Authenticity, Integrity and Proof of Existence for Long-Term Archiving: a Survey

Martín A. G. Vigil[1], Daniel Cabarcas[1],
Alexander Wiesmaier[2], and Johannes Buchmann[1]

[1] Technische Universität Darmstadt
Hochschulstraße 10, 64283 Darmstadt, Germany
{vigil,cabarcas,buchmann}@cdc.informatik.tu-darmstadt.de
[2] AGT Group (R&D) GmbH
Hilpertstraße 35, 64295 Darmstadt, Germany
awiesmaier@agtgermany.com

Abstract. Electronic archives are increasingly being used to store information that needs to be available for a long time such as land register information and medical records. In order for the data in such archives to remain useful, their integrity and authenticity must be protected over their entire life span. Also, in many cases it must be possible to prove that the data existed at a certain point in time. In this paper we survey solutions that provide long-term integrity, authenticity, and proof of existence of archived data. We analyze which trust assumptions they require and compare their efficiency. Based on our analysis, we discuss open problems and promising research directions.

Keywords: Long term; Archiving; Authenticity; Integrity; Proof of Existence

1 Introduction

Electronic archives are increasingly being used to store information that needs to be available for a long time. Recently, some land registers have migrated their data to digital media in order to reduce physical space and allow public access. Examples of such a service are the French land register of Alsace-Moselle [1] and the Estonian Land Register [2]. Electronic invoices can be used for tax purposes in the European Union according to Directive 2001/115/EC [3]. Therefore, the storage of invoices is necessary. For instance, Slovenia provides a web tool [4] for tax declaration that requires companies and individuals to store invoices regarding immovable property for at least 20 years [5]. In several countries, the health sector has to store patients' medical records for years. Digital media is the preferred storage means. The retention time for medical records varies. For instance, in Germany they are retained for 10 years after the end of a treatment [6], or 30 years if x-ray was used [7]. In the United Kingdom (UK), retention is mandatory until 10 years after the patient's death [8]. Patents are commonly preserved for several years. For instance, in the UK preserved periods can last up to 27 years [9].

In order for the data in such archives to remain useful, their *integrity* ("data have not been altered since their creation") and *authenticity* ("the origin of the data can be identified") must be protected over their entire life span. Also, in many cases *proof of existence* or *witnessed existence* ("a time reference when the data was witnessed can be identified")[3] is required. Achieving these goals is very challenging. Several protection solutions rely on cryptographic techniques. However, such techniques are not guaranteed to be secure in the future. For example, currently RSA signatures are being used in many solutions which are known to be vulnerable by quantum computer attacks. One approach to maintaining long-term protection is to regularly update digital signatures. However, this is an endless and complex process that requires keeping track of information concerning all the signatures generated in the past. Another approach is to avoid cryptography as much as possible by making use of the physical evidence of print media or by relying on trusted third parties. Given these different approaches, the question arises whether they are able to provide the expected long-term protection and how they compare.

The goal of this paper is to answer this question. We present, analyze, and compare the existing solutions. Our comparison refers to the trust assumptions required for long-term protection. This comparison allows us to assess the security level of the different solutions. For example, almost all solutions assume that digital signatures are updated before they become insecure. As cryptanalytic progress is hard to predict, it is unclear whether such an assumption is justified. We also compare the efficiency of the solutions, for example, their storage and computing time requirements as they are an indication for practical applicability. The existing solutions turn out not to be perfect. So our analysis naturally leads to open research questions.

In addition to integrity, authenticity, and witnessed existence, *long-term confidentiality* is another important protection goal. We do not discuss this problem here but refer the reader to [10].

This paper is organized as follows. Section 2 presents security components commonly used to protect digital data. Section 3 describes in detail existing solutions for long-term protection. Their security is compared in Section 4, and their efficiency in Section 5. Section 6 briefly discusses a number of solutions that fail to provide long-term protection. Finally, Section 7 provides a conclusion and proposes possible future work.

2 Background

In this section we introduce basic security components that are used as building blocks to protect data objects in digital archives.

[3] We use the term *witnessed existence* instead of *proof of existence* to indicate that such a proof is out of reach with current technology.

2.1 Notation

The literature covered by this survey uses inconsistent terminology. In order to be able to compare the different approaches, this paper uses the following terminology mostly taken from RFC 4810 [11], and augmented with a few additional terms:

- **long-term**: an unbounded period.
- **data object**: a unit of digital data. For example, a document or a photograph recorded as a digital file.
- **archive**: an infrastructure of equipment and policies to preserve trustworthy data objects.
- **archivist**: a trusted entity that is committed to preserve trustworthy data objects in an archive [12].
- **submitter**: an entity that submits data objects to an archive.
- **retriever**: an entity that retrieves data objects from an archive.
- **validation evidence**: the evidence necessary to demonstrate that a data object is trustworthy.
- **trust assumption**: reliance on something or someone that is taken for granted.

2.2 Cryptographic Hash Function

A *cryptographic hash function* or *hash function* is a mathematical function h that maps bit strings of arbitrary length to strings of fixed length [13] and satisfies the following properties: a) given an output $y = h(x)$, finding x is computationally infeasible ("pre-image resistance"); b) given an input x and the output $y = h(x)$, finding an input x', such that $x' \neq x$ and $h(x') = y$ is computationally infeasible ("second pre-image resistance"); and c) finding any different inputs x and x', such that $h(x) = h(x')$, is computationally infeasible ("collision resistant"). We refer to a hash function's output as *digest*, *hash value* or simply *hash*. An important application of hash functions is verifying the integrity of a data object.

The security of a hash function relies on the hardness of defeating one of the hash function's properties. Therefore, security fades out as computer power and cryptanalysis evolve. For example, the hash function MD5, which was considered secure, is no longer secure [14]. In addition, a brute force attack is always possible, thus, in the long term, no single hash function can be secure.

2.3 Merkle Trees

A *Merkle tree* [15] is an ordered binary tree that is constructed from a sequence of data objects $(d_1, \ldots, d_n)$ using a hash function h. The leaves of the tree are the hash values $h(d_i)$ for $1 \leq i \leq n$ (in that order). A node that is not a leaf is $h(l \| r)$, where the left (l) and the right (r) children are concatenated $(\|)$. Figure 1 shows an example of a Merkle tree.

Merkle trees can be used to verify whether a data object is an uncorrupted member of a sequence in the right position. The root of the tree serves as a

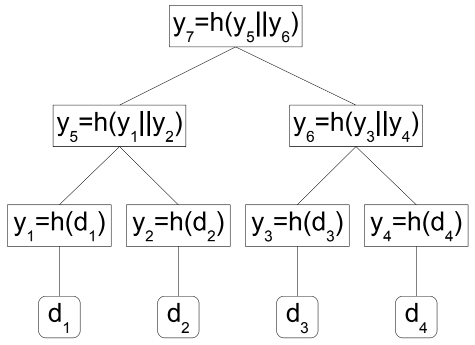

Fig. 1. A Merkle tree whose leaves $y_1, \ldots, y_4$ are digests of data objects $d_1, \ldots, d_4$, and nodes $y_5, \ldots, y_7$ are digests of the concatenation of their respective child nodes.

digest for the entire tree. A data object is verified with respect to the root. The verification is done by reconstructing the path from the leaf to the root using the so-called *authentication path*, which consists of the siblings of the nodes in the path. The verification effort grows logarithmically in the number of elements in the sequence [15].

2.4 Write-Once Media

A write-once medium, or write-once-read-many (WORM) memory, is a medium on which a data object can be written only a single time. Therefore, afterwards it is not possible to change the data object's copy on the medium. For example, optical disks. By writing a data object on write-once media, we can produce evidence of integrity for the data object.

