Trustworthy and Efficient Protection Schemes for Digital Archiving

Vom Fachbereich Informatik der

Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades

Doktor-Ingenieur (Dr.-Ing.)

von

MSc. Martín Augusto Gagliotti Vigil

geboren in Florianópolis, Brasilien.

Referenten: Prof. Dr. Johannes Buchmann
Prof. Dr. Ricardo Custódio

Tag der Einreichung: 28.05.2015
Tag der mündlichen Prüfung: 14.07.2015

Hochschulkennziffer: D17

Darmstadt 2015

List of Publications

[1] Johannes Braun, Andreas Hülsing, Alexander Wiesmaier, Martín Vigil, and Johannes A. Buchmann. How to avoid the breakdown of public key infrastructures - forward secure signatures for certificate authorities. In *Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, pages 53–68, 2012.

[2] Martín Vigil, Daniel Cabarcas, Johannes Buchmann, and Jingwei Huang. Assessing trust in the long-term protection of documents. In *2013 IEEE Symposium on Computers and Communications (ISCC 2013)*, pages 185–191, 2013. Cited on pages 29 and 68.

[3] Martín Vigil and Ricardo Custódio. Cleaning up the PKI for Long-Term Signatures. In *XII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 140–153, 2012. Cited on page 109.

[4] Martín Vigil, Cristian Thiago Moecke, Ricardo Felipe Custódio, and Melanie Volkamer. The notary based PKI - A lightweight PKI for long-term signatures on documents. In *Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, pages 85–97, 2012. Cited on page 68.

[5] Martín Vigil, Christian Weinert, Kjell Braden, Denise Demirel, and Johannes Buchmann. A performance analysis of long-term archiving techniques. In *2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems*, pages 878–889, 2014. Cited on page 47.

[6] Martín Vigil, Christian Weinert, Denise Demirel, and Johannes Buchmann. An efficient time-stamping solution for long-term digital archiving. In *IEEE 33rd In-*

ternational Performance Computing and Communications Conference (IPCCC 2014), pages 1–8, 2014. Cited on page 89.

[7] Martín Vigil, Johannes Buchmann, Daniel Cabarcas, Christian Weinert, and Alexander Wiesmaier. Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. *Computers & Security*, 50(0):16–32, 2015. Cited on pages 19, 29, and 47.

[8] Felipe Carlos Werlang, Ricardo Felipe Custódio, and Martín Vigil. A user-centric digital signature scheme. In *Public Key Infrastructures, Services and Applications - 10th European Workshop, EuroPKI 2013, Egham, UK, September 12-13, 2013, Revised Selected Papers*, pages 152–169, 2013.

Abstract

The amount of information produced in the last decades has grown notably. Much of this information only exists in the form of electronic documents and it has often to be stored for long periods. Therefore, digital archives are increasingly needed. However, for the documents to remain trustworthy while they are archived, they need to be protected by the archivists. Important protection goals that must be guaranteed are *integrity*, *authenticity*, *non-repudiation*, and *proof of existence*.

To address these goals, several protection schemes for digital archives have been designed. These schemes are usually based on cryptographic primitives, namely digital signatures and hash functions. However, since documents can be archived for decades or even indefinitely, the used cryptographic primitives can become insecure during the archival time. This is a serious issue because it can be exploited by attackers to compromise the protection goals of the archived documents. Therefore, a requirement for *long-term protection schemes* is to address the *aging of cryptography*, i.e. replacing the used primitives properly before they become insecure.

In this work we analyze and improve long-term protection schemes for digital archives. More precisely, we aim at answering three questions. (1) How do long-term protection schemes compare with respect to trustworthiness? (2) How do they differ in performance? (3) Can new schemes be designed, which generate more efficient and trustworthy evidence needed to establish the protection goals?

Although several protection schemes can be found in the literature, many of them fail in addressing the aging of cryptography. Therefore, our first step is to identify which existing schemes provide long-term protection with respect to integrity, authenticity, non-repudiation, and proof of existence.

Afterwards, to answer question (1) we analyze the trustworthiness of the long-term protection schemes using two approaches. In the first approach, we initially identify the required trust assumptions. Then, based on these assumptions, we compare the protection schemes.

In the second approach, we turn to quantifying the trustworthiness of the evidence generated by time-stamping and notarial schemes. To this end, we use a belief trust

model and design a reputation system. This leads to two further, more detailed answers to question (1). First, that trustworthiness depends on the reputation of the involved parties rather than the protection schemes themselves. Second, the trustworthiness of evidence tends to degrade in the long term. Therefore, we propose to use the reputation system to create incentives for the involved parties to build good reputation. This raises the trustworthiness of generated evidence, hence addressing question (3).

Next, we address question (2) by analyzing how schemes differ in performance using an analytical evaluation and experiments. More precisely, we measure the times needed to create and verify evidence, the space required to store evidence, and the communication necessary to generate evidence. Moreover, this analysis shows that while verifying evidence most of the time is spent on checking certificate chains.

The findings in the performance analysis provide us with directions for addressing question (3). We propose three new solutions that provide more efficient evidence. The first solution is a new notarial scheme that generates smaller evidence and that communicates less data than the existing notarial scheme. Novelties in our scheme include balancing the numbers of signatures that users and notaries verify, and using notaries as time-stamp authorities to provide proof of existence.

The second solution is based on the time-stamping scheme *Content Integrity Service* (CIS) and allows for faster evidence verification. To the best of our knowledge, CIS is the only scheme designed for an archive where documents are submitted and time-stamped sequentially but share the same sequence of time-stamps. However, in this case the validities of several time-stamps in this sequence may overlap. Consequently, many of these time-stamps need not be checked when verifying the time-stamp sequence for one document. We address this issue in our new scheme by using a data structure called *skip list*. The result is a time-stamp sequence where users can *skip* the time-stamps that are not necessary to guarantee the protection goals of one document. Using an analytical evaluation and experiments, we show that our scheme is notably faster than CIS.

The third solution is intended to reduce time spent on checking certificate chains when verifying evidence generated by time-stamping schemes. More precisely, we improve an existing public key infrastructure-based solution where the root certification authority generates smaller verification information for time-stamps. This verification information can be used to replace the certificate chains needed to verify time-stamps. However, this solution requires extra work from time-stamp authorities and the root certification authority, especially when the number of time-stamps grows significantly. In our solution, this issue is addressed such that this extra work

is independent of the number of time-stamps. Using an analytical evaluation we demonstrate the advantage of our solution.

Finally, we provide our conclusions and future work. In this thesis we design new solutions that allow for more efficient and trustworthy evidence of protection for archived documents. As future work, we suggest conducting more research in the direction of developing methods that address the decay of the trustworthiness of evidence over time.

Zusammenfassung

Ein großer Teil der in den letzten Jahrzehnten erzeugten Daten existiert nur noch in elektronischer Form und muss für einen langen Zeitraum gespeichert werden. Aus diesem Grund ist die Entwicklung sicherer elektronischer Archive wichtig. Dabei ist es erforderlich die Integrität, Authentizität und Nichtabstreitbarkeit von Dokumenten zu gewährleisten sowie einen Beweis für deren Existenz zu liefern.

Um diese Schutzziele zu erfüllen wurden verschiedene Verfahren entwickelt. Diese erzeugen in der Regel entsprechende Beweise mit Hilfe von kryptographischen Primitiven, wie beispielsweise digitale Signaturen oder Hashfunktionen. Allerdings müssen Dokumente oft für mehrere Jahrzehnte oder sogar für einen unbegrenzten Zeitraum aufbewahrt werden. Durch diese lange Speicherung der Dokumente ist anzunehmen, dass die Sicherheit der verwendeten kryptographischen Primitive nicht für den gesamten Zeitraum gewährleistet werden kann. Dies ist ein ernst zu nehmendes Risiko für die Langzeitsicherheit von archivierten Dokumenten, da es Angreifern die Möglichkeit bietet ein oder mehrerer Schutzziele zu verletzen. Aus diesem Grund ist eine wichtige Anforderung an die verwendeten Verfahren auch das Altern der eingesetzten kryptographischen Primitive zu berücksichtigen. Ein möglicher, häufig verwendeter, Ansatz ist die kryptographischen Primitive regelmäßig durch neuere und sicherere zu ersetzen und die entsprechenden Beweise zu aktualisieren, bevor die Sicherheit der alten Beweise gebrochen werden kann.

