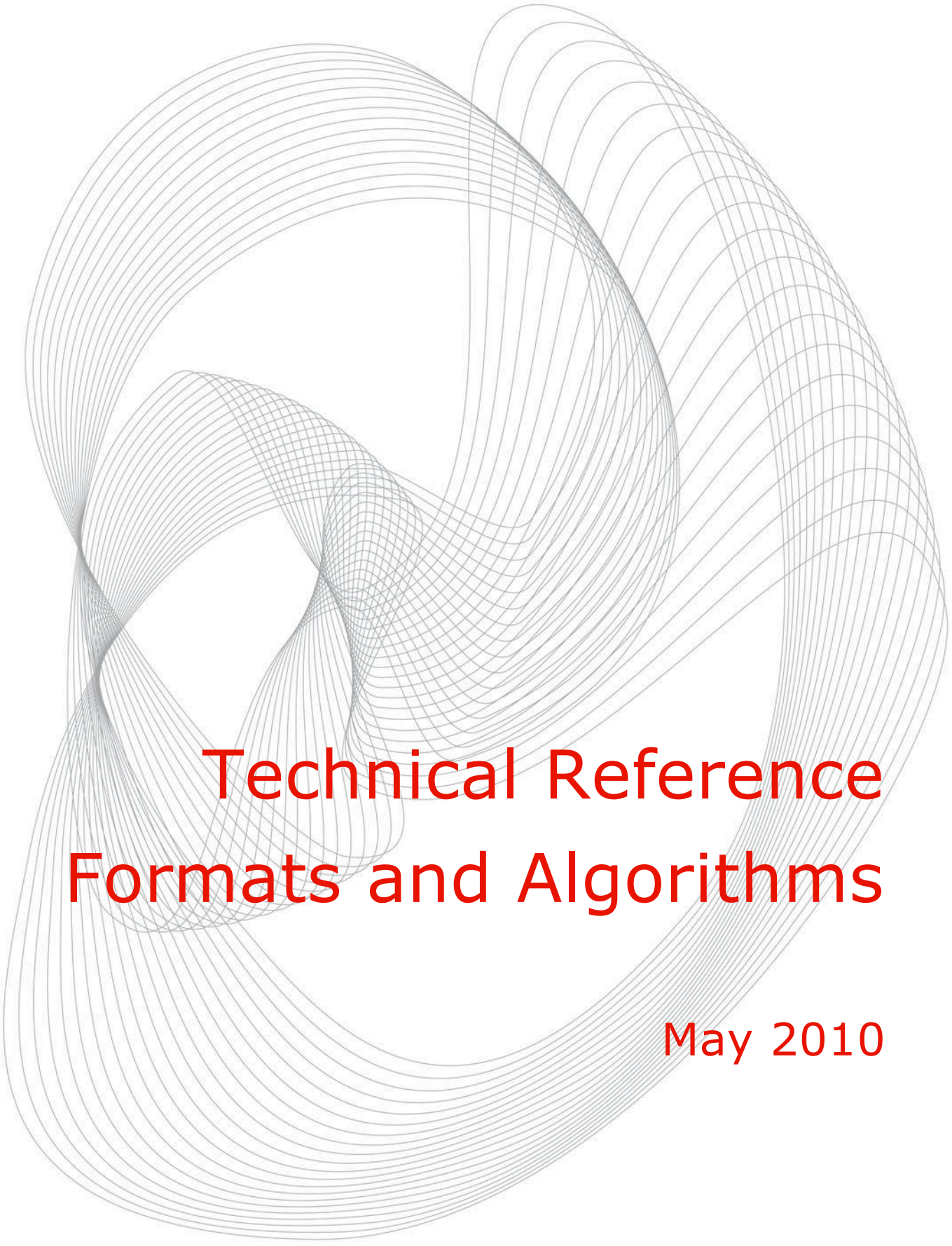
guardtime



Technical Reference
Formats and Algorithms

May 2010

guardtime

Contents

1. Overview

This document is a data format and algorithm specification appendix to the main technical reference on GuardTime timestamping system.

The aim is to define all the relevant data formats and verification procedures in detail sufficient for independent creation of verification tools, but the material is not intended to be usable without the main document.

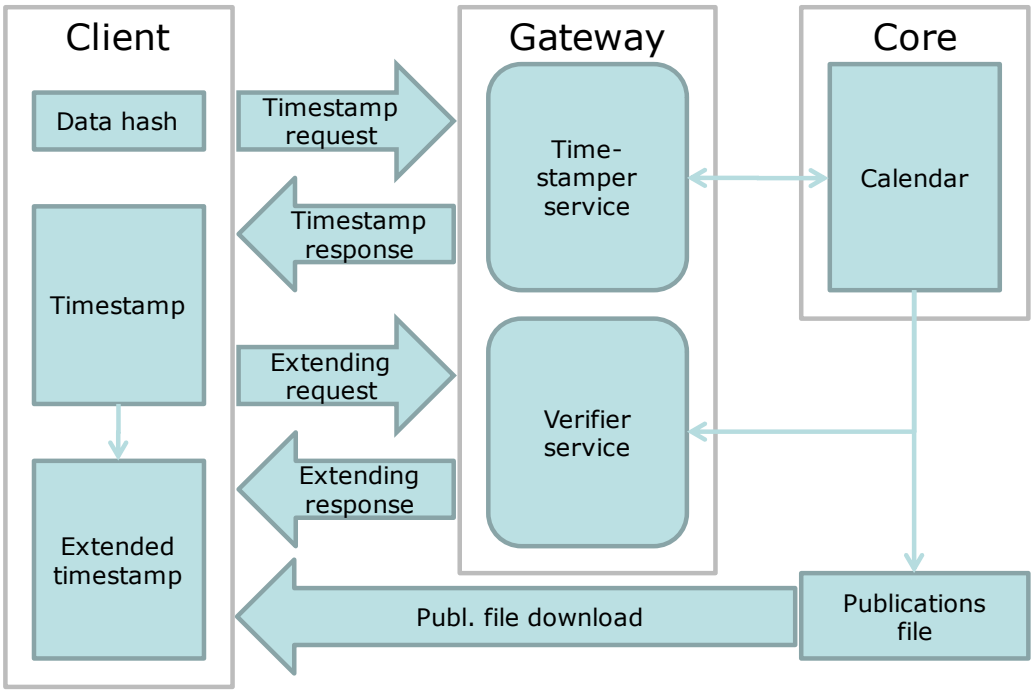
2. Introduction

GuardTime timestamping protocol and timestamp format are based on the IETF standard "Internet X.509 Public Key Infrastructure Time-Stamp Protocol", published as the RFC 3161. The RFC 3161 builds on the IETF standard "Cryptographic Message Syntax", published as the RFC 2630. While the following sections also summarize the contents of the RFC 3161 and RFC 2630, these summaries are provided only for readability of the current document and are not to be considered a substitute for the actual standards.

The concept of extending a timestamp is specific to the GuardTime TimeSignature algorithm and thus the relevant protocol is also defined by GuardTime.

Figure 1 illustrates the relationships between the data elements and messages a client of GuardTime timestamping service will encounter.

Figure 1: Data elements and messages.



Most of the data formats are presented in the ASN.1 notation, defined in "Information technology – Abstract Syntax Notation One (ASN.1): Specification of basic notation" published as the ITU-T X.680. The actual encoding used for transmission of messages and storage of timestamps is the DER formatting, defined in "Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", published as the ITU-T X.690.

Most of the examples of actual messages and timestamps in the following sections were originally produced using Peter Gutmann's *dumpasn1* utility and manually annotated.

3. Timestamping Request

The client initiates the timestamping process by sending a timestamping request.