A wide-visible medium is a write-once medium that ensures not only integrity, but also authenticity and witnessed existence. It is a *trusted* and public record whose long-term preservation at many locations makes tampering very difficult [16]. Therefore, we can produce evidence of authenticity for a data object by writing an authentic copy of it on wide-visible media. Examples of such media are broadcast protocols to spread data [17] and newspapers. Wide-visible media are extensively witnessed, thereby ensuring witnessed existence for data objects. This is because witnesses associate a data object on wide-visible media to the date *when* they came into contact with the data object. In addition, a wide-visible medium associates the witnessed existence of a data object with other objects written on the same medium. In the long term, it is not clear whether write-once and wide-visible media will be available in the future or how they will be accessed. For example, these media can disappear, due to physical deterioration, and witnesses can die.

2.5 Digital Signature

A *digital signature scheme* consists of three algorithms. The *key generation algorithm* generates a *secret signing key* and a *public verification key*. Given a data object and a secret signing key, the *signing algorithm* computes a *digital signature* on the data object. Correspondingly, given a data object, a signature on the data object, and a public verification key, the *verification algorithm* decides if the signature is valid or not valid. Examples of digital signature schemes currently used in practice are *RSA* [18] and variants of *ElGamal* [19,20].

There is no digital signature scheme that provides long-term security. The reason is that the security of the known digital signature schemes rely on the hardness of certain algorithmic problems. As computer power and cryptanalytic techniques evolve, this hardness fades out.

2.6 Public Key Infrastructure

The use of digital signatures relies on the availability of trustworthy public keys. *Public key infrastructures* (PKI) provide such keys by issuing and disseminating *certificates* [21]. Certificates are data objects that are digitally signed by some *Certification Authority* (CA), which asserts that a public key belongs to a certain subject. A CA can sign certificates for itself and other entities, such as subordinate CAs or final users. The sequence of certificates between the CA and a descendant certificate is called *certificate chain*. A common used standard for certificates is X.509 [22].

Certificates usually have an explicit expiration date. However, various circumstances may cause a certificate to become invalid before its expiration date. For instance, if the corresponding private key is compromised. In such case, the issuing CA revokes the certificate. A CA is responsible for disseminating the revocation status of unexpired certificates that the CA issued. Revocation status is often published using Certificate Revocation List [22] and Online Certificate Status Protocol [23].

The trustworthiness of a certificate fades in the long term. A reason is the reliance on the certificate's signature, whose security weakens over time. Another reason is that there may be insufficient revocation information. After the certificate expires, its revocation status might be no longer updated by the issuing CA. Consequently, the certificate's trustworthiness may not be guaranteed after expiration.

2.7 Timestamp

A trusted entity named *Timestamp Authority* (TSA) asserts the witnessed existence of a given data object by issuing a *timestamp*. A very common standard for timestamps is the *PKIX timestamp* [24]. It consists of a digital signature on the data object's digest together with the current date and time. Another type of timestamp consists of the digest of a data object published on a wide-visible medium (see Section 2.4).

A TSA can link a timestamp to a previous timestamp, thereby establishing an order of creation between them. To do so, the TSA creates a timestamp for the hash of the concatenation between a data object and an existing timestamp. This technique is called *hash linking* and is detailed in [25]. Also, a TSA can efficiently timestamp several data objects using a Merkle tree (see Section 2.3). The TSA calculates a Merkle tree, whose leaves are the digests of each data object, and timestamps the tree's root. This procedure is described in [16].

A single timestamp does not provide long-term security because it relies on a cryptographic hash function (see Section 2.2), a digital signature scheme (see Section 2.5) or wide-visible media (see Section 2.4), which all are subject to security deterioration over time.

2.8 The Lifetime of Cryptographic Algorithms

The security of cryptographic algorithms such as hash functions and digital signature algorithms is not everlasting (see Sections 2.2 and 2.5). Researchers and government agencies estimate for how long such algorithms will remain secure based on the best known cryptanalytic techniques, e.g. [26,27,28]. Estimates are to be revised whenever more efficient cryptanalytic techniques are discovered or computer power increases.

3 Solutions for Long-Term Protection

In this section we describe a number of solutions that provide long-term authenticity, integrity and/or witnessed existence for data objects in digital archives. To the best of our knowledge, only these solutions successfully address these protection goals in the long term.

3.1 ETSI Standards

In [29,30] the European Telecommunications Standards Institute (ETSI) proposes a solution that provides long-term evidence of integrity and witnessed existence for signed data objects. This evidence extends the trustworthiness of a data object's signature beyond the lifetimes of certificates, timestamps and cryptographic algorithms, thereby ensuring long-term authenticity. The solution is based on digital signatures (see Section 2.5) and PKIX timestamps (see Section 2.7).

There are three players in this solution: an archivist, who preserves a set of data objects in the archive; a retriever, who can verify integrity, authenticity and witnessed existence of data objects from the archive; and Timestamp Authorities (TSA) that issue timestamps on the archivist's request.

A data object is encapsulated in a data structure that we refer to as *archival package*. The standard requires that the data object is signed by its originator and the signature is included in the *archival package*.

For each data object, the archivist maintains a sequence of timestamps that is used as evidence for the data object's integrity and witnessed existence (this technique was originally proposed by Bayer et al. in [31]). Before requesting the first timestamp, the archivist collects the data object signer's certificate chain and revocation status and includes this data in the *archival package*. A first timestamp is issued on the *archival package* and is included in the *archival package*. The next timestamps are constructed as follows. Before the most recent timestamp's security is over (see Section 2.7), the archivist collects the corresponding TSA's certificate chain and revocation status, includes this data in the *archival package*, requests a new timestamp on the *archival package*, and includes the new timestamp in the *archival package*.

In order to verify integrity, witnessed existence and authenticity of a data object, a retriever verifies the signature on the data object, each timestamp, each certificate chain and each validation status included in the *archival package*. The verification of a timestamp consists of verifying the timestamp's signature using the corresponding TSA's public key. Note that the certificate chain and revocation status of the TSA that issued the most recent timestamp is not available in the *archival package*. The retriever collects this data by himself.

Indeed, this procedure proves integrity, witnessed existence and authenticity of the data object at any verification time. Suppose the first timestamp is issued at time t_1 on an *archival package* a_1 which consists of: the data object, its signature, and evidence that shows that the key used to verify the signature is trusted at time t_1. Given that the first timestamp is valid, it proves that a_1 was not modified between t_1 and time t_2, when a second timestamp is issued. Lets refer to the *archival package* at time t_2 as a_2. The second timestamp is issued on a_2 which consists of: a_1, the first timestamp, and evidence that shows that the TSA's key used to verify the first timestamp is trusted at time t_2. These three pieces of information constitute a proof that a_1 existed at time t_1 and that it has not changed since then. As long as the second timestamp is trusted, it proves that at time t_2 there was a proof that a_1 existed at time t_1. Therefore the second timestamp extends the proof that a_1 existed beyond t_2, even if the first timestamp is no longer trusted after t_2. In general, if for every $i = 1, \ldots, n$, the i^{th} timestamp is issued at time t_i and it is trusted between t_i and t_{i+1}, then, as long as the n^{th} timestamp is trusted, it proves at time t_{n+1} that there was a proof that a_1 existed at time t_1. Since a_1 contains the data object, integrity and witnessed existence follow. Moreover, since a_1 constitutes a proof that the data object was authentic at time t_1, authenticity follows.

There are two assumptions for the ETSI solution to guarantee long-term integrity, witnessed existence and authenticity of a data object: that the underlying PKIs and TSAs are reliable, and that the cryptographic algorithms are not unexpectedly broken.