Diese Arbeit analysiert und verbessert Verfahren, die für die Gewährleistung von Langzeitsicherheit digitaler Archive entwickelt wurden. Dabei werden insbesondere drei Forschungsfragen adressiert: (1) Wie unterscheiden sich die einzelnen Verfahren bezüglich ihrer Vertrauensannahmen? (2) Wie unterscheiden sich die einzelnen Verfahren bezüglich ihrer Effizienz? (3) Könne neue effizientere und vertrauenswürdigere Verfahren entwickeln werden?

Obwohl in den letzten Jahren bereits mehrere Verfahren zum Schutz von Dokumenten in Archiven entwickelt wurden adressieren viele nicht das Altern der verwendeten kryptographischen Primitive und sind daher nicht für die Langzeitarchivierung geeignet. Aus diesem Grund wird in dieser Arbeit zunächst identifiziert, welche

Systeme die Integrität, Authentizität und Nichtabstreitbarkeit von Dokumenten auf Dauer sicherstellen und einen entsprechenden Existenzbeweis liefern.

Danach, um Forschungsfrage (1) zu beantworten, werden die Vertrauensannahmen der einzelnen Schutzmechanismen analysiert. Hier verfolgt diese Arbeit zwei Ansätze. Zum einen werden die benötigten Vertrauensannahmen identifiziert und anschließend, basierend auf diesen Annahmen, die existierenden Verfahren miteinander verglichen.

Zum anderen wird die Vertrauenswürdigkeit in die zur Gewährleistung der Schutzziele verwendeten Beweise quantifiziert. Hierbei konzentriert sich diese Arbeit auf Verfahren, die Zeitstempel oder eine notarielle Beglaubigung zur Erstellung der Beweise verwenden. Dafür wird ein Reputationssystem vorgeschlagen, welches die Vertrauenswerte mit Hilfe eines so genannte „belief trust models“ berechnet. Dieser Ansatz erlaubt zwei detaillierte Antworten auf Frage (1) zu geben. Erstens, dass die Vertrauenswürdigkeit eines Beweises von der Reputation der bei der Erzeugung involvierten Parteien abhängt und nicht von dem verwendeten Schutzmechanismus selbst. Zweitens, dass die Vertrauenswürdigkeit des erzeugten Beweises abnimmt, je häufiger dieser aktualisiert wird. Darüber hinaus animiert das entwickelte Reputationssystem die involvierten Parteien sich vertrauenswürdig zu verhalten, um eine gute Reputation aufzubauen. Dies erhöht auch indirekt die Vertrauenswürdigkeit in die Gewährleistung der Schutzziele und adressiert somit ebenfalls Frage (3).

Anschließend widmet sich die Arbeit Frage (2), indem mit Hilfe einer analytischen Evaluation und entsprechenden Laufzeitexperimenten analysiert wird, wie sich die einzelnen Verfahren bezüglich der Effizienz unterscheiden. Dabei wird zum einen die Zeit gemessen, die benötigt wird um einen Beweis zu erzeugen und zu verifizieren. Zum anderen werden auch der Speicherbedarf der Beweise und die Komplexität der Kommunikation beim Erzeugen der Beweise ermittelt. Diese Analyse hat unter anderem gezeigt, dass bei der Verifizierung eines Beweises die Verifizierung der Zertifikatsketten den höchsten Rechenaufwand hat.

Auf diesem und anderen Ergebnissen der Effizienzanalyse aufbauend wurde Frage (3) adressiert. Um die Effizienz aktueller Verfahren zu verbessern werden drei neue Verbesserungsansätze vorgeschlagen. Der erste Ansatz ist ein neues notarielles Verfahren, welches kürzere Beweise erzeugt und einen geringeren Kommunikationsaufwand hat, als die bestehenden Ansätze. Eine wichtige Neuerung dieses Verfahrens ist eine Balance zwischen der Anzahl von Signaturen, die Benutzer verifizieren und denen die Notare überprüfen müssen zu schaffen. Darüber hinaus werden Notare als Zeitstempeldienste eingesetzt.

Die zweite Lösung baut auf dem Content Integrity Service (CIS) auf, bietet jedoch einen effizienteren Verifizierungsprozess von Beweisen. CIS ist unserem Kenntnis-

stand nach das einzige Verfahren für Archive, das sequentiell erzeugte und gespeicherte Dokumente mit Hilfe eines einzelnen Beweises schützt. Dieser besteht aus einer Sequenz von Zeitstempeln. Daher kann es bei einer sequentiellen Generierung und Archivierung von Dokumenten dazu kommen, dass Zeitstempel für diese generiert werden, deren Gültigkeitsdauer sich aber überschneidet. Bei der Verifizierung wird jedoch standardmäßig die Korrektheit aller Zeitstempel überprüft, obwohl dies bei überschneidungen nicht nötig ist. Das in dieser Arbeit vorgestellte neue Verfahren adressiert diesen Aspekt, indem eine Datenstruktur in Form von Skip-Listen zur Verwaltung der Zeitstempel eingesetzt wird. Dadurch werden bei der Verifizierung nur die Zeitstempel ausgewählt und überprüft, die entscheidend für die Gewährleistung der Schutzziele sind. Basierend auf einer analytischen Evaluation und Laufzeitexperimenten zeigen wir, dass der Verifizierungsprozess des in diese Arbeit vorgestellten Verfahrens deutlich effizienter ist als der von CIS.

Die dritte Lösung hat als Ziel den Aufwand beim überprüfen der Zertifikatsketten für die einzelnen Zeitstempel signifikant zu reduzieren. Ein Ansatz ist, dass die Autoritäten von Wurzelzertifikaten kompaktere Informationen für die Verifikation erzeugen und diese anschließend verwendet werden, um die langen Zertifikatsketten zu ersetzen. Dieser Ansatz bedeutet jedoch nicht nur einen Mehraufwand für die Zertifizierungsauthoritäten, sondern auch für die Zeitstempeldienste, die die Zeitstempel erzeugen. Erschwerend kommt hinzu, dass dieser Aufwand in Abhängigkeit von der Anzahl von Zeitstempel wächst. Die in dieser Arbeit vorgestellte Lösung verbessert diesen Ansatz, so dass der zusätzliche Aufwand unabhängig von der Anzahl der Zeitstempel ist. Die Vorteile dieser Lösung werden anhand einer analytischen Evaluation demonstriert.

Abschließend bietet diese Arbeit eine Zusammenfassung und einen Ausblick auf weitere Forschungsfragen. Im Rahmen dieser Arbeit wurde eine Vielzahl neuer Lösungen für Verfahren zum Schutz der Langzeitsicherheit von Dokumenten entwickelt, welche die aktuellen Verfahren bezüglich Effizienz und Vertrauenswürdigkeit verbessern. Eine mögliche zukünftige Forschungsaufgabe ist beispielsweise Methoden zu entwickeln, die den Verlust der Vertrauenswürdigkeit bei der Langzeitspeicherung adressieren.

Contents

1 Introduction

1.1 Motivation

In the last decades, the amount of information produced in digital form has grown notably. People, organizations, and machines have created more and more digitally-based contents. Turner et al. [50] estimate that the size of this *digital universe* should double every two years until 2020, reaching 44 zettabytes[1].

Since a large amount of this data only exists electronically, digital archives are often needed to securely and efficiently store this information for long periods of time. Moreover, digital archives have been also used to preserve data migrated from other types of media, such as paper and radio broadcast. Examples of organizations that use or will use digital archives are the following:

- *the Estonian land register*, which contained permanent digital records of about one million Estonian properties in 2014 [26];
- *the Irish Tax and Custom*, which stored 6.7 millions electronic tax returns only in 2013 [49];
- *the National Library of Norway*, which archived about 1 exabyte[2] of books, journals, newspapers, and webpages in 2007 [33];
- *the National Health Service* in the United Kingdom, which should adopt electronic health records by 2018 [13]. These records must be retained for up to 20 years [14].

The misuse of data stored in archives has always been a serious issue. For example, corrupted electronic medical records have misled physicians into prescribing wrong treatments that harmed or even killed patients [44]. Forged documents claimed

[1] 1 zettabyte is 2^{70} bytes.
[2] 1 exabyte is 2^{60} bytes.

to be from land registers have been used by criminals to carry out frauds [22]. Corporations have been involved in financial scandals, where the date of a stock option is forged to make it more valuable [9]. Moreover, dictatorships have tampered with historical documents in order to censor information [27].

To prevent these issues, digital archives must guarantee the *long-term* protection of their data, i.e. the *archived documents*. This means that such documents must be protected as long as they are in the archives. This time period may range from a few decades to many generations, for example, in the case of land registers. To cover long-term protection requirements, techniques are needed that provide everlasting protection. We will in this work therefore use the terms long-term and everlasting synonymously.