RFC 3161 defines a timestamping request to have the following structure:

```
TimeStampReq ::= SEQUENCE {
  version INTEGER,
  messageImprint MessageImprint,
  reqPolicy TSAPolicyId OPTIONAL,
  nonce INTEGER OPTIONAL,
  certReq BOOLEAN DEFAULT FALSE,
  extensions [0] IMPLICIT Extensions OPTIONAL }

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

TSAPolicyId ::= OBJECT IDENTIFIER
```

Field	Meaning
Version	Version number of the syntax. The current version is 1.
messageImprint	A digest of the datum to be timestamped, together with the identifier of the hash algorithm used to produce the digest value. It is up to the TSA to decide if the digest algorithm used to produce the digest value contained in the imprint is sufficiently collision-resistant and return an error response if it is not. GT specific: The list of currently supported algorithms is given in Section 11.
reqPolicy	If present, indicates the TSA policy under which the timestamp should be provided.
nonce	A presumably freshly generated random value that enables the client to check that the timestamp returned by the TSA was in fact generated in response to the particular request.

certReq	If this field is present and set to TRUE, the returned timestamp must contain the certificate for the key used to sign the timestamp.
Extensions	A generic way to add additional information. GT specific: extensions are not used nor supported in the current implementation.

Example 1: A timestamping request.

```
  0   54: SEQUENCE {                                                    // TimeStampReq, RFC3161
  2    1:   INTEGER 1                                                   // version
  5   49:   SEQUENCE {                                                 // messageImprint : MessageImprint
  7   13:     SEQUENCE {                                              // hashAlgorithm : AlgorithmIdentifier, RFC2459
  9    9:       OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1)     // algorithm
 20    0:       NULL                                                  // parameters
       :       }
 22   32:     OCTET STRING                                            // hashedMessage
       :       54 66 E3 CB A1 4A 84 3A 5E 93 B7 8E 3D 6A B8 D3 49 1E DC AC 7E 06 43 1C E1 A7 F4 98 28 C3 40 C3
       :     }
                                    // optional reqPolicy : TSAPolicyId not used
                                    // optional nonce : INTEGER not used
                                    // optional certReq : BOOLEAN not used
                                    // optional extensions : Extensions not used
       :   }
```

4. Timestamping Response

Upon receiving a request, the server returns a response that indicates a failure or a success and in case of success includes the timestamp.

RFC 3161 defines a timestamping response to have the following structure:

```
TimeStampResp ::= SEQUENCE {
  status PKIStatusInfo,
  timeStampToken TimeStampToken OPTIONAL }
```

Field	Meaning
status	Used to indicate the failure or success in responding to the timestamping request and in case of failure to indicate the cause.
timeStampToken	In case of success, contains the timestamp. The inner structure of this field is defined in Section 7.

RFC 3161, with reference to RFC 2510, defines the status info to have the following structure:

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

PKIStatus ::= INTEGER
```

Field	Meaning
status	Indicates the success or failure of the request. A value of 0 or 1 means success and in this case the response must contain a timestamp. For any other value, the response must not contain a timestamp. Please see RFC 3161 and RFC 2510 for detailed explanations of the values.

<table>
<tr><td>statusString</td><td>A free text message explaining the cause of the failure.</td></tr>
<tr><td>failInfo</td><td>A bit-field indicating the cause(s) of the failure:
<table>
<tr><th>Bit position</th><th>Meaning</th></tr>
<tr><td>0</td><td>Unrecognized or unsupported algorithm identifier in the message digest.</td></tr>
<tr><td>2</td><td>Transaction not permitted or supported.</td></tr>
<tr><td>5</td><td>The data submitted has the wrong format.</td></tr>
<tr><td>14</td><td>The TSA’s time source is not available.</td></tr>
<tr><td>15</td><td>The requested policy is not supported by the TSA.</td></tr>
<tr><td>16</td><td>The requested extension is not supported by the TSA.</td></tr>
<tr><td>17</td><td>The additional information requested could not be understood or is not available.</td></tr>
<tr><td>25</td><td>The request cannot be handled due to system failure.</td></tr>
</table></td></tr>
</table>

Example 2: A timestamping response indicating failure.

```
  0   67: SEQUENCE {                                                  // TimeStampResp, RFC3161
  2   65:   SEQUENCE {                                              // status : PKIStatusInfo
  4    1:     INTEGER 2                                             // status : PKIStatus (2 = rejection)
  7   56:     SEQUENCE {                                            // statusString : PKIFreeText
  9   54:       UTF8String
       :         '............Unrecognized or unsupported hash algorithm'
       :       }
 65    2:     BIT STRING 7 unused bits                              // failInfo : PKIFailureInfo
       :       80                                                  // bit 0 = badAlg
       :     }
                                  // optional timeStampToken : TimeStampToken not used in case of failure response
       :   }
```

Example 3: A timestamping response indicating success.

```
  0 3794: SEQUENCE {                                                  // TimeStampResp, RFC3161
  4    3:   SEQUENCE {                                                // status : PKIStatusInfo
  6    1:     INTEGER 0                                               // status : PKIStatus (0 = granted)
                                  // optional statusString : PKIFreeText not used
                                  // optional failInfo : PKIFailureInfo not used
       :     }
  9 3785:   SEQUENCE {                                                // timeStampToken : TimeStampToken
                                  // contents skipped in this example for brevity and explained in section 7
       :     }
       :   }
```

5. Extending Request

The purpose of the timestamp extending protocol is to supply the hash chains to link timestamps to Integrity Codes published in printed media.

We define an extending request to have the following structure:

```
CertTokenRequest ::= SEQUENCE {
  version INTEGER,
  historyIdentifier INTEGER,
  extensions [0] Extensions OPTIONAL }
```

Field	Meaning
version	Version number of the syntax. The current version is 1.
historyIdentifier	Identifier of the leaf of the calendar tree for which the hash chain connecting it to a published root is requested. Each leaf is identified by the number of seconds elapsed from 1970-01-01 00:00:00 UTC to the time of inserting that leaf to the tree. Essentially, this is a Unix time_t value, handled as a 64-bit unsigned integer.
extensions	A generic way to add additional information. Extensions are not used nor supported in the current implementation.

Example 4: An extending request.

```
   0    9: SEQUENCE {                                          // CertTokenRequest, GuardTime
   2    1:   INTEGER 1                                         // version
   5    4:   INTEGER 1229294488                                // historyIdentifier
                                 // optional extensions : Extensions not used
        :   }
```

6. Extending Response

Upon receiving a request, the server returns a response that indicates a failure or a success and in case of success includes the requested hash chain.

We define an extending response to have the following structure:

```
CertTokenResponse ::= SEQUENCE {
  status PKIStatusInfo,
  certToken [0] CertToken OPTIONAL }
```

<table>
<tr><td>Field</td><td>Meaning</td></tr>
<tr><td>status</td><td>Used to indicate the failure or success in responding to the timestamping request and in case of failure to indicate the cause. Defined and interpreted as specified in Section 4, except two new bit positions are defined for the failInfo field:
<table>
<tr><td>Bit position</td><td>Meaning</td></tr>
<tr><td>100</td><td>Timestamp cannot be extended yet, because the extending service is not aware of an Integrity Code having been published that would be recent enough to include the calendar leaf with the history identifier specified in the request.</td></tr>
<tr><td>101</td><td>Timestamp cannot be extended anymore, because the extending service does not have the calendar data going that far back in the history. Most likely older data has been deleted due to space constraints in the gateway and an attempt to extend the timestamp at another gateway may still succeed.</td></tr>
</table></td></tr>
<tr><td>certToken</td><td>In case of success, contains the requested hash chain and the corresponding publication information.</td></tr>
</table>

We define the hash chain and publication information to be formatted as follows:

```
CertToken ::= SEQUENCE {
  version INTEGER,
  history HashChain,
  publishedData PublishedData,
  pubReference SET OF OCTET STRING,
```

```
  extensions [0] Extensions OPTIONAL }
```

Field	Meaning
version	Version number of the syntax. The current version is 1.
history	The requested hash chain. The inner structure (which does NOT use the ASN.1 syntax) is described in Section 8.1.
publishedData	The Integrity Code to which the timestamp is connected by the hash chain in the previous field. The inner structure of this field is described in Section 8.3.
pubReference	References to printed media where the Integrity Code has been published. The inner structure of this field is described in Section 8.5.
extensions	A generic way to add additional information. Extensions are not used nor supported in the current implementation.

Example 5: An extending response indicating failure.

