U N I V E R S I T Y O F T A R T U
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science

Ahto Truu

Standards for Hash-Linking Based Time-Stamping Schemes

Master's Thesis (60 ECP)

Supervisor: prof. A. Buldas

Author: ..... " ....." 2010

Supervisor: ..... " ....." 2010

TARTU 2010

Contents

Introduction

The world is more and more relying on the accuracy of digital data. At the same time, most of the data is kept on easily and almost untraceably updateable media. There's an increasing need to have tools to ascertain the time when a datum was created or last modified, and to conclusively prove that it has not been tampered with since the last authorized change.

The most common approach today is to have a trusted third party, called time-stamping authority, add a (presumably accurate) time reading to the datum in question and sign the pair. This solution works only as far as the parties relying on the data are willing to trust the authority to neither collude with any of its customers nor issue false time-stamps by accident.

A way to reduce or even remove the need to trust the authority is to verifiably link the time-stamps to each other and/or externally occurring widely witnessed events, such as newspapers being published.

The one-wayness of cryptographic hash functions is somewhat similar to the one-wayness of the time itself, where information can flow from the past to the future, but not in the opposite direction. This property of hash functions has given rise to several schemes for building independently verifiable time-stamping systems.

For these systems to be usable in practice, the algorithms, protocols, and data formats have to be standardized so that all parties would have a common understanding of the meaning of each piece of evidence.

The goal of this paper is to survey existing standards for time-stamping systems based on linking data items with cryptographic hash functions and to investigate to what extent these standards are being followed by actual implementations of time-stamping services.

We review the relevant standards from four major bodies: the Internet Engineering Task Force, the European Telecommunications Standards Institute, the International Organization for Standardization, and the American National Standards Institute.

We also examine three time-stamping services — Surety, GuardTime, and CHRONOS — for their compatibility with the standards.

Chapter 1

Cryptographic Time-Stamping

The purpose of a cryptographic time-stamping service is to provide, upon request from the owner of some datum, evidence that can later be used to verify the existence of that datum at the time the request was submitted to the service. To avoid forgery, the evidence provided should be in a form that allows detection of any subsequent modifications of either the datum or the claimed time of its existence.

Until 1990, it was believed that the only way to provide such evidence is to introduce a trusted third party, called time-stamping authority (TSA), who would add a time reading to the datum submitted and then digitally sign the pair to produce a time-stamp token. The problem with this scheme is that it requires anyone relying on these tokens to trust the TSA.

Those who accept the time-stamps as evidence must believe that the TSA is not colluding with its customers to insert false time values in the tokens. Those who rely on the time-stamps to prove the integrity of their data have to trust that the TSA keeps its signing key secure, as even a single instance of unauthorized use of the key would cast a shadow of doubt on all tokens ever signed with it.

In the early 1990s it was discovered that cryptographic hash functions can be used to build efficient time-stamping schemes that do not require any relying parties to trust the TSA.

1.1 Hash Functions

A cryptographic *hash function* is a deterministic procedure that takes an arbitrary block of data and returns a fixed-size bit string, the *hash value*, such that any change to the data (whether accidental or intentional) would, with very high probability, cause the function to return a completely different hash value.

The input data are often called the *message*, and then the hash value is called the *message digest* or just *digest*.

For a cryptographic hash function H to be considered good, it should have the following four main properties:

- given any message m, the procedure to compute the digest $d = H(m)$ should be efficient;
- given any digest value d, it should be infeasible to find a message m such that $H(m) = d$; a function that has this property is called *preimage resistant*;
- given any message m_1, it should be infeasible to find a message m_2 such that $m_1 \neq m_2$, but $H(m_1) = H(m_2)$; a function that has this property is called *second preimage resistant*;
- it should be infeasible to find two messages m_1 and m_2 such that $m_1 \neq m_2$, but $H(m_1) = H(m_2)$; a function that has this property is called *collision resistant*.

A cryptographic hash function H can be used to time-stamp a message m without disclosing the message itself. One way to do it is to compute the digest $d = H(m)$ and publish it in a newspaper or an electronic message board that is mirrored on many servers.

If the digest is published in a widely witnessed medium, it will be very hard to remove or fake the evidence of when it was published. If the function H is preimage resistant, this means that the message m must have existed before the digest d was published. If the function H is second preimage resistant, then even disclosing the message m at a later date does not allow anyone else to find a slightly different message m' (for example, changing the name of the author on the original message m) that would have the same digest d.

In fact, this idea of time-stamping is quite old. It was used as early as in 1610 by Galileo Galilei. Galilei, having recently invented the telescope, discovered that Venus seemed to have phases that were inconsistent with the geocentric model of the world generally accepted at the time. However, the topic was quite sensitive and he did not want to publish his findings until he had double-checked them. On the other hand, he wanted to make sure he would get the credit of being the first to make the observation.

Thus, he published his findings as the anagram 'haec immatura a me iam frustra leguntur o.y.' (which means "these are at present too young to be read by me") and only later revealed that the letters should really be read in the order 'Cynthiae figuras aemulatur mater amorum' (which means "the mother of love imitates the shapes of Cynthia" — Venus has the same phases as the Moon). [17]

Another notable scientist to time-stamp his discoveries in a similar manner was Robert Hooke, who published two anagrams in 1676. The first of these anagrams he revealed in 1678 to be what we now know as the Hooke's law of elasticity and the second was shown in 1705, after Hooke's death, to describe the optimal shape of uniform freestanding arches. [23, 24, 25]

While this approach, when used with modern cryptographic hash functions, would allow the verification of time-stamp tokens to be independent of trust in any one third party, it would be rather expensive to publish a separate hash value for each individual document.

1.2 Hash-Linking

In 1990, Haber & Stornetta [20, 21] proposed a time-stamping scheme where each token issued by a TSA would include some information from the immediately preceding one and a reference to the immediately succeeding one. These back and forward references would link all the tokens ever issued by the TSA into a linearly ordered list.

To avoid the linking information growing larger in each successive token, and also to prevent anybody (including the TSA itself) from retroactively inserting additional tokens into the list, part of the information from the preceding token would be hashed using a cryptographic hash function before being included in the current one.

Then anyone wanting to establish that a time-stamp token is authentic and has not been issued falsely in a collusion between the TSA and its client could follow the chain in both directions. The verifier could contact owners of the older and newer tokens and check that the linking information in each pair of adjacent tokens matches and the time values in them agree with the claimed order of issuance.

Depending on how suspicious the verifier is, they could continue following the chain as far as needed in both directions, in order to satisfy themselves that so many parties colluding to fake a time-stamp token is too unlikely.

The main drawback of this scheme is that verification depends on all tokens being available to the verifier. To alleviate this, Haber & Stornetta also proposed a generalized version where, at the expense of k-fold increase of the size of the linking information, the verification would still work if among any k consecutive tokens at least one would be available to the verifier (k is a parameter chosen by the TSA at time-stamping time).

In 1991, Benaloh & de Mare [5] proposed to increase the efficiency of hash-linked time-stamping by having the time-stamping system operate in fixed-length rounds. The messages to be time-stamped within one round would be combined into a hierarchical structure from which a compact proof of participation could be extracted for each message. The aggregation structures would then be linked into a linear chain.

Bayer *et al* [4] described a similar idea and additionally suggested that the linear chain linking the aggregation structures could be replaced with another hierarchy to enable more efficient comparison of tokens from rounds distant in time. However, they did not elaborate on how the hierarchy should be maintained as new aggregation rounds would be added to it over time.

Both of these proposals traded efficiency for accuracy, as messages submitted wihin one round would be considered time-stamped simultaneously and could not be ordered in time.

From 1998 to 2000, Buldas *et al* [11, 10, 13] proposed a series of time-stamping schemes based on binary linking that allowed any two tokens to be ordered in time, even if they were issued within the same round.

1.3 Time-Stamping Standards

For time-stamping to be widely usable in practice, the time-stamp token formats, protocols to communicate to the time-stamping services, and verification procedures need to be standardized. In chapter 2, we review existing time-stamping related standards.

Standards from the Internet Engineering Task Force (IETF), the European Telecommunications Standards Institute (ETSI), the International Organization for Standardization (ISO), and the American National Standards Institute (ANSI) are included in the review. While there is no doubt that there are many local or specialized standards that have been left out, we believe the coverage to be comprehensive as far as general-purpose standards from internationally recognized bodies are concerned.

In reviewing the standards, we concentrate on technical specifications for time-stamp token formats and generally neglect the requirements placed on the equipment and operating procedures of a time-stamping service provider. Stapleton [42] has published an overview and comparison of the IETF, ISO, and ANSI standards from that perspective.

1.4 Time-Stamping Systems

In chapters 3 and 4, we review existing time-stamping services that issue hash-linking based tokens, organized by the underlying authenticated data structures.

In chapter 3, the binary tree based systems of Surety and GuardTime are covered. These two are commercial services available to the general public.

In chapter 4, we study the skip list based CHRONOS, a research prototype of the University of Pau (France). While the prototype is not a generally available service, we found the concept to be interesting enough to warrant its inclusion in the report.

We deliberately ignore providers of independently signed tokens, even though this category still encompasses the majority of publicly available services (both commercial and non-commercial) available in the world today.

We also leave out services such as the BALTICTIME [37] and the OCSP responder of Sertifitseerimiskeskus [41] that only use hash-linking in their internal audit logs. As the audit logs are not accessible to the end users, we believe they should not be considered to add full "widely-witnessed" quality to the service, even if control certificates are published for the benefit of accredited auditors.