3.2 Evidence Record Syntax

Evidence Record Syntax (ERS) is a solution for long-term protection of digital data proposed by Gondrom et al. in RFC 4998 [32]. The solution provides long-term evidence of integrity and witnessed existence for data objects. If data objects have been previously signed, long-term authenticity is guaranteed. The solution is based on Merkle trees (see Section 2.3) and PKIX timestamps (see Section 2.7).

There are three players in this solution: an archivist, who preserves a set of data objects in the archive; a retriever, who can verify integrity, authenticity and witnessed existence for data objects from the archive; and Timestamp Authorities (TSAs) that issue timestamps on the archivist's request.

In a nutshell, ERS works as follows. For each data object, the archivist maintains a structure called *Evidence Record*, which is used as evidence for the integrity and witnessed existence of the data object. An *Evidence Record* contains a set of timestamps[4]. A single timestamp can cover a group of data objects by using a Merkle tree, however, a copy of the timestamp is included in each object's *Evidence Record*. The archivist performs three different tasks to maintain *Evidence Records*:

– in an *initialization phase*, the archivist creates a Merkle tree from a group of data objects and includes a first timestamp to each of them;
– regularly, the archivist executes a *timestamp renewal*, in order to address the fading of the security of timestamps;
– and also on a regular basis, the archivist executes a *Merkle tree renewal*, in order to address the fading of the security of hash functions.

We now describe each task in detail. In the initialization phase, the archivist selects a group of data objects and a hash function, constructs a Merkle tree whose leaves are the hash values of the data objects, and requests a timestamp on the tree's root. The archivist includes in each data objects' *Evidence Record* the following:

– the identifier of the hash function used to construct the Merkle tree;
– the authentication path for the corresponding leaf of the Merkle tree;
– and the timestamp issued on the Merkle tree's root.

Timestamp renewal of a previously issued timestamp works as follows. The archivist requests a new timestamp on the hash of the old timestamp. The archivist includes the new timestamp in each *Evidence Record* that contains the old timestamp.

ERS does not specify how the archivist manages validation evidence for timestamps' signatures, namely certificates and revocation status. It is a common practice to store validation evidence for the previous timestamp's signature before requesting the new timestamp. Not doing so adds an extra trust assumption on the system. Namely, that the archivist is trusted to check this evidence before requesting a new timestamp. We assume that the archivist collects and stores validation evidence for the previous timestamp's signature before requesting the new timestamp.

[4] Actually, each timestamp is encapsulated in a structure called *Archive Timestamp*. We omit this detail to avoid confusion with PKIX timestamps.

Merkle tree renewal of a previously constructed Merkle tree works as follows. The archivist identifies the data objects $d_0, \ldots, d_{n-1}$ that originated this Merkle tree. The archivist constructs a new Merkle tree whose leaves $y_0, \ldots, y_{n-1}$ are given by

$$y_i = h'(h'(d_i)||h'(R_i)) \ , \tag{1}$$

where h' is a new hash function and R_i is the *Evidence Record*[5] of the i^{th} data object in its current state. As in the initialization phase, the archivist requests a timestamp on the Merkle tree's root, and includes in each data object's *Evidence Record*: the new hash function identifier, the authentication path for the corresponding leaf of the new Merkle tree, and the new root's timestamp.

It is the responsibility of the archivist to perform the corresponding renewals in a timely manner. That is, the archivist should execute a timestamp renewal before any timestamp becomes insecure, and a Merkle tree renewal before the hash function used to construct a Merkle tree becomes insecure.

In order to verify integrity and witnessed existence of a data object, a retriever verifies each of the timestamps in the object's *Evidence Record*. Timestamps created in the initialization phase or by Merkle tree renewal refer to the Merkle tree's root. Thus, in order to verify such a timestamp, the retriever reconstructs the root using the authentication path, which is stored in the *Evidence Record*. Timestamps created by timestamp renewal, refer to the previous timestamp.

This procedure indeed proves integrity and witnessed existence of the data object at any verification time. The first timestamp proves that the root of the Merkle tree existed since a time t_1. The witnessed existence of the root at time t_1, proves that the whole Merkle tree was constructed at that time. The authentication path proves that the data object was part of the Merkle tree. Suppose that the second timestamp is issued by timestamp renewal at time t_2. In this case, the second timestamp is issued on the hash of the first timestamp. We are assuming that the timestamp contains evidence that shows that the TSA's key used to verify the first timestamp was trusted at t_2. This two pieces of information constitute a proof that the root of the Merkle tree existed at time t_1. Therefore the second timestamp extends the proof of witnessed existence of the data object beyond t_2, even if the first timestamp is no longer trusted after t_2. In general, the n^{th} timestamp, issued by timestamp renewal at time t_n, extends the proof of witnessed existence of the data object beyond t_n, under the assumption that for every $i = 1, \ldots, n-1$, the i^{th} timestamp was issued at time t_i and it was trusted between t_i and t_{i+1}.

Now suppose that at time t_{n+1} the archivist runs a Merkle tree renewal. In this case, the $n+1^{th}$ timestamp proves that the root of the new Merkle tree existed since time t_{n+1}. The witnessed existence of the root at time t_{n+1} proves that the whole Merkle tree existed at that time. The authentication path proves that the leaf was part of the Merkle tree. Recall that the leaf is given by Equation 1. Hence, it proves the witnessed existence of the data object and its *Evidence*

[5] More precisely, only the sequence of all timestamps need to be hashed. The *Evidence Record* contains some extra information.

Record at time t_{n+1}. This constitutes a proof of the witnessed existence of the data object at time t_1 provided that the hash function used to construct the first Merkle Tree was trusted between t_1 and t_{n+1} and that for every $i = 1, \ldots, n$, the i^{th} timestamp was issued at time t_i and it was trusted between t_i and t_{i+1}. Figure 2 illustrates the proof.

In general, let $0 = n_1 < n_2 < \cdots < n_m < n_{m+1} = N$ be positive integers and $t_1 < \cdots < t_{N+1}$ be time references. Suppose that an initialization step is run on a data object at time t_1 using hash function h_1. Suppose Merkle tree renewals are run at times $t_{n_2} + 1, t_{n_3} + 1, \ldots, t_{n_m} + 1$, using hash functions $h_2, \ldots, h_m$ respectively. Suppose that timestamp renewal is run at time t_i for

$$i = n_1 + 2, \ldots, n_2, n_2 + 2, \ldots, n_3, n_3 + 2, \ldots\ldots, n_m, n_m + 2, \ldots, N.$$

Suppose that for $j = 1, \ldots, m$, the hash function h_j was trusted between t_{n_j+1} and $t_{n_{j+1}+1}$. Suppose that for $i = 1, \ldots, N$, the i^{th} timestamp was trusted between t_i and t_{i+1}. Then, verification of ERS evidence at time t_{N+1} proves integrity and witnessed existence of the data object. See Figure 2 for an illustration.

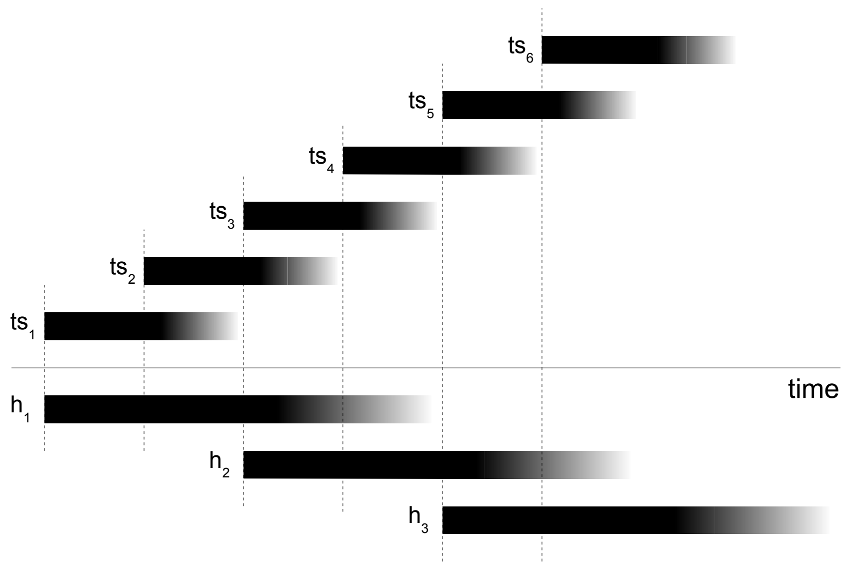

Fig. 2. Evidence of witnessed existence in ERS, where ts_1 to ts_6 are timestamps and h_1 to h_3 are hash functions. The solid color means that a timestamp or hash function is trusted; the gradient color means that trust is uncertain. Timestamps ts_2, ts_4 and ts_6 are created by timestamp renewal. Timestamps ts_2 and ts_3 are created by Merkle tree renewal.