```
  0   98: SEQUENCE {                                                  // CertTokenResponse, GuardTime
  2   96:   SEQUENCE {                                                // status : PKIStatusInfo, RFC3161
  4    1:     INTEGER 2                                               // status : PKIStatus (2 = rejection)
  7   75:     SEQUENCE {                                              // statusString : PKIFreeText
  9   73:       UTF8String
        :         '.............Data required for completing the request is not yet available'
        :       }
 84   14:     BIT STRING 3 unused bits                                // failInfo : PKIFailureInfo
                 00 00 00 00 00 00 00 00 00 00 00 00 08               // bit 100 = extendLater, GuardTime extension
        :     }
                                    // optional certToken : CertToken not used in case of failure response
        :   }
```

Example 6: An extending response indicating success.

```
  0 1033: SEQUENCE {                                                  // CertTokenResponse, GuardTime
  4    3:   SEQUENCE {                                                // status : PKIStatusInfo, RFC3161
```

```
   6    1:     INTEGER 0                                                    // status : PKIStatus (0 = granted)
                                    // optional statusString : PKIFreeText not used
                                    // optional failInfo : PKIFailureInfo not used
        :     }
   9 1024:   [0] {                                                         // certToken : CertToken, GuardTime
  13    1:     INTEGER 1                                                    // version
  16  972:     OCTET STRING                                                 // history : HashChain
                                    // to conserve space, the hash chain is not encoded in ASN.1
                                    // it is composed of a number of hash step entries, with each
                                    // entry having the following structure:
                                    //    1-byte code of the hash function to be used in this step (1 = SHA-256)
                                    //    1-byte direction indicator (0 = sibling is right-side neighbour; 1 = left-side)
                                    //    1-byte code of the hash function used to compute the sibling hash
                                    //    n-byte sibling hash (value of n is determined by the hash function)
                                    //    1-byte level restriction (upper limit on the number of entries that may precede)
        :       01 01 01 D9 8F 09 82 46 9F 5A F7 7C 9F F0 FD 81 C4 A7 7F 79 23 10 E6 01 83 57 69 0E 8D 28 FA 35 AB 17 27 FF
        :       01 01 01 CE 1C FD 78 65 A8 FF 7E DF 4F 5B 60 0D 91 CA 81 7A 7C B7 50 CC 0D F7 E0 4B 9A 8D B7 18 16 72 47 FF
        :       01 01 01 15 B8 D6 53 7C CC 84 EE F0 FF 71 83 6D C1 43 89 F0 E0 9B 3F D5 ED 2F FE 72 E8 9B E4 09 2A E6 15 FF
                                    // remaining 24 hash steps skipped for brevity
 992   41:     SEQUENCE {                                                   // publishedData : PublishedData
 994    4:       INTEGER 1234656000                                         // publicationIdentifier
1000   33:       OCTET STRING                                               // publicationImprint : DataImprint
                                    // the hash value in the root of the GuardTime Calendar tree:
                                    //    1-byte code of the hash function used to compute the root hash
                                    //    n-byte root hash (value of n is determined by the hash function)
        :         01 EE 1F BC 8F D3 FD 78 FD 11 B9 E2 67 DF 9A F2 36 11 B1 C5 BE 44 F0 20 AB 8B 14 19 C9 36 72 C4 D6
        :       }
1035    0:     SET {}                                                       // pubReference : SET OF OCTET STRING
                                    // optional extensions : Extensions not used
        :     }
        :   }
```

7. Timestamp

7.1. Cryptographic Message Syntax

RFC 3161 defines a timestamp to be a CMS message:

```
TimeStampToken ::= ContentInfo
```

RFC 2630 defines a CMS message to have the following structure:

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

ContentType ::= OBJECT IDENTIFIER
```

Field	Meaning
contentType	Type of the associated content.
content	The content itself.

RFC 3161 defines the content type of timestamps to be:

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

RFC 2630 defines the structure of a signed-data message to be:

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

```
CMSVersion ::= INTEGER

DigestAlgorithmIdentifiers ::= SET OF DigestAlgorithmIdentifier

CertificateSet ::= SET OF CertificateChoices

CertificateRevocationLists ::= SET OF CertificateList

SignerInfos ::= SET OF SignerInfo
```

Field	Meaning
version	Version number of the syntax. GT specific: GuardTime timestamps always use version 3.
digestAlgorithms	A collection of message digest algorithm identifiers. The intention in RFC 2630 is that digesting the signed data with all the listed algorithms yields the digests needed to verify all signatures in the signerInfos collection. RFC 3161 allows only one signature in a timestamp, thus a GuardTime timestamp always has exactly one item in this collection.
encapContentInfo	The signed content, consisting of a content type identifier and the content itself. RC 3161 further defines the structure of this field to be TSTInfo, as described below.
certificates	A collection of certificates. The intention in RFC 2630 is that the set of certificates be sufficient to contain chains from recognized certification authorities to all of the signers in the signerInfos field. GT specific: PKI-signed timestamps use this field to store the certificate for the private key used to sign the timestamp. Extended timestamps are not PKI-signed and thus do not have this field.
crls	A collection of certificate revocation lists (CRLs). GT specific: Since GuardTime uses white-listing of valid certificates instead of black-listing of invalid certificates, GuardTime timestamps do not have this field. See also Section 9 for more details on the publications file that contains the white-list of signing keys.

signerInfos	A collection of per-signer information items, each containing the signature and any signer-specific information needed to verify the signature. RFC 3161 further requires a timestamp to contain only the signature of the TSA.

RFC 2630 defines the encapsulated content of a signed-data message to be:

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

ContentType ::= OBJECT IDENTIFIER
```

Field	Meaning
eContentType	An object identifier that uniquely specifies the content type.
eContent	The content itself.

7.2. Timestamp Token Info

RFC 3161 defines the encapsulated content of timestamps to be of type:

```
id-ct-TSTInfo OBJECT IDENTIFIER ::= { iso(1) member-body(2 us(840) rsadsi(113549) pkcs(1) pkcs-9(9) smime(16) ct(1) 4}
```

and have the following structure:

```
TSTInfo ::= SEQUENCE {
  version INTEGER,
  policy TSAPolicyId,
  messageImprint MessageImprint,
  serialNumber INTEGER,
  genTime GeneralizedTime,
  accuracy Accuracy OPTIONAL,
  ordering BOOLEAN DEFAULT FALSE,
```

```
  nonce INTEGER OPTIONAL,
  tsa [0] GeneralName OPTIONAL,
  extensions [1] IMPLICIT Extensions OPTIONAL }

TSAPolicyId ::= OBJECT IDENTIFIER

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

Field	Meaning
version	Version number of the timestamp structure. GT specific: GuardTime timestamps always use version 1.
policy	The policy under which the timestamp was issued. If the request for a timestamp references a specific policy, the timestamping service must either issue a timestamp referencing the same policy or return an error response.
messageImprint	The digest of the datum timestamped. GT specific: The list of currently supported algorithms is given in Section 11.
serialNumber	An integer assigned to the timestamp by the TSA. The serial number combined with the name of the TSA must be unique. The users of timestamps must be able to handle values up to 160 bits.
genTime	The time at which the timestamp was created by the TSA. It is expressed as UTC (Coordinated Universal Time) time using the syntax YYYYMMDDhhmmss[.s...]Z. See RFC 3161 for more details. GT specific: This is the time when the timestamping request was received by the gateway, according to gateway's local clock; the time when the timestamp was registered in the GuardTime Calendar is encoded in the history hash chain embedded in the TimeSignature described below.
accuracy	The time deviation around the UTC time contained in genTime. The value zero must be assumed for any missing fields. Subtracting the accuracy from and adding it to the value contained in genTime gives the lower and upper limits to the actual time

	of creation of the timestamp by the TSA. If this field is missing, the accuracy of the timestamp can still be available through other means (for example, it may be specified by the timestamping policy).
ordering	If present and contains the value TRUE, any two timestamps from the same TSA can be ordered based on the genTime field. Otherwise two timestamps can only be ordered if the difference of the values of their genTime fields is greater than the sum of the values of their accuracy fields.
nonce	If the request for a timestamp contains a nonce, the timestamping service must issue a timestamp containing the same value.
tsa	Name of the TSA that issued the timestamp.
extensions	A generic way to add additional information. GT specific: GuardTime timestamps do not use this field.

