ext_index} + 1$. If $\text{ext_index} \leq \text{length} - 1$ and $\text{ext_index} \leq \text{ext_length}$ then proceed with step (c).
 - (f) $\text{length} = \text{length} + 1$. If $\text{length} \leq \text{ext_length} + \text{source_length}$ then proceed with step (b).
 - (g) Time stamp extension was unsuccessful, since source did not contain a suitable hash chain. Exit the operation with an error.
6. Initialize `chain_result` with a hash chain of length 0.
 7. Add the steps number $1 \dots \text{ext_index}$ of the hash chain `ext.linkingInfo` to the end of hash chain `chain_result`.
 8. Compute $\text{source_length} - \text{source_index}$ first hash steps of the hash chain contained in `source.linkingInfo`, using the DER-encoding of `source.timestampInfo` as input (if $\text{source_length} = \text{source_index}$ then output equals the input).
 9. Decode the hash step number $\text{source_length} - \text{source_index} + 1$ of the hash chain `source.linkingInfo`. Add a new hash step to the end of `chain_result`, filling the fields in as follows.
 - `alg_input` – the field `alg_const` of the hash step number $\text{source_length} - \text{source_index} + 1$ of the hash chain `source.linkingInfo`.
 - `direction_input` –
 - 1, if the field `direction_input` of the hash step number $\text{source_length} - \text{source_index} + 1$ of the hash chain `source.linkingInfo` is 0.
 - 0, if the field `direction_input` of the hash step number $\text{source_length} - \text{source_index} + 1$ of the hash chain `source.linkingInfo` is 1.
 - `alg_const` – the field `alg_input` of the hash step number $\text{source_length} - \text{source_index} + 1$ of the hash chain `source.linkingInfo`.
 - `H_const` – the hash of the result of the previous step, computed with the algorithm shown in the `alg_input` field of the hash step number $\text{source_length} - \text{source_index} + 1$ of `source.linkingInfo`.

10. Add the hash steps number `source_length-source_index+2...` `source_length` of the hash chain `source.linkingInfo` in the end of `chain_result`.
11. `stamp_result.version=ext.version`
12. `stamp_result.aggregations=ext.aggregations`
13. `stamp_result.timestampInfo=ext.timestampInfo`
14. `stamp_result.linkingInfo=chain_result`
15. `stamp_result.signatureInfo=chain_result.signatureInfo`
16. `stamp_result.extensions=ext.extensions`
17. Add the extensions contained in the field `source.extensions` to the field `stamp_result.extensions`.
18. Output `stamp_result`.

9 Transport Protocol

To send the requests and replies for time-stamping and extension, HTTP protocol version 1.1 (RFC2616) is used. MIME-types of the data elements are the following.

- Time-stamping request:
Content-Type: `application/x-linked-timestamp-request`
The content: DER-encoded `TimestampRequest` structure.
- Time-stamping reply:
Content-Type: `application/x-linked-timestamp`
The content: DER-encoded `LinkedTimestampToken` structure.
- Time stamp extension request:
Content-Type: `application/x-linked-extension-request`
The content: DER-encoded `ExtensionRequest` structure.
- Time stamp extension reply: the same as time-stamping reply.

For the type application/x-linked-timestamp the application can add optional parameters “name” and “filename”. Adding the file name helps to retain the message type in case when the time stamps are saved into a file. If these parameters are added, it is suggested to use the file name extension .LTS.

10 Supported cryptographic algorithms

All the users and providers of the time stamping service must support hash function SHA-1 ([SHA1]). All the users and providers of the time stamping service must support signature algorithm RSA together with hash algorithm SHA-1 (`sha1WithRSAEncryption`) according to the specification [RFC2313].

A ASN.1 module

```
TimeStampingProtocol { ... }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL --

IMPORTS
    Extensions, AlgorithmIdentifier
    FROM PKIX1Explicit88 { iso(1) identified-organization(3)
        dod(6) internet(1) security(5) mechanisms(5) pkix(7)
        id-mod(0) id-pkix1-explicit-88(1) }

HashChain ::= OCTET STRING

TimestampRequest ::= SEQUENCE {
    version          INTEGER { v1(1) },
    messageImprint  MessageImprint,
    extensions      Extensions OPTIONAL
}

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

```

}

LinkedTimestampToken ::= SEQUENCE {
    version            INTEGER { v1(1) },
    aggregations       HashChain,
    timestampInfo      TimestampInfo,
    linkingInfo        HashChain,
    signatureInfo      SignatureInfo,
    extensions         Extensions OPTIONAL
}

TimestampInfo ::= SEQUENCE {
    stampIdentifier    OCTET STRING,
    messageImprint     MessageImprint,
    genTime            GeneralizedTime,
    accuracy           [0] Accuracy OPTIONAL,
    historicalDigests  SEQUENCE OF MessageImprint,
    extensions         Extensions OPTIONAL
}

Accuracy ::= SEQUENCE {
    seconds            INTEGER                OPTIONAL,
    millis            [0] INTEGER (1..999)   OPTIONAL,
    micros            [1] INTEGER (1..999)   OPTIONAL
}

SignatureInfo ::= SEQUENCE {
    tbsInformation     TBSInformation,
    signatureAlgorithm AlgorithmIdentifier,
    -- signatureValue is calculated over DER-encoding of the
    -- tbsInformation field.
    signatureValue     OCTET STRING
}

TBSInformation ::= SEQUENCE {
    dataHash           MessageImprint,
    certHash           MessageImprint
}

ExtensionRequest ::= SEQUENCE {
    oldStamp           LinkedTimestampToken,
    newStampIdentifier OCTET STRING OPTIONAL,
    extensions         Extensions OPTIONAL
}

```

```

}

ExtensionResponse ::= LinkedTimestampToken

PublishedData ::= SEQUENCE {
    messageImprint    MessageImprint,
    stampIdentifier   OCTET STRING
}

END

```

B Base32 encoding

The following description is taken from the IETF draft [Jos02].

The Base 32 encoding is designed to represent arbitrary sequences of octets in a form that needs to be case insensitive but need not be humanly readable.

A 33-character subset of US-ASCII is used, enabling 5 bits to be represented per printable character. (The extra 33rd character, “=”, is used to signify a special processing function.)

The encoding process represents 40-bit groups of input bits as output strings of 8 encoded characters. Proceeding from left to right, a 40-bit input group is formed by concatenating 5 8bit input groups. These 40 bits are then treated as 8 concatenated 5-bit groups, each of which is translated into a single digit in the base 32 alphabet. When encoding a bit stream via the base 32 encoding, the bit stream must be presumed to be ordered with the most-significant-bit first. That is, the first bit in the stream will be the high-order bit in the first 8bit byte, and the eighth bit will be the low-order bit in the first 8bit byte, and so on.

Each 5-bit group is used as an index into an array of 32 printable characters. The character referenced by the index is placed in the output string. These characters, identified in Table 1, below, are selected from US-ASCII digits and uppercase letters.

Special processing is performed if fewer than 40 bits are available at the end of the data being encoded. A full encoding quantum is always completed at the end of a body. When fewer than 40 input bits are available in an input group, zero bits are added (on the right) to form an integral number of 5-bit groups. Padding at the end of the data is performed using the “=” character. Since all base 32 input

Table 1: The Base 32 Alphabet

Value	Encoding	Value	Encoding	Value	Encoding	Value	Encoding
0	A	9	J	18	S	27	3
1	B	10	K	19	T	28	4
2	C	11	L	20	U	29	5
3	D	12	M	21	V	30	6
4	E	13	N	22	W	31	7
5	F	14	O	23	X		
6	G	15	P	24	Y	(pad)	=
7	H	16	Q	25	Z		
8	I	17	R	26	2		

is an integral number of octets, only the following cases can arise:

1. the final quantum of encoding input is an integral multiple of 40 bits; here, the final unit of encoded output will be an integral multiple of 8 characters with no “=” padding,
2. the final quantum of encoding input is exactly 8 bits; here, the final unit of encoded output will be two characters followed by six “=” padding characters,
3. the final quantum of encoding input is exactly 16 bits; here, the final unit of encoded output will be four characters followed by four “=” padding characters,
4. the final quantum of encoding input is exactly 24 bits; here, the final unit of encoded output will be five characters followed by three “=” padding characters, or
5. the final quantum of encoding input is exactly 32 bits; here, the final unit of encoded output will be seven characters followed by one “=” padding character.

C Examples of Time Stamps

This appendix contains examples of the messages used in time-stamping protocols. These examples can be used as illustration of the data structures repre-

sented in the current specification or to test software.

All the examples are given both as decoded ASN.1 structures and as PEM-encoded byte strings that can be extracted from the text and used to test the protocol implementation.

C.1 Private Key and Certificate of the Time-Stamping Service Provider

C.1.1 Private Key of the Time-Stamping Service Provider

```
0 30 604: SEQUENCE {
4 02 1: INTEGER 0
7 02 129: INTEGER
: 00 9B 1A 19 5B CA 98 15 2F 96 8E 1C 17 FD 18 F7
: 56 C3 4F F7 EF 2C 6A 45 38 C8 4D C4 61 3C 17 06
: C2 F3 2A 77 05 87 7C 85 CA F4 04 6C C8 BB 71 0B
: 44 68 88 28 03 CF C8 E4 6D 24 86 BE B1 5E 74 63
: 1C 8B F5 9A C4 3E 75 E6 71 15 77 EC C4 89 55 C5
: 09 6C 87 29 23 8C 0A 73 C1 23 C0 5D 38 6B 9B BA
: 01 2C 2A 48 FA DD 03 AC 17 BA 57 29 69 F4 7F 3F
: 75 C9 A1 4B DF 03 0E 42 7B 57 E9 E5 29 98 27 4D
: FD
139 02 3: INTEGER 65537
144 02 128: INTEGER
: 07 03 28 10 59 59 84 E3 B0 E7 DD E7 4B BF 1C 37
: A4 FE F1 93 B3 AB 5E 53 D1 E2 8A 35 67 35 17 4E
: 2B 16 49 69 4B 95 DA AA B1 5B 9D DB 79 76 03 EF
: 64 D6 7B 10 A9 0D 49 1E 92 1F 31 71 ED 76 4F A0
: C1 C0 51 0B E9 AD 33 F6 25 01 D5 FC 15 70 4F D6
: 9D 39 27 9F BF A6 8D E8 2A 1B 06 35 97 67 21 33
: C8 0F 4C 38 50 D8 A5 73 F4 7E 84 DD 32 24 CB 45
: BC 20 FD 71 19 6A 1C C4 B0 27 0A 86 24 B0 77 15
275 02 65: INTEGER
: 00 CD 83 5E 12 69 B0 D1 31 3E 10 C1 78 FA AC DA
: EA FA BB 4E F4 66 99 0E 55 72 66 51 AA F0 38 1D
: 03 05 1B 00 FC 4E 2B 86 6B A4 3C D6 C3 85 3C 98
: A1 8C 62 B3 BF 61 B7 61 A8 98 E9 9E C2 70 A7 4B
: DB
342 02 65: INTEGER
: 00 C1 34 64 97 54 A1 58 59 EF C4 E9 A3 8F A9 94
: EF 9C 69 32 AA 3D B1 E3 54 04 B5 DC 27 21 FD 86
: 1A D0 7B 2B 57 E3 84 BF 10 92 A6 2B F7 D2 5C 5C
: D3 6D 4E 5B 1E 42 A0 BB C6 7B FA 99 C3 10 77 21
: 07
```

```

409 02 65: INTEGER
      : 00 8B 50 D1 95 B3 D6 2C 89 B0 24 55 B3 78 63 73
      : 68 35 9E 82 70 85 98 9E 31 E8 82 5A 3C 81 E9 D8
      : 28 F2 EE 12 65 F7 6B E4 0C D6 62 73 C8 49 6B 01
      : A1 09 F4 E3 1B 0A D7 F0 B9 05 27 E1 FA B7 B9 81
      : 5F
476 02 64: INTEGER
      : 76 41 CD 78 5B 52 CE B3 6D F1 47 58 D7 EC 62 BF
      : 2F 2B AC D9 4B 1E 01 19 42 30 D6 6A 15 82 0A 7D
      : 8B 32 78 DB 38 E5 DD D5 15 0E 25 47 71 39 1B 90
      : 46 0D 79 F4 51 AF F4 7F 8F 0F 5A E2 6C C7 9E 2D
542 02 64: INTEGER
      : 34 37 8A 97 D8 0A EF A4 5F A5 69 3B 77 A8 81 61
      : DD 51 FF C2 84 79 20 CB 1E 15 24 63 62 F7 55 E5
      : D2 A9 D7 A3 A0 49 6B 1C 90 12 08 E1 EA CE 54 3A
      : 61 11 8E 04 F7 F6 86 C7 9F CB A7 AE 10 F4 5B 30
      : }

```

C.1.2 Certificate of the Time-Stamping Service Provider