The procedure also guarantees long-term authenticity, provided that the data object was signed. In this case, before the archivist requests the first timestamp, he collects validation evidence (namely certificates and revocation status) for the data object's signature and includes this evidence in the signed data object.

There are two assumptions for ERS to guarantee long-term integrity, witnessed existence and authenticity of a data object: a) that the underlying PKIs and TSAs are reliable; and b) that the cryptographic algorithms are not unexpectedly broken.

3.3 Auditing Control Environment

Auditing Control Environment (ACE) is a solution for long-term protection of digital data proposed by Song and JaJa in [33]. It provides long-term evidence of integrity and witnessed existence for data objects. The solution produces two tiers of integrity evidence, allowing different levels of audits. The solution is based on Merkle trees (see Section 2.3) and wide-visible media (see Section 2.4).

We identify three players in this solution: an archivist, who preserves a set of data objects in the archive; an auditor who can verify integrity and witnessed existence for data objects from the archive; and the so-called Integrity Management System (IMS), which is a third-party service provider that generates integrity evidence on the archivist's request.

The IMS and the archivist agree on a hash function. The archivist registers a data object in ACE by submitting the data object's digest to the IMS. The IMS collects a number of registration requests and creates an *integrity token* (IT) for each data object as follows. The number of requests can be determined dynamically and it constitutes an *aggregation round*. The IMS creates a Merkle tree with the objects' digests of a round as its leaves. Each IT contains the object's digest, the time of registration and the corresponding authentication path in the Merkle tree. The IT is returned to the archivist, who stores it. The IMS stores the root of the Merkle tree. An IT and the root of the Merkle tree constitute the *first tier* of integrity evidence.

After a fixed time period, the IMS generates the *second tier* of integrity evidence. The IMS creates a Merkle tree whose leaves are digests of the first tier tree roots created during the time period. Figure 3 illustrates the two tier construction. Then, the IMS stores the root of this tree on a wide-visible medium. This root is the witness value of the state of the archive.

Before the hash function becomes insecure, the IMS and the archivist agree on a new hash function. The archivist reregisters in ACE each object that was registered with the previous hash function. To do so, the hash over the concatenation of the data object with the corresponding IT is calculated and submitted to the IMS. ACE considers two kinds of audits, a first tier audit and a second tier audit. As a maintenance policy, the archivist carries out first tier audits on a regular basis. An audit can also be requested by a retriever of a data object to verify its integrity. Second tier audits can be triggered by a disagreement in a first tier audit or also carried out as a maintenance policy.

A first tier audit works as follows. The auditor requests the data object and corresponding IT from the archive. The auditor then computes a digest of the object and compares it with the digest stored in the IT. If the values are equal,

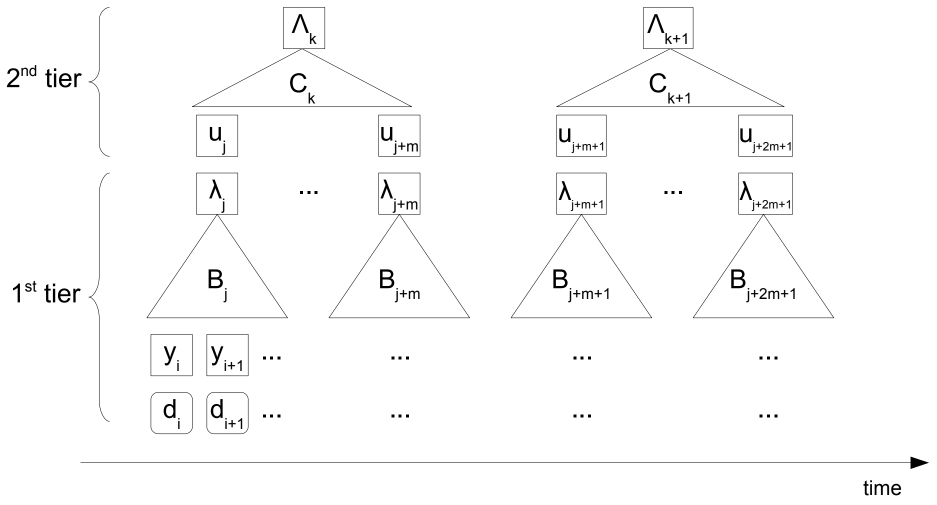

Fig. 3. Integrity Evidence in ACE, where d_i is a data object with digest y_i, B_j is a first tier Merkle tree with root λ_j, u_j is the digest of λ_j and C_k is a second tier Merkle tree with root Λ_k.

the auditor requests from the IMS the first tier tree root corresponding to the time specified in the IT. Then, the auditor computes the root of the Merkle tree using the digest and authentication path stored in the IT and compares it to the value provided by the IMS. If the values are equal, the auditor accepts, otherwise, rejects the object. If an object has been registered multiple times, the procedure is repeated for each IT.

Under the assumption that the IMS is reliable, it is easy to see that this procedure indeed proves integrity and witnessed existence of a data object. Moreover, if the object is appropriately reregistered before the hash function becomes insecure, the first tier audit proves integrity and witnessed existence of the data object at any verification time. The first IT proves the witnessed existence and integrity of the data object from the time it is issued t_0 until the time t_1 when a second IT is issued. Because the first IT is attached to the object before reregistering it, the second IT extends the proof of witnessed existence beyond t_1, even if the first IT is no longer trusted after t_1.

As we will see next, the second tier audit provides a proof of integrity that is independent of the IMS reliability. Moreover, it exposes any wrong doing of the IMS, thus providing an incentive for the IMS to act correctly.

In the second tier audit, we assume that the auditor has carried out the first tier audit. Then, the auditor requests from the IMS the second tier authentication path corresponding to the first tier root. Using this path and the first tier root, the auditor computes the second tier Merkle tree root and compares it to the corresponding value stored on a wide-visible medium. If the values are equal, the auditor accepts, otherwise, rejects the object.

Under the assumption that a wide-visible medium is trustworthy, it is easy to see that this procedure indeed proves integrity and witnessed existence of a data object. Moreover, if the IMS fails to provide a second tier authentication path that yields the value stored on a wide-visible medium, then the auditor can conclude that the IMS provided incorrect values.

There are two assumptions for ACE to guarantee long-term integrity and witnessed existence of a data object: a) that the wide-visible media are trustworthy; and b) that the cryptographic algorithms are not unexpectedly broken.

3.4 Content Integrity Service

Content Integrity Service (CIS) is a solution for long-term protection of digital data proposed by Haber et al. in [34,35]. It provides long-term evidence of integrity and witnessed existence for data objects. The solution allows objects to be transformed and produces an auditable record of every transformation. The solution is based on timestamps (see Section 2.7)

There are four main players in this solution: an archivist, who preserves a set of data objects in the archive; a retriever, who can verify integrity and witnessed existence for data objects from the archive; Timestamp Authorities (TSAs) that issue timestamps on the archivist's request; and a submitter, who submits data objects to be archived and can request changes on them.