7.3. Signer Info

RFC 2630 defines the per-signer information to have the following structure:

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

CMSVersion ::= INTEGER

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

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

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

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

AttributeValue ::= ANY

SignatureValue ::= OCTET STRING
```

Field	Meaning
version	Version number of the syntax. GT specific: GuardTime timestamps always use version 1.
signerIdentifier	Specifies the signer's public key certificate to be used for verifying the signature. GT specific: GuardTime timestamps always use the issuerAndSerialNumber option of the SignerIdentifier structure.
digestAlgorithm	Identifies the message digest algorithm used by the signer. The digest is computed on the EncapsulatedContent value and stored in the message-digest field in the SignedAttributes structure.
signedAttrs	Identifies the attributes whose values are protected by the signature. RFC 2630 specifies that this field is mandatory for the content type used by timestamps and at least the content-type and message-digest attributes must be present. RFC 3161 further specifies that ESSCertID attribute must be present. GT specific: GuardTime timestamps do not include the ESSCertID attribute. The extended timestamps do not have any PKI-based signatures or certificates, thus the attribute makes no sense for them. But the signedAttrs structure can't be modified during the extending process, so the attribute must be omitted in the PKI-signed form as well.
signatureAlgorithm	Identifies the signature algorithm used by the signer to generate the digital signature. GT specific: GuardTime timestamps always use id-gt-TimeSignatureAlg.

signature	The digital signature itself. GT specific: GuardTime timestamps always use TimeSignature specified below.
unsignedAttrs	A collection of attributes that are not signed. GT specific: GuardTime timestamps do not use this field.

8. Time Signature

We define the object identifier for the GuardTime TimeSignature algorithm to be:

```
id-gt-TimeSignatureAlg OBJECT IDENTIFIER ::= { iso(1) org(3) dod(6) internet(1) private(4) enterprise(1) GuardTime(27868) algorithm(4) 1}
```

We also define the signature value of a TimeSignature to have the following structure:

```
TimeSignature ::= SEQUENCE {
  location HashChain,
  history HashChain,
  publishedData PublishedData,
  pkSignature [0] SignatureInfo OPTIONAL,
  pubReference [1] SET OF OCTET STRING OPTIONAL }

HashChain ::= OCTET STRING
```

Field	Meaning
location	The hash chain that describes the path in the GuardTime aggregator tree from the leaf that issued the timestamp to the top. The inner structure (which does NOT use the ASN.1 syntax) is described below.
history	The hash chain that describes the path in the GuardTime calendar from the leaf corresponding to the second when the timestamp was issued to the root corresponding to the second when the control publication was generated. The inner structure (which does NOT use the ASN.1 syntax) is described below. The shape of this hash chain fixes the time when the timestamp was registered in the GuardTime calendar, relative to the publicationIdentifier field in the PublishedData structure. The algorithm to extract the time value of the leaf (history identifier) from the shape of the hash chain and the time value of the root (publication identifier) is given in Section 8.3.
publishedData	The hash value of the root node of the calendar tree and the time when the value was taken. For extended timestamps, this corresponds to an Integrity Code. For non-extended timestamps, the structure and meaning is the same, but the information is not published in physical media and thus has to be protected by the PKI-based signature in the following field.
pkSignature	The PKI-based signature used to protect the publishedData field of a timestamp that is not yet extended. Not used in extended

	timestamps.
pubReference	References to printed media where the Integrity Code has been published. Not used in PKI-signed timestamps. The inner structure of this field is described in Section 8.5.

8.1. Hash Chain

HashChain represents a hash chain extracted from a Merkle tree. See the main reference document for a general discussion of Merkle trees and hash chains.

HashChain is an OCTET STRING that is a concatenation of a number of HashStep items. The items are listed in the bottom-up order, from the leaf towards the root of the tree. The number of HashStep items is not explicitly encoded in the structure, but determined by the length of the individual items and the length of the HashChain. It is a format error if a hash chain does not consist of a number of whole HashStep items.

A HashStep item is an OCTET STRING that consists of the following fields:

Field	Length	Meaning
algorithm	1 octet	The GuardTime identifier of the hash algorithm used to perform the hashing of input data in this step. The identifiers of all supported algorithms are given in Section 11. Any value not listed there is a format error.
direction	1 octet	The direction indicator. A value of 0 means the sibling hash from the next field is to be concatenated to the hash value of input data from the left. A value of 1 means the sibling hash from the next field is to be concatenated to the hash value of input data from the right. Any value other than 0 or 1 is a format error.
sibling	variable	The sibling hash value of type DataImprint needed to compute the common ancestor in the Merkle tree from which the hash chain has been extracted. The inner structure (which does NOT use the ASN.1 syntax) is described below.
level	1 octet	The level restriction that indicates an upper limit on how many levels may be below the current one in the Merkle tree, or equivalently, how many HashStep items may precede the current one in the HashChain. The restriction is observed in the location hash chain only.

The computation of the hash chain is done by the algorithm ComputeHashChain. The idea is to follow the hash computation steps affected by the input data value in the tree from which the hash chain was extracted. Therefore, starting from the input data in a leaf of the tree, we proceed upwards, on each step hashing the value from the previous step and concatenating it to the sibling to get the value to pass on as the data value to the parent node. More formally:

```
ComputeHashChain:
Input:
	inputData : OCTET STRING
	hashChain : HashChain,
		viewed as a concatenation step[1] | step[2] | … | step[m],
		where step[i] : HashStep
Output:
	outputData : OCTET STRING
Definition:
	outputData := inputData
	for i := 1 to m do
		if step[i].direction == 0 then
			outputData := step[i].sibling |
				step[i].algorithm | ComputeHash(step[i].algorithm, outputData) |
				step[i].level
		else if step[i].direction == 1 then
			outputData := step[i].algorithm | ComputeHash(step[i].algorithm, outputData) |
				step[i].sibling |
				step[i].level
		else
			error
		endif
	endfor
```

8.2. Data Imprint

DataImprint represents a hash value together with an identifier of the algorithm that was used to compute the hash value. Because the GuardTime time signatures contain a lot of hash values, there was a need for a more compact syntax than the ASN.1-based MessageImprint of RFC 3161.

A DataImprint is an OCTET STRING that consists of the following fields:

Field	Length	Meaning
algorithm	1 octet	The GuardTime identifier of the hash algorithm that was used to compute the hash value in the following field. The identifiers of all supported algorithms are given in Section 11. Any value not listed there is a format error.
hashValue	variable	A hash value computed from some data using the algorithm specified by the previous field. The length of the hash value depends on the algorithm.

8.3. Published Data

We define the published data field to have the following structure:

```
PublishedData ::= SEQUENCE {
  publicationIdentifier INTEGER,
  publicationImprint DataImprint }

DataImprint ::= OCTET STRING
```

Field	Meaning
publicationIdentifier	This is the number of seconds elapsed from 1970-01-01 00:00:00 UTC to the time of computing the hash value in the following field. Essentially, this is a Unix time_t value, handled as a 64-bit unsigned integer.
publicationImprint	This is the hash value from the root node of the calendar tree at the moment indicated by the previous field. The inner structure (which does NOT use the ASN.1 syntax) is described in Section 8.2.

The computation of the history identifier from the publication identifier and the shape of the hash chain is done by the algorithm ComputeHistoryID. The algorithm, whose derivation is given in the main reference document, is as follows:

```
ComputeHistoryID:
Input:
```

```
      publicationID : INTEGER
      hashChain : HashChain,
            viewed as a concatenation step[1] | step[2] | … | step[m],
            where step[i] : HashStep
Output:
      historyID : INTEGER
Definition:
      historyID := 0
      for i := m downto 1 do
            offset := ExtractHighestOneBit(publicationID)
            if step[i].direction == 1 then
                  // sibling is on the right side, that is
                  // history ID is in the left sub-tree
                  publicationID := offset - 1
            else if step[i].direction == 0 then
                  // sibling is on the left side, that is
                  // history ID is in the right sub-tree
                  historyID := historyID + offset
                  publicationID := publicationID - offset
            else
                  error
            endif
      endfor

ExtractHighestOneBit:
Input:
      input : INTEGER
Output:
      output : INTEGER
Definition:
      if input == 0 then
            error
      endif
      output := 1
      while input > 0 do
            input := input shifted right 1 bit
            output := output shifted left 1 bit
      endwhile
```

The verification of the match between the hash chains and the publication imprint is done by the algorithm CheckHashChains:

```
CheckHashChains:
Input:
	signerInfo : SignerInfo
	timeSignature : TimeSignature
Output:
	A success or error status
Definition:
	// the correspondence between the ASN.1 OIDs and GuardTime IDs is defined in Section 11
	algClient := GuardTimeID(signerInfo.digestAlgorithm)
	output := ComputeHash(algClient, DER(signerInfo.signedAttrs)
	// hash chain computation is defined in Section 8.1
	output := ComputeHashChain(output, timeSignature.location)
	output := ComputeHashChain(output, timesignature.history)
	// the first byte of a DataImprint is the hash algorithm ID
	algServer := timeSignature.publishedData.publicationImprint[1]
	output := algServer | ComputeHash(algServer, output)
	// the computed value must match the one in the timestamp
	if output == timeSignature.publishedData.publicationImprint then
		success
	else
		error
	endif
```

8.4. PKI Signature

We define the PKI-based signature to be represented by the following structure:

```
SignatureInfo ::= SEQUENCE {
  signatureAlgorithm AlgorithmIdentifier,
  signatureValue OCTET STRING,
  keyCommitmentRef [0] SET OF OCTET STRING OPTIONAL }
```

Field	Meaning

signatureAlgorithm	Identifier of a conventional PKI signature algorithm.
signatureValue	Output of the signature algorithm.
keyCommitmentRef	References to printed media where the digest of the signing key has been published. The inner structure is described in Section 8.5.

8.5. Bibliographic References

A bibliographic reference to a printed medium is an OCTET STRING that consists of the following fields:

Field	Length	Type	Meaning
version	2 octets	16-bit integer	Version number. Currently only version 1 is defined and any other value is a format error.
content	variable	variable	The content itself. If version is 1, then the content is an UTF-8 string, as defined in RFC 3629.

Example 7: An unextended timestamp.

```
  0 3785: SEQUENCE {                                                  // TimeStampToken, RFC3161
                                                                      // same as ContentInfo, RFC2630
  4    9:   OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)       // contentType
 15 3770:   [0] {
 19 3766:     SEQUENCE {                                             // content : SignedData
 23    1:       INTEGER 3                                           // version : CMSVersion
 26   15:       SET {                                               // digestAlgorithms : DigestAlgorithmIdentifiers
 28   13:         SEQUENCE {                                        // AlgorithmIdentifier, RFC2459
 30    9:           OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1)  // algorithm
 41    0:           NULL                                            // parameters
       :           }
       :         }
 43  151:       SEQUENCE {                                          // encapContentInfo : EncapsulatedContentInfo
```

```
 46   11:          OBJECT IDENTIFIER tSTInfo (1 2 840 113549 1 9 16 1 4)       // eContentType : ContentType
 59  135:          [0] {
 62  132:            OCTET STRING, encapsulates {                              // eContent
 65  129:              SEQUENCE {                                              // TSTInfo, RFC3161
 68    1:                INTEGER 1                                             // version
 71   11:                OBJECT IDENTIFIER                                     // policy : TSAPolicyId
       :                  id-gt-tsPolicy 1 v1 (1 3 6 1 4 1 27868 2 1 1)
 84   49:                SEQUENCE {                                            // messageImprint : MessageImprint
 86   13:                  SEQUENCE {                                          // hashAlgorithm : AlgorithmIdentifier, RFC2459
 88    9:                    OBJECT IDENTIFIER                                 // algorithm
       :                      sha-256 (2 16 840 1 101 3 4 2 1)
 99    0:                    NULL                                              // parameters
       :                    }
101   32:                  OCTET STRING                                        // hashedMessage
       :                    54 66 E3 CB A1 4A 84 3A 5E 93 B7 8E 3D 6A B8 D3 49 1E DC AC 7E 06 43 1C E1 A7 F4 98 28 C3 40 C3
       :                  }
135   16:                INTEGER                                               // serialNumber
       :                  49 AB A6 22 00 02 00 01 00 01 00 00 00 5B AD 46
153   15:                GeneralizedTime 02/03/2009 09:25:54 GMT               // genTime : GeneralizedTime, RFC2459
                                     // note that this is the time when the timestamping request was
                                     // received by the gateway, according to gateway's local clock;
                                     // the time when the timestamp was registered in the GuardTime
                                     // Calendar is encoded in the history hash chain below
170    3:                SEQUENCE {                                            // accuracy : Accuracy
172    1:                  INTEGER 1                                           // seconds
                                     // optional millis : INTEGER not used
                                     // optional micros : INTEGER not used
       :                  }
                                     // optinal ordering : BOOLEAN not used
                                     // optinal nonce : INTEGER not used
175   20:                [0] {                                                 // tsa : GeneralName, RFC2459
177   18:                  [6] 'ATS1.guardtime.com'                            // uniformResourceIdentifier : IA5String
       :                  }
                                     // optional extensions : Extensions not used
       :                } // TSTInfo ends
       :              }
       :            }
       :          } // EncapsulatedContentInfo ends
```

```
 197  708:       [0] {                                                    // certificates : CertificateSet
                                  // contents skipped in this example for brevity
                                  // as they are not specific to timestamping
        :         }
 909 2876:       SET {                                                    // signerInfos : SignerInfos, RFC2630
 913 2872:         SEQUENCE {                                             // SignerInfo
 917    1:           INTEGER 1                                            // version : CMSVersion
 920   43:           SEQUENCE {                                           // signerIdentifier : SignerIdentifier
                                                                          // we use IssuerAndSerialNumber option
 922   38:             SEQUENCE {                                         // issuer : Name = RDNSequence
 924   13:               SET {                                            // RelativeDistinguishedName
 926   11:                 SEQUENCE {                                     // AttributeTypeAndValue
 928    3:                   OBJECT IDENTIFIER commonName (2 5 4 3)       // type
 933    4:                   PrintableString 'TSA2'                       // value
        :                   }
        :                 }
 939   21:               SET {
 941   19:                 SEQUENCE {                                     // AttributeTypeAndValue
 943    3:                   OBJECT IDENTIFIER organizationName (2 5 4 10) // type
 948   12:                   PrintableString 'GuardTime AS'               // value
        :                   }
        :                 }
        :               }
 962    1:             INTEGER 1                                          // serialNumber
        :             }
 965   13:           SEQUENCE {                                           // digestAlgorithm : DigestAlgorithmIdentifier
 967    9:             OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1) // algorithm
 978    0:             NULL                                               // parameters
        :             }
 980   77:           [0] {                                                // signedAttrs : SignedAttributes
 982   26:             SEQUENCE {                                         // Attribute
 984    9:               OBJECT IDENTIFIER contentType (1 2 840 113549 1 9 3) // attrType
 995   13:               SET {                                            // attrValues
 997   11:                 OBJECT IDENTIFIER                              // AttributeValue
        :                   tSTInfo (1 2 840 113549 1 9 16 1 4)
        :                 }
        :               }
1010   47:             SEQUENCE {                                         // Attribute
```