In the systems we do review, we consider the standards-compatibility on two levels. First we investigate if the linking mechanism used (more precisely, the hash links extracted from the underlying authenticated data structure) could be embedded in the data structures designated for that purpose by the standards. Then we check if the messages exchanged and tokens issued by the services are actually formatted in accordance with the standards.

Chapter 2

Time-Stamping Standards

In the following sections we will review the major standards related to time-stamping, with focus on their support for producing hash-linked tokens.

2.1 ASN.1

All of the time-stamping standards reviewed in the following sections employ Abstract Syntax Notation One (ASN.1) to define their message formats.

ASN.1 consists of a data definition language to describe data structures and several sets of encoding rules to specify encoding and decoding of populated instances of those structures as bit strings. It is a joint standard published simultaneously by the Telecommunication Standardization Sector (ITU-T) of the International Telecommunication Union (ITU) and the Joint Technical Committee One (JTC1) of the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC).

Below we try to summarize the bare minimum needed to comprehend the material cited from the time-stamping standards in the following sections. To develop a working knowledge of the notation, at least the material of the parts of the standard that cover the basic notation (ITU-T X.680 [33], ISO/IEC 8824-1 [28]) and the basic and distinguished encoding rules (ITU-T X.690 [34], ISO/IEC 8825-1 [29]) has to be absorbed.

In the most general level, the ASN.1 notation includes a set of basic data

types and a set of constructs to combine them.

Most of the basic data types, such as `INTEGER` and `BOOLEAN`, behave quite as would be expected by anyone with any programming experience.

The `OCTET STRING` and `ANY` types can be used for keeping any data, with the principal difference between them being that an ASN.1 data parser should in general not expect to be able to interpret the contents of an `OCTET STRING`, but may assume that a value of type `ANY` embeds another ASN.1 object (whose internal structure is not necessarily known in advance).

The `OBJECT IDENTIFIER` type represents a globally unique identifier. The identifiers are managed in hierarchical manner. An organization is assigned a prefix and gets the responsibility to manage allocation of all identifiers with that prefix and often further delegates different branches to different units. For example, the prefix 2.16.840 is assigned to the USA, which designated 2.16.840.1.101 for the government, which in turn assigned 2.16.840.1.101.3.4 to NIST for algorithm identifiers, which in turn defined the identifier of the SHA2-256 hash function to be 2.16.840.1.101.3.4.2.1.

The `SEQUENCE` models an ordered collection of variables that may in general be of any ASN.1 type. For example, the ASN.1 snippet

```
s :: SEQUENCE {
  i INTEGER,
  b BOOLEAN
}
```

defines a structure `s` consisting of two fields, an integer `i` and a boolean `b`.

A field in a structure may be designated as `OPTIONAL`, which means the value may be absent in some instances of the structure; the `DEFAULT` keyword also declares a field optional, but additionally specifies a default value to be assumed in case the field is absent.

The `SEQUENCE OF` models an ordered and `SET OF` an unordered collection of variables that are all of the same type.

For encoding data under either the basic or distinguished rules, each of the data types and constructs is assigned a tag and a value is encoded as the type tag, followed by the length of the value, followed by the value itself.

For a non-basic data type, the value consists of the concatenated encodings

of the members. Note that the field names are not included in the encoded form, which may present a problem in parsing structures with several optional members. For example, given the type

```
t1 :: SEQUENCE {
  a INTEGER DEFAULT 1,
  b INTEGER DEFAULT 2
}
```

and the value `{ 0 }`, it is not possible to determine if the value `0` is intended for the field `a` (and the field `b` should take the default value `2`) or for the field `b` (and the field `a` should take the default value `1`).

To resolve such ambiguities, it is possible to tag members of a `SEQUENCE`. For example, given the type

```
t2 :: SEQUENCE {
  a [1] INTEGER DEFAULT 1,
  b [2] INTEGER DEFAULT 2
}
```

and the value `{ [2] 0 }`, it is now clear that the value `0`, having been tagged with `2`, is intended for the field `b` and the field `a` is the one that should assume its default value `1`.

The `EXPLICIT` and `IMPLICIT` designations can be safely ignored by all readers who do not intend to manually encode and decode ASN.1 data.

The basic encoding rules (BER) allow for more than one possible encoding for some data elements. Since this is not acceptable in some situations (in particular for time-stamped or digitally signed material), the distinguished encoding rules (DER) remove the ambiguities.

2.2 IETF

The Internet Engineering Task Force (IETF) is an international community of network designers, operators, vendors, and researchers concerned with the evolution of the Internet architecture and the smooth operation of the

Internet. It is open to any interested individual. While many participants engage in IETF activities as part of their work, the IETF officially always views them as individuals, not as company representatives.

The IETF's products are documents published as Requests For Comments (RFCs). The RFCs relevant to time-stamping have been produced by the Network Working Group. As the name might suggest, not all RFCs are standards, some are just informational.

2.2.1 RFC 3161

The IETF time-stamping specification was published as RFC 3161: Internet X.509 Public Key Infrastructure Time-Stamp Protocol [1] and is based on the CMS syntax published as RFC 2630: Cryptographic Message Syntax [26]. The official status of both of them is "proposed standard".

The RFC 3161 only defines a mechanism for producing individually signed independent time-stamp tokens. While this may seem to make it irrelevant for the current work dedicated to hash-linked tokens, it is not so. All the standards described in the following sections re-use the basic structures defined in the RFC 3161.

2.2.2 Protocol

The basic interaction protocol defined in the RFC 3161 is very simple: a client sends to the service a time-stamping request containing a hash value of the datum to be time-stamped and the service returns a response.

The RFC 3161 defines the syntax of a time-stamping request to be:

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

The field `version` contains the version number of the request syntax. The current version is `1`.

The field `messageImprint` contains a digest of the datum to be stamped, together with the identifier of the hash algorithm used to produce the digest value:

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

The field `reqPolicy`, if present, contains the object identifier of the time-stamping policy under which the token should be provided. The policy is a statement from the TSA regarding the terms of service. The informational RFC 3628 (see section 2.2.4) discusses the issues that a TSA should address in its policy.

```
TSAPolicyId ::= OBJECT IDENTIFIER
```

The field `nonce`, if present, contains a presumably freshly generated random value that enables the client to check that the token returned by the TSA was in fact generated in response to this particular request.

If the field `certReq` is present and contains `TRUE`, the returned time-stamp token must include the certificate for the key used by the TSA to sign it.

The field `extensions` is a placeholder for a generic way to add additional information. The RFC 3161 does not define any extensions.

The RFC 3161 defines the syntax of a time-stamping response to be:

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

The field `status` is used to indicate the success or failure in processing the time-stamping request and in case of failure to indicate the cause.

In case of success, the field `timeStampToken` contains the freshly issued time-stamp token. The inner structure of this field is defined in section 2.2.3.

The RFC 3161, with reference to the RFC 2510: Internet X.509 Public Key Infrastructure Certificate Management Protocols [2], defines the status info to have the following structure:

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

PKIStatus ::= INTEGER
```

The field `status` indicates the success or failure in processing the request. A value `0` or `1` means success and in this case the response must contain a time-stamp token. Any other value indicates a failure and in this case the response must not contain a time-stamp token.

The field `statusString` is a free text message explaining the cause of the failure. The field `failInfo` is a bit-field indicating the cause(s) of the failure.

The RFC 3161 and RFC 2510 contain detailed explanations of the possible values of the `status` and `failInfo` fields.

2.2.3 Token Format

The RFC 3161 defines a time-stamp token to be a CMS message and the RFC 2630 defines a CMS message to have the following structure:

```
TimeStampToken ::= ContentInfo

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

ContentType ::= OBJECT IDENTIFIER
```

The `contentType` field determines the expected syntax and interpretation of the `content` field. The RFC 3161 defines time-stamp tokens to use the `SignedData` type and the RFC 2630 defines the content of a `SignedData` 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
```

The field `version` indicates the version number of the syntax of the structure, which the RFC 2630 defines to be `3` for `SignedData` messages.

The field `digestAlgorithms` is a collection of message digest algorithm identifiers, with the intention that digesting the signed data with the listed algorithms yields the digests needed to verify the signatures in the `signerInfos` collection. Since the RFC 3161 allows for only one signature in a time-stamp token, it is expected that any token has just one item in this collection, but this is not an actual requirement.

The field `encapContentInfo` embeds the signed content, consisting again of a content type identifier and the content itself.

The field `certificates` is a collection of certificates, with the intention that the set will be sufficient to contain chains from recognized certification authorities to all of the signers in the `signerInfos` field, although the set is allowed to be either incomplete or redundant, or both.

The field `crls` is a collection of certificate revocation lists (CRLs), with the intention that the set will be sufficient to establish the validity of the certificates given in the `certificates` field; again, the set is allowed to be either incomplete or redundant, or both.

The field `signerInfos` is defined in the RFC 2630 to be a collection of per-signer information items, each containing the signature value and any signer-specific data needed to verify the signature. The RFC 3161 restricts this for time-stamp tokens and requires the collection to contain just the signature of the TSA that issued the token.