Much as in the ETSI standards described in Section 3.1, in CIS, the archivist maintains a sequence of timestamps for each data object that is used as evidence for its integrity and witnessed existence. The first timestamp is issued on the data object and when the most recent timestamp is about to expire, the archivist requests a new timestamp on the hash calculated over the data object together with all its previous timestamps.

The most important difference to the ETSI standards is that CIS allows data objects to be transformed. Typical transformations that may occur are format conversions, annotations, additions of metadata, and later modifications of the data object. When a submitter requests to change a data object, the archivist stores the modified data object, together with a description of the transformation. Then, the archivist requests a new timestamp on the hash of the concatenation of the old object, the new object, the transformation description and previous timestamps. The goal is to memorize – and enable later verification of – the transformations, while preserving the assurance of integrity all the way back to that of the original form of the data object.

The authors of CIS argue that PKIX timestamps are not suitable for long-term archiving. The reason is that the trustworthiness of these timestamps relies on digital signatures, whose signing keys are hard to protect over time. Alternatively, they propose that TSAs issue timestamps using hash linking and commit them to wide-visible media (see Section 2.7). They defend that this technique is more reliable because the trustworthiness of timestamps depends solely on the wide-visible media and the collision resistance of hash functions.

The verification algorithm is analogous to that of ETSI. Following the analysis described in Section 3.1, it is easy to see that the procedure proves integrity and witnessed existence of a data object at any verification time.

The authors of CIS implemented a prototype of CIS within HP's Digital Media Platform (DMP). This is an architecture for content storage and processing operations [34]. The solution presents an XML-based interface for service interaction, and a graph data model for storing documents. The authors of CIS claim that the DMP Repository Abstraction is convenient for handling the integrity metadata used by the CIS.

There are two assumptions for CIS to guarantee long-term integrity and witnessed existence of a data object: a) that the wide-visible media are trustworthy; and b) that the cryptographic algorithms are not unexpectedly broken.

3.5 Lots of Copies Keep Stuff Safe

Lots of Copies Keep Stuff Safe (LOCKSS) is a solution for protection of digital data described in [36]. It guarantees integrity for archived data objects. The solution consists of establishing a peer-to-peer network of archives with replicated content. On a regular basis, archives communicate with each other to compare their copies. They use a voting protocol to identify corrupted copies.

LOCKSS was developed for libraries to preserve access to journals and other archival information published on the Web. In this context, it is assumed that retrievers of a particular library trust their library. LOCKSS does not provide any evidence of integrity for data objects to retrievers. Instead, it is used by legitimate libraries to maintain reliable copies of their digital content.

The overall goal of LOCKSS design is that with high probability legitimate archivists have a correct version of their content despite failures and attacks, and with low probability a powerful adversary can damage a significant proportion of the legitimate archives without detection.

LOCKSS works as follows. Archivists establish a peer-to-peer network with other archives. On a regular basis, peers vote on large archival units (AUs), normally a year's run of a journal. A LOCKSS peer invites into its poll a number of peers, hoping they will offer votes on their version of the AU. An invited peer computes a fresh digest of its own version of the AU, which it returns in a vote. If the caller of the poll receives votes that overwhelmingly agree with its own version of the AU (a landslide win), it is satisfied. If it receives votes that overwhelmingly disagree with its own version of the AU (a landslide loss), it repairs its AU by fetching the copy of a voter who disagreed, and invites peers to vote on the new AU. This process is repeated until landslide wins. If the result of the poll justifies neither a landslide win nor a landslide loss (an inconclusive poll), then the caller raises an alarm to attract human attention to the situation.

We present here a simplified version of LOCKSS opinion poll protocol for illustration. Archivist A wants to verify the integrity of its copy of an object d_i, where i is the unique identification for d in LOCKSS. Archivist A chooses a hash function h, generates a nounce N_A and sends a *poll* request (i, h, N_A) to a number of archives that have copies of the object. When an archive B receives the request, it generates a nounce N_B, computes $y_B = h(N_A \| N_B \| d'_i)$, where d'_i is its copy of the object, and sends the response $r_B = (N_B, y_B)$ to A. Then A evaluates $y'_B = h(N_A \| N_B \| d_i)$ and compares y_B with y'_B. Archivist A does the same for each response and if the majority of comparisons succeed, then A considers its local copy d_i uncorrupted. Otherwise, A replaces his local copy d_i by a copy d'_i from a peer that answered the poll, and starts the protocol again using d'_i as its local copy.

The detailed version of the protocol (see [36]) includes some additional features:

– The poller chooses two sets of poll participants, an inner circle of trusted peers and a outer circle of new peers. Only the inner circle is used for deciding on the AU. By polling the outer circle, the poller tests the reliability of newcomers for expanding its set of trusted peers.
– A confidential channel between archivists is established using a key exchange protocol before starting the protocol.
– LOCKSS requires that pollers as well as voters expend provable computational effort [37], that is, they are required to prove that they spent computational effort proportional to the underlying protocol operation (hashing of an AU).

The protocol indeed protects the integrity of the archive. The simplified protocol allows legitimate archivists to detect the damage of a data object, and to repair it. This is true only if there are enough trusted peers with healthy copies of the data object. In order to defend the system from an attack, LOCKSS makes it costly for an adversary to sway an opinion poll in his favor or to waste peers' resources by requiring provable computational effort. Archivists also defend from malicious peers by only considering previously tested peers. Finally, by raising an alarm when a peer determines that a poll is inconclusive, LOCKSS makes it likely that an attack is detected before it progresses.

There are three assumptions for LOCKSS to guarantee long-term integrity of a data object: a) that the archivist that serves the retriever is reliable; b) that the majority of peers are run by reliable archivists; and c) that an archivist can find enough reliable peers with copies of its content.

LOCKSS is not vulnerable to the unexpected break of a cryptographic algorithm because no evidence of integrity is built. Moreover, the absence of integrity evidence allows LOCKSS to easily address format obsolescence of archived data objects by just updating them.

3.6 Optimized Certificates

Optimized Certificates (OCs) is a solution proposed by Custódio et al. in [38] that provides long-term evidence of authenticity and witnessed existence for signed data objects. This evidence extends the trustworthiness of a data object's signature beyond the lifetimes of the corresponding certificates, timestamps and cryptographic algorithms, thereby ensuring long-term authenticity.

In this solution, the original validation evidence for a data object's signature (namely certificates, their revocations status, and PKIX timestamps) is *summarized* within a single attestation. A reliable party called *notary* produces such an attestation, therewith asserting that: a) the signer's certificate is valid to verify the data object's signature, b) the signed data object is uncorrupted, and c) the existence of the data object's signature was witnessed. The attestation is called *Optimized Certificate (OC)* and is almost a copy of the data object signer's certificate. The OC has the structure of an X.509 certificate to keep compatibility with existing Public Key Infrastructures. The OC contains the signer's public key and name. Its validity period is only the moment the reliable party issued the OC. Therefore it is not revocable. The OC includes the data object's signature and digest and a date on which these data were witnessed. These data are structured as an X.509 Extension. Thus, the OC works as a timestamp for the signed data object. The notaries that issue the OCs have traditional X.509 certificates, which are issued by a Root CA. These certificates can be revoked.

There are three players in this solution: the archivist, who preserves a set of data objects in the archive; the notaries that issue OCs on the archivist's request, and a retriever, who can verify integrity, witnessed existence and authenticity for data objects from the archive using OCs.