```
1012    9:                OBJECT IDENTIFIER                                        // attrType
        :                   messageDigest (1 2 840 113549 1 9 4)
1023   34:                SET {                                                    // attrValues
1025   32:                  OCTET STRING                                           // AttributeValue
        :                     37 26 FD 3A 3A E5 79 0B AA 77 AD 7C DF D0 0B FF 20 C3 32 92 DE F3 73 DE BD 7C B9 7F 4D 2C 63 38
        :                   }
        :                 }
        :               }
1059   14:             SEQUENCE {                                                 // signatureAlgorithm : SignatureAlgorithmIdentifier
1061   10:               OBJECT IDENTIFIER                                        // algorithm
        :                   id-gt-TimeSignatureAlg (1 3 6 1 4 1 27868 4 1)
1073    0:               NULL                                                     // parameters
        :               }
1075 2710:             OCTET STRING, encapsulates {                               // signature : SignatureValue
1079 2706:               SEQUENCE {                                               // TimeSignature, GuardTime
1083 1476:                 OCTET STRING                                           // location : HashChain
                                     // to conserve space, the hash chain is not encoded in ASN.1
                                     // it is composed of a number of hash step entries, with each
                                     // entry having the following structure:
                                     //    1-byte code of the hash function to be used in this step (1 = SHA-256)
                                     //    1-byte direction indicator (0 = sibling is right-side neighbour; 1 = left-side)
                                     //    1-byte code of the hash function used to compute the sibling hash
                                     //    n-byte sibling hash (value of n is determined by the hash function)
                                     //    1-byte level restriction (upper limit on the number of entries that may precede)
        :                    01 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 02
        :                    01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
        :                    01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04
                                     // remaining 38 hash steps skipped for brevity
2563  900:                 OCTET STRING                                           // history : HashChain
                                     // the encoding is the same as for the location hash chain
        :                    01 00 01 8F 79 CB A8 69 3A 5A 01 1B 0F 1F 43 21 03 F2 59 44 13 77 07 97 42 F4 87 50 F1 88 B3 D1 9C ED 0F FF
        :                    01 00 01 19 17 68 D5 BB F1 E0 25 A7 66 E9 C9 F4 0D FC A4 F4 D5 D6 70 CE AF 43 34 B2 7A DC 6C 7B 2B AE 9A FF
        :                    01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF
                                     // remaining 22 hash steps skipped for brevity
3467   41:                 SEQUENCE {                                             // publishedData : PublishedData
3469    4:                   INTEGER 1236038400                                   // publicationIdentifier
3475   33:                   OCTET STRING                                         // publicationImprint : DataImprint
                                     // the hash value in the root of the GuardTime Calendar tree:
```

```
                                    //    1-byte code of the hash function used to compute the root hash
                                    //    n-byte root hash (value of n is determined by the hash function)
        :                     01 33 FE 03 E7 98 90 20 CE C1 C8 DD F8 B3 17 0E 20 AC A3 55 3D A7 7C 32 9C 72 14 A4 0A BA 20 39 E7
        :                   }
3510  275:                 [0] {                                                 // pkSignature : SignatureInfo
3514   13:                   SEQUENCE {                                          // signatureAlgorithm : AlgorithmIdentifier
3516    9:                     OBJECT IDENTIFIER                                 // algorithm
        :                       sha256WithRSAEncryption (1 2 840 113549 1 1 11)
3527    0:                     NULL                                              // parameters
        :                     }
3529  256:                   OCTET STRING                                        // signatureValue
                                    // contents skipped in this example for brevity
                                    // this is a standard RSA signature, just a very large integer
        :                   }
        :                 }
        :               }
        :             }
        :           }
        :         }
        :       }
        :     }
        :   }
```

Example 8: An extended timestamp.

```
   0 2938: SEQUENCE {                                                          // TimeStampToken, RFC3161
                                                                               // same as ContentInfo, RFC2630
   4    9:   OBJECT IDENTIFIER signedData (1 2 840 113549 1 7 2)               // contentType
  15 2923:   [0] {
  19 2919:     SEQUENCE {                                                     // content : SignedData
  23    1:       INTEGER 3                                                    // version : CMSVersion
  26   15:       SET {                                                        // digestAlgorithms : DigestAlgorithmIdentifiers
  28   13:         SEQUENCE {                                                 // AlgorithmIdentifier, RFC2459
  30    9:           OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1)       // algorithm
  41    0:           NULL                                                     // parameters
        :           }
        :         }
  43  151:       SEQUENCE {                                                   // encapContentInfo : EncapsulatedContentInfo
  46   11:         OBJECT IDENTIFIER tSTInfo (1 2 840 113549 1 9 16 1 4)      // eContentType : ContentType
```

```
 59  135:         [0] {
 62  132:           OCTET STRING, encapsulates {                          // eContent
 65  129:             SEQUENCE {                                          // TSTInfo, RFC3161
 68    1:               INTEGER 1                                         // version
 71   11:               OBJECT IDENTIFIER                                 // policy : TSAPolicyId
       :                 id-gt-tsPolicy 1 v1 (1 3 6 1 4 1 27868 2 1 1)
 84   49:               SEQUENCE {                                        // messageImprint : MessageImprint
 86   13:                 SEQUENCE {                                      // hashAlgorithm : AlgorithmIdentifier, RFC2459
 88    9:                   OBJECT IDENTIFIER                             // algorithm
       :                     sha-256 (2 16 840 1 101 3 4 2 1)
 99    0:                   NULL                                          // parameters
       :                   }
101   32:                 OCTET STRING                                    // hashedMessage
       :                   54 66 E3 CB A1 4A 84 3A 5E 93 B7 8E 3D 6A B8 D3 49 1E DC AC 7E 06 43 1C E1 A7 F4 98 28 C3 40 C3
       :                 }
135   16:               INTEGER                                           // serialNumber
       :                 49 45 8B 97 00 02 00 01 00 01 00 00 00 59 25 0C
153   15:               GeneralizedTime 14/12/2008 22:41:27 GMT           // genTime : GeneralizedTime, RFC2459
                                     // note that this is the time when the timestamping request was
                                     // received by the gateway, according to gateway's local clock;
                                     // the time when the timestamp was registered in the GuardTime
                                     // Calendar is encoded in the history hash chain below
170    3:               SEQUENCE {                                        // accuracy : Accuracy
172    1:                 INTEGER 1                                       // seconds
                                     // optional millis : INTEGER not used
                                     // optional micros : INTEGER not used
       :                 }
                                     // optinal ordering : BOOLEAN not used
                                     // optinal nonce : INTEGER not used
175   20:               [0] {                                             // tsa : GeneralName, RFC2459
177   18:                 [6] 'ATS1.guardtime.com'                        // uniformResourceIdentifier : IA5String
       :                 }
                                     // optional extensions : Extensions not used
       :               } // TSTInfo ends
       :             }
       :           }
       :         } // EncapsulatedContentInfo ends
                                     // optional certificates : CertificateSet not used in extended timestamps
```

```
197 2741:       SET {                                                   // signerInfos : SignerInfos, RFC2630
201 2737:         SEQUENCE {                                            // SignerInfo
205    1:           INTEGER 1                                          // version : CMSVersion
208   43:           SEQUENCE {                                         // signerIdentifier : SignerIdentifier
                                                                       // we use IssuerAndSerialNumber option
210   38:             SEQUENCE {                                       // issuer : Name = RDNSequence
212   13:               SET {                                          // RelativeDistinguishedName
214   11:                 SEQUENCE {                                   // AttributeTypeAndValue
216    3:                   OBJECT IDENTIFIER commonName (2 5 4 3)     // type
221    4:                   PrintableString 'TSA2'                     // value
       :                   }
       :                 }
227   21:               SET {
229   19:                 SEQUENCE {                                   // AttributeTypeAndValue
231    3:                   OBJECT IDENTIFIER organizationName (2 5 4 10) // type
236   12:                   PrintableString 'GuardTime AS'             // value
       :                   }
       :                 }
       :               }
250    1:             INTEGER 1                                        // serialNumber
       :             }
253   13:           SEQUENCE {                                         // digestAlgorithm : DigestAlgorithmIdentifier
255    9:             OBJECT IDENTIFIER sha-256 (2 16 840 1 101 3 4 2 1) // algorithm
266    0:             NULL                                             // parameters
       :             }
268   77:           [0] {                                              // signedAttrs : SignedAttributes
270   26:             SEQUENCE {                                       // Attribute
272    9:               OBJECT IDENTIFIER contentType (1 2 840 113549 1 9 3) // attrType
283   13:               SET {                                          // attrValues
285   11:                 OBJECT IDENTIFIER                            // AttributeValue
       :                   tSTInfo (1 2 840 113549 1 9 16 1 4)
       :                 }
       :               }
298   47:             SEQUENCE {                                       // Attribute
300    9:               OBJECT IDENTIFIER                              // attrType
       :                 messageDigest (1 2 840 113549 1 9 4)
311   34:               SET {                                          // attrValues
313   32:                 OCTET STRING                                 // AttributeValue
```