```

0 30 424: SEQUENCE {
4 30 273: SEQUENCE {
8 A0 3: [0] {
10 02 1: INTEGER 2
      : }
13 02 1: INTEGER 0
16 30 13: SEQUENCE {
18 06 9: OBJECT IDENTIFIER
      : sha1withRSAEncryption (1 2 840 113549 1 1 5)
29 05 0: NULL
      : }
31 30 26: SEQUENCE {
33 31 24: SET {
35 30 22: SEQUENCE {
37 06 3: OBJECT IDENTIFIER commonName (2 5 4 3)
42 13 15: PrintableString 'Ajatempliteenus'
      : }
      : }
      : }
59 30 30: SEQUENCE {
61 17 13: UTCTime '020430073445Z'
76 17 13: UTCTime '020530073445Z'
      : }
91 30 26: SEQUENCE {
93 31 24: SET {

```

```

95 30 22: SEQUENCE {
97 06 3: OBJECT IDENTIFIER commonName (2 5 4 3)
102 13 15: PrintableString 'Ajatempliteenus'
: }
: }
: }
119 30 159: SEQUENCE {
122 30 13: SEQUENCE {
124 06 9: OBJECT IDENTIFIER rsaEncryption (1 2 840 113549 1 1 1)
135 05 0: NULL
: }
137 03 141: BIT STRING 0 unused bits
: 30 81 89 02 81 81 00 9B 1A 19 5B CA 98 15 2F 96
: 8E 1C 17 FD 18 F7 56 C3 4F F7 EF 2C 6A 45 38 C8
: 4D C4 61 3C 17 06 C2 F3 2A 77 05 87 7C 85 CA F4
: 04 6C C8 BB 71 0B 44 68 88 28 03 CF C8 E4 6D 24
: 86 BE B1 5E 74 63 1C 8B F5 9A C4 3E 75 E6 71 15
: 77 EC C4 89 55 C5 09 6C 87 29 23 8C 0A 73 C1 23
: C0 5D 38 6B 9B BA 01 2C 2A 48 FA DD 03 AC 17 BA
: 57 29 69 F4 7F 3F 75 C9 A1 4B DF 03 0E 42 7B 57
: E9 E5 29 98 27 4D FD 02 03 01 00 01
: }
: }
281 30 13: SEQUENCE {
283 06 9: OBJECT IDENTIFIER
: sha1withRSAEncryption (1 2 840 113549 1 1 5)
294 05 0: NULL
: }
296 03 129: BIT STRING 0 unused bits
: 61 E0 29 9E 05 E1 89 A4 BB D1 A8 8B C6 16 95 B4
: E5 BE F0 A2 46 63 81 BE D8 93 11 58 26 F6 9A A3
: D1 62 89 F4 F7 3F E3 62 88 0B 3B 95 28 BD 7A B5
: 03 64 7D 66 22 8A F4 0D D8 40 BF E1 FA BB 51 34
: 25 2E AB 17 F1 6C 89 BB 44 D4 C6 89 11 DE D5 E4
: 0D 6D 5C F5 C3 6D C0 84 DB F1 2A DF 0B 38 F5 66
: 75 69 79 B8 1D D6 19 E5 C2 58 FE 9E 63 48 F3 1C
: B6 70 91 8C D7 65 ED A5 9A F1 82 8E 8E BC AF 98
: }

```

C.2 Time-stamped Messages

C.2.1 Time-stamped Message 1

The message: "12345678901234567890"
Time-stamping request:

```

0 30 38: SEQUENCE {
2 02 1:  INTEGER 1
5 30 33:  SEQUENCE {
7 30 9:   SEQUENCE {
9 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
16 05 0:   NULL
      :   }
18 04 20:  OCTET STRING
      :   7E 0A 12 42 BD 8E F9 04 4F 27 DC A4 5F 5F 72 AD
      :   5A 11 25 BF
      :   }
      :   }

```

C.2.2 Time-stamped Message 2

The message: “abcdefghijklmnopqrstuvwxy”

Time-stamping request:

```

0 30 38: SEQUENCE {
2 02 1:  INTEGER 1
5 30 33:  SEQUENCE {
7 30 9:   SEQUENCE {
9 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
16 05 0:   NULL
      :   }
18 04 20:  OCTET STRING
      :   32 D1 0C 7B 8C F9 65 70 CA 04 CE 37 F2 A1 9D 84
      :   24 0D 3A 89
      :   }
      :   }

```

C.2.3 Time-stamped Message 3

The message: “ABCDEFGHIJKLMNPOQRSTUVWXYZ”

Time-stamping request:

```

0 30 38: SEQUENCE {
2 02 1:  INTEGER 1
5 30 33:  SEQUENCE {
7 30 9:   SEQUENCE {
9 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
16 05 0:   NULL
      :   }
18 04 20:  OCTET STRING
      :   80 25 6F 39 A9 D3 08 65 0A C9 0D 9B E9 A7 2A 95

```

```

:      62 45 45 74
:      }
:      }

```

C.3 Time Stamps without Aggregation

From all the time stamps of this section, it must be possible to extract the time value using the algorithm presented in Section 8.1 and taking the certificate of Subsection C.1.2 as trust anchor.

C.3.1 Time Stamp for Message 1

```

0 30 638: SEQUENCE {
4 02 1: INTEGER 1
7 04 0: OCTET STRING
9 30 360: SEQUENCE {
13 04 9: OCTET STRING
:      08 00 00 00 00 00 11 D9 80
24 30 33: SEQUENCE {
26 30 9: SEQUENCE {
28 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05 0: NULL
:      }
37 04 20: OCTET STRING
:      7E 0A 12 42 BD 8E F9 04 4F 27 DC A4 5F 5F 72 AD
:      5A 11 25 BF
:      }
59 18 15: GeneralizedTime '20020430151018Z'
76 A0 11: [0] {
78 30 9: SEQUENCE {
80 02 1: INTEGER 5
83 A0 4: [0] {
85 02 2: INTEGER 500
:      }
:      }
:      }
89 30 280: SEQUENCE {
93 30 33: SEQUENCE {
95 30 9: SEQUENCE {
97 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
104 05 0: NULL
:      }
106 04 20: OCTET STRING
:      64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D

```

```

      :           F7 85 B9 06
      :           }
128 30 33: SEQUENCE {
130 30 9:  SEQUENCE {
132 06 5:  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
139 05 0:  NULL
      :           }
141 04 20: OCTET STRING
      :           03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
      :           A6 91 CC 2C
      :           }
163 30 33: SEQUENCE {
165 30 9:  SEQUENCE {
167 06 5:  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
174 05 0:  NULL
      :           }
176 04 20: OCTET STRING
      :           88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      :           D4 7D B7 77
      :           }
198 30 33: SEQUENCE {
200 30 9:  SEQUENCE {
202 06 5:  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
209 05 0:  NULL
      :           }
211 04 20: OCTET STRING
      :           B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      :           A6 4A 7F AE
      :           }
233 30 33: SEQUENCE {
235 30 9:  SEQUENCE {
237 06 5:  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
244 05 0:  NULL
      :           }
246 04 20: OCTET STRING
      :           C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
      :           F2 A8 A3 DA
      :           }
268 30 33: SEQUENCE {
270 30 9:  SEQUENCE {
272 06 5:  OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
279 05 0:  NULL
      :           }
281 04 20: OCTET STRING
      :           90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42

```

```

:          5B DC 2F 52
:          }
303 30 33: SEQUENCE {
305 30 9: SEQUENCE {
307 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
314 05 0: NULL
:          }
316 04 20: OCTET STRING
:          49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
:          B5 1F 4F 17
:          }
338 30 33: SEQUENCE {
340 30 9: SEQUENCE {
342 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
349 05 0: NULL
:          }
351 04 20: OCTET STRING
:          FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
:          6E 83 D8 8B
:          }
:          }
:          }
373 04 46: OCTET STRING
:          00 00 00 93 ED DB EE 22 57 F3 E1 BE CE 3A EE 86
:          A3 5A 67 B6 D1 15 5D 00 00 00 93 ED DB EE 22 57
:          F3 E1 BE CE 3A EE 86 A3 5A 67 B6 D1 15 5D
421 30 218: SEQUENCE {
424 30 70: SEQUENCE {
426 30 33: SEQUENCE {
428 30 9: SEQUENCE {
430 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
437 05 0: NULL
:          }
439 04 20: OCTET STRING
:          38 09 FC 40 A9 A3 4A E7 BC AB 37 22 9C A1 90 BC
:          A4 85 F4 3D
:          }
461 30 33: SEQUENCE {
463 30 9: SEQUENCE {
465 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
472 05 0: NULL
:          }
474 04 20: OCTET STRING
:          7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
:          AE 3A 27 8F

```

```

      :      }
      :      }
496 30 13: SEQUENCE {
498 06  9:   OBJECT IDENTIFIER
      :       sha1withRSAEncryption (1 2 840 113549 1 1 5)
509 05  0:   NULL
      :      }
511 04 128: OCTET STRING
      :       3C C2 30 DC E3 B2 96 DB 42 FB 94 C8 9F CF 53 5E
      :       3C 2A 31 92 7C CE 7B 25 90 96 FB 27 F6 AD 5A 49
      :       DC 6E 7E AF F6 08 06 3B CA 0C 34 FF 88 7C 0A 0D
      :       CE CC 2F AF B5 21 FA E5 D5 04 E7 17 6B 7A A6 A9
      :       11 2E 77 36 CD AE 58 F0 2F 3E 81 26 F2 4E B0 F9
      :       1B 47 AC 63 C0 24 99 45 09 CF BD 9A 80 42 72 52
      :       25 DD 7E CB D5 BC A2 F8 82 55 CE D1 CB AB 86 B8
      :       C0 6F 3D DA AD 1F DC 6F A0 52 25 E6 1E C0 DC E1
      :       }
      :      }
      :     }

```

C.3.2 Time Stamp for Message 2

```

 0 30 789: SEQUENCE {
 4 02  1:   INTEGER 1
 7 04  0:   OCTET STRING
 9 30 465: SEQUENCE {
13 04  9:   OCTET STRING
      :       08 00 00 00 00 00 11 D9 8B
24 30 33: SEQUENCE {
26 30  9:   SEQUENCE {
28 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05  0:       NULL
      :      }
37 04 20:   OCTET STRING
      :       32 D1 0C 7B 8C F9 65 70 CA 04 CE 37 F2 A1 9D 84
      :       24 0D 3A 89
      :      }
59 18 15: GeneralizedTime '20020430151022Z'
76 A0 11: [0] {
78 30  9:   SEQUENCE {
80 02  1:       INTEGER 5
83 A0  4:   [0] {
85 02  2:       INTEGER 500
      :      }
      :     }
      :    }

```

```

89 30 385: SEQUENCE {
93 30 33:   SEQUENCE {
95 30 9:    SEQUENCE {
97 06 5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
104 05 0:    NULL
      :
      }
106 04 20:   OCTET STRING
      :      64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
      :      F7 85 B9 06
      :
      }
128 30 33:   SEQUENCE {
130 30 9:    SEQUENCE {
132 06 5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
139 05 0:    NULL
      :
      }
141 04 20:   OCTET STRING
      :      03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
      :      A6 91 CC 2C
      :
      }
163 30 33:   SEQUENCE {
165 30 9:    SEQUENCE {
167 06 5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
174 05 0:    NULL
      :
      }
176 04 20:   OCTET STRING
      :      88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      :      D4 7D B7 77
      :
      }
198 30 33:   SEQUENCE {
200 30 9:    SEQUENCE {
202 06 5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
209 05 0:    NULL
      :
      }
211 04 20:   OCTET STRING
      :      B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      :      A6 4A 7F AE
      :
      }
233 30 33:   SEQUENCE {
235 30 9:    SEQUENCE {
237 06 5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
244 05 0:    NULL
      :
      }
246 04 20:   OCTET STRING
      :      C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
      :      F2 A8 A3 DA

```