Initially, the archivist replaces the validation evidence for a data object' signature by an OC. For each signed data object, the archivist requests an OC from a notary by submitting the signed data object's digest and signature, the signer's certificate chain and revocation status of the certificates in this chain. If there are timestamps, they are also submitted to the notary together with the TSAs' certificate chains and revocation status of the included certificates. The notary returns an OC. The archivist replaces the validation evidence available in the signed data object by the OC and the notary's certificate.

On a regular basis, the archivist monitors the OC's trustworthiness for each signed data object. An OC becomes untrustworthy if the cryptographic algorithms become insecure or the notary's certificate expires. Before an OC becomes untrustworthy, the archivist renews it. The archivist submits the existing OC, the corresponding issuer certificate, and the signed data object's digest, which is calculated with a secure hash function, to a notary. In turn, the notary issues a new OC. The archivist replaces the old OC and the corresponding issuer certificate with the new OC and the notary's certificate.

A notary receives OC requests, which comprise the signed data object's digest and either a) certificate chains, revocation statuses, PKIX timestamps, and the data object's signature or b) an OC and its issuer's certificate. The notary verifies whether certificates are valid and the cryptographic algorithms are still secure. If the verification succeeds, the notary creates an OC. He sets the OC's public key to be the data object signer's public key. He defines the OC's validity period to be the current time. He inserts into the OC a witnessed existence evidence that comprises: a) the data object's signature, which was either directly submitted by the archivist or is taken from the submitted OC; b) the signed data object's digest; and c) a date, which is taken either from a submitted timestamp (the older timestamp, if more than one were submitted), from the submitted OC, or is the current time. The notary signs the OC and returns it.

Given a signed data object, the OC and the notary's certificate, the retriever checks authenticity by verifying if: a) the signature is valid under the public key in the OC, b) the OC's signature is valid under the public key in the notary's certificate, c) the OC's validity period is within the validity period of the notary's certificate, d) the signature on the notary's certificate is valid under the public key in Root CA's certificate, and e) the notary's certificate is neither expired nor revoked. The retriever checks integrity and witnessed existence by verifying if the signed data object's digest and signature equal the digest and signature included in the OC.

Note that updating OCs does not add further data. Rather validation evidence is replaced by new objects created with up to date cryptographic algorithms and certificates. Therefore, the number of objects in the validation evidence is constant.

This solution indeed ensures authenticity, integrity and witnessed existence of a signed data object if a retriever trusts all involved notaries. Trust is required because the archivist discards the original validation evidence, which therefore cannot be verified in the future. Thus, given an OC issued at t_1, i.e. the first OC that is directly replacing the original signer's certificate, a retriever infers that the signer's certificate was valid at t_1, since a trusted notary *properly* verified it at t_1. Like the signer's certificate, the retriever infers the validity for the OCs. That is, for every $i = 2, \ldots, n$, given the i^{th} OC issued at t_i, the retriever infers that the $i - 1^{th}$ OC was valid at t_i. Thus, if the retriever can verify the validity of the current OC, i.e. the n^{th} at t_n, then he infers that the 1^{st} OC issued at t_1 was valid. Consequently, the retriever infers that the signer's certificate was valid at t_1.

There are three assumptions for the OC solution to guarantee long-term integrity, witnessed existence and authenticity of a data object: a) that the underlying PKIs and TSAs are reliable, b) all notaries are reliable, and c) that the cryptographic algorithms are not unexpectedly broken.

4 Capabilities and Requirements

In this section we compare the surveyed solutions in regard to three aspects: a) the security goals that they achieve, b) the assumptions that need to be made in order to achieve such goals, and c) their support for format migration. Table 1 summarizes the security goals and assumptions.

The security goals assessed in this survey are long-term integrity, authenticity and witnessed existence where integrity is the minimum goal. The two other goals include integrity.

Long-term evidence of witnessed existence is achieved by all solutions except LOCKSS. Three of the solutions surveyed (ETSI, ERS and OC) provide long-term authenticity. In all three cases authenticity is achieved in the same way. Data objects are stored together with a digital signature so an evidence

Table 1. The classification of surveyed solutions in regard to achieved security goals and required trust assumptions

		Solutions					
		ETSI	ERS	ACE	CIS	LOCKSS	OC
Security Goals	Integrity	✓	✓	✓	✓	✓	✓
	Witnessed Existence	✓	✓	✓	✓		✓
	Authenticity	✓	✓				✓
Trust Assumptions	PKI	✓	✓				✓
	TSA	✓	✓				✓
	No unexpected break	✓	✓	✓	✓		✓
	Wide-visible media			✓	✓		
	Reliable actors					✓	✓
	External copies					✓	

of witnessed existence of the signed data objects proves authenticity. ACE and CIS provide evidence of witnessed existence. They do not provide authenticity proofs under the given trust assumptions. However, evidence of authenticity could easily be provided, by signing data-objects and a PKI that certifies public keys.

The different solutions require a number of assumptions in order to guarantee long-term protection. We identified them as follows.

– **PKI:** There are reliable Public Key Infrastructures.
– **TSA:** There are reliable Timestamp Authorities.
– **No unexpected break:** Cryptographic algorithms are not unexpectedly broken.
– **Wide-visible media:** There exist trustworthy media, that make data widely observable.
– **Reliable actors:** There are reliable actors other than trusted PKIs and TSAs.
– **External copies:** There exist a number of reliable copies of relevant data objects. The number is a parameter.

In order to analyze the solutions we group them according to their assumptions. ETSI and ERS form one group, ACE and CIS form a second group, while OC and LOCKSS are analyzed separately.

The ETSI and ERS solutions use the same assumptions. They rely heavily on digital signatures, thus, on PKIs. Also, they rely on TSAs and do not tolerate unexpected breaks.

ACE and CIS rely mainly on wide-visible media for long-term protection. They regularly publish a commitment on wide-visible media that allows making the current state of the archive verifiable. They use hash functions in order to make the commitment small and do not tolerate an unexpected break of this hash function. The wide-visible media only allow dating the witnessed existence to a precision of a few weeks. In order to make this more precise, ACE proposes to use TSAs.

OC requires similar assumptions as ETSI and ERS, and, in addition, notaries as additional reliable actors. This leads to efficiency improvements by avoiding the storage of many signatures. LOCKSS relies heavily on the archivists themselves as reliable actors. Also, LOCKS assumes the archivists to be able to compare with reliable copies.

Format obsolescence is not in the focus of this survey. However, it is important for long-term archives. Format migration requires to alter content, thus it may compromise integrity. From the solutions surveyed, only ACE, CIS and LOCKSS provide support for format migration.

5 Efficiency Comparison

In this section we compare the different solutions with respect to their efficiency. In fact, we mainly concentrate on the storage overhead. Additionally, we briefly analyze the archivist's effort to maintain evidence in each solution. This comparison is not conclusive because the surveyed sources not always provide enough details about efficiency.

The ETSI standards produces a significant storage overhead. For each data object, the system stores a chain of timestamps that grows over time. In addition, the system stores validation evidence for each timestamp signature. Troncoso et al. argue in [39] that this overhead is affordable, based on Moore's [40] and Kryder's [41] laws. The archivist has to maintain a separate timestamp chain for each data object, adding new timestamps whenever necessary. According to [42], the ETSI standards "do not provide scalable and cost efficient solutions for long-term archiving, because the timestamp renewal would require a new qualified timestamp for each archive object".

ERS requires roughly the same storage overhead as ETSI. The Merkle tree construction allows for a single timestamp to protect several data objects. However, the same timestamp is replicated in each data object. Compared to ETSI, the archivist needs to request fewer timestamps. On the other hand, the archivist needs to maintain the Merkle trees. New data objects cannot be inserted as leaves into an existing and timestamped Merkle tree. Therefore, a new Merkle tree has to be created and timestamped whenever new data objects are to be archived. As a consequence, in a long-term scenario there would be several Merkle trees that will require regular and individual renewals of timestamps and Merkle trees.