Important protection goals are *integrity*, *authenticity*, *non-repudiation*, *proof of existence*, and *confidentiality*. However, providing these protection goals for archived documents is challenging. For instance, digital signatures are often used to provide authenticity for digital documents. But, digital signatures become insecure when their security properties are defeated by advances in computer power and cryptanalytic techniques. For example, today's attacks can defeat the security of 512-bit RSA signatures [10] which were considered secure in 1990. Therefore, single digital signatures cannot provide long-term protection.

Numerous solutions have been designed to help archivists guarantee long-term protection for documents. However, only few solutions can achieve the protection goals in the presence of aging cryptographic primitives. Although this is not a new topic, no comparative analysis of all schemes achieving the protection goals in the long term has been provided so far. Such an analysis could help archivists to select the protection scheme that meets their needs best. Moreover, selecting the appropriate scheme at the very beginning can be important for archivists. The reason is that, since schemes generate distinct evidence, it may be hard to replace a scheme after it has been employed.

Furthermore, the long-term protection schemes should be analyzed with respect to non-security properties. Two very important properties that have not received the attention they deserve are trustworthiness and performance. The trust assumptions and correspondingly the trustworthiness of these schemes vary. Moreover, to determine trustworthiness of the evidence generated for an archived document, further questions still have to be answered. How reliable were the parties involved in each protection scheme at the time evidence was created? Is the current reputation of the involved parties important for the trustworthiness of evidence created far in the past? Does the number of involved parties affect the trustworthiness of the generated evidence? Answers to these questions not only allow to select the appropriate

protection schemes, but also help to predict whether the evidence will be accepted as trustworthy by future users.

Finally, a performance analysis is desirable when selecting schemes because they may perform differently under distinct scenarios. For example, digital archives where documents are submitted in batches can benefit from schemes that produce a single evidence for a batch of documents rather than for each document individually. In other cases, however, documents can be submitted sequentially or updated frequently, thereby requiring a different solution. After selecting the protection scheme, the performance analysis also helps digital archives and involved trusted parties to allocate the necessary resources. Furthermore, an analysis that indicates where performance can be further improved is helpful to design more efficient solutions.

1.2 Contributions

The focus of this work is on analyzing and improving schemes that provide the protection goals integrity, authenticity, non-repudiation, and proof of existence for archived documents in the long term. Confidentiality is not covered here because it requires solutions that are very different from those used to achieve the other protection goals. For instance, the solutions for confidentiality use a distinct type of cryptographic primitives and generate no evidence that can be used to demonstrate that this protection goal is achieved. An overview of solutions that guarantee long-term confidentiality is provided by Braun et al. [4].

Therefore, we first survey in Chapter 3 the protection schemes that guarantee one or more of the aforementioned protection goals. The result is an overview, divided into two parts. The first part contains the schemes that achieve these protection goals in the long term. The second part presents some schemes that cannot provide long-term protection, showing why they fail. More precisely, the long-term protection schemes described in the first part can be classified into three groups. The first group contains *time-stamping schemes*, to which the solutions *Advanced Electronic Signatures* (AdES), *Content Integrity Service* (CIS), and *Evidence Record Syntax* (ERS) belong. The second group contains *notarial schemes*, where we describe the solution *Cumulative Notarizations* (CN). The third group is *replication schemes*, to which *Lots of Copies Keep Stuff Safe* (LOCKSS) belongs. We describe these five schemes systematically so that they can be compared with respect to trustworthiness and performance.

Next, we analyze the long-term protection schemes with respect to trustworthiness and performance. In the trustworthiness analysis found in Chapter 4, we assess

to what extent retrievers can trust these schemes to guarantee the protection goals for an archived document. This analysis consists of two parts. The first part is a qualitative analysis, where we compare the schemes as to the required trust assumptions. This analysis indicates that the time-stamping and notarial schemes tend to be more trustworthy than LOCKSS. The reason is that, contrary to LOCKSS, time-stamping and notarial schemes provide evidence that can be used by any retriever to check whether the protection goals are indeed fulfilled for an archived document.

The second part is a quantitative analysis, where we approximate the trustworthiness of the evidence generated by the time-stamping and notarial schemes. To this end, we use a *belief trust model* which allow us to estimate how likely evidence is to indeed guarantee the promised protection goals. Since this trust model uses as input the reputation of the parties involved in the generation of evidence, we propose a long-term reputation system. In this reputation system, participants verify evidence and use the verification result to rate the involved parties. By analyzing the trust model and the reputation system, we draw two important observations. First, that if the evidence for a document is updated for several years, involving many parties, then its trustworthiness tends to degrade. Second, that the reputation system allows for using incentives for the involved parties to build good reputation. This helps to raise the trustworthiness of involved parties and, consequently, the trust retrievers put in the generated evidence.

Performance is addressed in Chapter 5 by a performance analysis of the time-stamping and notarial schemes. They are analyzed with respect to time, space, and communication complexity. Time refers to the durations needed to initialize, update, and verify evidence. Space refers to the size of the generated evidence. Communication complexity refers the size of the messages exchanged while generating evidence. Our performance analysis has two parts. In the first part we evaluate performance analytically, that is, without taking specific cryptographic primitives into account. In the second part, we evaluate performance empirically by carrying out experiments with prototypical implementations. More precisely, we simulate the protection of documents in an archive for 100 years using well-known cryptographic primitives and distinct signature lifetimes. The analysis shows interesting results. First, the notarial scheme CN is the most time-efficient scheme at the cost of the highest communication complexity. Second, ERS is the fastest time-stamping scheme when generating evidence because it hashes the minimum data required to address the aging of the used cryptography. Third, when verifying evidence in all schemes, most of the time is spent on checking certificate chains. This is even worse for the time-stamping schemes because they accumulate certificate chains whereas the notarial scheme does not. Moreover, the experiments indicate that a solution

to mitigate this issue in time-stamping schemes is to use longer signature lifetimes. Conversely, this does not apply for the notarial scheme.

Next, we turn to designing three new solutions to improve the performance of the protection schemes. The first solution is a new notarial scheme named *Attested Certificates* (AC) that addresses the performance issues identified by analyzing CN. This is done in Chapter 6. AC is more efficient than CN with respect to communication and space. Communication is reduced by leaving the verification of the signatures on documents to retrievers. Thus, contrary to CN, documents and their signatures are not sent to notaries. Space is reduced by generating smaller evidence which, as opposed to CN, contains neither old notary signatures nor time-stamps. We demonstrate that AC is superior to CN by a performance analysis. Moreover, we compare CN and AC with respect to their trust assumptions, showing that the solutions provide a similar level of trustworthiness.

The second solution that we propose is based on the time-stamping solution CIS and is called *Content Integrity Service with Skip Lists* (CISS). The scheme is to be used in digital archives where subsets of documents share the same sequence of time-stamps. Note that this approach reduces the effort of an archivist, who needs to maintain only some time-stamp sequences rather than one sequence for each document in the archive. Although also ERS can be used for document sets, it has the limitation that the documents of a set must be submitted to an archive and time-stamped at the same time. However, in some cases this is not possible. CIS is the most suitable solution for this scenario; however, CIS has performance issues that we address in CISS. The main issue is that when submitting and time-stamping documents sequentially, the generated evidence may contain several time-stamp whose validities overlap. It follows that many time-stamps in the sequence need not be checked by retrievers when they verify the sequence for single documents. We address this issue in CISS by using a data structure called *skip list*. We show how to build time-stamp sequences with skip lists allowing retrievers to select the time-stamps that are needed to guarantee the protection goals of single documents, while skipping the other time-stamps during verification. Moreover, CISS is designed to address the aging of cryptography efficiently like ERS. That is, CISS hashes only the minimal data when dealing with the aging of cryptography. As before, we provide an analytical evaluation and experiments to show that CISS verifies evidence significantly faster than CIS.

The third solution aims at preventing retrievers from spending excessive time on certificate chain verifications when using time-stamping schemes. In Chapter 8 we improve the solution by Silva et al. [45], where certificate chains needed to verify time-stamps are replaced by smaller proofs. These proofs consist of single signatures

created by a root certification authority (CA) as follows. First, the root CA checks that the certificate chain of a time-stamp authority is valid and then re-signs the time-stamps issued by that time-stamp authority. Our solution is called *re-signed certificates* and uses a trust anchor (not necessarily a root CA) to re-sign only the certificates of the time-stamp authorities. The re-signed certificates are requested by archivists when they update their time-stamp sequences. This leads to a more efficient solution when a time-stamp authority issues a large amount of time-stamps during its lifetime. Finally, we analyze the performance of both solutions analytically, showing that ours indeed reduces the effort of time-stamp authorities while also speeding up the verification process for retrievers.