```

:
}
268 30 33: SEQUENCE {
270 30 9: SEQUENCE {
272 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
279 05 0: NULL
:
}
281 04 20: OCTET STRING
: 90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
: 5B DC 2F 52
:
}
303 30 33: SEQUENCE {
305 30 9: SEQUENCE {
307 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
314 05 0: NULL
:
}
316 04 20: OCTET STRING
: 49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
: B5 1F 4F 17
:
}
338 30 33: SEQUENCE {
340 30 9: SEQUENCE {
342 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
349 05 0: NULL
:
}
351 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B
:
}
373 30 33: SEQUENCE {
375 30 9: SEQUENCE {
377 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
384 05 0: NULL
:
}
386 04 20: OCTET STRING
: 21 85 2A 39 7E 3A 9B CF EB 18 AD F4 7C 9F 42 EE
: 5B 5A 65 88
:
}
408 30 33: SEQUENCE {
410 30 9: SEQUENCE {
412 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
419 05 0: NULL
:
}
421 04 20: OCTET STRING
: 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E 1C
: 11 0C 9A 86

```

```

:
}
443 30 33: SEQUENCE {
445 30 9: SEQUENCE {
447 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
454 05 0: NULL
:
}
456 04 20: OCTET STRING
: AC 6E 6F C1 B4 C0 0D 0D F4 D5 6A 36 D6 04 D9 22
: 2E 7B 85 2E
:
}
:
}
:
}
478 04 92: OCTET STRING
: 00 00 00 7D 62 30 D0 D7 4B FB 2A 57 4C F4 5B 11
: CB 14 3A 52 9B FD 6C 00 00 00 AC 6E 6F C1 B4 C0
: 0D 0D F4 D5 6A 36 D6 04 D9 22 2E 7B 85 2E 00 00
: 00 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E
: 1C 11 0C 9A 86 00 00 00 92 D3 12 18 C5 BA 9B B7
: C8 FD 69 BC BE 8D B4 20 9D 86 81 27
572 30 218: SEQUENCE {
575 30 70: SEQUENCE {
577 30 33: SEQUENCE {
579 30 9: SEQUENCE {
581 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
588 05 0: NULL
:
}
590 04 20: OCTET STRING
: 64 3D 5C 8C 6F 42 2F BB 24 EC 59 76 23 88 CD E0
: C6 6E C5 37
:
}
612 30 33: SEQUENCE {
614 30 9: SEQUENCE {
616 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
623 05 0: NULL
:
}
625 04 20: OCTET STRING
: 7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
: AE 3A 27 8F
:
}
:
}
647 30 13: SEQUENCE {
649 06 9: OBJECT IDENTIFIER
: sha1withRSAEncryption (1 2 840 113549 1 1 5)
660 05 0: NULL
:
}

```

```

662 04 128:    OCTET STRING
:             06 0B D1 99 6C 4F 94 18 28 42 9D 6A 11 0C 44 7F
:             F5 15 5F 5A 4F 7C 0A 37 4B 2F 6D 7D 80 B2 A7 ED
:             65 7B 74 FD 3F 39 5F 92 E0 49 52 03 FF 82 83 20
:             F3 EA C4 C0 89 61 9C 9C E7 3C 1B 78 8E E2 DE 72
:             18 C9 53 06 C1 CE E9 63 19 F5 92 A7 1E F4 BF 83
:             3B D7 32 A3 1C 69 9B 9E B2 19 96 AE 90 B2 A7 79
:             36 89 E6 57 D3 0C 8D 0C F0 CC 52 F9 F2 BE 8F 33
:             E3 70 78 80 4C 30 26 BF ED 39 5A 05 B5 52 56 91
:             }
:         }

```

C.3.3 Time Stamp for Message 3

```

0 30 753: SEQUENCE {
4 02 1:   INTEGER 1
7 04 0:   OCTET STRING
9 30 452: SEQUENCE {
13 04 9:   OCTET STRING
:         08 00 00 00 00 00 11 D9 A9
24 30 33: SEQUENCE {
26 30 9:   SEQUENCE {
28 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05 0:   NULL
:         }
37 04 20: OCTET STRING
:         80 25 6F 39 A9 D3 08 65 0A C9 0D 9B E9 A7 2A 95
:         62 45 45 74
:         }
59 18 15: GeneralizedTime '20020430151212Z'
76 30 385: SEQUENCE {
80 30 33: SEQUENCE {
82 30 9:   SEQUENCE {
84 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
91 05 0:   NULL
:         }
93 04 20: OCTET STRING
:         64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
:         F7 85 B9 06
:         }
115 30 33: SEQUENCE {
117 30 9:   SEQUENCE {
119 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
126 05 0:   NULL
:         }

```

```

128 04 20:      OCTET STRING
      :      03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
      :      A6 91 CC 2C
      :      }
150 30 33:      SEQUENCE {
152 30  9:      SEQUENCE {
154 06  5:      OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
161 05  0:      NULL
      :      }
163 04 20:      OCTET STRING
      :      88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      :      D4 7D B7 77
      :      }
185 30 33:      SEQUENCE {
187 30  9:      SEQUENCE {
189 06  5:      OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
196 05  0:      NULL
      :      }
198 04 20:      OCTET STRING
      :      B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      :      A6 4A 7F AE
      :      }
220 30 33:      SEQUENCE {
222 30  9:      SEQUENCE {
224 06  5:      OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
231 05  0:      NULL
      :      }
233 04 20:      OCTET STRING
      :      C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
      :      F2 A8 A3 DA
      :      }
255 30 33:      SEQUENCE {
257 30  9:      SEQUENCE {
259 06  5:      OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
266 05  0:      NULL
      :      }
268 04 20:      OCTET STRING
      :      90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
      :      5B DC 2F 52
      :      }
290 30 33:      SEQUENCE {
292 30  9:      SEQUENCE {
294 06  5:      OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
301 05  0:      NULL
      :      }

```

```

303 04 20:      OCTET STRING
      :          49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
      :          B5 1F 4F 17
      :          }
325 30 33:      SEQUENCE {
327 30  9:          SEQUENCE {
329 06  5:              OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
336 05  0:              NULL
      :          }
338 04 20:      OCTET STRING
      :          FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
      :          6E 83 D8 8B
      :          }
360 30 33:      SEQUENCE {
362 30  9:          SEQUENCE {
364 06  5:              OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
371 05  0:              NULL
      :          }
373 04 20:      OCTET STRING
      :          2F B6 96 1C 14 BF 41 39 51 D4 16 3F FF FA 63 93
      :          90 3D 2D 8E
      :          }
395 30 33:      SEQUENCE {
397 30  9:          SEQUENCE {
399 06  5:              OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
406 05  0:              NULL
      :          }
408 04 20:      OCTET STRING
      :          CB 52 83 D0 E0 DE 68 79 23 37 BF 16 49 52 AF 9A
      :          72 9B C7 A3
      :          }
430 30 33:      SEQUENCE {
432 30  9:          SEQUENCE {
434 06  5:              OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
441 05  0:              NULL
      :          }
443 04 20:      OCTET STRING
      :          67 76 4B 62 B4 A8 A0 36 8B D0 C3 93 F2 63 BE 5F
      :          4A F7 B1 7C
      :          }
      :          }
      :          }
465 04 69:      OCTET STRING
      :          00 00 00 A0 E0 AE E2 8B 10 AF 53 CD 3D 5F 93 50
      :          B6 0E F0 EF DD 43 8C 00 00 00 67 76 4B 62 B4 A8

```

```

:      A0 36 8B D0 C3 93 F2 63 BE 5F 4A F7 B1 7C 00 00
:      00 EC 27 A5 02 88 42 B9 09 D2 8C 41 2B 18 48 3C
:      9F DA A2 43 D4
536 30 218: SEQUENCE {
539 30 70:   SEQUENCE {
541 30 33:     SEQUENCE {
543 30 9:      SEQUENCE {
545 06 5:        OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
552 05 0:        NULL
:              }
554 04 20:      OCTET STRING
:              BB 07 04 B0 F9 B1 92 9E AA 45 DC E8 DA 27 1B 59
:              61 79 CA 2C
:              }
576 30 33: SEQUENCE {
578 30 9:   SEQUENCE {
580 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
587 05 0:     NULL
:         }
589 04 20: OCTET STRING
:         7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
:         AE 3A 27 8F
:         }
:     }
611 30 13: SEQUENCE {
613 06 9:   OBJECT IDENTIFIER
:         sha1withRSAEncryption (1 2 840 113549 1 1 5)
624 05 0:   NULL
:     }
626 04 128: OCTET STRING
:         53 F9 64 7E 5F 3F 42 BD B7 54 17 CD 7F 20 51 78
:         37 60 FD B7 7D 6B F6 D7 49 E5 5F C9 3F D2 D6 A2
:         12 6A BE 58 F4 A4 5E D9 8C 80 9F D0 E3 28 D2 F9
:         43 AE D1 F6 86 D7 DD 55 AA EB FE E4 83 97 B9 FF
:         C5 5E 15 D3 54 76 0D 74 0C 91 87 49 2F EC 46 94
:         34 47 46 A5 AF D8 4C 92 5C F3 27 D6 73 9D 4A 1E
:         12 A8 FF D0 66 61 21 D3 1E 6D 5D 14 18 B8 25 8C
:         51 1F DE 11 5C 58 A3 2F 34 B8 CD F1 EA 8F 99 60
:     }
: }

```

C.3.4 Comparing Time Stamps

Comparison of time stamps C.3.1 and C.3.2 must give `vresult`="not earlier" and `vtype`="unknown", since the difference of time values of the time

stamps does not exceed the accuracy of the time values.

Comparison of time stamps C.3.1 and C.3.3 must give vresult="earlier" and vtype="indirect".

C.4 Time Stamp Extension

C.4.1 Extending the Time Stamp C.3.1 to Time Stamp C.3.3

Extension request

```
0 30 653: SEQUENCE {
4 30 638: SEQUENCE {
8 02 1: INTEGER 1
11 04 0: OCTET STRING
13 30 360: SEQUENCE {
17 04 9: OCTET STRING
: 08 00 00 00 00 00 11 D9 80
28 30 33: SEQUENCE {
30 30 9: SEQUENCE {
32 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
39 05 0: NULL
: }
41 04 20: OCTET STRING
: 7E 0A 12 42 BD 8E F9 04 4F 27 DC A4 5F 5F 72 AD
: 5A 11 25 BF
: }
63 18 15: GeneralizedTime '20020430151018Z'
80 A0 11: [0] {
82 30 9: SEQUENCE {
84 02 1: INTEGER 5
87 A0 4: [0] {
89 02 2: INTEGER 500
: }
: }
: }
93 30 280: SEQUENCE {
97 30 33: SEQUENCE {
99 30 9: SEQUENCE {
101 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
108 05 0: NULL
: }
110 04 20: OCTET STRING
: 64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
: F7 85 B9 06
: }
```

```

132 30 33: SEQUENCE {
134 30 9: SEQUENCE {
136 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
143 05 0: NULL
:
}
145 04 20: OCTET STRING
: 03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
: A6 91 CC 2C
:
}
167 30 33: SEQUENCE {
169 30 9: SEQUENCE {
171 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
178 05 0: NULL
:
}
180 04 20: OCTET STRING
: 88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
: D4 7D B7 77
:
}
202 30 33: SEQUENCE {
204 30 9: SEQUENCE {
206 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
213 05 0: NULL
:
}
215 04 20: OCTET STRING
: B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
: A6 4A 7F AE
:
}
237 30 33: SEQUENCE {
239 30 9: SEQUENCE {
241 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
248 05 0: NULL
:
}
250 04 20: OCTET STRING
: C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
: F2 A8 A3 DA
:
}
272 30 33: SEQUENCE {
274 30 9: SEQUENCE {
276 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
283 05 0: NULL
:
}
285 04 20: OCTET STRING
: 90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
: 5B DC 2F 52
:
}

```

```

307 30 33: SEQUENCE {
309 30 9: SEQUENCE {
311 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
318 05 0: NULL
: }
320 04 20: OCTET STRING
: 49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
: B5 1F 4F 17
: }
342 30 33: SEQUENCE {
344 30 9: SEQUENCE {
346 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
353 05 0: NULL
: }
355 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B
: }
: }
: }
377 04 46: OCTET STRING
: 00 00 00 93 ED DB EE 22 57 F3 E1 BE CE 3A EE 86
: A3 5A 67 B6 D1 15 5D 00 00 00 93 ED DB EE 22 57
: F3 E1 BE CE 3A EE 86 A3 5A 67 B6 D1 15 5D
425 30 218: SEQUENCE {
428 30 70: SEQUENCE {
430 30 33: SEQUENCE {
432 30 9: SEQUENCE {
434 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
441 05 0: NULL
: }
443 04 20: OCTET STRING
: 38 09 FC 40 A9 A3 4A E7 BC AB 37 22 9C A1 90 BC
: A4 85 F4 3D
: }
465 30 33: SEQUENCE {
467 30 9: SEQUENCE {
469 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
476 05 0: NULL
: }
478 04 20: OCTET STRING
: 7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
: AE 3A 27 8F
: }
: }

```

```

500 30 13: SEQUENCE {
502 06 9:   OBJECT IDENTIFIER
      :   sha1withRSAEncryption (1 2 840 113549 1 1 5)
513 05 0:   NULL
      :   }
515 04 128: OCTET STRING
      :   3C C2 30 DC E3 B2 96 DB 42 FB 94 C8 9F CF 53 5E
      :   3C 2A 31 92 7C CE 7B 25 90 96 FB 27 F6 AD 5A 49
      :   DC 6E 7E AF F6 08 06 3B CA 0C 34 FF 88 7C 0A 0D
      :   CE CC 2F AF B5 21 FA E5 D5 04 E7 17 6B 7A A6 A9
      :   11 2E 77 36 CD AE 58 F0 2F 3E 81 26 F2 4E B0 F9
      :   1B 47 AC 63 C0 24 99 45 09 CF BD 9A 80 42 72 52
      :   25 DD 7E CB D5 BC A2 F8 82 55 CE D1 CB AB 86 B8
      :   C0 6F 3D DA AD 1F DC 6F A0 52 25 E6 1E C0 DC E1
      :   }
      :   }
646 04 9:   OCTET STRING
      :   08 00 00 00 00 00 11 D9 A9
      :   }

```

Extended time stamp. From this time stamp it must be possible to extract the time value using the algorithm presented in Section 8.1 and taking the certificate of Subsection C.1.2 as trust anchor.