The RFC 2630 defines the encapsulated content of a `SignedData` message to have the following structure:

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

ContentType ::= OBJECT IDENTIFIER
```

As in `ContentInfo`, also in `EncapsulatedContentInfo`, the `eContentType` field determines the syntax and interpretation of the `eContent` field.

The RFC 3161 defines the encapsulated content to be of type `TSTInfo`:

```
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
```

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

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

The field `version` indicates the version number of the syntax of the structure. Only version 1 is currently defined.

The field `policy` contains the identifier of the policy under which the time-stamp token was issued by the TSA.

The field `messageImprint` contains the digest of the time-stamped datum. Obviously, the contents of this field must be the same as the corresponding field in the time-stamping request for which the token was generated.

The field `serialNumber` contains an integer assigned to the token by the TSA; the RFC 3161 requires that the serial number combined with the name of the TSA must be unique, and consequently requires clients to be able to handle values up to 160 bits long.

The field `genTime` contains the time at which the token was created by the TSA. It is expressed as Coordinated Universal Time (UTC) using the syntax `YYYYMMDDhhmmss[.s...]Z`, where the terminating `Z` indicates that the time is given in UTC.

The field `accuracy` defines the time deviation around the time contained in `genTime`. Subtracting the accuracy from and adding it to the value contained in `genTime` gives the lower and upper bounds to the actual time of creation of the token by the TSA. The value zero must be assumed for any missing sub-fields within this structure. If the whole structure is missing, the accuracy of the time can still be available through other means (for example, it may be specified by the time-stamping policy).

If the field `ordering` is present and contains the value `TRUE`, any two tokens from the same TSA can be ordered based on the values in their `genTime` fields. Otherwise two tokens 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.

The field `nonce` must contain the value from the corresponding field in the request, or be missing if the field was absent in the request.

The field `tsa` contains the name of the TSA that issued the token.

The field `extensions` is a placeholder location for a generic way to add additional information to the time-stamp tokens. The RFC 3161 does not define any extensions.

The 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

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
```

The field `version` contains the version number of syntax of the `SignerInfo` structure. Currently versions `1` and `3` are defined, and their use depends on the way the signer is identified in the `SignerIdentifier` structure.

The field `signerIdentifier` specifies the signer's public key certificate to be used for verifying the signature.

The field `digestAlgorithm` identifies the message digest algorithm used by the signer. The digest is computed on the DER-encoded form of the `EncapsulatedContent` structure and stored in the message-digest attribute in the `signedAttrs` collection.

The field `signedAttrs` is a collection of type/value pairs of the attributes whose values are protected by the signature. The RFC 2630 specifies that this field is mandatory for the `SignedData` type and at least the `content-type` and `message-digest` attributes must be present. The RFC 3161 further specifies that the `ESSCertID` attribute must be present.

The field `signatureAlgorithm` identifies the algorithm used by the signer to generate the signature and the field `signature` contains the signature value.

The field `unsignedAttrs` is a collection of attributes that are not signed.

Figure 2.1 illustrates how the different hashing and signing steps are layered to have the time-stamped data and the time value protected by the signature in the time-stamp token.

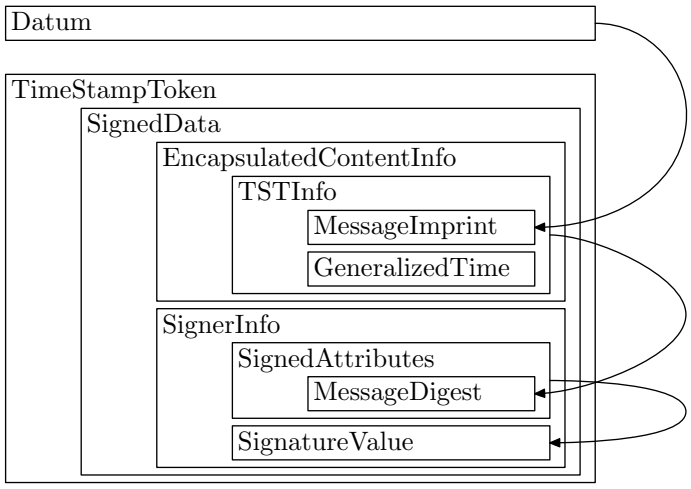

Figure 2.1: Data dependencies in an RFC 3161 time-stamp token.

2.2.4 RFC 3628

The RFC 3628: Policy Requirements for Time-Stamping Authorities [39] defines a recommended baseline time-stamping policy. The official status of this RFC is "informational".

The initial version of the baseline policy was developed in cooperation between the IETF and the ETSI and the technical content of the RFC 3628 is identical to the first edition of the ETSI TS 102 023 (see section 2.3.2).

2.3 ETSI

The European Telecommunications Standards Institute (ETSI) is a non-profit organization that produces standards for information and communication technologies. ETSI's purpose is to produce and perform the maintenance of the technical standards and other deliverables as required by its member organizations. The membership is not limited to European organizations.

Time-stamping related work is done in the Electronic Signatures and Infrastructures Technical Committee (ESI).

2.3.1 TS 101 861

The TS 101 861: Time Stamping Profile [15] is an application profile on top of the IETF RFC 3161 (see section 2.2.1). The profile specifies the minimal list of algorithms and optional fields that implementations must support, etc. It does not introduce any technological innovation over the RFC 3161 and is mentioned here only for the completeness of coverage.

2.3.2 TS 102 023

The TS 102 023: Policy Requirements for Time-Stamping Authorities [16] defines a recommended baseline time-stamping policy.

The initial version of the baseline policy was developed in cooperation between the ETSI and the IETF and the technical content of the first edition of the TS 102 023 was identical to the IETF RFC 3628 (see section 2.2.4).

2.4 ISO/IEC

The International Organization for Standardization (ISO) is a network of national standards institutes, with membership limited to one organization per country, the one "most representative of standardization in its country".

The time-stamping related work is done in the Joint Technical Committee One (JTC1) that the ISO formed together with the International Electrotechnical Commission (IEC), a global organization dedicated to preparing and publishing international standards for electrical, electronic and related technologies.

2.4.1 ISO/IEC 18014

The standard ISO/IEC 18014: Information Technology — Security Techniques — Time-Stamping Services is published in three parts.

The ISO/IEC 18014-1: Time-Stamping Services — Part 1: Framework [30] describes the general model of time-stamping and defines the basic protocols of interaction with the time-stamping service.

The ISO/IEC 18014-2: Time-Stamping Services — Part 2: Mechanisms Producing Independent Tokens [31] describes, as the title implies, the generation and verification of time-stamp tokens that are independent of each other and can be verified individually. The mechanism based on digital signatures is compatible with the RFC 3161 (see section 2.2.1). Two additional mechanisms are defined: one based on message authentication codes and the other based on archival of evidence by the TSA. Since we are not interested in either of these mechanisms, we will mostly ignore the Part 2 for the rest of our discussion.

The ISO/IEC 18014-3: Time-Stamping Services — Part 3: Mechanisms Producing Linked Tokens [32] describes the mechanisms for producing time-stamp tokens linked to each other to enhance the security of the tokens and reduce the level of trust that both the requestors and verifiers need to place in the TSA. It gives the general model for services of this type, defines the data structures and protocols used to interact with such a service, and also discusses some possible implementations.

2.4.2 Model

In the ISO/IEC 18014-3 model, a TSA producing linked time-stamp tokens is expected to use cryptographic hash functions to link each new token to other tokens previously generated by the TSA in a way that ensures that a false token cannot be inserted into the collection. The process includes an optional aggregation step, a linking step, and a publishing step.

A linking step consists of forming a verifiable binding between the new token and links produced by the TSA previously. The linking method must ensure that the most recently produced link provides a cryptographic summary of all time-stamp tokens that ever participated in the linking process.

A TSA may perform linking operations on aggregated groups rather than individual tokens. An aggregation step takes a group of events as inputs and produces a verifiable cryptographic link between each event and the rest of the group. The resulting aggregate value is then linked by the TSA in a way similar to the case of the single time-stamp token. The TSA should assign the same time-value to all tokens involved in an aggregation step.

The goal of the publishing step is to distribute the links in order to make them "widely witnessed" and thus generate independently verifiable statements of when the tokens were issued by the system. A common way is to publish the links in a newspaper or on a message board that is mirrored on many servers. The value published should at least depend on all tokens generated by the TSA since the previous publishing event.

2.4.3 Protocol

The ISO/IEC 18014 extends the protocol given by the RFC 3161 (see section 2.2.2) in two ways:

- In addition to the time-stamping request and response messages of the RFC 3161, the ISO/IEC 18014 also defines verification request and response messages to enable a client to request the information needed to link a particular time-stamp token to a value published by the TSA.
- The ISO/IEC 18014 defines three types of extensions that a client may include in the time-stamping requests.

Verification

The format of a verification request is as follows:

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

The field `version` contains the version number of the request syntax. The current version is `1`.

The field `tst` contains the time-stamp token that needs to be linked to a control publication.

The field `requestID`, if present, contains a value that enables the client to match the response returned by the TSA to the request.

The format of a verification response is as follows:

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

The field `version` contains the version number of the response syntax. The current version is `1`.

The field `status` is used to indicate the success or failure in processing the verification request and in case of failure to indicate the cause in the same manner as in the time-stamping protocol (see section 2.2.2).

The field `tst` contains the time-stamp token from the request, but updated with the linking info that connects it to a control publication.

The field `requestID` must contain the value from the corresponding field in the request, or be missing if the field was absent in the request.

Extensions

The general format of an extension is

```
Extension ::= SEQUENCE {
  extnId OBJECT IDENTIFIER,
  critical BOOLEAN DEFAULT FALSE,
  extnValue OCTET STRING
}
```

The `extnId` contains the type of the extension and the field `extnValue` a value whose interpretation depends on the type specified by `extnId`.

If the field `critical` is present and set to `TRUE`, the extension is deemed critical. Any party that receives a message containing an unknown critical extension must issue an error response and refuse to process the message. Unknown non-critical extensions should be ignored.