CIS offers some advantages in storage overhead over ETSI and ERS. Despite similarities with ETSI, the use of timestamps based on wide-visible media has the advantage that there is no need to store validation evidence. For a single data object, CIS potentially requires fewer timestamps than the ETSI and ERS solutions. The reason is that timestamps in CIS rely only on the security of hash functions only and no PKIX timestamps are needed. However, in terms of management, both the CIS and ETSI solutions require similar effort by the archivist who needs to maintain a chain of timestamps for each data object.

ACE produces less storage overhead than ETSI and ERS. The evidence of integrity stored in a single IT is presumably smaller than a PKIX timestamp because there is no need to store validation evidence for signatures (namely certificates and revocation status). Moreover, one would expect fewer ITs for a single object than PKIX timestamps in ETSI and ERS. This is because IT relies only on hash functions. The witness values that are stored on wide-visible media are also relatively small. The authors of ACE estimate that, if IMS produces one witness value per day, the total size of all the witness values over a year is around 100 KB. The archivist's effort to maintain the ITs for each data object is similar to ERS and CIS. There is, however, an extra burden for IMS, which stores Merkle tree roots for each round. The authors implemented a prototype and show the results in [33].

For LOCKSS to work, the archived data objects must be replicated across archives. Therefore, LOCKSS has a huge storage overhead. Also, archives must invest considerable computational resources to run polls on a regular basis. The authors of LOCKSS claim that acceptable performance can be achieved at a low cost and they give explicit values in [36].

OC has very moderate storage requirements that, in contrast to the other surveyed solutions, does not depend on the number of updates on the validation evidence. This is because there is no accumulation of evidence, rather old validation evidence is replaced by new one using up to date cryptographic algorithms and certificates. In terms of management, OC is equal to ETSI, CIS and ACE. That is, the archivist needs to monitor each data object's OC and request a new one from a notary in a timely manner.

6 Solutions that Fail to Provide Long-Term Protection

For completeness, we present a number of solutions that fail to provide long-term authenticity, integrity and witnessed existence and we briefly explain why they fail to be secure in the long term. They are interesting because they introduce extra protection goals and use different approaches. For further examples we refer the reader to [43].

Authentic Timestamps [44] is a solution for archival storage. Besides integrity and witnessed existence it also produces evidence of *non existence* for data objects. The authors introduce a data structure that implements an append-only, persistent, authenticated dictionary by combining ideas from Merkle and Patricia trees. The solution uses PKIX timestamps (see Section 2.7) for witnessed existence. The scheme does not address the weakening of hash algorithms or timestamp signatures hence it does not provide long-term protection.

Key Archive Service and Timestamp (KASTS) is an archiving system proposed by Maniatis and Baker in [45]. The system aims at extending the validity of digital signatures beyond the expiration dates of the corresponding certificates. It does so by timestamping a signature at the time it is produced and archiving old signature verification keys. Verification keys are stored by a trusted third party named *Key Archive Service*. Validation evidence is structured in authenticated search trees [46] and only the roots of the trees are timestamped. KASTS keeps only the most recent timestamp, thus a retriever lacks evidence to evaluate the chronological order of authenticated search trees. KASTS does not address the weakening of the hash algorithms used to construct the authenticated search trees. Therefore, KASTS does not provide long-term protection.

Certified Accountable Tamper-Evident Storage (CATS) is a network storage system proposed by Yumerefendi and Chase in [47]. It provides evidence of witnessed existence, non existence, integrity and authenticity for data objects. The evidence is based on authenticated search trees, digital signatures (see Section 2.5) and wide-visible media (see Section 2.4). The focus of CATS is on providing means of accountability for the players of an archival solution. A CATS server maintains a shared directory of data objects where users can read and write. Clients can verify that the server is faithful and cannot deny their operations. CATS does not address the obsolescence of cryptographic algorithms, hence it does not provide long-term protection.

Maniatis et al. propose in [48] to build a network of Timestamp Authorities called Prokopius. They use a peer-to-peer archival storage network as their wide-visible medium to commit timestamps. This allows clients to verify timestamps issued by formerly trusted TSAs, even if they are no longer in service. Such TSA can be used by an archivist to provide protection for data objects. The solution is based on timestamps, Merkle trees and Byzantine Fault Tolerant protocols[6]. Although Prokopius increases the reliability of traditional timestamps, it does not address long-term protection as defined in this survey. In particular it does not deal with the obsolescence of cryptographic algorithms, which will compromise the evidences of authenticity and witnessed existence.

Victorian Electronic Record Strategy (VERS) is a solution for long-term protection of digital data developed by a team commissioned by the Australian government [50,51]. VERS is a prototype intended to test archiving techniques to maintain government records. The solution provides evidence of integrity and authenticity for data objects. The solution is based on digital signatures (see Section 2.5). An important design decision of VERS is not to use Public Key Infrastructures (see Section 2.6). The authors argue that CAs are likely to disappear during the archival period of data objects and, therefore, it is better not rely on certificates. Without certificates and revocation data, there is no proof that binds the signer's identification and public key. Alternatively, the authors suggest to confirm the binding between the signer's identification and public key by: a) comparing the public key with other records signed by the same signer, and b) the archivist maintaining a signed certification record that binds the identification of an authorized user in VERS and the corresponding public key. VERS allows authorized users to resign an archived data object, but not its signatures. Without resigning signatures, the obsolescence of cryptographic algorithms is not addressed. As a consequence, VERS fails to provide long-term protection.

[6] The Byzantine failure assumption [49] models systems whose components may behave in unexpected ways due to hardware failures, network congestion or malicious attacks. Byzantine failure-tolerant algorithms must cope with such failures.

7 Conclusion and Future Work

We surveyed solutions for long-term protection of digital data, describing and comparing them in regard to their protection goals and trust assumptions. A brief comparative analysis of the solutions' efficiency was also provided. Additionally, we identified a number of archival solutions that fail to provide long-term protection, and showed why they fail.

A number of open problems arise from our analysis. First, there are desirable properties that are not covered by the long-term protection solutions. One example is evidence of non-existence of a document at a certain point in time. Next, it would be desirable to know how trust assumptions can be reduced or adjusted to meet the requirements of a particular archive. Also, it would be interesting to perform a rigorous analysis of the long-term protection solutions using the recently developed security models [52,53]. In addition, a comprehensive efficiency analysis would allow comparing the surveyed solutions. This analysis should in particular include scalability issues.

Given the information presented in this survey we come to the preliminary conclusion that CIS and ACE appear to be the best solutions. Both can address all protection goals that we discussed in this paper, require fewer trust assumptions than other solutions, and support format migration. However, there is a caveat: it is unclear whether the wide-visible media assumption is plausible in the long term. In terms of efficiency, the storage overhead in CIS and ACE seems lower than ETSI, ERS and LOCKSS. Nevertheless, OC excels in this measure, at the cost of the stronger trust assumption of a notary.

Cryptographers would argue that replacing trust in actors with mathematical assumptions would make the overall security much more convincing. More generally, finding solutions that also address long-term confidentiality, other possible protection goals, and the balance between cryptographic protection and reliance on trusted parties appears to be an interesting and important research direction.

8 Acknowledgments

Martín A. G. Vigil is supported by *Coordenação de Aperfeiçoamento de Pessoal de Nível Superior* (CAPES/Brazil).