```

0 30 800: SEQUENCE {
4 02 1:   INTEGER 1
7 04 0:   OCTET STRING
9 30 360: SEQUENCE {
13 04 9:   OCTET STRING
      :   08 00 00 00 00 00 11 D9 80
24 30 33: SEQUENCE {
26 30 9:   SEQUENCE {
28 06 5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05 0:   NULL
      :   }
37 04 20: OCTET STRING
      :   7E 0A 12 42 BD 8E F9 04 4F 27 DC A4 5F 5F 72 AD
      :   5A 11 25 BF
      :   }
59 18 15: GeneralizedTime '20020430151018Z'
76 A0 11: [0] {
78 30 9:   SEQUENCE {
80 02 1:   INTEGER 5
83 A0 4:   [0] {
85 02 2:   INTEGER 500
      :   }

```

```

      :
      : }
      : }
89 30 280: SEQUENCE {
93 30 33: SEQUENCE {
95 30 9: SEQUENCE {
97 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
104 05 0: NULL
      :
      : }
106 04 20: OCTET STRING
      : 64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
      : F7 85 B9 06
      :
      : }
128 30 33: SEQUENCE {
130 30 9: SEQUENCE {
132 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
139 05 0: NULL
      :
      : }
141 04 20: OCTET STRING
      : 03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
      : A6 91 CC 2C
      :
      : }
163 30 33: SEQUENCE {
165 30 9: SEQUENCE {
167 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
174 05 0: NULL
      :
      : }
176 04 20: OCTET STRING
      : 88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      : D4 7D B7 77
      :
      : }
198 30 33: SEQUENCE {
200 30 9: SEQUENCE {
202 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
209 05 0: NULL
      :
      : }
211 04 20: OCTET STRING
      : B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      : A6 4A 7F AE
      :
      : }
233 30 33: SEQUENCE {
235 30 9: SEQUENCE {
237 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
244 05 0: NULL
      :
      : }
246 04 20: OCTET STRING

```

```

:           C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
:           F2 A8 A3 DA
:           }
268 30 33: SEQUENCE {
270 30 9: SEQUENCE {
272 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
279 05 0: NULL
:           }
281 04 20: OCTET STRING
:           90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
:           5B DC 2F 52
:           }
303 30 33: SEQUENCE {
305 30 9: SEQUENCE {
307 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
314 05 0: NULL
:           }
316 04 20: OCTET STRING
:           49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
:           B5 1F 4F 17
:           }
338 30 33: SEQUENCE {
340 30 9: SEQUENCE {
342 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
349 05 0: NULL
:           }
351 04 20: OCTET STRING
:           FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
:           6E 83 D8 8B
:           }
:           }
:           }
373 04 207: OCTET STRING
:           00 00 00 93 ED DB EE 22 57 F3 E1 BE CE 3A EE 86
:           A3 5A 67 B6 D1 15 5D 00 01 00 21 4A AE 69 F5 E2
:           E1 F8 88 8E 44 7C 5D 20 94 CC 57 D2 81 18 00 01
:           00 1E 25 E2 E1 E3 85 25 9C 65 22 C1 55 19 A1 CA
:           8B 07 EE 6E 83 00 01 00 98 DA 5E F8 0E FA 58 D7
:           42 4F 20 AA FE 25 6D B1 EF F0 5D 16 00 01 00 CF
:           C0 8C FA 92 8C 8D 98 B7 8C 40 A8 EB 3F E7 2D 62
:           94 8D 55 00 01 00 D8 51 1C 1A 2E 1C A9 0D 53 EE
:           BC 1C E7 0F BA 14 6E 74 69 3A 00 00 00 93 ED DB
:           EE 22 57 F3 E1 BE CE 3A EE 86 A3 5A 67 B6 D1 15
:           5D 00 01 00 CB 52 83 D0 E0 DE 68 79 23 37 BF 16
:           49 52 AF 9A 72 9B C7 A3 00 01 00 DB 2E 39 6E 68

```

```

      :      02 BA C0 55 16 E0 5E 55 E3 94 4D 32 C9 93 02
583 30 218: SEQUENCE {
586 30   70: SEQUENCE {
588 30   33: SEQUENCE {
590 30    9: SEQUENCE {
592 06    5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
599 05    0: NULL
      :      }
601 04   20: OCTET STRING
      :      BB 07 04 B0 F9 B1 92 9E AA 45 DC E8 DA 27 1B 59
      :      61 79 CA 2C
      :      }
623 30   33: SEQUENCE {
625 30    9: SEQUENCE {
627 06    5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
634 05    0: NULL
      :      }
636 04   20: OCTET STRING
      :      7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
      :      AE 3A 27 8F
      :      }
      :      }
658 30   13: SEQUENCE {
660 06    9: OBJECT IDENTIFIER
      :      sha1withRSAEncryption (1 2 840 113549 1 1 5)
671 05    0: NULL
      :      }
673 04  128: OCTET STRING
      :      53 F9 64 7E 5F 3F 42 BD B7 54 17 CD 7F 20 51 78
      :      37 60 FD B7 7D 6B F6 D7 49 E5 5F C9 3F D2 D6 A2
      :      12 6A BE 58 F4 A4 5E D9 8C 80 9F D0 E3 28 D2 F9
      :      43 AE D1 F6 86 D7 DD 55 AA EB FE E4 83 97 B9 FF
      :      C5 5E 15 D3 54 76 0D 74 0C 91 87 49 2F EC 46 94
      :      34 47 46 A5 AF D8 4C 92 5C F3 27 D6 73 9D 4A 1E
      :      12 A8 FF D0 66 61 21 D3 1E 6D 5D 14 18 B8 25 8C
      :      51 1F DE 11 5C 58 A3 2F 34 B8 CD F1 EA 8F 99 60
      :      }
      :      }

```

C.4.2 Extending the time stamp C.3.2 to a String

Published string: "GAXDAIJQ-BEDAKKYO-AMBBUBIA-AQKJR2UL-O7B7ON2L-BTQAOTGT-QR6JCTA7-YBCAICII-AAAAAAAA-CHM3Q==="

Extension request

```

0 30 804: SEQUENCE {
4 30 789:   SEQUENCE {
8 02  1:     INTEGER 1
11 04  0:    OCTET STRING
13 30 465:   SEQUENCE {
17 04  9:    OCTET STRING
      :      08 00 00 00 00 00 11 D9 8B
28 30 33:   SEQUENCE {
30 30  9:    SEQUENCE {
32 06  5:    OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
39 05  0:    NULL
      :      }
41 04 20:   OCTET STRING
      :      32 D1 0C 7B 8C F9 65 70 CA 04 CE 37 F2 A1 9D 84
      :      24 0D 3A 89
      :      }
63 18 15:   GeneralizedTime '20020430151022Z'
80 A0 11:   [0] {
82 30  9:    SEQUENCE {
84 02  1:    INTEGER 5
87 A0  4:    [0] {
89 02  2:    INTEGER 500
      :      }
      :      }
93 30 385:  SEQUENCE {
97 30 33:   SEQUENCE {
99 30  9:    SEQUENCE {
101 06  5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
108 05  0:   NULL
      :     }
110 04 20:  OCTET STRING
      :     64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
      :     F7 85 B9 06
      :     }
132 30 33:  SEQUENCE {
134 30  9:   SEQUENCE {
136 06  5:   OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
143 05  0:   NULL
      :     }
145 04 20:  OCTET STRING
      :     03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
      :     A6 91 CC 2C
      :     }
167 30 33:  SEQUENCE {

```

```

169 30 9: SEQUENCE {
171 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
178 05 0:     NULL
      :     }
180 04 20:    OCTET STRING
      :     88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      :     D4 7D B7 77
      :     }
202 30 33:    SEQUENCE {
204 30 9:     SEQUENCE {
206 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
213 05 0:     NULL
      :     }
215 04 20:    OCTET STRING
      :     B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      :     A6 4A 7F AE
      :     }
237 30 33:    SEQUENCE {
239 30 9:     SEQUENCE {
241 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
248 05 0:     NULL
      :     }
250 04 20:    OCTET STRING
      :     C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
      :     F2 A8 A3 DA
      :     }
272 30 33:    SEQUENCE {
274 30 9:     SEQUENCE {
276 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
283 05 0:     NULL
      :     }
285 04 20:    OCTET STRING
      :     90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
      :     5B DC 2F 52
      :     }
307 30 33:    SEQUENCE {
309 30 9:     SEQUENCE {
311 06 5:     OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
318 05 0:     NULL
      :     }
320 04 20:    OCTET STRING
      :     49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
      :     B5 1F 4F 17
      :     }
342 30 33:    SEQUENCE {

```

```

344 30 9: SEQUENCE {
346 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
353 05 0: NULL
:
: }
355 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B
:
: }
377 30 33: SEQUENCE {
379 30 9: SEQUENCE {
381 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
388 05 0: NULL
:
: }
390 04 20: OCTET STRING
: 21 85 2A 39 7E 3A 9B CF EB 18 AD F4 7C 9F 42 EE
: 5B 5A 65 88
:
: }
412 30 33: SEQUENCE {
414 30 9: SEQUENCE {
416 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
423 05 0: NULL
:
: }
425 04 20: OCTET STRING
: 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E 1C
: 11 0C 9A 86
:
: }
447 30 33: SEQUENCE {
449 30 9: SEQUENCE {
451 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
458 05 0: NULL
:
: }
460 04 20: OCTET STRING
: AC 6E 6F C1 B4 C0 0D 0D F4 D5 6A 36 D6 04 D9 22
: 2E 7B 85 2E
:
: }
:
: }
482 04 92: OCTET STRING
: 00 00 00 7D 62 30 D0 D7 4B FB 2A 57 4C F4 5B 11
: CB 14 3A 52 9B FD 6C 00 00 00 AC 6E 6F C1 B4 C0
: 0D 0D F4 D5 6A 36 D6 04 D9 22 2E 7B 85 2E 00 00
: 00 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E
: 1C 11 0C 9A 86 00 00 00 92 D3 12 18 C5 BA 9B B7
: C8 FD 69 BC BE 8D B4 20 9D 86 81 27
576 30 218: SEQUENCE {

```

```

579 30 70: SEQUENCE {
581 30 33: SEQUENCE {
583 30 9: SEQUENCE {
585 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
592 05 0: NULL
:
: }
594 04 20: OCTET STRING
: 64 3D 5C 8C 6F 42 2F BB 24 EC 59 76 23 88 CD E0
: C6 6E C5 37
:
: }
616 30 33: SEQUENCE {
618 30 9: SEQUENCE {
620 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
627 05 0: NULL
:
: }
629 04 20: OCTET STRING
: 7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
: AE 3A 27 8F
:
: }
:
: }
651 30 13: SEQUENCE {
653 06 9: OBJECT IDENTIFIER
: sha1withRSAEncryption (1 2 840 113549 1 1 5)
664 05 0: NULL
:
: }
666 04 128: OCTET STRING
: 06 0B D1 99 6C 4F 94 18 28 42 9D 6A 11 0C 44 7F
: F5 15 5F 5A 4F 7C 0A 37 4B 2F 6D 7D 80 B2 A7 ED
: 65 7B 74 FD 3F 39 5F 92 E0 49 52 03 FF 82 83 20
: F3 EA C4 C0 89 61 9C 9C E7 3C 1B 78 8E E2 DE 72
: 18 C9 53 06 C1 CE E9 63 19 F5 92 A7 1E F4 BF 83
: 3B D7 32 A3 1C 69 9B 9E B2 19 96 AE 90 B2 A7 79
: 36 89 E6 57 D3 0C 8D 0C F0 CC 52 F9 F2 BE 8F 33
: E3 70 78 80 4C 30 26 BF ED 39 5A 05 B5 52 56 91
:
: }
:
: }
797 04 9: OCTET STRING
: 08 00 00 00 00 00 11 D9 B8
:
: }

```

Extended time stamp. From this time stamp it must be possible to extract the time value using the algorithm presented in Section 8.1 and taking the string above as trust anchor.