1.3 Structure

This thesis is organized as follows. In Chapter 2 we introduce the terminology and components used in the protection schemes addressed in this work. We also explain how the lifetime of cryptography has been predicted and how trustworthiness can be approximated. Chapter 3 provides an overview of existing protection schemes. These schemes are analyzed with respect to trustworthiness and performance in Chapters 4 and 5, respectively. In Chapter 6 we propose a new notarial scheme by improving the performance of the existing CN solution. Next, in Chapter 7 we design a more efficient time-stamping scheme based on the existing CIS solution. In Chapter 8 we propose a solution to the performance issue of checking evidence containing long certificate chains. Finally, in Chapter 9 we draw our conclusions and give directions for future work.

2 Preliminaries

In this chapter we provide the background for this thesis. We start by defining the terminology that we use in the following chapters. Next, describe the protection goals, cryptographic primitives, and components that are particularly important for our work. The protection goals described are integrity, authenticity, non-repudiation, and proof of existence. The cryptographic primitives are hash functions and digital signatures. In the description of the components, we distinguish between two types. The first type refers to the components that use the cryptographic primitives and includes Merkle trees, public key infrastructures, widely visible media, and time-stamps. However, since hash functions and digital signatures have limited lifetimes, we also discuss how researchers and governments have predicted for how long these primitives can be used securely. The second type of components does not use cryptographic primitives and includes trust models and skip lists.

2.1 Terminology

We define the following terms that we use along this work.

Long term : a very long or endless period of time.

Document : any digital content.

Digital archive : a service that stores documents for the long term.

Protection scheme : a solution to be used to protect documents stored in a digital archive.

Archivist : the entity that runs a digital archive together with a protection scheme.

Evidence : the data provided by a protection scheme to demonstrate that a document has been protected properly.

Retriever : a third party that obtains documents from an archive and use the evidence provided for the documents to verify their protection.

2.2 Protection goals

In this section we present the protection goals addressed in our work. We first define them and then explain how they are related.

The protection goals we deal with are *integrity*, *authenticity*, *non-repudiation*, and *proof of existence*. Integrity means that a document has not been altered. Authenticity means that its origin can be identified. Non-repudiation prevents an originator from repudiating that he or she is the originator of a document. Proof of existence allows to identify a time reference when a document existed.

Integrity, authenticity, non-repudiation, and proof of existence are closely related. Integrity and proof of existence are the basis. Stating that a document has not been changed includes a statement since when this is true. Thus, proof of existence is desirable. Conversely, proof of existence makes only sense with evidence of integrity. Also, authenticity and non-repudiation make only sense if there is also an evidence of integrity. Therefore, in this work when we say that authenticity, non-repudiation or proof of existence are provided, this implicitly means that integrity is also guaranteed.

2.3 Cryptographic hash functions

Cryptographic hash functions are central building blocks in cryptography. For example, they are used to reduce integrity verification of large documents to integrity verification of short bit strings. A hash function is a map $h : \{0,1\}^* \to \{0,1\}^n$, where $\{0,1\}^*$ denotes the set of all bit strings of arbitrary length and n is some fixed positive integer. This map is assumed to be collision resistant, that is, finding $x, x' \in \{0,1\}^*$ with $x \neq x'$ and $h(x) = h(x')$ is infeasible. A typical choice for a hash function is SHA-3 [40] where $n \in \{224, 256, 384, 512\}$. For a more detailed discussion of hash function see [6].

2.4 Merkle trees

Merkle trees [37] go a step further than hash functions. They use hash functions and a binary tree construction to reduce integrity verification of a sequence of documents $d_1, ..., d_m$, m being a power of 2, to the integrity verification of the root of the tree.

The construction is as follows. A Merkle tree uses a cryptographic hash function h. The leaves of the tree are $h(d_1), ..., h(d_m)$, in that order. Any internal node (including the root) is constructed as the hash $h(y_a||y_b)$ of the concatenation of the left child y_a and the right child y_b of the node. Figure 2.1 shows an example of a Merkle tree where $m = 4$.

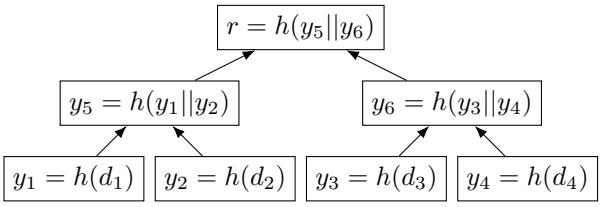

Figure 2.1: A Merkle tree for the documents d_1, d_2, d_3, and d_4. The leaves y_1, y_2, y_3, and y_4, the inner nodes y_5 and y_6, and the root r are computed using hash function h.

To verify the integrity of a document $d_i \in \{d_1, \ldots, d_m\}$, a verifier is given d_i, the root r of the Merkle tree constructed as explained above and the *authentication path* for d_i. This path contains $\log_2 m$ hashes, which are the siblings of the nodes in the path from the leaf $h(d_i)$ to the root r. For instance, the authentication path of leaf y_1 in Figure 2.1 comprises hashes y_2 and y_6. The verifier uses this authentication path to construct a root r'. For example, given the document d_1 and the authentication path $\{y_2, y_6\}$, the verifier computes $r' = h(h(h(d_1)||y_2)||y_6)$. Next she compares r' with r. If the roots are equal, then d_i is uncorrupted.

2.5 Signatures

Digital signatures provide authenticity and non-repudiation for documents. A digital signature scheme consists of three algorithms. The key generation algorithm generates a secret signing key and a public verification key. Given a document and the secret signing key, the signature algorithm computes a signature on the document. Correspondingly, given the document, a signature on the document, and a public verification key, the verification algorithm decides whether the signature is valid. A valid signature proves that the document has not been changed since it was signed (integrity) and that the holder of the secret signing key is the origin of the document (authenticity). Also, the authenticity can be verified by anyone (non-repudiation). The digital signature schemes currently used in practice are RSA, DSA, and ECDSA [32]. For detailed descriptions of these schemes, see [6].

The validity of a digital signature relies not only on the verification algorithm to attest that the signature is correct, but also on the public key to be valid. This is

discussed in Section 2.7.

We are aware that for performance reasons signature schemes are often used together with hash functions. More precisely, to sign a document a signer first computes a hash of the document and then computes a signature on this hash. However, in this work we assume a hash function is not needed to sign documents, unless we explicitly mention it. The reason is to make the presented protocols involving signatures easier to comprehend.

In practice, signature schemes are often used together with hash functions because of, for example, performance issues. More precisely, to sign a document a signer first computes a hash of the document and then computes a signature on this hash. However, we seldom make this procedure explicit in this work. The reason is to make the presented protocols involving signatures easier to comprehend.

2.6 Public key infrastructures

The use of digital signatures relies on the availability of trustworthy public keys. A *public key infrastructure* (PKI) provides such keys. We explain the hierarchical PKI, which is the most popular model. Another model of PKI is the Web of Trust [63].

A PKI consists of one or several certification authorities (CA). A CA signs a certificate to assert that a public key belongs to a certain subject. Moreover, the CA can also assert specific properties of the public key (e.g. for which purpose this key can be used). A certificate comes with a date when the certificate is first valid and an expiration date. CAs can sign certificates for themselves and other entities, such as subordinate CAs or end users. Therefore, the trustworthiness of a public key may depend on the validity of all certificates in a chain of certificates (see [7, sec. 6.1]).

Certificates may become invalid before they expire. For example, the secret signature key corresponding to the certified public key may be compromised. Therefore, CAs operate revocation services that enable the revocation of invalid certificates and provide the related revocation information for users. Depending on the policies of a CA, it may provide no revocation information for expired certificates. For more details on certificate revocation, see [7, chap. 5].

2.7 Signature verification

Checking the validity of a signature is much more involved than just applying the verification algorithm. For example, it is necessary to check the validity of the public

key used in the verification. Since signature verification is an important ingredient of all schemes described in the next chapters, we present the process of verifying a signature s at a time t in detail. The time t is usually the moment when the verification occurs (i.e. the *current* time) or the time provided by a time-stamp. In some particular cases, t can be the time when the signature s was created.

In the first step, a verifier collects the necessary *verification information* for the signature s. This information includes a chain of certificates and the current revocation information for each certificate in the chain. The first element in the chain is issued by a CA called *trust anchor*, which the verifier must trust. The last element is a certificate for the public key required to verify s. The other certificates in the chain certify the public key required to verify the signature on the next certificate in the chain.