References

1. Ministère de la Justice: Livre Foncier
2. Centre of Registers and Information Systems: e-Land Register
3. : Council Directive 2001/115/EC of 20 December 2001 amending Directive 77/388/EEC with a view to simplifying, modernising and harmonising the conditions laid down for invoicing in respect of value added tax. Official Journal L 015 , 17/01/2002 P. 0024 - 0028
4. Tax Administration of the Republic of Slovenia: Electronic Tax Operations (2012)
5. European Comission: VAT in the European Community

6. Bundesärztekammer: (Muster-)Berufsordnung für die in Deutschland tätigen Ärztinnen und Ärzte (2011)
7. Bundesregierung der Justiz: Verordnung über den Schutz vor Schäden durch Röntgenstrahlen (Röntgenverordnung - RöV) (1987)
8. Department of Health: Annex D1: Health Records Retention Schedule
9. Intellectual Property Office: Retention and Disposal Policy for Patent Related Records (jun 2012)
10. Braun, J., Buchmann, J., Mullan, C., Wiesmaier, A.: Long term confidentiality: a survey. Designs, Codes and Cryptography (2012) Accepted for publication, to appear. Preliminary version: Cryptology ePrint Archive, Report 2012/449.
11. Wallace, C., Pordesch, U., Brandner, R.: Long-Term Archive Service Requirements. RFC 4810 (Informational) (March 2007)
12. Duranti, L.: Continuity and transformation in the role of the archivist: The findings of the interpares project. Journal of the Japan Society for Archival Science **3**(6) (2007) 27–41
13. Menezes, A.J., Vanstone, S.A., Oorschot, P.C.V.: Handbook of Applied Cryptography. 1st edn. CRC Press, Inc., Boca Raton, FL, USA (1996)
14. Stevens, M., Sotirov, A., Appelbaum, J., Lenstra, A., Molnar, D., Osvik, D., de Weger, B.: Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In Halevi, S., ed.: Advances in Cryptology - CRYPTO 2009. Volume 5677 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2009) 55–69
15. Merkle, R.C.: A Certified Digital Signature. In Brassard, G., ed.: Advances in Cryptology - CRYPTO '89. Volume 435 of Lecture Notes in Computer Science., Springer (1989) 218–238
16. Bayer, D., Haber, S., Stornetta, W.: Improving the efficiency and reliability of digital time-stamping. Sequences II: Methods in Communication, Security, and Computer Science (1993) 329–334
17. Benaloh, J., de Mare, M.: Efficient broadcast time-stamping. Clarkson University Department of Mathematics and Computer Science TR (1991) 91–1
18. Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM **21**(2) (1978) 120–126
19. Gamal, T.E.: On computing logarithms over finite fields. In Williams, H.C., ed.: CRYPTO. Volume 218 of Lecture Notes in Computer Science., Springer (1985) 396–402
20. Johnson, D., Menezes, A., Vanstone, S.A.: The elliptic curve digital signature algorithm (ecdsa). Int. J. Inf. Sec. **1**(1) (2001) 36–63
21. Kohnfelder, L.M.: Towards a practical public-key cryptosystem. Technical report, Massachusetts Institute of Technology (1978)
22. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008)
23. Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard) (June 1999) Updated by RFC 6277.
24. Adams, C., Cain, P., Pinkas, D., Zuccherato, R.: Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161 (Proposed Standard) (August 2001) Updated by RFC 5816.
25. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Advances in Cryptology Crypto 90 **3**(2) (1991) 437–455

26. Lenstra, A.K., Verheul, E.R.: Selecting cryptographic key sizes. J. Cryptology **14**(4) (2001) 255–293
27. Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen). (2010)
28. National Institute of Standards and Technology: Recommendation for Key Management – Part 1: General (Revised). (2007)
29. ETSI: XML Advanced Electronic Signatures (XAdES). 1.8.3 edn. Number TS 101 903. (jan 2010)
30. ETSI: CMS Advanced Electronic Signatures (CAdES). 1.7.4 edn. Number TS 101 733. (Jul. 2010)
31. Bayer, D., Haber, S., Stornetta, S.: Improving the efficiency and reliability of digital time-stamping. In: Sequences II: Methods in Communication, Security, and Computer Science (Proceedings of the Sequences Workshop, 1991), Springer-Verlag, New York (1993) 329–334
32. Gondrom, T., Brandner, R., Pordesch, U.: Evidence Record Syntax (ERS) (2007)
33. Song, S., JaJa, J.: Techniques to audit and certify the long-term integrity of digital archives. International Journal on Digital Libraries **10**(2-3) (2009) 123–131
34. Haber, S.: Content Integrity Service for Long-Term Digital Archives. In: Archiving 2006, Ottawa, Canada, IS&T (2006) 159–164
35. Haber, S., Kamat, P., Kamineni, K.: A content integrity service for digital repositories. (2008)
36. Maniatis, P., Roussopoulos, M., Giuli, T.J., Rosenthal, D.S.H., Baker, M.: The LOCKSS peer-to-peer digital preservation system. ACM Transactions on Computer Systems **23**(1) (2005) 2–50
37. Dwork, C., Naor, M.: Pricing via processing or combatting junk mail. In Brickell, E., ed.: Advances in Cryptology - CRYPTO 92. Volume 740 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (1993) 139–147
38. Custódio, R.F., Vigil, M.A.G., Romani, J., Pereira, F.C., da Silva Fraga, J.: Optimized Certificates - A New Proposal for Efficient Electronic Document Signature Validation. In Mjø lsnes, S.F., Mauw, S., Katsikas, S.K., eds.: EuroPKI. Volume 5057 of Lecture Notes in Computer Science., Springer (2008) 49–59
39. Troncoso, C., De Cock, D., Preneel, B.: Improving secure long-term archival of digitally signed documents. In: Proceedings of the 4th ACM international workshop on Storage security and survivability. StorageSS '08, New York, NY, USA, ACM (2008) 27–36
40. Moore, G., et al.: Cramming more components onto integrated circuits. Proceedings of the IEEE **86**(1) (1998) 82–85
41. Walter, C.: Kryder's law. Scientific American **293**(2) (2005) 32
42. Huehnlein, D., Korte, U., Langer, L., Wiesmaier, A.: A Comprehensive Reference Architecture for Trustworthy Long-Term Archiving of Sensitive Data. In: New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on. (2009) 1–5
43. Storer, M.W., Greenan, K., Miller, E.L.: Long-term threats to secure archives. In: Proceedings of the second ACM workshop on Storage security and survivability, ACM (2006) 9–16
44. Oprea, A., Bowers, K.D.: Authentic Time-Stamps for Archival Storage (2009)
45. Maniatis, P., Baker, M.: Enabling the Archival Storage of Signed Documents. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies. FAST '02, Berkeley, CA, USA, USENIX Association (2002)

46. Buldas, A., Laud, P., Lipmaa, H.: Accountable certificate management using undeniable attestations. In: Proceedings of the 7th ACM conference on Computer and communications security, ACM (2000) 9–17
47. Yumerefendi, A.R., Chase, J.S.: Strong accountability for network storage. Trans. Storage **3**(3) (2007)
48. Giuli, P., Maniatis, P., Giuli, T., Baker, M.: Enabling the Long-Term Archival of Signed Documents through Time Stamping. In: Proceedings of the 1st USENIX Conference on File and Storage Technologies. FAST '02, USENIX Association (2002) 31–46
49. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3) (1982) 382–401
50. Victoria - Public Record Office: Victorian Electronic Records Strategy Final Report. 4 edn. (1998)
51. Waugh, A., Wilkinson, R., Hills, B., Dell'oro, J.: Preserving digital information forever. In: Proceedings of the fifth ACM conference on Digital libraries. DL '00, New York, NY, USA, ACM (2000) 175–184
52. Müller-Quade, J., Unruh, D.: Long-term security and universal composability. In Vadhan, S., ed.: Theory of Cryptography. Volume 4392 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2007) 41–60
53. Canetti, R., Cheung, L., Kaynar, D., Lynch, N., Pereira, O.: Modeling computational security in long-lived systems. In van Breugel, F., Chechik, M., eds.: CONCUR 2008 - Concurrency Theory. Volume 5201 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 114–130