```

0 30 928: SEQUENCE {
4 02 1: INTEGER 1

```

```

7 04 0: OCTET STRING
9 30 465: SEQUENCE {
13 04 9: OCTET STRING
: 08 00 00 00 00 00 11 D9 8B
24 30 33: SEQUENCE {
26 30 9: SEQUENCE {
28 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05 0: NULL
: }
37 04 20: OCTET STRING
: 32 D1 0C 7B 8C F9 65 70 CA 04 CE 37 F2 A1 9D 84
: 24 0D 3A 89
: }
59 18 15: GeneralizedTime '20020430151022Z'
76 A0 11: [0] {
78 30 9: SEQUENCE {
80 02 1: INTEGER 5
83 A0 4: [0] {
85 02 2: INTEGER 500
: }
: }
: }
89 30 385: SEQUENCE {
93 30 33: SEQUENCE {
95 30 9: SEQUENCE {
97 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
104 05 0: NULL
: }
106 04 20: OCTET STRING
: 64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
: F7 85 B9 06
: }
128 30 33: SEQUENCE {
130 30 9: SEQUENCE {
132 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
139 05 0: NULL
: }
141 04 20: OCTET STRING
: 03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
: A6 91 CC 2C
: }
163 30 33: SEQUENCE {
165 30 9: SEQUENCE {
167 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
174 05 0: NULL

```

```

      :
      }
176 04 20:   OCTET STRING
      :       88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
      :       D4 7D B7 77
      :
      }
198 30 33:   SEQUENCE {
200 30  9:     SEQUENCE {
202 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
209 05  0:       NULL
      :
      }
211 04 20:   OCTET STRING
      :       B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
      :       A6 4A 7F AE
      :
      }
233 30 33:   SEQUENCE {
235 30  9:     SEQUENCE {
237 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
244 05  0:       NULL
      :
      }
246 04 20:   OCTET STRING
      :       C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
      :       F2 A8 A3 DA
      :
      }
268 30 33:   SEQUENCE {
270 30  9:     SEQUENCE {
272 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
279 05  0:       NULL
      :
      }
281 04 20:   OCTET STRING
      :       90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
      :       5B DC 2F 52
      :
      }
303 30 33:   SEQUENCE {
305 30  9:     SEQUENCE {
307 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
314 05  0:       NULL
      :
      }
316 04 20:   OCTET STRING
      :       49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
      :       B5 1F 4F 17
      :
      }
338 30 33:   SEQUENCE {
340 30  9:     SEQUENCE {
342 06  5:       OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
349 05  0:       NULL

```

```

:
:
351 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B
:
:
373 30 33: SEQUENCE {
375 30 9: SEQUENCE {
377 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
384 05 0: NULL
:
:
386 04 20: OCTET STRING
: 21 85 2A 39 7E 3A 9B CF EB 18 AD F4 7C 9F 42 EE
: 5B 5A 65 88
:
:
408 30 33: SEQUENCE {
410 30 9: SEQUENCE {
412 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
419 05 0: NULL
:
:
421 04 20: OCTET STRING
: 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E 1C
: 11 0C 9A 86
:
:
443 30 33: SEQUENCE {
445 30 9: SEQUENCE {
447 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
454 05 0: NULL
:
:
456 04 20: OCTET STRING
: AC 6E 6F C1 B4 C0 0D 0D F4 D5 6A 36 D6 04 D9 22
: 2E 7B 85 2E
:
:
:
:
:
478 04 230: OCTET STRING
: 00 00 00 7D 62 30 D0 D7 4B FB 2A 57 4C F4 5B 11
: CB 14 3A 52 9B FD 6C 00 00 00 AC 6E 6F C1 B4 C0
: 0D 0D F4 D5 6A 36 D6 04 D9 22 2E 7B 85 2E 00 00
: 00 07 CD 59 65 B5 13 43 B4 E9 A8 10 AD 9F 31 3E
: 1C 11 0C 9A 86 00 01 00 7D 3B 07 25 91 63 F6 4C
: 7C 40 B0 CF D1 65 A1 30 C7 CA A1 EF 00 00 00 21
: 85 2A 39 7E 3A 9B CF EB 18 AD F4 7C 9F 42 EE 5B
: 5A 65 88 00 01 00 D8 51 1C 1A 2E 1C A9 0D 53 EE
: BC 1C E7 0F BA 14 6E 74 69 3A 00 00 00 93 ED DB
: EE 22 57 F3 E1 BE CE 3A EE 86 A3 5A 67 B6 D1 15

```

```

      :      5D 00 01 00 0C DF D8 FB D3 7E 35 16 1A 8C 2A 60
      :      35 39 9B E7 F4 39 AE 64 00 01 00 9F DA 92 83 48
      :      B8 69 2F 55 90 27 EB 59 DC ED 5C 58 24 54 A6 00
      :      01 00 98 8B 3D FC C2 A9 12 D8 88 D1 7E 05 67 C9
      :      E3 F1 67 0B D4 52
711 30 218: SEQUENCE {
714 30 70: SEQUENCE {
716 30 33: SEQUENCE {
718 30 9: SEQUENCE {
720 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
727 05 0: NULL
      :      }
729 04 20: OCTET STRING
      :      98 EA 8B 77 C3 F7 37 4B 0C E0 07 4C D3 84 7C 91
      :      4C 1F C0 44
      :      }
751 30 33: SEQUENCE {
753 30 9: SEQUENCE {
755 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
762 05 0: NULL
      :      }
764 04 20: OCTET STRING
      :      7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
      :      AE 3A 27 8F
      :      }
      :      }
786 30 13: SEQUENCE {
788 06 9: OBJECT IDENTIFIER
      :      sha1withRSAEncryption (1 2 840 113549 1 1 5)
799 05 0: NULL
      :      }
801 04 128: OCTET STRING
      :      94 F3 DC 20 3D 0D A6 96 B0 71 67 DF 4E 30 FF 88
      :      41 44 7A BE FA 95 C1 D9 B9 76 70 87 67 DC E0 BB
      :      75 03 73 21 DC 33 4F 4D 8A DF F5 37 80 5C 18 0F
      :      2F 2E 8F 68 09 DB EA A7 F1 63 F0 C4 E4 75 4E 35
      :      0E E7 F7 29 DC D7 9F 53 72 17 68 78 F8 E7 DF 71
      :      6D 59 49 41 A3 F1 98 C4 16 53 39 AE FB 7C AA 9B
      :      90 01 09 30 99 B2 45 63 F7 42 11 28 9F 61 1A 6F
      :      84 DC 62 8A 95 F2 83 0D 20 CB CA BA 08 89 AB 26
      :      }
      :      }

```

C.4.3 Comparing Time Stamps

Comparison of time stamps C.4.1 and C.4.2 (using the certificate C.1.2 and the string C.4.2 for verification) must give `vresult="earlier"` and `vtype="direct"`.

C.4.4 Extending One Time Stamp with the Help of Another

It is possible to extend time stamp C.4.1 to the stamp C.4.2. From the result it must be possible to extract the time value using the algorithm presented in Section 8.1 and taking the string from Subsection C.4.2 as trust anchor.

Extension result

```
0 30 823: SEQUENCE {
4 02 1: INTEGER 1
7 04 0: OCTET STRING
9 30 360: SEQUENCE {
13 04 9: OCTET STRING
: 08 00 00 00 00 00 11 D9 80
24 30 33: SEQUENCE {
26 30 9: SEQUENCE {
28 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
35 05 0: NULL
: }
37 04 20: OCTET STRING
: 7E 0A 12 42 BD 8E F9 04 4F 27 DC A4 5F 5F 72 AD
: 5A 11 25 BF
: }
59 18 15: GeneralizedTime '20020430151018Z'
76 A0 11: [0] {
78 30 9: SEQUENCE {
80 02 1: INTEGER 5
83 A0 4: [0] {
85 02 2: INTEGER 500
: }
: }
: }
89 30 280: SEQUENCE {
93 30 33: SEQUENCE {
95 30 9: SEQUENCE {
97 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
104 05 0: NULL
: }
106 04 20: OCTET STRING
: 64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
: F7 85 B9 06
```

```

:
}
128 30 33: SEQUENCE {
130 30 9: SEQUENCE {
132 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
139 05 0: NULL
:
}
141 04 20: OCTET STRING
: 03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
: A6 91 CC 2C
:
}
163 30 33: SEQUENCE {
165 30 9: SEQUENCE {
167 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
174 05 0: NULL
:
}
176 04 20: OCTET STRING
: 88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
: D4 7D B7 77
:
}
198 30 33: SEQUENCE {
200 30 9: SEQUENCE {
202 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
209 05 0: NULL
:
}
211 04 20: OCTET STRING
: B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
: A6 4A 7F AE
:
}
233 30 33: SEQUENCE {
235 30 9: SEQUENCE {
237 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
244 05 0: NULL
:
}
246 04 20: OCTET STRING
: C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
: F2 A8 A3 DA
:
}
268 30 33: SEQUENCE {
270 30 9: SEQUENCE {
272 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
279 05 0: NULL
:
}
281 04 20: OCTET STRING
: 90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
: 5B DC 2F 52

```

```

:
}
303 30 33: SEQUENCE {
305 30 9: SEQUENCE {
307 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
314 05 0: NULL
:
}
316 04 20: OCTET STRING
: 49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
: B5 1F 4F 17
:
}
338 30 33: SEQUENCE {
340 30 9: SEQUENCE {
342 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
349 05 0: NULL
:
}
351 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B
:
}
:
}
:
}
373 04 230: OCTET STRING
: 00 00 00 93 ED DB EE 22 57 F3 E1 BE CE 3A EE 86
: A3 5A 67 B6 D1 15 5D 00 01 00 21 4A AE 69 F5 E2
: E1 F8 88 8E 44 7C 5D 20 94 CC 57 D2 81 18 00 01
: 00 1E 25 E2 E1 E3 85 25 9C 65 22 C1 55 19 A1 CA
: 8B 07 EE 6E 83 00 01 00 98 DA 5E F8 0E FA 58 D7
: 42 4F 20 AA FE 25 6D B1 EF F0 5D 16 00 01 00 CF
: C0 8C FA 92 8C 8D 98 B7 8C 40 A8 EB 3F E7 2D 62
: 94 8D 55 00 01 00 D8 51 1C 1A 2E 1C A9 0D 53 EE
: BC 1C E7 0F BA 14 6E 74 69 3A 00 00 00 93 ED DB
: EE 22 57 F3 E1 BE CE 3A EE 86 A3 5A 67 B6 D1 15
: 5D 00 01 00 0C DF D8 FB D3 7E 35 16 1A 8C 2A 60
: 35 39 9B E7 F4 39 AE 64 00 01 00 9F DA 92 83 48
: B8 69 2F 55 90 27 EB 59 DC ED 5C 58 24 54 A6 00
: 01 00 98 8B 3D FC C2 A9 12 D8 88 D1 7E 05 67 C9
: E3 F1 67 0B D4 52
606 30 218: SEQUENCE {
609 30 70: SEQUENCE {
611 30 33: SEQUENCE {
613 30 9: SEQUENCE {
615 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
622 05 0: NULL
:
}
624 04 20: OCTET STRING

```