ExtMethod Extension

The function of the `ExtMethod` extension is to enable a client to request the TSA to use a specific method for generating a time-stamp token. The extension is submitted to the server in the `extensions` field of `TimeStampReq` and the server, in addition to using one of the requested methods for generating the time-stamp token, also copies the extension into the `extensions` field of `TSTInfo` of the generated token. The `extnValue` field of an `ExtMethod` extension embeds the following structure:

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

Method ::= OBJECT IDENTIFIER
```

ExtHash Extension

The function of the `ExtHash` extension is to enable a client to submit for time-stamping more than one hash value derived from the same datum. The extension is submitted to the server in the `extensions` field of `TimeStampReq`

and the server processes it by copying the extension into the `extensions` field of `TSTInfo` of the generated time-stamp token. The `extnValue` field of an `ExtHash` extension embeds the following structure:

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

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

ExtRenewal Extension

The function of the `ExtRenewal` extension is to enable a client to submit an existing time-stamp token to be time-stamped together with the hash value of a datum. The extension is submitted to the server in the `extensions` field of `TimeStampReq` and the server processes it by copying the extension into the `extensions` field of `TSTInfo`. The `extnValue` field of an `ExtRenewal` extension embeds the following structure:

```
ExtRenewal ::= TimeStampToken
```

Time-stamping a datum and an existing time-stamp token allows the client to insulate the existing token from the impending failure of any cryptographic function used by the token. The time value in the new token indicates that the old token existed before the cryptographic function failed and thus could not have been falsely generated by taking advantage of the weakness. Therefore, the time value in the old token can be relied on as the actual time when the datum was time-stamped.

2.4.4 Token Format

The ISO/IEC 18014-3 extends the time-stamp token format given by the RFC 3161 (see section 2.2.3) to allow two new time-stamp token generation methods, the “linked” and the “linked-and-signed” method.

Linked Token

A time-stamp token generated using the "linked" method is encapsulated in the `DigestedData` type instead of the `SignedData` type. The RFC 2630 [26] defines a `DigestedData` message to have the following structure:

```
DigestedData ::= SEQUENCE {
  version CMSVersion,
  digestAlgorithm DigestAlgorithmIdentifier,
  encapContentInfo EncapsulatedContentInfo,
  digest Digest
}

Digest ::= OCTET STRING
```

The meaning of the `version`, `digestAlgorithm`, and `encapContentInfo` fields is the same as for the `SignedData` case (see section 2.2.3).

The `digest` field should contain a DER-encoded `BindingInfo` structure (see section 2.4.5) that links the `TSTInfo` embedded in `encapContentInfo` to the other tokens produced by the TSA.

Figure 2.2 illustrates how the hashing and linking steps are layered to have the time-stamped data and the time value protected in the time-stamp token.

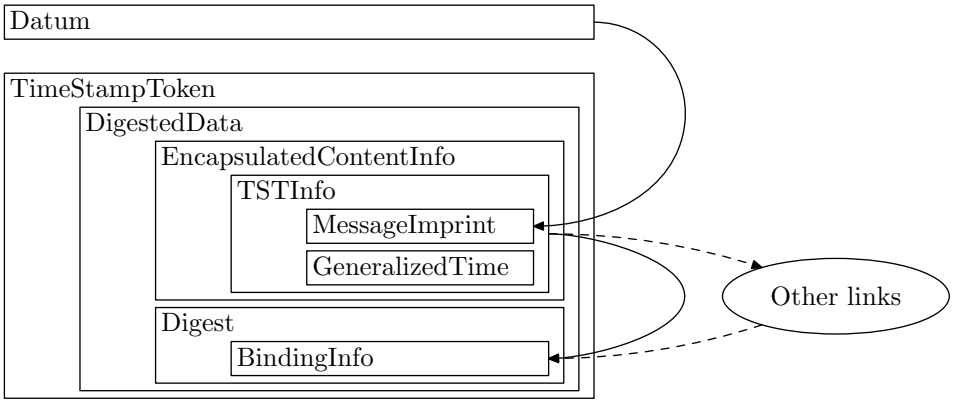

Figure 2.2: Data dependencies in a linked time-stamp token.

A time-stamp token generated using the "linked" method has to be verified based on the linking info, as explained in section 2.4.5.

Linked-and-Signed Token

A time-stamp token generated using the "linked-and-signed" method is encapsulated in the `SignedData` type (see section 2.2.3), with the `signedAttrs` collection containing an extra attribute.

The `tsp-signedData` attribute should contain a DER-encoded `BindingInfo` structure (see section 2.4.5) that links the `TSTInfo` structure embedded in `encapContentInfo` to the other tokens produced by the TSA.

Note that, since `tsp-signedData` is a signed attribute, the linking process has to be performed before the token is signed by the TSA.

Figure 2.3 illustrates how the hashing, linking, and signing steps are layered to have the time-stamped data and the time value protected in the time-stamp token.

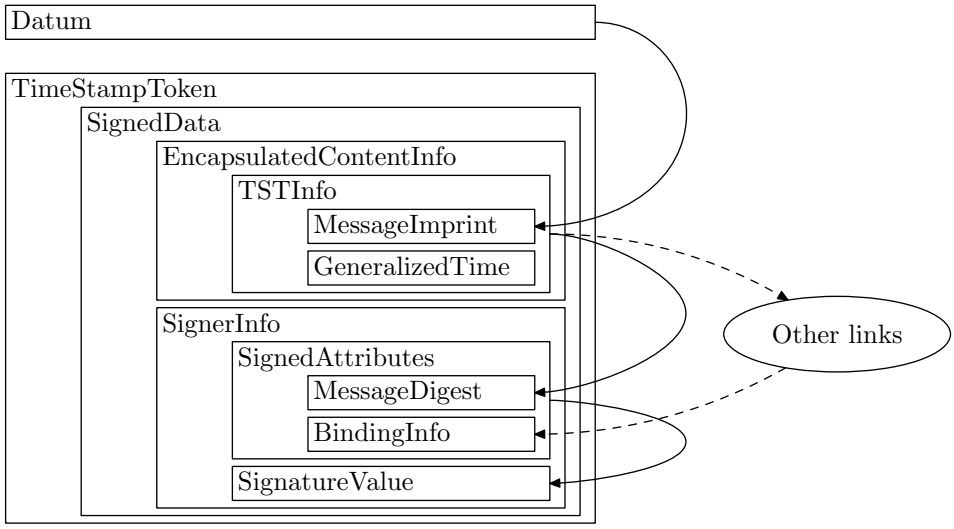

Figure 2.3: Data dependencies in a linked-and-signed time-stamp token.

A time-stamp token generated using the "linked-and-signed" method may be verified based either on the linking info, as explained in section 2.4.5, or the signature, at verifier's choice.

2.4.5 Linking Info Format

The linking information for both the "linked" and the "linked-and-signed" time-stamp tokens is represented in a structure that follows the general model described in section 2.4.2:

```
BindingInfo ::= SEQUENCE {
  version Version,
  msgImprints MessageImprints,
  aggregate [0] Chains OPTIONAL,
  links Links,
  publish [1] Chains OPTIONAL,
  extensions [2] Extensions OPTIONAL
}

MessageImprints ::= SEQUENCE SIZE (1..MAX) OF MessageImprint
```

The field `version` contains the version number of this data structure. The current version is `1`.

The field `msgImprints` contains the digest or digests computed on the encapsulated `TSTInfo` structure.

The field `aggregate` contains the results of aggregating the `msgImprints` field with other members of the aggregation. The field may contain more than one group of aggregation data when multi-tiered aggregation schemes are supported.

The field `links` contains the the results of linking operations (from prior time values) that were used as input to the linking operation for the time value in the encapsulated `TSTInfo` digested in `msgImprints`. It always includes the result of the linking operation for the immediately preceding time value, which represents the running summary of the cumulative linking operations so far. If the `aggregate` field is present, the contents of the `links` field are linked to the result of the computation of the `aggregate` field. If the `aggregate` field is absent, the contents of the `links` field are linked to the contents of the `msgImprints` field.

The field `publish`, if present, contains the results of aggregating the time-stamp token with publishing data.

The field `extensions` is a placeholder for a generic way to add additional information. Two types of extensions, `ExtName` and `ExtTime`, are defined in the ISO/IEC 18014.

A sequence of operations representing an aggregation or publishing process is contained in the `Chain` structure:

```
Chains ::= SEQUENCE SIZE (1..MAX) OF Chain
Chain ::= SEQUENCE {
  algorithm ChainAlgorithmIdentifier,
  links Links
}
```

The field `algorithm` contains the object identifier of the algorithm used to compute the chain of links and the field `links` contains the links that form the elements of the chain.

The ISO/IEC 18014 mentiones some algorithms that could be used, but does not specify any of them in sufficient detail for implementation, nor assign any identifiers. So, for now, this should be considered a framework that any actual provider has to fill in.

A single linking operation or a single step of an aggregation is represented by the `Link` data structure:

```
Links ::= SEQUENCE SIZE (1..MAX) OF Link
Link ::= SEQUENCE {
  algorithm [0] LinkAlgorithmIdentifier OPTIONAL,
  identifier [1] INTEGER OPTIONAL,
  members Nodes
}
```

The field `algorithm` contains the object identifier of the algorithm used to compute the link. The field may be absent if the algorithm used to compute the link is specified at a higher level, for example in the containing `Chain` structure.

The field `identifier` contains a local link identifier. This is used when the link is referred to (used as input) by other links.

The field `members` contains the data items or identifiers of data items to be linked in this step.

A single input element to a linking operation or to a step of an aggregation is represented by the `Node` structure:

```
Nodes ::= SEQUENCE SIZE (1..MAX) OF Node
Node ::= CHOICE {
  imprints [0] Imprints,
  reference [1] INTEGER
}

Imprints ::= SEQUENCE SIZE (1..MAX) OF Imprint
Imprint ::= OCTET STRING
```

If the `imprints` field is present, the node contains the actual data items to be used as input to the current operation.

If the `reference` field is present, the node identifies another `Link` structure whose result is to be used as input to the current operation; in this case the value of the `reference` field corresponds to the value of the `identifier` field of the `Link` structure whose results are to be included in the current operation.

The value `0` in the `reference` field indicates that the data item to be used as input is obtained from a source external to the local group of links being computed. One example of such source is the `msgImprints` field.

Verification

According to the ISO/IEC 18014-3, the verification of a `BindingInfo` structure is performed as follows:

- the verifier uses the verification protocol to communicate to the issuing TSA or a third-party service that has access to the issuing TSA's summary links;
- if there is no summary link value for the time value in the submitted time-stamp token's encapsulated `TSTInfo` structure, the verification fails;

- the encapsulated `TSTInfo` structure is checked against all hash values in the `msgImprints` in the `BindingInfo`; if there is a mismatch, the verification fails;
- if the `aggregate` field is present in the `BindingInfo` structure:
  - the aggregation algorithm is applied to the `msgImprints` and `aggregate` fields assuming that a `0 reference` value within a `Node` refers to the `msgImprints` field;
  - the linking algorithm is applied to the result and the `links` field assuming that a `0 reference` value within a `Node` refers to the previous result;
- if the `aggregate` field is absent:
  - the linking algorithm is applied to the `msgImprints` and `links` fields assuming that a `0 reference` value within a `Node` refers to the `msgImprints` field;
- the result of the linking algorithm is compared to the archived summary link for the time value in the time-stamp token's encapsulated `TSTInfo` structure.[1]

ExtName Extension

The function of the `ExtName` extension is to enable the TSA to assign, for auditing or record-keeping purposes, names to the steps of the time-stamping process. The extension may be added to `extensions` field of `BindingInfo`. The `extnValue` field of an `ExtName` extension embeds a `GeneralName` structure, as defined in the RFC 2459 [27].

ExtTime Extension

The function of the `ExtName` extension is to enable the TSA to record, for auditing or record-keeping purposes, times of performing the steps of the time-stamping process. The extension may be added to `extensions` field

[1] Curiously, the field `publish` of `BindingInfo` is not referenced at all.

of `BindingInfo`. The `extnValue` field of an `ExtName` extension embeds a `GeneralizedTime` value.

2.5 ANSI

The American National Standards Institute (ANSI) coordinates the development and use of voluntary consensus standards in the United States and represents United States in the ISO, the IEC, etc. The ANSI itself does not develop standards; rather it accredits other organizations as standards developers.

The time-stamping related work has been done by the Accredited Standards Committee X9 (ASC X9), accredited by ANSI to develop, establish, maintain, and promote standards for the financial services industry.

2.5.1 ANS X9.95

From our viewpoint, the ANS X9.95: Trusted Time Stamp Management and Security [3] is very similar to the ISO/IEC 18014 (see section 2.4.1).

Compared to the ISO/IEC 18014, the ANS X9.95 removes the time-stamping scheme based on archival of evidence by the TSA and adds one based on signing the time-stamp tokens with extremely short-lived keys. Neither of the changes are relevant to generating hash-linked time-stamp tokens.

The ANS X9.95 is much more detailed in the procedural requirements that the TSA is expected to follow to maintain the accuracy of its clocks and the security of its systems, but these changes are also outside the scope of the current work.

2.5.2 Formats

In addition to the ASN.1 syntax used exclusively for all protocol messsages and tokens by other standards we have surveyed, the ANS X9.95 adds the possibility to encode all these items in XML. However, no system we know of actually uses the XML based syntax, so we abstain from citing the format specification here.

The ASN.1 based formats in ANS X9.95, including the standard extensions, both for the time-stamping requests and the linking information, are the same as in the ISO/IEC 18014.

The only significant change for hash-linked time-stamp token formats is the definition of a Merkle tree based aggregation algorithm.

Verification

Compared to the ISO/IEC 18014, the ANS X9.95 corrects the omission in the specification of the verification procedure and specifies that, after performing the aggregation and linking steps of the hash chains, also the publishing steps have to be executed before arriving at a final result that can be compared to a control publication distributed by the TSA.

Chapter 3

Binary Tree Based Linking

Binary trees are ubiquitous in computer science, and therefore it should not surprise anyone to encounter them in time-stamping as well. In the following sections we will study systems where the tokens are linked into binary tree based structures.

3.1 Merkle Trees

A Merkle tree (figure 3.1, left) is a binary tree where leaf nodes are populated with some hash values and each non-leaf node contains the hash value of the concatenation of its child nodes. The concept was introduced by Merkle [35, 36] in order to enable efficient signing of multiple documents with a single digital signature.

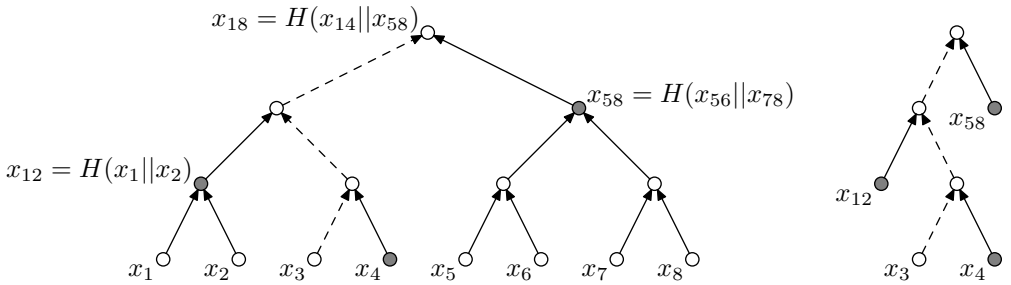

Figure 3.1: A Merkle tree (left) and the hash chain for x_3 (right).

Having a signature on the hash value in the root node of the tree, a chain of about $\log N$ hashing steps (figure 3.1, right) can be extracted for each one of the N leaf nodes to prove independently from all other values in the tree that the hash value in that particular leaf indeed participated in the computation that produced the signed root hash value.

For example, to show that the value x_3 participated, the claimant needs to additionally produce the three pairs (x_4, R), (x_{12}, L), (x_{58}, R), where the second member of each pair indicates the order in which the "incoming" hash value and the first member of the pair should be concatenated. This is enough to re-compute $x_{34} = H(x_3||x_4)$, $x_{14} = H(x_{12}||x_{34})$, $x_{18} = H(x_{14}||x_{58})$.

A relying party can then perform the computation and compare the final result with the originally signed root hash value. If the hash function H used is cryptographically secure, then a match of the two values is a strong indication that the value x_3 submitted by the claimant is indeed the same as what was in the tree at the time the root value was signed.

Bayer *et al* [4] were the first to explicitly propose using Merkle trees to increase the efficiency of hash-linking based time-stamping schemes. The aggregation method by Benaloh & de Mare [5] is specified in less detail, but the idea is essentially the same.

Buldas *et al* [14, 12, 8, 9] have extensively researched theoretical security bounds of such schemes in the context of time-stamping.

3.2 Surety

Surety is an information assurance software and services company founded in 1994 by Bellcore scientists, including Stuart Haber and W. Scott Stornetta who pioneered the concept of hash-linking based timestamping. The company currently has offices in the USA and Korea.

We didn't succeed in locating an official technical reference for the system, so some of the following descriptions are based on secondary sources.

3.2.1 Architecture

According to a 1997 paper by Haber & Stornetta [22], the initial version of Surety's time-stamping system (figure 3.2) used Merkle trees to aggregate requests received in time-stamping rounds and a linear hash chain to link the aggregate hashes of the rounds. There is no mention of publishing control certificates or enabling users of the service to access the hash chain.

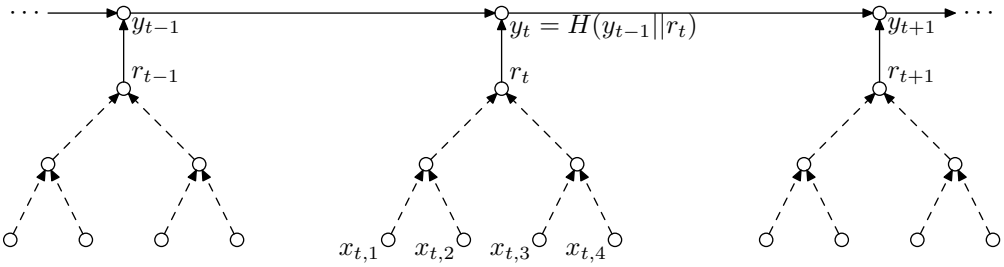

Figure 3.2: Surety's old system with Merkle aggregation and linear linking. In round t, the requests $x_{t,i}$ are aggregated into a Merkle tree (dashed arrows). The root hash r_t is then linked into a linear hash chain (solid arrows).

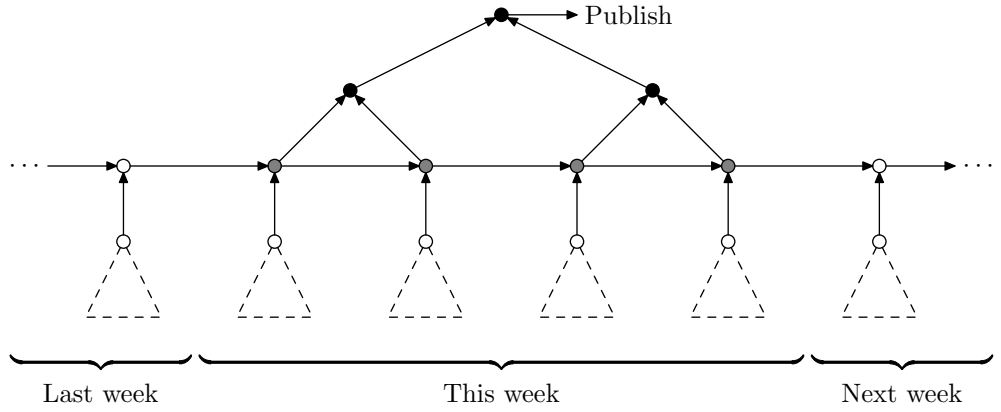