```
         :                   1C AD 2F 08 E2 6D 41 1D 72 BD DF C1 8E 3A E1 21 57 C5 5C 8D C0 C2 D7 07 0F 43 4C 82 D3 38 89 1A
         :                 }
         :               }
         :             }
 347   14:           SEQUENCE {                                              // signatureAlgorithm : SignatureAlgorithmIdentifier
 349   10:             OBJECT IDENTIFIER                                     // algorithm
         :               id-gt-TimeSignatureAlg (1 3 6 1 4 1 27868 4 1)
 361    0:             NULL                                                  // parameters
         :             }
 363 2575:           OCTET STRING, encapsulates {                            // signature : SignatureValue
 367 2571:             SEQUENCE {                                            // TimeSignature, GuardTime
 371 1548:               OCTET STRING                                        // location : HashChain
                                     // to conserve space, the hash chain is not encoded in ASN.1
                                     // it is composed of a number of hash step entries, with each
                                     // entry having the following structure:
                                     //    1-byte code of the hash function to be used in this step (1 = SHA-256)
                                     //    1-byte direction indicator (0 = sibling is right-side neighbour; 1 = left-side)
                                     //    1-byte code of the hash function used to compute the sibling hash
                                     //    n-byte sibling hash (value of n is determined by the hash function)
                                     //    1-byte level restriction (upper limit on the number of entries that may precede)
         :                   01 00 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 02
         :                   01 00 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 03
         :                   01 01 01 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 04
                                     // remaining 40 hash steps skipped for brevity
1923  972:               OCTET STRING                                        // history : HashChain
                                     // the encoding is the same as for the location hash chain
         :                   01 01 01 D9 8F 09 82 46 9F 5A F7 7C 9F F0 FD 81 C4 A7 7F 79 23 10 E6 01 83 57 69 0E 8D 28 FA 35 AB 17 27 FF
         :                   01 01 01 CE 1C FD 78 65 A8 FF 7E DF 4F 5B 60 0D 91 CA 81 7A 7C B7 50 CC 0D F7 E0 4B 9A 8D B7 18 16 72 47 FF
         :                   01 01 01 15 B8 D6 53 7C CC 84 EE F0 FF 71 83 6D C1 43 89 F0 E0 9B 3F D5 ED 2F FE 72 E8 9B E4 09 2A E6 15 FF
                                     // remaining 22 hash steps skipped for brevity
2899   41:               SEQUENCE {                                          // publishedData : PublishedData
2901    4:                 INTEGER 1234656000                                // publicationIdentifier
2907   33:                 OCTET STRING                                      // publicationImprint : DataImprint
                                     // the hash value in the root of the GuardTime Calendar tree:
                                     //    1-byte code of the hash function used to compute the root hash
                                     //    n-byte root hash (value of n is determined by the hash function)
         :                   01 EE 1F BC 8F D3 FD 78 FD 11 B9 E2 67 DF 9A F2 36 11 B1 C5 BE 44 F0 20 AB 8B 14 19 C9 36 72 C4 D6
         :                 }
```

```
                                        // optional pkSignature : SignatureInfo not used in extended timestamps
        :                 }
        :               }
        :             }
        :           }
        :         }
        :       }
        :     }
```

9. Publications File

Publications file is the device for electronic delivery of Integrity Codes and the list of currently valid PKI signing keys for verification of timestamps for which there is no Integrity Code available yet.

The file consists of a fixed header, followed by a variable number of publication data cells, followed by a variable number of certificate hash cells, followed by one set of publication references, followed by one digital signature to check the integrity and authenticity of the publications file.

We define the structure of the header to be as follows:

Field	Type	Meaning
version	16-bit unsigned integer	Version. Currently only version 1 is defined and any other value is a format error.
firstPublication	64-bit unsigned integer	Identifier of the earliest publication in the file.
publicationsBegin	32-bit unsigned integer	Offset of the first publication data cell in the file.
publicationCellSize	16-bit unsigned integer	Size of a publication data cell. This is equal to 8+max(sizeof(publicationImprint)). As long as all publications are produced using SHA-256, the cell size is 8+(1+32)=41 octets.
publicationsCount	32-bit unsigned integer	Number of the publication data cells in the file.
certHashesBegin	32-bit unsigned integer	Offset of the first certificate hash cell in the file.
certHashCellSize	16-bit unsigned integer	Size of a certificate hash cell. This is equal to 8+max(sizeof(certHash)). As long as all certificates are hashed with SHA-256, the cell size is 8+(1+32)=41 octets.
certHashesCount	16-bit unsigned integer	Number of the certificate hash cells in the file.
pubReferenceBegin	32-bit unsigned integer	Offset of the bibliographic reference block in the file.
signatureBlockBegin	32-bit unsigned integer	Offset of the signature block in the file.

After the header, a number of publication data cells follow. We define the structure of a publication data cell to be:

Field	Type	Meaning
publicationIdentifier	64-bit unsigned integer	Publication identifier. The number of seconds from 1970-01-01 00:00:00 UTC to the moment the publication was created.
publicationImprint	DataImprint	Root hash of the GuardTime calendar tree, formatted as defined in Section 8.2.
padding	octet string	publicationCellSize-8-sizeof(publicationImprint) zero bytes to even out the cell sizes.

After the publication data cells, a number of certificate hash cells follow. We define the structure of a certificate hash cell to be:

Field	Type	Meaning
certTime	64-bit unsigned integer	Certificate publication time. The number of seconds from 1970-01-01 00:00:00 UTC to the moment the validity of the certificate starts.
certHash	DataImprint	Hash value of the public key of the PKI key pair, formatted as defined in Section 8.2.
padding	octet string	certHashCellSize-8-sizeof(certHash) zero bytes to even out the cell sizes.

After the certificate hash cells, a set of bibliographic references for the most recent publication in the file follows. This is a DER-encoded ASN.1 SET OF OCTET STRING. See Section 8.5 for more details.

After the bibliographic references, a PKCS#7 digital signature of all the preceding bytes (that is, from the beginning of the header to the end of the bibliographic references) follows. See RFC 2630 for more details.

When verifying the signature on the publications file, the following steps have to be performed:

1. Check that the signature block is a properly formed detached PKCS#7 signature on all the preceding bytes in the publications file.
2. Check that the signature contains a certificate on the signing key issued to publications@guardtime.com that has the digitalSignature key usage bit set (see “Internet X.509 Public Key Infrastructure. Certificate and CRL Profile”, published as RFC 2459 for details).

3. Check that the certificate chain embedded in the signature leads to a trusted root certificate.

If any of the checks fail, the publications file is to be considered invalid and must not be used.

Example 9: GuardTime publications file.