```

:          98 EA 8B 77 C3 F7 37 4B 0C E0 07 4C D3 84 7C 91
:          4C 1F C0 44
:          }
646 30 33: SEQUENCE {
648 30 9: SEQUENCE {
650 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
657 05 0: NULL
:          }
659 04 20: OCTET STRING
:          7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
:          AE 3A 27 8F
:          }
:          }
681 30 13: SEQUENCE {
683 06 9: OBJECT IDENTIFIER
:          sha1withRSAEncryption (1 2 840 113549 1 1 5)
694 05 0: NULL
:          }
696 04 128: OCTET STRING
:          94 F3 DC 20 3D 0D A6 96 B0 71 67 DF 4E 30 FF 88
:          41 44 7A BE FA 95 C1 D9 B9 76 70 87 67 DC E0 BB
:          75 03 73 21 DC 33 4F 4D 8A DF F5 37 80 5C 18 0F
:          2F 2E 8F 68 09 DB EA A7 F1 63 F0 C4 E4 75 4E 35
:          0E E7 F7 29 DC D7 9F 53 72 17 68 78 F8 E7 DF 71
:          6D 59 49 41 A3 F1 98 C4 16 53 39 AE FB 7C AA 9B
:          90 01 09 30 99 B2 45 63 F7 42 11 28 9F 61 1A 6F
:          84 DC 62 8A 95 F2 83 0D 20 CB CA BA 08 89 AB 26
:          }
:          }
:          }

```

C.5 Time Stamps with Aggregation

In this example, the messages 1, 2 and 3 (Subsections C.2.1, C.2.2 and C.2.3) are aggregated using Merkle's tree (see Figure 4).

The time stamp for Message C.2.1

```

0 30 812: SEQUENCE {
4 02 1: INTEGER 1
7 04 46: OCTET STRING
:          00 01 00 32 D1 0C 7B 8C F9 65 70 CA 04 CE 37 F2
:          A1 9D 84 24 0D 3A 89 00 01 00 80 25 6F 39 A9 D3
:          08 65 0A C9 0D 9B E9 A7 2A 95 62 45 45 74
55 30 465: SEQUENCE {
59 04 9: OCTET STRING
:          08 00 00 00 00 00 11 D9 C5

```

```

70 30 33: SEQUENCE {
72 30 9: SEQUENCE {
74 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
81 05 0: NULL
:
: }
83 04 20: OCTET STRING
: CF 40 4A 5A 75 45 AF 82 BA 6C 07 CF 60 96 C5 30
: 0F D5 03 95
:
: }
105 18 15: GeneralizedTime '20020502104029Z'
122 A0 11: [0] {
124 30 9: SEQUENCE {
126 02 1: INTEGER 5
129 A0 4: [0] {
131 02 2: INTEGER 500
:
: }
:
: }
135 30 385: SEQUENCE {
139 30 33: SEQUENCE {
141 30 9: SEQUENCE {
143 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
150 05 0: NULL
:
: }
152 04 20: OCTET STRING
: 64 CA 6E 3D AF 1F BC 18 0A D0 D2 EB AA 3B 07 2D
: F7 85 B9 06
:
: }
174 30 33: SEQUENCE {
176 30 9: SEQUENCE {
178 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
185 05 0: NULL
:
: }
187 04 20: OCTET STRING
: 03 2D 06 16 6B E8 0C 5E D0 1B 59 A1 C8 B4 1D DA
: A6 91 CC 2C
:
: }
209 30 33: SEQUENCE {
211 30 9: SEQUENCE {
213 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
220 05 0: NULL
:
: }
222 04 20: OCTET STRING
: 88 97 B9 1F 97 D5 3B 77 C0 C5 F6 88 24 AB F4 DB
: D4 7D B7 77

```

```

:
}
244 30 33: SEQUENCE {
246 30 9: SEQUENCE {
248 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
255 05 0: NULL
:
}
257 04 20: OCTET STRING
: B1 7F EB 89 71 3B 10 E9 6F 19 9E 31 72 9F 47 AA
: A6 4A 7F AE
:
}
279 30 33: SEQUENCE {
281 30 9: SEQUENCE {
283 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
290 05 0: NULL
:
}
292 04 20: OCTET STRING
: C2 50 15 11 7C D3 49 D4 CB CF 1D 16 6D E9 B8 83
: F2 A8 A3 DA
:
}
314 30 33: SEQUENCE {
316 30 9: SEQUENCE {
318 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
325 05 0: NULL
:
}
327 04 20: OCTET STRING
: 90 33 FD 7C 74 8F 4F C7 28 A4 4C 0C 71 02 E2 42
: 5B DC 2F 52
:
}
349 30 33: SEQUENCE {
351 30 9: SEQUENCE {
353 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
360 05 0: NULL
:
}
362 04 20: OCTET STRING
: 49 97 3D 1B 87 CD 1D FB 6B B2 C3 89 C0 0B DD 56
: B5 1F 4F 17
:
}
384 30 33: SEQUENCE {
386 30 9: SEQUENCE {
388 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
395 05 0: NULL
:
}
397 04 20: OCTET STRING
: FE 7C 17 2D 92 35 78 D6 8D 0F EE 44 42 07 79 2C
: 6E 83 D8 8B

```

```

:
}
419 30 33: SEQUENCE {
421 30 9: SEQUENCE {
423 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
430 05 0: NULL
:
}
432 04 20: OCTET STRING
: 41 A9 2F 81 AC D5 B5 CF 1A 5A 22 2F 29 57 B6 46
: 10 03 09 7A
:
}
454 30 33: SEQUENCE {
456 30 9: SEQUENCE {
458 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
465 05 0: NULL
:
}
467 04 20: OCTET STRING
: 0E 1C A6 AC 70 4A 37 C7 0E 7E 17 53 DB B8 28 A8
: EC 77 C7 4F
:
}
489 30 33: SEQUENCE {
491 30 9: SEQUENCE {
493 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
500 05 0: NULL
:
}
502 04 20: OCTET STRING
: FB 4C 0F 7B 0D 60 DE CB D8 4D 18 6D 0E 7B 1E FC
: 7B 3B AD 8E
:
}
:
}
524 04 69: OCTET STRING
: 00 00 00 82 52 AC 6F 86 13 7F CA C5 DF FA 0D 97
: 14 1A 89 D0 AC 85 FF 00 00 00 FB 4C 0F 7B 0D 60
: DE CB D8 4D 18 6D 0E 7B 1E FC 7B 3B AD 8E 00 00
: 00 EE 99 2C DA 90 58 CA 78 41 D6 71 64 0C 84 BC
: AC 0A B6 79 B5
595 30 218: SEQUENCE {
598 30 70: SEQUENCE {
600 30 33: SEQUENCE {
602 30 9: SEQUENCE {
604 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
611 05 0: NULL
:
}
613 04 20: OCTET STRING
: 85 EE B6 AD 5F 22 A3 97 A2 28 A1 41 27 BD 7B 05

```

```

      :          CE 28 FF 24
      :          }
635 30 33: SEQUENCE {
637 30 9: SEQUENCE {
639 06 5: OBJECT IDENTIFIER sha1 (1 3 14 3 2 26)
646 05 0: NULL
      :          }
648 04 20: OCTET STRING
      :          7E C3 90 80 95 2A 4A 38 75 D8 8E 34 E0 DF B9 F4
      :          AE 3A 27 8F
      :          }
      :          }
670 30 13: SEQUENCE {
672 06 9: OBJECT IDENTIFIER
      :          sha1withRSAEncryption (1 2 840 113549 1 1 5)
683 05 0: NULL
      :          }
685 04 128: OCTET STRING
      :          75 D0 C6 E1 C4 56 1A FA 5C E2 61 29 25 1D 19 5A
      :          0B 4B 9E 34 41 54 6A 3F 45 CD 55 6E BA 58 29 2F
      :          C6 39 2F 73 70 F5 01 34 55 D9 8F 35 26 6C D6 BC
      :          D7 73 14 40 95 6E 35 AD 8A EE 26 B0 0C 91 B0 4D
      :          69 66 B5 EF 8D D0 58 B8 6A 7E 0B DE 93 9A 65 F4
      :          83 D2 1F D6 91 A4 7A 17 84 71 27 6E C4 D2 27 D7
      :          F9 96 C0 F8 32 C7 9F D4 BD 7C 08 FB 30 71 40 F5
      :          85 ED 10 F7 2C 81 81 70 2A E4 5D FA DD 30 D0 E0
      :          }
      :          }
      :          }

```

C.6 The Above Messages in PEM-format

C.6.1 Private Key of the Time-stamping Service Provider C.1.1