Figure 3.3: Surety's new system with Merkle aggregation, linear linking, and Merkle publishing. This week's new linking hashes (gray nodes) are aggregated into a Merkle tree (black nodes) and the root of the tree is published.

According to a 2005 technical report from GMU [44] and Surety's own 2008 whitepaper [43], they have added another layer of Merkle trees to aggregate the linked hashes for weekly publishing (figure 3.3).

The language in the control publication (figure 3.4) and analysis of sample tokens seems to indicate that only the new linking hashes added since the last publication are included in each subsequent one.


NOTICES & LOST AND FOUND
(5100-5102)
Universal Registry Entries:
Zone 2 -
dS8492cgVOFAoP9kyE1XzMOrQ
HgEwzkVbVafNvIkUz99qvq8/ME
p5y9EFSG8XxzMBaIGQQ==
Zone 3 -
JnFCg+HCmvhj8GmmUP7VZna71
NgZup/RfuKUQNzCHWXMuqLK
durxHQV5pSHLqBGPRiy+mg==
These base64-encoded values represent the combined fingerprints of all digital records notarized by Surety between 2009-06-03Z 2009-06-09Z.
www.surety.com 571-748-5800

Figure 3.4: Surety's control publication in the New York Times.

This approach to aggregating, linking, and publishing seems to fit very well into the general model shared by the ISO/IEC 18014 and the ANS X9.95.

3.2.2 Formats

Due to lack of documentation, we can't judge the standards-compliance of the time-stamping and verification requests and responses of Surety's service. We did, however, manually parse and annotate a sample time-stamp token and discovered no deviations from the ANS X9.95 specification.

Surety's whitepaper [43] claims ANS X9.95 and ISO/IEC 18014-3 compliance and in the light of the above we have quite no grounds to contest that claim, aside from the fact that the Merkle tree based aggregation algorithm, while standard in the ANS X9.95, looks like a non-standard extension relative to the ISO/IEC 18014. But as there are no algorithms defined in the ISO/IEC 18014, any working implementation would be an extension.

Since Surety's tokens use the `DigestedData` encapsulation, they are definitely not RFC 3161 compatible.

3.3 GuardTime

GuardTime is a data integrity service provider founded in 2006 by cryptographers Märt Saarepera and Ahto Buldas. The company currently has offices in Estonia, Singapore, and Japan.

The descriptions that follow are based on public specifications released by the company.

3.3.1 Architecture

According to the specification [18], GuardTime's time-stamping system uses Merkle trees to aggregate requests received in one-second time-stamping rounds and one constantly growing Merkle tree (the *calendar tree*) to link the aggregate values of the rounds (figure 3.5). The hash value currently in the root of the calendar tree is published monthly as a control certificate that encompasses all tokens issued by the service until that time.

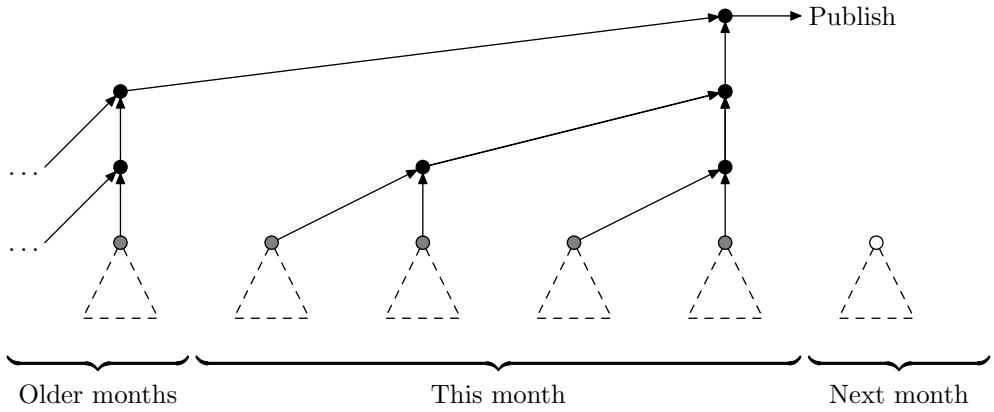

Figure 3.5: GuardTime's system with Merkle aggregation and combined Merkle linking/pubishing. This months's new aggregate hashes together with the full past (gray nodes) are linked into a Merkle tree (black nodes) and the root of the tree is published.

Since the calendar tree is updated continuously and the hash chains extracted from it are included in each time-stamp token issued, it could be viewed as the linking mechanism in the general model shared by the ISO/IEC 18014 and the ANS X9.95.

On the other hand, a snapshot of the same calendar tree is also used to perform the publishing operation from the general model. While this observation would make it conceivable to embed the hash chain extracted from the calendar tree in the `publish` field of the `BindingInfo` structure, we would then face the problem that there is nothing left to be put into the mandatory `links` field. The field could also not be populated by a single-step identity function as it is required to include a link summarizing all rounds preceding the current one.

So, in the end it seems that, if GuardTime were to use the `BindingInfo` structure in their time-stamp tokens, the hash chains extracted from the calendar tree should go to the `links` field and only the information regarding the control publication should be put into the `publish` field. In reality they use a quite different encapsulation for their hash chains, though.

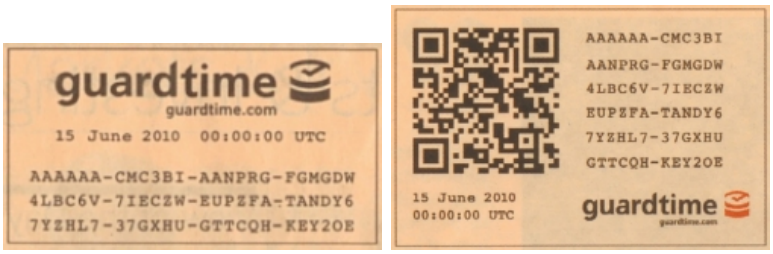

Figure 3.6: GuardTime's control publications in the Financial Times (left) and Äripäev (right).

An interesting property arises from the fixed length of the aggregation rounds and the fact that the merging order in building the calendar tree from the aggregation round summaries is deterministic. The shape of the hash chain (that is, the information whether the sibling hash is on the left or on the right in each hashing step) uniquely determines the position of the leaf relative to the root of the tree. Since it is known when the root was extracted from the calendar tree, the shape of the hash chain effectively encodes the time value for the aggregation round.

However, taking advantage of that information would certainly be a non-standard extension, even if the hash chains were encoded into otherwise compatible `BindingInfo` structure.

3.3.2 Formats

According to the format specification [19]:

- the time-stamping request and response messages used by GuardTime are compatible with the RFC 3161 specification (and therefore also with the ISO/IEC 18014 and the ANS X9.95);
- the verification request and response messages are completely non-standard;
- the time-stamp token format is based on using the `SignedData` encapsulation and treating the hash chains as a signature produced using a non-standard signing algorithm.

We did also parse and annotate a sample time-stamp token and discovered two minor deviations from the RFC 3161, both related to the closed-system PKI keys used to temporarily authenticate the calendar tree hash chains while there is no control publication available yet:

- the RFC 3161 requires the `ESSCertID` attribute to be present in the `signedAttrs` field of the `SignerInfo` structure, but it is absent in the GuardTime tokens;
- the RFC 3161 requires the `tsa` field, if present in the `TSTInfo` structure, to contain one of the subject names included in the certificate that is to be used to verify the token, but in a GuardTime token the field instead contains the hostname of the gateway that issued the token.

For both issues, it could be argued that the RFC 3161 requirements technically do not apply because the PKI key is used to sign the root hash value from the calendar tree, rather than the `signedAttrs` structure, as was the intention when the RFC requirements were drafted.

Perhaps more importantly, in a time-stamp token from GuardTime, unlike the general RFC 3161 case, the PKI signature is a temporary authentication device. Therefore it is sensible to avoid polluting permanent components of the token with references to a temporary helper object.

Of course, the main practical issue is that a generic RFC 3161-compliant client is unable to parse and verify the GuardTime-specific `TimeSignature` structure.

Since the GuardTime solution presents the hash chains as a signature scheme rather than a hash-linking scheme, the requirements for "linked" and "linked-and-signed" tokens in the ISO/IEC 18014 and the ANS X9.95 sense do not apply. The compatibility concerns when viewing the tokens as "signed" are the same as relative to the RFC 3161.

Chapter 4

Skip List Based Linking

In the following sections we will study a data structure called skip list and a time-stamping system where the tokens are linked into a skip list based authenticated dictionary.

4.1 Skip Lists

A skip list (figure 4.1) is a linked list augmented with extra pointers to allow the list to be traversed faster by skipping some entries (hence the name). Each entry is assigned a level, a level k entry has pointers for levels $0 \ldots k$, and a level i pointer points to the next entry whose level is i or higher.

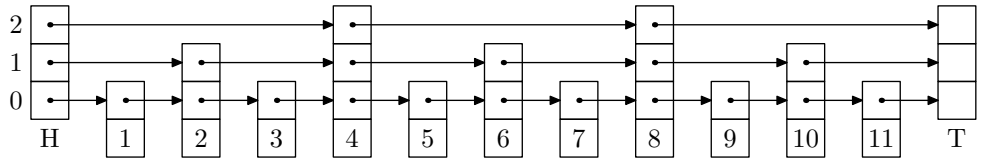

Figure 4.1: A skip list. H is head, T is tail, $1 \ldots 11$ are data, $0 \ldots 2$ are levels.

Skip lists were introduced in 1990 by Pugh [40] as an alternative to binary search trees. In Pugh's initial design the assignment of levels to entries was randomized, which gave $O(\log n)$ time searches, insertions, and deletions in an n-entry list on average, but could sometimes degrade to much worse (similarly to QuickSort, the degenerate cases are very rare and can mostly

be ignored in practice). Nevertheless, in 1992 Munro *et al* [38] proposed several deterministic versions that have guaranteed $O(\log n)$ time searches, insertions, and deletions.