In the second step, the verifier checks that the signature algorithms and hash functions that were used to create s and the signatures on all certificates and revocation information are secure at time t. She also checks that the key sizes in the signature algorithms are sufficient to make the signatures secure.

In the third step, the verifier checks that none of the certificates is revoked at time t by using the previously collected revocation information.

The fourth step is to verify that t is in the validity interval of all certificates. The standard signature validity model is called *shell model*. For other validity models see [7, chap. 6].

The fifth and last step is to apply the verification algorithm on all signatures.

Moreover, certificates can contain key properties as mentioned in Section 2.6. In this case, an additional step is needed to check that the contained key properties are guaranteed. For more details on the verification of key properties see [12, sec. 6].

2.8 Signature-based time-stamps

A time-stamp on a document shows that this document existed at a certain point in time and has not been changed since. This time is usually provided as a calendar date. Nevertheless, a time-stamp can also use a relative date, showing that the document existed before another document. However, since it is unclear how to use relative dates to verify whether cryptographic algorithms were used while they were secure, we will present time-stamps using calendar dates. For time-stamps using relative dates, we refer to reader to the *hash chain*-based scheme by Maniatis and Baker [34].

A time-stamp is issued by a trusted third party named *time-stamp authority* (TSA). Such time-stamps are created as follows. First, some entity computes the hash $h(d)$ of some document d. Then the entity requests a time-stamp on $h||h(d)$ from a TSA. Here, h stands for the identifier of the hash function used to compute the hash $h(d)$. The TSA creates a time-stamp by signing $h||h(d)||t$, where t is the time when the time-stamp is created. The time-stamp consists of h, t, and the signature on $h||h(d)||t$. The *verification information* for the time-stamp is the verification information needed to verify the signature on $h||h(d)||t$.

A time-stamp on a document d is verified as follows. First, the verifier extracts the hash function h and the time reference t from the time-stamp. Then, she computes $h(d)$ and verifies the signature on $h||h(d)||t$ contained in the time-stamp as explained in Section 2.7.

2.9 Widely Visible Media-based time-stamps

A time-stamp on a document can also be constructed by publishing the hash of the document on *widely visible media* (WVM). An example for WVM is a newspaper. The assumption is that WVM is preserved for a long time in such a way that it is impossible to modify WVM. In addition, WVM must carry a date on which it was created. In the case of a newspaper, this is the date of publication of the newspaper. This type of time-stamps has been used in practice by Surety, LLC [48]. A time-stamp created by Surety, LLC and published in *The New York Times* is illustrated in Figure 2.2. For more information regarding WVM, see [1].

Generating a WVM-based time-stamp on a document d works as follows. Some entity selects a hash function identified by h, calculates $h(d)$, and submits both $h(d)$ and h to a TSA. The TSA publishes $h(d)$ and h on WVM. The verification information for this time-stamp allows a verifier to locate and access the time-stamp. If several hashes are submitted to the TSA, it may reduce the size of the time-stamp by constructing a Merkle tree whose leaves are the submitted hashes. The TSA then publishes the root of the Merkle tree and the hash function identifier h. In this case, the verification information for each time-stamped document d should also include the authentication path from $h(d)$ to the published Merkle tree root.

To verify a WVM-based time-stamp on the document d, the verifier extracts the creation time of the WVM and the hash function identifier h. She computes the hash $h(d)$ and compares it with the hash stored in the WVM. If the TSA used a Merkle tree, the verifier first uses $h(d)$ and the corresponding authentication path to compute the root. Then, she compares the computed root with the root published


NOTICES & LOST AND FOUND
(5100-5102)
Public Notices
Lost & Found 5100-5102
Universal Registry Entries:
Zone 2 WF/PQxqyo9iQUuJtJ/YAPdu2
OZGOFRkWOZqQRIFuy+D/h5Gp
Zone 3 3x30YXa020fuZtPDKNPAc0az
dHNhO4AjBBroZgXetz2dtgdg
These base64-encoded values represent the combined fingerprints of all digital records notarized by Surety between 2003-10-01Z - 2003-10-07Z
www.surety.com 703-707-9901
Commercial Notices

Figure 2.2: A widely visible media-based time-stamp issued in *The New York Times* [47].

on the WVM. If the roots match, then document d existed on the date when the WVM was published and d has not been changed since.

2.10 Lifetime of cryptographic algorithms

The security of most cryptographic algorithms and their parameters is not everlasting. Over the years, constant advances in computer power and cryptanalytic techniques tend to defeat the security properties of the algorithms and their parameters. In the case of hash functions, they become insecure when collisions can be found. As to digital signature algorithms or key sizes, they are considered insecure when signatures can be forged (for details, see [20]).

To estimate for how long these algorithms are expected to be secure for the chosen parameters (e.g. key sizes), Lenstra [31] proposed a model based on the best known attacks and current trends, such as the expected progress of (published) cryptanalysis and Moore's Law [38]. Likewise, governmental agencies provide recommendations on how to choose cryptographic algorithms and parameters (e.g. [8, 39]). These predictions and recommendations are based on the observed development of technology and are revised on a regular basis.

2.11 Trust

In this section we present how the trustworthiness of parties and information can be quantified. We first provide the definitions of trust, trustworthiness, and reputation. Next, we explain how reputation and trust can be quantified by using *scores* and present important operators on such scores.

2.11.1 Definitions

Among many definitions of trust, the two most used are *reliability trust* and *decision trust*. Reliability trust is defined by Gambetta [19] as the subjective probability by which the individual A expects that another individual B performs a given action on which A's welfare depends. Decision trust extends reliability trust by also taking into account the risk that one party (i.e. an individual) is willing to take upon trusting another party or information [36]. We refer to decision trust as *trust*.

Trust is commonly used to approximate the trustworthiness of a party or information. Wierzbicki [61] defines trustworthiness as an "objective, context-dependent property of deserving trust". Since it is an intrinsic quality, it is hard to assess trustworthiness in practice. This is why trust is commonly used to approximate the trustworthiness of a party or information.

Moreover, trust and reputation are related. Roughly, trust is between two parties or a party and an information. Reputation is the aggregated experience of a community on a party or information. The reputation is often quantified by a *reputation score*. A party or an information that has high reputation is frequently trusted by many parties in that community; a party outside that community may also use that reputation to make his or her trust decisions.

Reputation systems can be centralized or distributed. In a centralized reputation system, an authority collects experiences reported by a community. This authority is trusted to preserve the reported experiences and typically to compute the reputation scores of parties or information. In contrast, there is no trusted authority in distributed reputation systems. In this case, members of a community share their experiences and compute the reputation scores from these experiences by themselves.

2.11.2 Quantifying and operating on reputation

Reputation can be quantified by reputation scores as follows. Assume an experience by one participant c on another participant or information z is a boolean representing whether c is satisfied after interacting with z. From these experiences, the reputation

score $\vec{X}_z^C = (r_z^C, s_z^C)$ is computed, where C identifies a community of participants that interacted with z, r_z^C is the sum of the positive experiences on z, and s_z^C is the sum of the negative experiences on z.

Important operators on reputation scores are *weighted averaging* and *addition*. Weighted averaging can be used to combine reputations scores of the same party or information when these scores are provided by communities that are not equally trusted. That is, the experiences by one community are more reliable than the experiences by another community. Thus, assume that reputation scores $\vec{X}_z^{C_1}, \ldots, \vec{X}_z^{C_n}$ were computed from experiences on z by communities $C_1, \ldots, C_n$, respectively. Moreover, assume that experiences by communities $C_1, \ldots, C_n$ have non-negative weights $w_1, \ldots, w_n$ and occurred at the same time (i.e. their experiences are dependent). For example, the interactions with z provide the same outcomes for the communities. Then the weighted average reputation score can be computed by

$$\vec{X}_z^{C_1,\ldots,C_n} = \left(\frac{\sum_{i=1}^n w_i r_z^{C_i}}{\sum_{i=1}^n w_i}, \frac{\sum_{i=1}^n w_i s_z^{C_i}}{\sum_{i=1}^n w_i}\right). \tag{2.1}$$

Two reputation scores of the same party or information can be added using the addition operator as follows. Assume $\vec{X}_v^{C_1}$ and $\vec{X}_v^{C_2}$ are the reputations of z and are computed from the experiences provided by communities C_1 and C_2, respectively. Moreover, assume that the experiences of C_1 and C_2 occurred at different times (i.e. their experiences are independent). For example, the interactions with z provide distinct outcomes for the communities. Then, the addition of $\vec{X}_v^{C_1}$ and $\vec{X}_v^{C_2}$ can be computed by

$$\vec{X}_z^{C_1} + \vec{X}_z^{C_2} = (r_z^{C_1} + r_z^{C_2}, s_z^{C_1} + s_z^{C_2}). \tag{2.2}$$

2.11.3 Quantifying and operating on trust

Trust can be quantified by *trust scores*. To determine them, we use reputation scores and a trust model. Well-established trust models are belief models. They are a probability framework where probabilities are assigned not only to the possible outcomes but also to uncertainty. They receive as input the reputation score of a party or information and compute a trust score as output. The more experiences the reputation score contains, the lower is the uncertainty and the more reliable is the approximation of trustworthiness. Examples of such belief trust models are *Subjective Logic* [25] and *CertainTrust* [43].