```

-----BEGIN RSA PRIVATE KEY-----
MIICXAIBAAKBgQCbGhlbypgVL5aOHBf9GpdWw0/37yxqRTjITcRhPbcGwvMqdwWH
fIXK9ARsyLtxC0RoiCgDz8jkbSSGvrFedGMci/WaxD515nEVd+zEiVXFCWyHKSOM
CnPBI8BdOGubugEsKkj63QOsF7pXKWn0fz91yaFL3wMOQntX6eUpmCdN/QIDAQAB
AoGABwMoEFlZhOOw593nS78cN6T+8ZOzq15T0eKKNWc1F04rFklpS5XaqrFbndt5
dgPvZnZ7EKkNSR6SHzFx7XZPoMHAUQvprTP2JQHv/BVwT9adOsefv6aN6CobBjWX
ZyEzyA9MOFDYpXP0foTdmITLRbwg/XEZahzEsCcKhiSwdxUCQQDNg14SabDRMT4Q
wXj6rNrqrto9GaZDlVyZlGq8DgdAwUbaPxoK4ZrpDzWw4U8mKGMYrO/YbdhqJjp
nsJwp0vbAeAwTRkl1ShWfnvxOmjj6mU75xpMqo9seNUBLXcJyH9hhrQeytX44S/
EJKmK/fSXfzTbU5bHkKgu8Z7+pnDEHchBwJBAlTQ0ZWzliYJsCRVs3hjc2glnojw
hZieMeiCWjyB6dgo8u4SZfdr5AzWYnPiSWsBoQn04xsK1/C5BSfh+re5gV8CQHZZ
zXhbUs6zbfFHWnfsYr8vK6zZSx4BGUlwmoVggp9izJ42zjl3dUVDiVHcTkbkEYN
-----

```

efRRr/R/jw9a4mzHni0CQDQ3ipfYCu+kX6VpO3eogWHdUf/ChHkgyx4VJGNI91X1
0qnXo6BJaxyQEgjh6s5UOmERjgT39obHn8unrhD0WzA=
-----END RSA PRIVATE KEY-----

C.6.2 Certificate of the Time-stamping Service Provider C.1.2

-----BEGIN CERTIFICATE-----
MIIBqDCCARGgAwIBAgIBADANBgkqhkiG9w0BAQUFADAaMRgwFgYDVQQDEw9BamF0
ZW1wbG10ZWVudXMwHhcNMDIwNDMwMDczNDQ1WhcNMDIwNTMwMDczNDQ1WjAaMRgw
FgYDVQQDEw9BamF0ZW1wbG10ZWVudXMwZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJ
AoGBAJsaGvKmBUvlo4cF/0Y91bDT/fvLGpFOMhNxGE8FwbC8yp3BYd8hcr0BGzI
u3ELRgiIKAPPyORtJIa+sV50YxyL9ZrEPnXmcRV37MSJvcUJbIcpI4wKc8EjwF04
a5u6ASwqSPrdA6wXulcpafR/P3XJoUvfAw5Celfp5SmYJ039AgMBAAEwDQYJKoZI
hvcNAQEFBQADgYEAYeApngXhiaS70aiLxhaVtOW+8KJGY4G+2JMRWCb2mqPRYon0
9z/jYogLO5UovXq1A2R9ZiKK9A3YQL/h+rtrNCUuqxfxbIm7RNTGiRHeleQNbVz1
w23AhNvxKt8LOPVmdWl5uB3WGeXCWP6eY0jzHLZwkYzXZe2lmvGCjo68r5g=
-----END CERTIFICATE-----

C.6.3 Time-stamped Message C.2.1

MTIzNDU2Nzg5MDEyMzQ1Njc4OTA=

C.6.4 Time-stamping request Corresponding to Message C.2.1

MCYCAQEwITAJBgUrDgMCGGUABBR+ChJCvY75BE8n3KRfX3KtWhElvw==

C.6.5 Time-stamped Message C.2.2

YWJjZGVmZ2hpamtsbW5vcHFyc3R1dnd4eXo=

C.6.6 Time-stamping request Corresponding to Message C.2.2

MCYCAQEwITAJBgUrDgMCGGUABBQy0Qx7jP1lcMoEzjfyoz2EJA06iQ==

C.6.7 Time-stamped Message C.2.3

QUJDREVGR0hJSktMTU5PUFFSU1RVVldYWVo=

C.6.8 Time-stamping request Corresponding to Message C.2.3

MCYCAQEwITAJBgUrDgMCGGUABBSAJW85qdMIzQrJDZvppyqVYkVfda==

C.6.9 Time Stamp C.3.1

MIICfgIBAQQAMIIBaAQJCAAAAAAAAAEdmAMCEwCQYFKw4DAHoFAAAQUfgoSQR20+QRP
J9ykXl9yrVoRjB8YDzIwMDIwNDMwMTUxMDE4WqALMAkCAQWgBAICafQwggEYMCEw
CQYFKw4DAHoFAAAQUZMpuPa8fvBgK0NLrqsHLfeFuQYwITAJBgUrDgMCGGUABBRQD
LQYWa+gMXtAbWaHI tB3appHMLDAhMAkGBSsOAwIaBQAEFIiXuR+XlTt3wMX2iCSr
9NvUfbd3MCEwCQYFKw4DAHoFAAAQUSX/riXE7E0lvGZ4xcp9HqqZKf64wITAJBgUr
DgMCGGUABBTcUBURfNNJlMvPHRZt6biD8qij2jAhMAkGBSsOAwIaBQAEFJAz/Xx0
j0/HKKRMDHEC4kJb3C9SMCEwCQYFKw4DAHoFAAAQUSZc9G4fNHftrss0JwAvdVrUf
TxcwITAJBgUrDgMCGGUABBT+fBctkjV41o0P7kRCB3ksboPYiwQuAAAAk+3b7iJX
8+G+zjruhqNaZ7bRFV0AAACT7dviIlfz4b700u6GolpnttEVXTCB2jBGMCEwCQYF
Kw4DAHoFAAAQUOAn8QkmjSue8qzcinKGQvKSF9D0wITAJBgUrDgMCGGUABBR+w5CA
lSpKOHXYjjTg37n0rjonjzANBqkqhkiG9w0BAQUFAASBgDzCMNzjpsbbQvuUyJ/P
U148KjGSfM57JZCW+yf2rVpJ3G5+r/YIBjvKDDT/iHwKDC7ML6+1Ifrl1QTnF2t6
pgkRLnc2za5Y8C8+gSbyTrD5G0esY8AkmuUJz72agEJyUiXdfsvVvKL4glX00cur
hrjAbz3arR/cb6BSJeYewNzh

C.6.10 Time Stamp C.3.2

MIIDFQIBAQQAMIIB0QQJCAAAAAAAAAEdmLMCEwCQYFKw4DAHoFAAAQUtEMe4z5ZXDK
BM438qGdhCQNOokYDzIwMDIwNDMwMTUxMDIyWqALMAkCAQWgBAICafQwggGBMCEw
CQYFKw4DAHoFAAAQUZMpuPa8fvBgK0NLrqsHLfeFuQYwITAJBgUrDgMCGGUABBRQD
LQYWa+gMXtAbWaHI tB3appHMLDAhMAkGBSsOAwIaBQAEFIiXuR+XlTt3wMX2iCSr
9NvUfbd3MCEwCQYFKw4DAHoFAAAQUSX/riXE7E0lvGZ4xcp9HqqZKf64wITAJBgUr
DgMCGGUABBTcUBURfNNJlMvPHRZt6biD8qij2jAhMAkGBSsOAwIaBQAEFJAz/Xx0
j0/HKKRMDHEC4kJb3C9SMCEwCQYFKw4DAHoFAAAQUSZc9G4fNHftrss0JwAvdVrUf
TxcwITAJBgUrDgMCGGUABBT+fBctkjV41o0P7kRCB3ksboPYizAhMAkGBSsOAwIa
BQAEFCGFKjl+OpvP6xit9HyfQu5bWmWIMCEwCQYFKw4DAHoFAAAQUB8lZZbUTQ7Tp
qBCtnzE+HBEMmoYwITAJBgUrDgMCGGUABBSsbm/BtMANDfTVajbWBNkiLnuFLgRc
AAAAfwiw0NdL+ypXTPRbEcsUOlKb/WwAAACsbm/BtMANDfTVajbWBNkiLnuFLgAA
AAfNWWW1E0006agQrZ8xPhwRDJqGAAAAktMSGMW6m7fI/Wm8vo20IJ2GgScwgdow
RjAhMAkGBSsOAwIaBQAEFGQ9XIxvQi+7J0xZdiOIzedGbsU3MCEwCQYFKw4DAHoF
AAQUfsOQgJUqsjh12I404N+59K46J48wDQYJKoZIhvcNAQEFBQAEgYAGC9GZbE+U
GChCnWoRDER/9RVfwk98CjdLL219gLNk7WV7dP0/OV+S4E1SA/+CgyDz6sTAiWGC
nOc8G3i04t5yGm1TBsHO6WMZ9ZKnHvS/gzvXMqMcaZueshmWrpCyp3k2ieZX0wyN
DPDMUvnyvo8z43B4gEwwJr/toVoFtVJWkQ==

C.6.11 Time Stamp C.3.3

MIIC8QIBAQQAMIIBxAQJCAAAAAAAAAEdmpMCEwCQYFKw4DAHoFAAAQUgCVvOanTCGUK
yQ2b6acq1WJFRXQYDzIwMDIwNDMwMTUxMjEyWjCCAYEwITAJBgUrDgMCGGUABBRk
ym49rx+8GARQ0uuqOwct94W5BjAhMAkGBSsOAwIaBQAEFAMtBhZr6Axe0BtZoci0
HdqmkcwsMCEwCQYFKw4DAHoFAAAQUiJe5H5fVO3fAxfaiJKv029R9t3cwITAJBgUr
DgMCGGUABBSxf+uJcTsQ6W8ZnjFyn0eqpkp/rjAhMAkGBSsOAwIaBQAEFMJQFRF8
00nUy88dFm3puIPyqKPAMCEwCQYFKw4DAHoFAAAQUkDP9fHSPT8copEwMcQLiQlvc
L1IwITAJBgUrDgMCGGUABBRJlz0bh80d+2uyw4nAC91WtR9PFzAhMAkGBSsOAwIa

BQAEFP58Fy2SNXjWjQ/uREIHeSxug9iLMCEwCQYFKw4DAHoFAAQUL7aWHBS/QT1R
1BY///pjk5A9LY4wITAJBgUrDgMCGGUABBTLUoPQ4N5oeSM3vxZJUq+acpvHozAh
MAkGBSsOAwIaBQAEFGd2S2K0qKA2i9DDk/Jjvl9K97F8BEUAAACg4K7iixCvU809
X5NQtg7w791DjAAAAGd2S2K0qKA2i9DDk/Jjvl9K97F8AAAA7CelAohCuQnSjEER
GEg8n9qiQ9QwgdowRjAhMAkGBSsOAwIaBQAEFLsHBLD5sZKeqkXc6NonG1lhecoc
MCEwCQYFKw4DAHoFAAQUfsOQgJUqSjh12I404N+59K46J48wDQYJKoZihvcNAQEF
BQAEgYBT+WR+Xz9CvbdUF81/IFF4N2D9t31r9tdJ5V/JP9LWohJqvlj0pF7ZjICf
00Mo0vldrth2htfdVarr/uSDl7n/xV4V01R2DXQMkYdJL+xGlDRHRqWv2EySXPMn
1nOdSh4SqP/QZmEh0x5tXRQYuCWMUR/eEVxYoy80uM3x6o+ZYA==

C.6.12 Time Stamping Request C.4.1

MIICjTCCAn4CAQEEADCCAWgECQgAAAAAABHZgDAhMAkGBSsOAwIaBQAEFH4KEkK9
jvkETyfcP9fcq1aESW/GA8yMDAyMDQzMDelMTAxOFqgCzAJAgEFoAQCAgH0MIIB
GDAhMAkGBSsOAwIaBQAEFGTKbj2vH7wYctDS66o7By33hbkGMCEwCQYFKw4DAHoF
AAQUAy0GFmvoDF7QG1mhyLQd2qaRzCwwITAJBgUrDgMCGGUABBSI17kfl19U7d8DF
9ogkq/Tb1H23dzAhMAkGBSsOAwIaBQAEFLF/641xOxDpbxmeMXKfR6qmSn+uMCEw
CQYFKw4DAHoFAAQUwLAVEXzTSdTLzX0Wbem4g/Koo9owITAJBgUrDgMCGGUABBSQ
M/18di9PxyikTAXxAuJCW9wvUjAhMAkGBSsOAwIaBQAEFEEmXPRuHzR37a7LDicAL
3ValH08XMCEwCQYFKw4DAHoFAAAQU/nwXLZi1eNaND+5EQgd5LG6D2IsELgAAAJPt
2+4iV/Phvs467oaJWme20RVdAAAAk+3b7iJX8+G+zjruhqNaZ7bRFV0wgdowRjAh
MAkGBSsOAwIaBQAEFDgJ/ECpo0rnvKs3IpyhkLykhfQ9MCEwCQYFKw4DAHoFAAAQU
fsOQgJUqSjh12I404N+59K46J48wDQYJKoZihvcNAQEFBQAEgYA8wJdC47KW20L7
lMifz1NePCoxknzOeyWQlvsn9q1aSdxufq/2CAY7ygw0/4h8Cg3OzC+vtSH65dUE
5xdregapES53Ns2uWPAvPoEm8k6w+RtHrGPAJlJFCc+9moBCcll1l3X7L1byi+IJV
ztHLq4a4wG892q0f3G+gUiXmHsDc4QQJCAAAAAAAEdmp

C.6.13 Time Stamp C.4.1

MIIDIAIBAQQAMIIBaAQJCAAAAAAAEdmAMCEwCQYFKw4DAHoFAAAQUfgoSQR20+QRP
J9ykX19yrVoRjB8YDzIwMDIwNDMwMTUxMDE4WqALMAkCAQWgBAICafQwggEYMCEw
CQYFKw4DAHoFAAAQUZMpuPa8fvBgK0NLRqjsHLfeFuQYwITAJBgUrDgMCGGUABBD
LQYwa+gMXtAbWaHIItB3appHMLDAhMAkGBSsOAwIaBQAEFTiXuR+X1Tt3wMX2iCSr
9NvUfbd3MCEwCQYFKw4DAHoFAAAQUsX/riXE7E0lvGZ4xcp9HqqZKf64wITAJBgUr
DgMCGGUABBTcUBURfNNJ1MvPHRZt6biD8qiJ2jAhMAkGBSsOAwIaBQAEFJAz/Xx0
j0/HKKRMDHEC4kJb3C9SMCEwCQYFKw4DAHoFAAAQUSZc9G4fNHftrssOJwAvdVrUf
TxcwITAJBgUrDgMCGGUABBT+fBctkjV41o0P7kRCB3ksboPYiwsBzwAAAJPt2+4i
V/Phvs467oaJWme20RVdAAEAIUquafXi4fiIjkr8XSCUzFfSgRgAAQAEJeLh44U
nGUiwVUZocqLB+5ugwABAJjaXvgO+lJXQk8gqv4lbbHv8F0WAAEAz8CM+pKMjzi3
jEC06z/nLWKUjVUAAQDYURwaLhypDVPuvBznD7oUbnRpOgAAAJPt2+4iV/Phvs46
7oaJWme20RVdAAEAy1KD00DeaHk jN78WSVKvmnKbx6MAAQDbLjluaAK6wFUW4F5V
45RNMsmTAjCB2jBGMCEwCQYFKw4DAHoFAAAQUuwcEsPmxkp6qRdzo2icbWWF5yiw
ITAJBgUrDgMCGGUABBR+w5Ca1SpKOHXYjTg37n0rjonjzANBgkqhkiG9w0BAQUF
AASBgFP5ZH5fP0K9t1QXzX8gUXg3YP23fWv210nlX8k/0taiEmq+WPSkXtmMgJ/Q