The skip list shown on figure 4.1 is a *perfect* skip list: each pointer on level i points exactly 2^i entries forward. Skip lists where insertions and deletions may occur at arbitrary positions can't be kept perfect efficiently, but append-only skip lists can.

For the insertions to always go to the end of the list, new entries have to arrive in non-decreasing order of keys. While a data structure that requires entries to be added in the order of keys is not very useful in general, it is just perfect for time-stamping.

4.2 CHRONOS

CHRONOS, unlike the other systems reviewed in this work, is not a publicly available commercial service. It is a research prototype built in the University of Pau based on design by Kaouthar Blibech and Alban Gabillon.

There is no technical specification available, the following discussion is based on the academic papers of Blibech & Gabillon [6, 7].

4.2.1 Architecture

CHRONOS uses skip list based aggregation (figure 4.2) to ensure provable total order of requests within each round. What was considered a single entry (with multiple pointers) in the generic skip list is now viewed as multiple nodes linked to each other via hashing. Among the nodes representing one entry, the highest one is called a *plateau node*.

An aggregation round builds the skip list as follows:

1. The head of the list (column H on the figure) is seeded with the last round's summary hash (node A on the figure).

2. As new entries are added to the list, each new node is computed by hashing together one or two data items. Data always flows from the re-

quest upwards to the higher-level nodes corresponding to that request; data flows to the right only from the plateau nodes.

3. When a new request is added to the list, its index and *head proof* are returned to the requestor at once. The head proof contains most recent items from every level, except non-plateau nodes are excluded. The rationale is that the head proof is the minimal set of nodes that summarizes the state of the half-constructed skip list for that moment. For example, for request 6 on the figure, only the nodes I and J are needed; there is no entry for level 1 in the head proof.

4. When the round is finished accepting requests, the tail of the list (column T) is computed and the value in the highest level of the tail (node Z on the figure) becomes the summary hash for this round, which is published as a control certificate.

5. Then a second response is sent to each requestor, containing the summary hash and the *tail proof* of the request. The tail proof contains all the extra nodes that are needed to re-compute the summary hash from the request. For request 6, the nodes K and L are the tail proof.

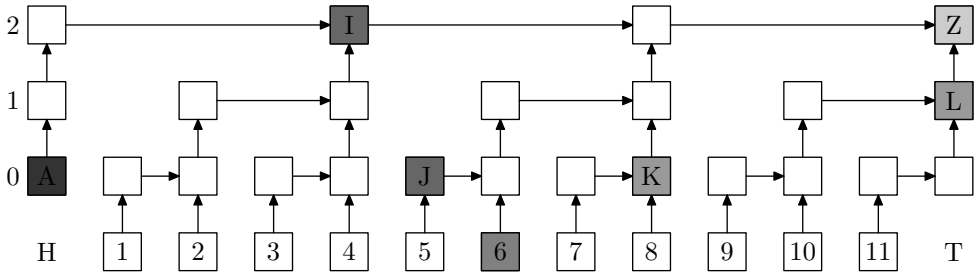

Figure 4.2: CHRONOS's aggregation graph for a round with 11 requests. Arrows show data flow. Node A is seeded with last round's summary hash. Node Z will contain this round's summary hash. Nodes I and J are the head proof and nodes K and L the tail proof for the request 6.

As can be seen from above, the aggregation rounds are linked linearly in CHRONOS, and each round is published. Because the requests are totally ordered within rounds, the service can afford to make the rounds so long that it is feasible to publish all round summaries without any further aggregation for publishing.

The head and tail proofs can't be used in isolation, however. They have to be merged to create a hash chain that proves participation of a request in the round. For example, the hash chain computation for request 6 on figure 4.2 would need the node J from the head proof first, then node K from the tail proof, then node I from the head and then node L from the tail proof.

The hash chain obtained by merging the head and tail proofs encompasses the functions of all three mechanisms — aggregating, linking, and publishing — of the general model shared by the ISO/IEC 18014 and the ANS X9.95. Since the `links` member of the `BindingInfo` is mandatory, it seems sensible to put the hash chain there to satisfy the formal requirement.

However, an important consideration is that a generic verifier performing just the hash chain re-computation will only prove that the request participated in the round, but is unable to use the hash chains to establish the relative temporal order of two timestamps from the same round. Taking advantage of that additional information would be a non-standard extension.

4.2.2 Formats

The CHRONOS project claims ISO/IEC 18014 compliance. Unfortunately, we were unable to obtain sample time-stamp tokens from the service and there is no token format specification, so we can't offer any evidence neither to support nor to counter this claim.

Summary

The goal of this work was to survey the existing standards for hash-linking based time-stamping systems and investigate the compatibility of actual systems with those standards.

Relevant standards from four major bodies — the Internet Engineering Task Force (IETF), the European Telecommunications Standards Institute (ETSI), the International Organization for Standardization (ISO), and the American National Standards Institute (ANSI) — were reviewed.

From our review we can conclude that in the historic progression of the technical standards from the IETF RFC 3161 to the ISO/IEC 18014 to the ANSI ANS X9.95, each following standard has taken care to remain backwards compatible with the predecessor for those users who do not need the new features added in the more recent specification.

We discovered only one case where a feature that was present in an older standard was revoked by a newer one. This is the "archive" time-stamping method specified in the ISO/IEC 18014, but not in the ANS X9.95. Considering that supporting any given method is optional for any service, and especially in view of the security risks that this method carried, we feel its absence from the ANSI standard is really not a great loss for anyone.

We also examined the following time-stamping services for their compatibility with the technical standards mentioned above.

The Surety time-stamping service claims full compliance with the ANS X9.95 standard and we found no evidence to the contrary. The service also claims compliance with the ISO/IEC 18014, regarding which the only reservation is that the Merkle tree based aggregation algorithm used by Surety is not defined in the ISO standard and could therefore be viewed as a non-standard

extension. However, the ISO standard does not define any algorithms at all, so any working system would be a non-standard extension in this sense. The service neither claims nor delivers any compatibility with the RFC 3161 specification.

The GuardTime time-stamping service claims partial compliance with the RFC 3161 standard, as they package their hash-linking based evidence in the form of a non-standard signature scheme for inserting it into otherwise RFC 3161-like time-stamp tokens. Since their tokens are based on the forward-compatible RFC 3161 specification, they are just as compliant with the ISO and ANSI standards.

The CHRONOS prototype service claims compliance with the ISO/IEC 18014 standard. While there is no evidence that it could not be compatible, there is also no proof, as we were unable to obtain sample tokens for independent analysis.

Räsiahelatel põhinevate ajatemplisüsteemide standardid

Magistritöö

Ahto Truu

Kokkuvõte

Inimkond sõltub üha enam informatsioonist, mida sageli hoitakse kergesti ja märkamatult muudetavatel andmekandjatel. Seoses sellega kasvab vajadus vahendite järele, mis võimaldaks tuvastada, millal mingi andmeobjekt loodi ja kontrollida, et seda pole volitamata isikute poolt muudetud.

Tänapäeval levinuim lahendus on tuua selleks süsteemi usaldusväärne kolmas osapool, ajatempliteenus, mis lisab andmetele (eeldatavasti õige) aja ja signeerib saadud paari digitaalselt. See lahendus töötab ainult eeldusel, et kõik osapooled usuvad, et teenus ei võltsi ajatempleid ei meelega (näiteks mõne kliendi tellimusel) aga kogemata (näiteks süsteemi tõrke tõttu).

Üks võimalus vähendada või isegi üldse loobuda vajadusest ajatempliteenust usaldada on siduda ajatemplid omavahel ja/või mingite väliste sündmustega, näiteks ajalehtede ilmumisega.

Krüptograafiliste räsifunktsioonide ühesuunalisus on mõnevõrra sarnane aja enda ühesuunalise kulgemisega — informatsioon liigub minevikust tuleviku suunas, kuid mitte vastupidi. Räsifunktsioonide ühesuunalisusel põhinevad mitu skeemi sõltumatult kontrollitavate ajatemplisüsteemide loomiseks.