The two aforementioned models are based on the Beta Probability Density Function and are isomorphic, i.e. trust scores can be mapped from one model to an-

other. Subjective Logic has the advantage of using mathematics which is easier to comprehend. On the other hand, CertainTrust allows for a user-friendly graphical representation of trust scores. However, since we will not use graphical representations of trust scores and both models can be interchanged (they are isomorphic), we select Subjective Logic.

Trust scores are called *opinions* in Subjective Logic. An opinion of party a about the truth of statement z is defined as a triple ω_z^a of real numbers (b, d, u) in $[0, 1]$ such that $b + d + u = 1$. In this triple, b stands for the belief that z is true, d stands for the disbelief that z is true, and u stands for the uncertainty whether z is true.

A reputation score can be mapped to an opinion and vice versa. The mapping function is as follows. It receives as input a reputation score $\vec{X}_z^C$ and the prior knowledge about z, that is, the information about z in the absence of experiences on z. Jøsang et al. set the prior knowledge to one, i.e. $r_0 = s_0 = 1$. The function outputs an opinion $\omega_v = (b_v, d_v, u_v)$ such that

$$b_z = \frac{r_z^C}{r_z^C + s_z^C + r_0 + s_0}, d_z = \frac{s_z^C}{r_z^C + s_z^C + r_0 + s_0}, d_z = \frac{r_0 + s_0}{r_z^C + s_z^C + r_0 + s_0}. \tag{2.3}$$

An important operator on opinions is the conjunction operator ("$\wedge$"). It is analogous to a logic AND and computes the conjunctive truth of two independent opinions as follows. Assume there exist two opinions ω_v and ω_z about v and z respectively. By computing $\omega_v \wedge \omega_z$ we obtain the new opinion $\omega_{v \wedge z}$ such that

$$\omega_{v \wedge z} = \begin{cases} b_{v \wedge z} = b_v b_z \\ d_{v \wedge z} = d_v + d_z - d_v d_z \\ u_{v \wedge z} = u_v u_z + b_v u_z + b_z u_v. \end{cases} \tag{2.4}$$

Note that the presented operators on reputation and trust scores are commutative and associative.

2.12 Skip lists

Skip lists [42] are ordered data structures that can be used to search for elements in a data set efficiently. To this end, several links between the elements are generated and stored in *internal* linked lists.

When an element is added to the skip list, the element is stored in one or more internal linked list. More precisely, the linked lists are organized in levels. All elements added to the skip list are stored in the internal linked list L_0 in the first

level (level 0). Half of those elements are also stored in the linked list L_1 in level 1, a quarter of the elements are stored in the linked list L_2 in level 2, and so on. In general, the internal linked list L_j in level $j \in \mathbb{Z}_{\geq 0}$ stores $1/2^j$ of the elements added to the skip list. Figure 2.3 illustrates a skip list containing three internal linked lists and the added elements $e_0, \ldots, e_4$.

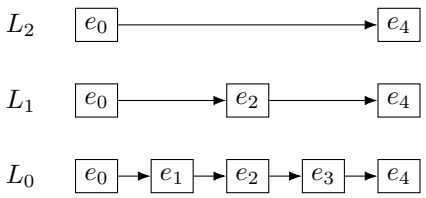

Figure 2.3: A skip list containing the elements e_0, e_1, e_2, e_3, and e_4. These elements are stored in the internal linked lists L_0, L_1, and L_2.

There are two types of skip lists, which differ in how elements are assigned to linked lists. The first type is the deterministic skip list, where we can predict which linked lists contain a given element by using the order in which the elements were added to the skip list. The second type is the probabilistic skip list, where the elements to be stored in an internal linked list are selected randomly using a distribution probability.

The skip lists that we will use in Chapter 7 need to fulfill two requirements. First, since we will need to time-stamp specific elements and links from a skip list, it should be possible to determine the position of these elements before generating the time-stamp. Second, after a time-stamp is created, the links should not be changed. Since when removing an element we need to redefine the links, the removal of elements should be prevented. To meet these requirements, we therefore select skip lists that are deterministic and append-only.

We next explain how to initialize and add new elements to a deterministic, append-only skip list. To initialize the skip list, the internal linked list L_0 in level 0 is created. At this moment, L_0 is empty and there exist no further linked lists.

To add the first element e_0 to the initialized skip list, we just add it to the empty linked list L_0. Since there are no other elements in L_0, we create no links to e_0.

For $i > 0$, the addition of the next elements e_i requires extra steps as follows. We not only add e_i to the linked list L_0 but also create a link from element e_{i-1} to e_i. Furthermore, we may need to add e_i to other linked lists and generate links from other elements to e_i. To illustrate these steps, we first describe the addition of the next four elements e_1, e_2, e_3, and e_4 to the skip list. Then, we show a generalization of this process. Figure 2.3 depicts the skip list after e_4 is added.

1. The second element e_1 is added to the linked list L_0. A link from e_0 to e_1 is created.

2. The third element e_2 is added to the linked list L_0 and a link from e_1 to e_2 is generated. Moreover, an empty linked list L_1 in the second level is created containing e_0, e_2, and a link from e_0 to e_2.

3. The fourth element e_3 is added to the linked list L_0. A link from e_2 to e_3 is created.

4. The fifth element e_4 is added to the linked lists L_0 and L_1. Next, two links are created: one from e_3 to e_4 in L_0, and another from e_2 to e_4 in L_1. Furthermore, an empty linked list L_3 is created containing e_0, e_4, and the link from e_0 to e_4.

In general, for $i > 0$ the $(i+1)$th element e_i added to the skip list is stored in the linked lists $L_0, \ldots, L_g$. To determine $g \in \mathbb{Z}_{\geq 0}$ we use the formula

$$g = \log_2(i/m), \tag{2.5}$$

where m is odd as shown by Maniatis and Baker [34]. For example, for $i = 4$, the formula is satisfied by $m = 1$ and $g = 2$. Then, for each $0 \leq j \leq g$, we create a link from element e_k to element e_i in the linked list L_j such that $k = i - 2^j$. Note that the first element e_0 is stored in all linked lists.

3 Protection schemes for long-term archiving

This chapter provides an overview of existing protection schemes that provide one or more of the protection goals integrity, authenticity, non-repudiation, and proof of existence for archived documents. This overview is presented in two parts. The first part contains the schemes that guarantee the mentioned protection goals in the long term. In this part we distinguish among schemes using time-stamp sequences, notarial attestations, and replication. The second part of the overview provides schemes that use the above approaches or that achieve extra protection goals (e.g. proof of non-existence) but fail in addressing the aging of cryptography. The content of this chapter was published in [58].

3.1 Time-stamping schemes

In this section we present the first type of long-term protection solutions for digital archives. We start by explaining why single time-stamps cannot provide long-term protection. Next, we introduce time-stamp sequences and describe the solutions that use this approach.

In our descriptions, an *archivist* maintains documents in the archive. In particular, the archivist is responsible for generating *evidence* that can be used to verify that a document existed at some point in time and has not been changed since (proof of existence). If the document was digitally signed by its originator, the evidence also provides authenticity and non-repudiation for each document. The evidence is stored together with the document. To create such evidence, the archivist requests time-stamps from a *time-stamp authority* (TSA). Also, *retrievers* obtain documents from the archive and use the evidence provided for the documents to verify their various protection goals.

Signatures must be verified in some schemes. This is done by using the proce-

dure explained in Section 2.7. Such a signature verification uses certain verification information. The default is that the retriever collects up-to-date verification information. For example, certificates and their most recent revocation information. In some cases, the retriever is provided with stored verification information. This will be mentioned explicitly.

3.1.1 The validity of a time-stamp

As explained Chapter 2, time-stamps provide proof of existence for documents. However, single time-stamps based on widely visible media (WVM) or on signatures cannot provide long-term protection.