4yjs+UOu0faG191Vquv+5IOXuf/FXhXTVHYNdAyRh0kv7EaUNEdGpa/YTJJC8yfw

c51KHhKo/9BmYSHTHm1dFbi4JYxRH94RXFi jLzS4zfHqj51g

C.6.14 Time Stamping Request C.4.2

MIIDJCCAxUCAQEEADCCAdEECQgAAAAAABHZizAhMAkGBSsOAwIaBQAeFDLRDHuM
+WVwygTON/KhnYQkDTqJGA8yMDAyMDQzMDElMTAyMlqgCzAJAgEFoAQCAgH0MIIB
gTAhMAkGBSsOAwIaBQAeFGTKbj2vH7wYCTDS66o7By33hbkGMCEwCQYFKw4DAhOF
AAQUAy0GFmvoDF7QG1mhyLQd2qaRzCwwITAJBgUrDgMCGGUABBSI17kfl9U7d8DF
9ogkq/Tb1H23dzAhMAkGBSsOAwIaBQAeFLF/641xOxDpbxmeMXKfR6qmSn+uMCEw
CQYFKw4DAhOFAAQUwLAVEXzTSdTLzx0Wbem4g/Koo9owITAJBgUrDgMCGGUABBSQ
M/18dI9PxyikTAXxAuJCW9wvUjAhMAkGBSsOAwIaBQAeFEmXPRuHzR37a7LDicAL
3ValH08XMCEwCQYFKw4DAhOFAAQU/nwXLZi1eNaND+5EQgd5LG6D2IswITAJBgUr
DgMCGGUABBSQhhSo5fjqbz+sYrfr8n0LuW1pliDAhMAkGBSsOAwIaBQAeFAfNWWW1
E0006agQrZ8xPhwRDJqGMCEwCQYFKw4DAhOFAAQUrG5vwbTADQ301Wo21gTZIi57
hS4EXAAAH1iMNDXS/sqV0z0WxHLFDpSm/1sAAAARg5vwbTADQ301Wo21gTZIi57
hS4AAAAHzV1ltRNDtOmoEK2fMT4cEQyahgAAAJLTEh jFupu3yPlpvL6NtCCdhoEn
MIHaMEYwITAJBgUrDgMCGGUABBRkPVyMb0IvuyTsWXYjiM3gx7FNzAhMAkGBSsO
AwIaBQAeFH7DkICVKko4ddiONODfufSuOiePMA0GCSqGSIB3DQEBBQUABIGABgvR
mWxPlBgoQplqEQxEf/UVX1pPfAo3Sy9tfYCyp+1le3T9PzlfkuBJUGP/goMg8+rE
wI1hnJznPBt4juLechjJUwbBzuljgFWSpx70v4M71zKjHGmbnrIZlq6Qsqd5Nonm
V9MMjQzWzFL58r6PM+NweIBMMCa/7TlaBbVSVpEECQgAAAAAABHZuA==

C.6.15 Time Stamp C.4.2

MIIDoAIBAQQAMIIB0QQJCAAAAAAEEdmLMCEwCQYFKw4DAhOFAAQUmtEMe4z5ZXDK
BM438qGdhCQNOokYDzIwMDIwNDMwMTUxMDIyWqALMAkCAQWgBAICafQwggGBMCEw
CQYFKw4DAhOFAAQUZMpuPa8fvBgK0NLRqjsHLfeFuQYwITAJBgUrDgMCGGUABBSQD
LQYwa+gMXtAbWaHIitB3appHMLDAhMAkGBSsOAwIaBQAeFIIiXuR+X1Tt3wMX2iCSr
9NvUfbd3MCEwCQYFKw4DAhOFAAQUsX/riXE7E0lvGZ4xcp9HqqZKf64wITAJBgUr
DgMCGGUABBTcUBURfNNJ1MvPHRzt6biD8qij2jAhMAkGBSsOAwIaBQAeFJAz/Xx0
j0/HKKRMDHEC4kJb3C9SMCEwCQYFKw4DAhOFAAQUUSZc9G4fNHftrssOJwAvdVrUf
TxcwITAJBgUrDgMCGGUABBT+fBctkjV41o0P7kRCB3ksboPYizAhMAkGBSsOAwIa
BQAeFCGFKj1+OpvP6xit9HyfQu5bWmWIMCEwCQYFKw4DAhOFAAQUB81ZZbUTQ7Tp
qBCtnzE+HBEMmoYwITAJBgUrDgMCGGUABBSsbm/BtMANDfTVajbWBNkiLnuFLgSB
5gAAAH1iMNDXS/sqV0z0WxHLFDpSm/1sAAAARg5vwbTADQ301Wo21gTZIi57hS4A
AAAHzV1ltRNDtOmoEK2fMT4cEQyahgABAH07ByWRY/ZMfECwz9FloTDHyqHvAAAA
IYUqOX46m8/rGK30fJ9C71taZYgAAQDYURwaLhypDVPuvBznD7oUbnRpOgAAAJPt
2+4iv/Phvs467oajWme20RVdAAEADN/Y+9N+NRYa jCpgNTmb5/Q5rmQAAQCf2pKD
SLhpL1WQJ+tZ301cWCRUpgABAjiLPfzCqRLYiNF+BwfJ4/FnC9RSMIHaMEYwITAJ
BgUrDgMCGGUABBSY6ot3w/c3Swzgb0zThHyRTB/ARDAhMAkGBSsOAwIaBQAeFH7D
kICVKko4ddiONODfufSuOiePMA0GCSqGSIB3DQEBBQUABIGALPPcID0NppawcWff
TjD/ieFEer76lchZuXZwh2fc4Lt1A3Mh3DNPTYrf9TeAXBgPLY6PaAnb6qfxY/DE
5HVONQ7n9ync159Tchdoepjn33FtWUlBo/GYxBZTOa77fKqbkAEJMjmyRWP3QhEo
n2Eab4TcYoqv8oMNIIMvKugijqyY=

C.6.16 Time Stamp C.4.4

MIIDNwIBAQQAMIIBaAQJCAAAAAAAAAEdmAMCEwCQYFKw4DAhOFAAAQUfgoSQR20+QRP
J9ykXl9yrVoRjB8YDzIwMDIwNDMwMTUxMDE4WqALMAkCAQWgBAICafQwggEYMCEw
CQYFKw4DAhOFAAAQUZMpuPa8fvBgK0NLrQjsHLfeFuQYwITAJBgUrDgMCGGUABBQD
LQYWa+gMXtAbWaHI tB3appHMLDAhMAkGBSsOAwIaBQAEFIiXuR+XlTt3wMX2iCSr
9NvUfbd3MCEwCQYFKw4DAhOFAAAQUSX/riXE7E0lvGZ4xcp9HqqZKf64wITAJBgUr
DgMCGGUABBTCUBURfNnJlMvPHRZt6biD8qij2jAhMAkGBSsOAwIaBQAEFJAz/Xx0
j0/HKKRMDHEC4kJb3C9SMCEwCQYFKw4DAhOFAAAQUSZc9G4fNHftrss0JwAvdVrUf
TxcwITAJBgUrDgMCGGUABBT+fBctkjV4l0P7kRCB3ksboPYiwsB5gAAAjPt2+4i
V/Phvs467oaJwme20RVdAAEAIUquafXi4fiIjkr8XSCUzFfSgRgAAQAEJeLh44U1
nGUiwVUZocqLB+5ugwABAjJaXvgO+lJXQk8gqv4lbbHv8F0WAAEAz8CM+pKmjZi3
jEC06z/nLWKUjVUAAQDYURwaLhypDVPuvBznD7oUbnRpOgAAAjPt2+4iV/Phvs46
7oaJwme20RVdAAEADN/Y+9N+NRYaJcPgNTmb5/Q5rmQAAQCf2pKDSLhpl1WQJ+tZ
30lcWCRUpgABAjilPpzCqRLYiNF+BwfJ4/FnC9RSMIHAMEYwITAJBgUrDgMCGGUA
BBSY6ot3w/c3Swzgb0zThHyRTB/ARDAhMAkGBSsOAwIaBQAEFH7DkICVKko4ddiO
NODfufSuOiePMA0GCSqGSIB3DQEBBQUABIGALPPCID0NppawcWffTjD/iEFEer76
lchZuXZwh2fc4Lt1A3Mh3DNPTYrf9TeAXBgPLy6PaAnb6qfxY/DE5HVONQ7n9ync
159Tchdoepjn33FtWUlBo/GYxBZTOa77fKqbkAEJMjmyRWP3QhEon2Eab4TcYoqV
8oMNI MvKugiJqyY=

C.6.17 Time Stamp C.5

MIIDLAIBAQQUAAEAMtEMe4z5ZXDKBM438qGdhCQNOokAAQCAJW85qdMIZQRJDZvp
pyqVYkVfDCCAdEECQgAAAAABHZxTAhMAkGBSsOAwIaBQAEFM9ASlp1Ra+CumwH
z2CWxTAP1QOVGA8yMDAyMDUwMjEwNDAYOVqgCzAJAgEFoAQCAgH0MIIBGTahMAkG
BSsOAwIaBQAEFGTKbj2vH7wYctDS66o7By33hbkGMCEwCQYFKw4DAhOFAAAQUAy0G
FmvoDF7QG1mhyLQd2qaRzCwwITAJBgUrDgMCGGUABBSI17kfl9U7d8DF9ogkq/Tb
1H23dzAhMAkGBSsOAwIaBQAEFLF/64lxOxDpbxmeMXKfR6qmSn+uMCEwCQYFKw4D
AhOFAAAQUw1AVEXzTSdTLzx0Wbem4g/Koo9owITAJBgUrDgMCGGUABBSQM/18dI9P
xyikTAXxAuJCW9wvUjAhMAkGBSsOAwIaBQAEFEmXPRuHzR37a7LDicAL3Va1H08X
MCEwCQYFKw4DAhOFAAAQU/nwXLZi1eNaND+5EQgd5LG6D2IswITAJBgUrDgMCGGUA
BBRbqS+BrNW1zxpaii8pV7ZGEAMJejAhMAkGBSsOAwIaBQAEFA4cpqxsJfHDn4X
U9u4KKjsd8dPMCEwCQYFKw4DAhOFAAAQU+0wPewlg3svYTRhtDnse/Hs7rY4ERQAA
AIJSrG+GE3/Kxd/6DZcUGonQrIX/AAAA+0wPewlg3svYTRhtDnse/Hs7rY4AAADu
mSzakFjKeEHwCQMhLysCrZ5tTCB2jBGMCEwCQYFKw4DAhOFAAAQUhe62rV8io5ei
KKFBJ717Bc4o/yQwITAJBgUrDgMCGGUABBR+w5CALSpKOHXYjjTg37n0rjonjzAN
BgkqhkiG9w0BAQUFAASBgHXQuHEVhr6XOJhKSUdGVols540QVRqP0XNVW66Wckv
xjkvc3D1ATRv2Y81JmzWvNdZFEcvbjWtiu4msAyRsE1pZrXvjdBuGp+C96TmmX0
g9If1pGkeheEcSduxNIin1/mWwPgyx5/UvXwI+zBxQPWF7RD3LIGBcCrkXfrdMNDg

References

- [ABSW01] Arne Ansper, Ahto Buldas, Märt Saarepera, and Jan Willemsen. Improving the availability of time-stamping services. In Vijay Varadharajan and Yi Mu, editors, *6th australasian conference, acisp 2001*, volume 2119 of *LNCS*, pages 360–375, Sydney, Australia, July 2001. Springer-Verlag.
- [BL98] Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *Proc. 1st international conference on information security and cryptology – icisc’98*, pages 3–13, Seoul, Korea, December 1998.
- [BLLV98] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Willemsen. Time-stamping with binary linking schemes. In *Advances in cryptology – crypto’98*, volume 1462 of *LNCS*, pages 486–501, Santa Barbara, 1998. Springer-Verlag.
- [BLS00] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *Public key cryptography – pkc’2000*, volume 1751 of *LNCS*, pages 293–305, Melbourne, Australia, January 2000. Springer-Verlag.
- [SHA1] D. Eastlake and P. Jones. US Secure Hash Algorithm 1 (SHA1). RFC3174, September 2001.
- [HS91] Stuart Haber and W.Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.
- [X.680] International Telecommunication Union. Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation. Recommendation X.680, July 2002.
- [X.690] International Telecommunication Union. Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER). Recommendation X.690, July 2002.
- [Jos02] S. Josefsson. Base Encoding of Data. IETF draft draft-josefsson-base-encoding-04, February 2002.

- [RFC2313] B. Kaliski. PKCS #1: RSA Encryption. RFC2313, March 1998.
- [Lip99] Helger Lipmaa. *Secure and efficient time-stamping schemes*. PhD thesis, Tartu University, 1999.
- [Mer80] Ralph C. Merkle. Protocols for public key cryptosystems. In *Proceedings of the 1980 IEEE symposium on security and privacy*, pages 122–134, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, April 1980. IEEE Computer Society Press.
- [Mer89] Ralph C. Merkle. A certified digital signature. In *Advances in cryptography – Crypto’89*, volume 435 of *LNCS*, pages 218–238, Santa Barbara, 1989. Springer-Verlag.
- [Wil02] Jan Willemsen. *Size-efficient interval time stamps*. PhD thesis, Tartu University, 2002.
- [X.509] Information technology – Open systems interconnection – The Directory: Public-key and attribute certificate frameworks. ITU-T recommendation X.509, March 2000.