Selleks, et neid süsteeme praktikas edukalt juurutada, tuleb standardida nende kasutatavad algoritmid, protokollid ja andmevormingud. Vastasel juhul ei tarviste süsteemi kasutajad jõuda üksmeelele ühe või teise tõendi tähenduse osas.

Käesoleva töö eesmärk oli uurida räsiahelatel põhinevate ajatempliskeemide standardeid ja reaalsete süsteemide vastavust neile standarditele.

Töö esimeses osas vaatlesime nelja suure standardiasutuse — Internet Engineering Task Force (IETF), European Telecommunications Standards Institute (ETSI), International Organization for Standardization (ISO) ja American National Standards Institute (ANSI) — standardeid.

Nende asutuste välja antud standardid — IETF RFC 3161, ISO/IEC 18014 ja ANSI ANS X9.95 — ühilduvad omavahelt üldiselt väga hästi. Me leidsime ainult ühe näite, kus uuem standard ei toeta mingit vanemas standardis kirjeldatud võimalust.

Töö teises osas uurisime olemasolevaid räsiahelatel põhinevaid ajatempliteenuseid ja nende ühilduvust eelnimetatud standarditega.

Surety lubab oma ajatempliteenuse olevat ANS X9.95-ühilduva ja me ei leidnud ühtki põhjust vastupidist väita. ISO/IEC 18014-ühilduvuse osas on ainus reservatsioon asjaolu, et Surety teenuses kasutatav agregeerimisalgoritm ei ole ISO standardis kirjeldatud ja seda võiks seega lugeda mittestandardseks laienduseks. Tasub aga silmas pidada, et ISO standard ei defineeri ühtki konkreetset agregeerimisalgoritmi ja seega oleks igasugune töötav süsteem sellest seisukohast võttes mittestandardne laiendus. RFC 3161'ga Surety teenus ei ühildu.

GuardTime väidab oma ajatempliteenuse olevat põhimõtteliselt RFC 3161-ühilduva. Nende lahendus pakendab räsiahelatel põhineva tõendusmaterjali mittestandardse signatuuri kujule, et see siis muus osas RFC 3161-ühilduva vormiga ajatemplile lisada. Kuna ISO ja ANSI standardid on RFC 3161 laiendused, on GuardTime'i ajatemplid nendega täpselt sama ühilduvad kui RFC 3161'ga.

CHRONOSe projekt väidab oma prototüübi olevat ISO/IEC 18014-ühilduva. Me ei näe küll olulist põhjust, miks see peaks võimatu olema, aga kuna meil ei õnnestunud hankida selle teenuse ajatemplite näidiseid, ei saa me seda väidet ka kinnitada.

Bibliography

[1] Carlisle Adams, Pat Cain, Denis Pinkas, and Robert Zuccherato. Internet X.509 public key infrastructure time-stamp protocol (TSP). IETF RFC 3161, 2001. `http://www.ietf.org/rfc/rfc3161.txt` [2010-07-01].

[2] Carlisle Adams and Stephen Farrell. Internet X.509 public key infrastructure certificate management protocols. IETF RFC 2510, 1999. `http://www.ietf.org/rfc/rfc2510.txt` [2010-07-01].

[3] ANSI ASC X9. Trusted time stamp management and security. ANS X9.95, 2005.

[4] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334. Springer, 1992.

[5] Josh Benaloh and Michael de Mare. Efficient broadcast time-stamping. Technical report, Clarkson University, 1991.

[6] Kaouthar Blibech and Alban Gabillon. CHRONOS: an authenticated dictionary based on skip lists for timestamping systems. In *SWS '05: Proceedings of the 2005 Workshop on Secure Web Services*, pages 84–90. ACM, 2005.

[7] Kaouthar Blibech and Alban Gabillon. A new timestamping scheme based on skip lists. In *Computational Science and Its Applications — ICCSA 2006*, pages 395–405. Springer, 2006.

[8] Ahto Buldas and Aivo Jürgenson. Does secure time-stamping imply collision-free hash functions? In *Provable Security — ProvSec 2007*. Springer, 2007.

[9] Ahto Buldas, Aivo Jürgenson, and Margus Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In *Information Security and Privacy — ACISP 2010*. Springer, 2010.

[10] Ahto Buldas and Peeter Laud. New linking schemes for digital time-stamping. In *Proceedings of The 1st International Conference on Information Security and Cryptology — ICISC '98*, pages 3–14. Korea Institute of Information Security and Cryptology, 1998.

[11] Ahto Buldas, Peeter Laud, Helger Lipmaa, and Jan Villemson. Time-stamping with binary linking schemes. In *Advances in Cryptology — CRYPTO '98*, pages 486–501. Springer, 1998.

[12] Ahto Buldas and Sven Laur. Do broken hash functions affect the security of time-stamping schemes? In *Applied Cryptography and Network Security — ACNS 2006*. Springer, 2006.

[13] Ahto Buldas, Helger Lipmaa, and Berry Schoenmakers. Optimally efficient accountable time-stamping. In *Public Key Cryptography — PKC '00*, pages 293–305. Springer, 2000.

[14] Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In *Advances in Cryptology — ASIACRYPT 2004*, pages 500–514. Springer, 2004.

[15] ETSI ESI. Time stamping profile. ETSI TS 101 861 v1.3.1, 2006.

[16] ETSI ESI. Policy requirements for time-stamping authorities. ETSI TS 102 023 v1.2.2, 2008.

[17] Owen Gingerich. The Galileo affair. *Scientific American*, 247(2):133–143, 1982.

[18] GuardTime AS. GuardTime technical reference. GuardTime, 2010. `http://www.guardtime.com/downloads/GuardTimeTechRef.pdf` [2010-07-01].

[19] GuardTime AS. GuardTime technical reference — formats and algorithms. GuardTime, 2010. `http://www.guardtime.com/downloads/GuardTimeTechRefApp.pdf` [2010-07-01].

[20] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In *Advances in Cryptology — CRYPTO '90*, pages 437–455. Springer, 1991.

[21] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *Journal of Cryptology*, 3(2):99–111, 1991.

[22] Stuart Haber and W. Scott Stornetta. Secure names for bit-strings. In *CCS '97: Proceedings of the 4th ACM conference on Computer and communications security*, pages 28–35. ACM, 1997.

[23] Robert Hooke. *Description of helioscopes*. Printed by T.R. for John Martyn, London, 1676.

[24] Robert Hooke. *Lectures de potentia restitutiva, or, Of spring*. Printed for John Martyn, London, 1678.

[25] Robert Hooke and Richard Waller. *The posthumous works of Robert Hooke*. Royal Society, London, 1705.

[26] Russell Housley. Cryptographic message syntax. IETF RFC 2630, 1999. `http://www.ietf.org/rfc/rfc2630.txt` [2010-07-01].

[27] Russell Housley, Warwick Ford, Tim Polk, and David Solo. Internet X.509 public key infrastructure certificate and CRL profile. IETF RFC 2510, 1999. `http://www.ietf.org/rfc/rfc2459.txt` [2010-07-01].

[28] ISO/IEC JTC1. Abstract Syntax Notation One (ASN.1): Specification of basic notation. ISO/IEC 8824-1, 2002.

[29] ISO/IEC JTC1. ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). ISO/IEC 8825-1, 2002.

[30] ISO/IEC JTC1. Time-stamping services — Part 1: Framework. ISO/IEC 18014-1, 2002.

[31] ISO/IEC JTC1. Time-stamping services — Part 2: Mechanisms producing independent tokens. ISO/IEC 18014-2, 2002.

[32] ISO/IEC JTC1. Time-stamping services — Part 3: Mechanisms producing linked tokens. ISO/IEC 18014-3, 2004.

[33] ITU-T. Abstract Syntax Notation One (ASN.1): Specification of basic notation. ITU-T X.680, 2002.

[34] ITU-T. ASN.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). ITU-T X.690, 2002.

[35] Ralph C. Merkle. *Secrecy, Authentication, and Public Key Systems*. PhD thesis, Stanford University, 1979.

[36] Ralph C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, pages 122–134, 1980.

[37] R. Miskinis, D. Smirnov, E. Urba, A. Burokas, B. Malysko, P. Laud, and F. Zuliani. Digital time stamping system based on open source technologies. In *Frequency Control Symposium, 2009 Joint with the 22nd European Frequency and Time forum*, pages 700–705. IEEE, 2009.

[38] J. Ian Munro, Thomas Papadakis, and Robert Sedgewick. Deterministic skip lists. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 367–375. Society for Industrial and Applied Mathematics, 1992.

[39] D. Pinkas, N. Pope, and J. Ross. Policy requirements for time-stamping authorities (TSAs). IETF RFC 3628, 2003. `http://www.ietf.org/rfc/rfc3628.txt` [2010-07-01].

[40] William Pugh. Skip lists: a probabilistic alternative to balanced trees. *Commun. ACM*, 33(6):668–676, 1990.

[41] Sertifitseerimiskeskus AS. Time-stamping service principles. Sertifitseerimiskeskus, 2002. `http://www.sk.ee/pages.php/0203040506` [2010-07-01].

[42] Jeff Stapleton. Trusted time stamp standards: A comparison and guideline of ANS X9.95. Technical report, Information Assurance Consortium, Innové LLC, 2007. http://www.infoassurance.org/Public%20Docs/TTSS%2020070429%20IAC.pdf [2010-07-01].

[43] Surety, LLC. Ensuring record integrity with AbsoluteProof. Surety, 2008.

[44] Michael Thimblin, NagaSree Chandu Kamisetty, Padmanabhan Raman, and Anupama Paila. Implementation of an evidentiary record validation utility and security analysis for Surety's AbsoluteProof. Technical report, George Mason University, 2005. http://bass.gmu.edu/courses/ECE543/project/reports_2005/TIMESTAMPING_report.pdf [2010-07-01].