The validity of a WVM-based time-stamp relies on the security of the cryptographic hash function used to create the time-stamp. As shown in Section 2.10, this security is not everlasting.

Likewise, signature-based time-stamps also rely on cryptographic hash functions but additionally on digital signatures which also expire. For example, when the certificates required to verify them expires.

Therefore, along this work we refer to the *validity of a time-stamp* as the period that starts when the time-stamp is created and ends when the used hash function or signature becomes insecure.

3.1.2 Time-stamp sequences

Time-stamp sequences were first introduced by Bayer et al. [1] to prolong the validity of time-stamps. In this section, we explain how the intervals for time-stamp renewal are chosen when generating such sequences.

First, an initial time-stamp T_1 is created at time t_1 that, depending on the scheme, provides proof of existence for a single document or a sequence of documents. When necessary, new time-stamps $T_2, T_3, \ldots$ are created at times $t_2 < t_3 < \ldots$ that prolong the validity of the previous time-stamp beyond its expiration time. We will see that in order for the prolongation to work, it is necessary that the previous time-stamp is still valid when a new time-stamp is created. In the following, we explain how time-stamp creation times can be chosen such that this condition is satisfied.

Assume that the time-stamps $T_1, T_2, \ldots$ are based on WVM. Such time-stamps expire when the hash functions $h_1, h_2, \ldots$ used in their construction become insecure. Hence, when a new time-stamp T_k is created ($k > 1$), the hash function h_{k-1} used to create the previous time-stamp T_{k-1} must be still secure. Also, to construct T_k, a hash function h_k must be still secure at time t_{k+1}. So choosing h_k and t_{k+1}

needs to predict the lifetimes of hash functions. Section 2.10 discusses how this can be done. One possibility of choosing the creation times of the time-stamps is as follows. Determine an expected lifetime τ_h of cryptographic hash functions as described in Section 2.10 and create the time-stamps at times $t_k = t_1 + (k-1)\tau_h$, $k \geq 1$. This construction method works as long as there is no unexpected break of a hash function. This issue will be discussed in more detail in Chapter 4.

Now assume that the time-stamps used in the sequence are based on signatures. Then the selection of the construction times $t_1, t_2, \ldots$ is slightly more complicated because the time-stamps $T_1, T_2, \ldots$ use hash functions $h_1, h_2, \ldots$ and digital signatures $s_1, s_2, \ldots$ in their construction. Hence, when a new time-stamp T_k is created, h_{k-1} and s_{k-1} must be still secure. To construct T_k, a hash function h_k must be chosen and a signature s_k must be created that are still secure at time t_{k+1}. So choosing h_k, s_k, and t_{k+1} requires predicting the lifetimes of hash functions and signatures. Since the expected lifetime τ_h of hash functions is typically longer than the expected lifetime τ_s of signatures, the time-stamp creation time can be chosen as $t_k = t_1 + (k-1)\tau_s$. At time t_k a new hash function is only chosen when $t_{k+1} > t_{k-1} + \tau_h$. Otherwise, $h_k = h_{k-1}$ is used. This construction method works as long as a hash function and signature are not suddenly compromised. This issue will be discussed in more detail in Chapter 4.

3.1.3 Advanced Electronic Signatures

Advanced Electronic Signatures (AdES) [17, 18] is a protection scheme in which an archivist maintains a sequence of time-stamps for each document in the archive. The first time-stamp in such a sequence provides proof of existence for the document at a certain point in time. The following time-stamps validate the previous time-stamps, thereby extending their validity beyond their expiration. Retrievers verify proof of existence of a document by verifying the respective time-stamp sequence. The original AdES uses signature-based time-stamps but it also works with WVM-based time-stamps.

We explain the construction of an AdES time-stamp sequence on a document d in more detail. For the selection of creation times, hash functions, and signatures we refer to Section 3.1.2. Initially, the archivist selects a hash function h_1 and requests the first time-stamp T_1 on $h_1||h_1(d)$ which is issued at time t_1.

Now suppose for $k > 1$ that time-stamps T_j have been issued at times t_j using hash functions h_j, $1 \leq j < k$. The next time-stamp T_k is created at time t_k when T_{k-1} has not expired. Thus, the archivist selects a hash function h_k. This may be the previous hash function h_{k-1} or a new hash function depending on

whether h_{k-1} is expected to be still secure at time t_{k+1}. The archivist also collects up-to-date verification information V_{k-1} for T_{k-1} and requests a time-stamp T_k on $h_k||h_k(d||T_1||V_1||\ldots||T_{k-1}||V_{k-1})$. At the current time $t \in [t_k, t_{k+1}]$, the evidence stored with each document consists of time-stamps $T_1, \ldots, T_k$ and verification information $V_1, \ldots, V_{k-1}$.

Now we describe how a retriever can use the evidence for document d to verify its proof of existence for t_1, i.e. that it existed at time t_1 and has not been changed since. Let $k \geq 1$ and suppose that the retriever wishes to perform this verification at the current time $t \in [t_k, t_{k+1}]$. Then she verifies time-stamps T_j, $1 \leq j \leq k$. To verify T_k, she constructs $y = h_k(d||T_1||V_1||\ldots||T_{k-1}||V_{k-1})$ and checks that T_k is a valid time-stamp on $h_k||y$ at time t. This time-stamp proves that d, $T_1, \ldots, T_{k-1}$, and $V_1, \ldots, V_{k-1}$ existed at time t_k and have not been changed since.

Now suppose for $1 \leq j < k$ that the retriever was able to verify T_{j+1}. This means that $T_1, \ldots T_j$, and $V_1, \ldots, V_j$ existed at time t_{j+1} and have not been changed since. In order to verify T_j, she constructs $y = h_j(d||T_1||V_1||\ldots||T_{j-1}||V_{j-1})$ and verifies that, given verification information V_j, time-stamp T_j is a valid time-stamp on $h_j||y$ at time t_{j+1}. For $j = 1$ this shows that document d existed at time t_1 and has not been changed since. Note that for $j = 1$ the sequences $T_1, \ldots, T_{j-1}$ and $V_1, \ldots, V_{j-1}$ are empty.

As for time-stamps, the document d may consist of a primary document and a signature on it. In this case, the sequence provides not only the proof of existence of the primary document at time t_1, but also its authenticity and non-repudiation. Note that the signature on d and the corresponding verification information must be added to the evidence and time-stamped when the time-stamp sequence is initialized.

3.1.4 Content Integrity Service

As AdES, Content Integrity Service (CIS) [23] protects documents by time-stamp sequences. CIS is intended to use WVM-based time-stamps, but it also supports signature-based time-stamps. Moreover, CIS allows an archivist to time-stamp distinct documents sequentially such that the documents share the same time-stamp sequence. This feature is useful when time-stamped documents are changed and the changes must also be time-stamped. For example, when the original format of a document needs to be changed.

We describe how time-stamp sequences are created in CIS using WVM-based time-stamps. Again we refer to Section 3.1.2 for the selection of the times when time-stamps are created and when new hash functions are chosen. The initial document to be time-stamped is d. The archivist selects a hash function h_1, computes the hash

$h_1(d)$, and requests a time-stamp T_1 on $h_1||h_1(d)$ from a TSA. The TSA creates T_1 at time t_1.

Now assume for $k > 1$ that time-stamps T_j have been created at times t_j using hash functions h_j, $1 \leq j < k$. The next time-stamp T_k is created before T_{k-1} expires. For this purpose, the archivist chooses a potentially new hash function h_k and determines the hash y_k to be time-stamped as follows. First, he computes $y_1 = h_k(d)$. Next, for $1 \leq j < k$ the archivist computes $y_{j+1} = h_k(y_j||h_k(T_j||V_j))$, where V_j contains verification information required to verify time-stamp T_j at time t_{j+1}. While verification information V_j with $j < k-1$ is already available, the archivist must collect new verification information for T_{k-1} at time t_k. Also, the archivist may need to time-stamp an additional document d'. In this case, hash y_k is set to $h_k(y_{k-1}||h_k(T_{k-1}||V_{k-1})||h_k(d'))$. Next, the archivist requests a time-stamp T_k on $h_k||y_k$ from a TSA. Note that the construction of y_j in this step and the creation of y_j in step j are very similar. The only difference is that in step j is used the hash function h_j and in step k is used the hash function h_k.