```
00 01                                                                   // version
00 00 00 00 48 03 f0 00                                                 // earliest publication identifier
00 00 00 24                                                             // offset of first publication cell in file
00 29                                                                   // publication cell size
00 00 00 0b                                                             // number of publication cells
00 00 01 e7                                                             // offset of first certificate hash cell in file
00 29                                                                   // certificate hash cell size
00 04                                                                   // number of ceritificate hash cells
00 00 02 8b                                                             // offset of bibliographic references in file
00 00 02 8d                                                             // offset of digital signature in file
                                    // publication cells, 0x0000000b = 11 of them, 0x0029 = 41 bytes each
                                    //    8-byte publication identifier, time_t value
                                    //    1-byte code of the hash function used to compute the root hash
                                    //    n-byte root hash (value of n is determined by the hash function)
                                    //    m-byte padding with 0x00 for shorter hash values to make all cells the same size
00 00 00 00 48 03 f0 00 01 de 7f bb e0 93 25 1c 42 ac 69 f1 a7 4a 4b 00 d5 42 3d a6 d4 9f aa 55 c7 a2 d9 61 43 3f 3f 57 e5
00 00 00 00 48 2b 7d 00 01 2a 48 20 cd 4c 4b a8 2a 83 a7 7c 17 0c 15 d6 db 9d b4 17 46 29 a5 1c f3 d7 87 fa 44 81 3a 3c 94
00 00 00 00 48 54 5b 80 01 08 ff ab 20 49 7c fc 0c 11 b4 51 bf e1 21 d5 6a 64 fc a2 53 8f ce f5 fc aa 5d e9 0c d7 83 e2 65
00 00 00 00 48 7b e8 80 01 db 86 30 41 5d 83 97 09 29 1b 4c 59 a4 49 69 82 30 7c ce 7d 1b d9 1b de dd 21 af c6 76 3a 79 34
00 00 00 00 48 a4 c7 00 01 6b bb 38 13 70 ff 7b a4 58 70 9f 5e 84 98 da 88 3c 8b a7 ed 9b 73 39 50 01 16 a1 e1 aa fa 19 2a
00 00 00 00 48 cd a5 80 01 5d 56 20 78 cb 6b 31 a1 85 a2 8e d2 71 34 31 3b 76 96 0d f1 c5 e9 df d2 53 ee a7 84 68 cd 49 da
00 00 00 00 48 f5 32 80 01 37 f4 e7 c1 cb 00 17 2c 0d 5f 4c 25 cc 93 2a 67 44 54 62 6a aa a8 a5 a2 f8 72 1b 04 69 48 53 ac
00 00 00 00 49 1e 11 00 01 4c c5 72 75 63 ac 33 4f b8 24 29 b7 1d ed 1c c2 7e 0a b9 b1 9a 0b 4c 79 92 10 9c 1b ad 64 81 da
00 00 00 00 49 45 9e 00 01 c0 28 7e fa 1e 41 ca 14 13 37 12 b6 e4 3f 47 c2 c4 05 e1 e6 a0 d5 f8 7b f9 a3 5c 04 87 b0 93 08
00 00 00 00 49 6e 7c 80 01 f6 d7 c9 4d f9 60 8a db c8 2a 67 0b 17 65 a3 62 60 97 9e 9b c9 ca 7a bb 19 22 1d d1 ce 15 e6 33
00 00 00 00 49 97 5b 00 01 ee 1f bc 8f d3 fd 78 fd 11 b9 e2 67 df 9a f2 36 11 b1 c5 be 44 f0 20 ab 8b 14 19 c9 36 72 c4 d6
                                    // certificate hash cells, 0x0004 = 4 of them, 0x0029 = 41 bytes each
                                    //    8-byte time_t value for certificate publication time
                                    //    1-byte code of the hash function used to compute the hash of the public key
                                    //    n-byte hash of the public key (value of n is determined by the hash function)
                                    //    m-byte padding with 0x00 for shorter hash values to make all cells the same size
00 00 00 00 46 8e 6a a5 01 d1 48 eb f6 f7 e6 c6 d5 26 02 62 04 65 18 9f 71 79 2e 89 f9 cb ed ad 92 0d ec e5 af a4 ec 1b c1
```

```
00 00 00 00 46 8e 6a c0 01 fe cd 05 3e f2 77 fa 2a cc 26 41 59 c1 4a 3f cd 9a 46 32 8e 12 05 8f 40 59 3e 90 af c5 fb ad 5f
00 00 00 00 47 f3 8f 1c 01 33 87 16 99 0b 1e 4f 7d ac 52 9b 56 c3 1c 06 54 c1 c3 d5 c3 f9 08 4f 03 2e 5a 3b 3f d5 c7 c6 af
00 00 00 00 47 f3 8f b0 01 ff 99 1e 47 d0 a1 76 d8 06 a4 73 13 24 ae 3d 62 23 a7 3a 28 36 9b 24 cd a0 a3 44 6c 78 3b de da
                                       // bibliographic references for the last publication cell
                                       // DER-encoded ASN.1 SET OF OCTET STRING
                                       // empty set in this example
31 00
                                       // PKCS#7 signature of all preceding bytes using SHA-256 as the signature hash function
30 82 0c a1 06 09 2a 86 48 86 f7 0d 01 07 02 a0 82 0c 92 30 82 0c 8e 02 01 01 31 0f 30 0d 06 09
                                       // remaining 3205 bytes of the signature skipped for brevity
```

10. Publication String

A publication string contains a PublishedData structure (see Section 8.3), formatted in a way suitable of printed media and manual entry into a verification tool. Towards the first goal, the representation is limited to letters and numbers only; towards the second goal, it embeds a checksum to detect typing errors.

The publication string is constructed as follows:

1. The publication data is assembled as a concatenation of
   - the publication identifier, represented as a 64-bit unsigned integer, with the bits ordered from the most significant to the least significant;
   - the publication imprint, consisting of the 1-byte identifier of the hash algorithm and the hash value itself.
2. Then the CRC-32 checksum of the publication data is computed (see "Error-correcting procedures for DCEs using asynchronous-to-synchronous conversion", published as ITU-T V.42) and appended to the data.
3. Then the resulting sequence is represented in base 32 (see "The Base16, Base32, and Base64 Data Encodings", published as RFC 4648) and optionally broken into groups of 6 or 8 characters by dashes.

Example 10: GuardTime publication string for 2009-02-15 00:00:00 UTC, formatted for printing.

```
AAAAAA-CJS5NQ-AAPOD6-6I7U75-PD6RDO-PCM7PZ-V4RWCG-Y4LPSE-6AQKXC-YUDHET-M4WE23-XFPW6G
```

Example 11: GuardTime publication string for 2009-02-15 00:00:00 UTC, raw data.

```
                                        // 8-byte integer 1234656000, the time_t value for 2009-02-15 00:00:00
00 00 00 00 49 97 5B 00
                                        // 1-byte code of the hash function used to compute the root hash
                                        // n-byte root hash (value of n is determined by the hash function)
01 EE 1F BC 8F D3 FD 78 FD 11 B9 E2 67 DF 9A F2 36 11 B1 C5 BE 44 F0 20 AB 8B 14 19 C9 36 72 C4 D6
                                        // 4-byte CRC-32 checsum of the preceding bytes
EE 57 DB C6
```

11. Supported Hash Algorithms

The RFC 3161 that GuardTime timestamps are based on requires the timestamping authority to check that the hash value submitted for timestamping has been created using a sufficiently strong algorithm. Table 1 lists the hash functions currently supported in client hashes. SHA-256 is recommended as the default unless the client has a reason to prefer another algorithm.

All internal hashing by the GuardTime service (cross-links between sections of a timestamp, building of aggregation trees, and building of the calendar tree) is currently done using the SHA-256 algorithm.

Table 1: Hash functions supported by GuardTime.

Name	Digest length	ASN.1 OID	GuardTime ID
SHA-1	20 bytes	1.3.14.3.2.26	0
SHA-256	32 bytes	2.16.840.1.101.3.4.2.1	1
RIPEMD-160	20 bytes	1.3.36.3.2.1	2
SHA-224	28 bytes	2.16.840.1.101.3.4.2.4	3
SHA-384	48 bytes	2.16.840.1.101.3.4.2.2	4
SHA-512	64 bytes	2.16.840.1.101.3.4.2.3	5