We explain which evidence is stored and how it is verified. Suppose that, in the course of construction of the time-stamp sequence, documents $d_1, \ldots, d_m$ have been time-stamped at times $t_1, \ldots, t_m$, respectively. At the current time $t \in [t_k, t_{k+1}]$, where $t_k \geq t_m$, the evidence stored with $d_1, \ldots, d_m$ consists of $t_1, \ldots, t_m$, the time-stamps $T_1, \ldots, T_k$, and the verification information $V_1, \ldots, V_{k-1}$.

To verify the proof of existence of a document $d_i \in \{d_1, \ldots, d_m\}$ at the current time t, a retriever does as follows. First, she identifies the time-stamp T_i applied on d_i using the time reference t_i. Then, she verifies the time-stamp sequence $T_i, \ldots, T_k$. This verification is analogous to the verification in AdES.

3.1.5 Evidence Record Syntax

In the Evidence Record Syntax (ERS) solution [3, 21], an archivist protects a sequence of documents $d_1, \ldots, d_m$ by using a single time-stamp sequence and Merkle trees. Time-stamps are signed, but ERS could also use WVM-based time-stamps.

We start by explaining how the archivist initializes a sequence of time-stamps. Initially, he selects a cryptographic hash function h_1 and constructs a Merkle tree M_1 with leaves $h_1(d_1), \ldots, h_1(d_m)$. Next, he requests a time-stamp T_1 on $h_1||r_1$ from a TSA, where r_1 is the root of tree M_1. The TSA creates T_1 at time t_1. In Figure 3.1, this procedure is used to create time-stamp T_1. Also, for each document d_i the archivist creates evidence to be used by a verifier. This evidence consists of the authentication path $A_{i,1}$ from the leaf $h(d_i)$ to the root r_1 in M_1, $1 \leq i \leq m$.

Now assume for $k > 1$ that Merkle trees $M_1, \ldots, M_{k-1}$ with roots $r_1, \ldots, r_{k-1}$ and

time-stamps $T_1, \ldots, T_{k-1}$ have been created. The next time-stamp, hash function, and Merkle tree to be constructed are T_k, h_k, and M_k, respectively. Section 3.1.2 explains when the next time-stamp must be created and whether a new hash function must be selected. If no new hash function is selected, then the archivist sets $h_k = h_{k-1}$, collects verification information V_{k-1} for T_{k-1}, and requests a time-stamp T_k on $h_k||h_k(T_{k-1}||V_{k-1})$ from a TSA. The TSA creates T_k at time t_k. In Figure 3.1, this procedure is used to create time-stamps $T_2, \ldots, T_{k-1}$.

If a new hash function h_k is selected, then a new Merkle tree M_k is constructed. The ith leaf is the hash of the concatenation of the following data: the document d_i, the authentication paths for the leaves corresponding to d_i in all previously constructed Merkle trees, the previously issued time-stamps $T_1, \ldots, T_{k-1}$, and their verification information $V_1, \ldots, V_{k-1}$. Once M_k is created, the archivist requests time-stamp T_k on $h_k||r_k$ from a TSA, where r_k is the root of M_k. Time-stamp T_k is then created at time t_k. This procedure is used to generate time-stamp T_k in Figure 3.1.

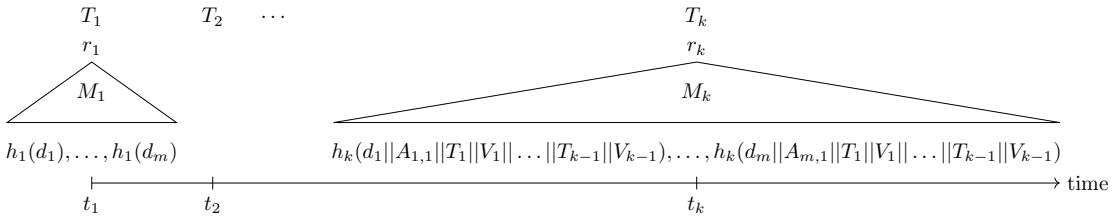

Figure 3.1: The Merkle trees M_1 and M_k created at the times t_1 and t_k are used to time-stamp the documents $d_1, \ldots, d_m$. The created roots are r_1 and r_k. The authentication paths for the documents in M_1 are $A_{1,1}, \ldots, A_{m,1}$. The verification informations $V_1, \ldots, V_{k-1}$ are needed to verify the time-stamps $T_1, \ldots, T_{k-1}$.

The evidence stored with the documents $d_1, \ldots, d_m$ is as follows. At the current time $t \in [t_k, t_{k+1}]$, the evidence consists of the authentication paths $A_{i,1}, \ldots, A_{i,k}$ for the leaves corresponding to documents d_i $(1 \leq i \leq m)$ in the Merkle trees $M_1, \ldots, M_k$, time-stamps $T_1, \ldots, T_k$, and verification information $V_1, \ldots, V_{k-1}$.

To verify proof of existence for the document $d_i \in \{d_1, \ldots, d_m\}$, a retriever uses the evidence as follows. At the current time $t \in [t_k, t_{k+1}]$, she uses the evidence for d_i to construct the leaf corresponding to d_i in the last Merkle tree M_k. Next, she uses the authentication path $A_{i,k}$ in the evidence to construct the root r_k of this tree. She collects verification information V_k for the time-stamp T_k on $h_k||r_k$ and checks that T_k is valid at time t. If this check is successful and $k = 1$, then the verification is complete. Otherwise, k is replaced by $k - 1$, t is replaced by t_k, the verification information V_k for T_k is extracted from the evidence, and the procedure is repeated

using the verification information provided in the evidence for d_i. Note that not in all steps new Merkle trees were created. Therefore, new roots of Merkle trees must only be created and verified when they are used for the first time. When this is not the case, the retriever verifies whether T_k is a valid time-stamp on $h_k||h_k(T_{k-1}||V_{k-1})$.

Finally, note that like CIS, ERS allows the archivist to use a single time-stamp sequence as proof of existence for a set of documents. However, in ERS all documents in the set must be time-stamped at the same time.

3.2 Notarial schemes

In the previous section, we have seen how sequences of time-stamps can extend the proof of existence of documents indefinitely. *Notarial attestations* are a different approach to solving the same problem but constructing simpler evidence. In this section we first introduce notarial attestations. Afterwards, we describe a long-term notarial scheme.

3.2.1 Notarial attestations

The idea is that time-stamps are replaced by *notarial attestations*. In such attestations, a trusted third party called *notary* attests by his signature that one or more signatures were valid at certain times, for example, when the attestation was created. Retrievers who trust the notary only need to verify the notary's signature but not the attested signatures. In order to extend the validity of the original signatures indefinitely, this process needs to be repeated since the signatures of the notaries expire. In this case, creation times, hash functions, and signatures must be selected as explained in Section 3.1.2. However, retrievers need to verify only the most recent notary signature. This efficiency improvement requires stronger trust assumptions as will be explained in Chapter 4.

3.2.2 Cumulative Notarizations

In the Cumulative Notarizations [30] solution, an archivist requests notarial attestations as evidence for documents in the archive. However, in contrast to the time-stamping schemes, verification information for signatures is not accumulated. We next detail how this evidence is generated and which objects must be stored. Finally, we explain how the evidence is verified.

Assume that the archivist has a document d, a signature s on d, and the certificate c required to verify s. To initialize evidence, he proceeds as follows. First, he selects

a hash function h and requests a time-stamp T on $h||h(d||s)$ from a TSA. Next, he requests a notarial attestation (i.e. signature) from a notary by sending d, s, c, and T. The notary is a trusted third party. The notary collects verification information for s and T and checks that s was valid at the creation time of T and that T is valid at the current time. The notary creates and returns a signature s_1 on $d||s||c||T$. The archivist stores s_1 and T together with d, s, and c.

Now assume for $k > 1$ that the archivist has obtained signatures $s_1, \ldots, s_{k-1}$. He requests the next signature s_k at time t_k, before s_{k-1} becomes insecure. To that end, he sends $d, s, c, T, s_1, \ldots, s_{k-1}$ to a notary. This notary is not necessarily the same who created s_{k-1}. The notary obtains verification information for s_{k-1}, checks that s_{k-1} is valid at the current time, creates a signature s_k on $d||s||c||T||s_1|| \ldots ||s_{k-1}$, and returns s_k to the archivist. The archivist stores s_k together with d, s, c, T, and $s_1, \ldots, s_{k-1}$.

The evidence for the document d is as follows. At the current time $t \in [t_k, t_{k+1}]$, the evidence consists of the signature s on d, the certificate c needed to verify s, the time-stamp T on $h||h(d||s)$, and the notary signatures $s_1, \ldots, s_k$.

Finally, a retriever uses the evidence to check the authenticity, non-repudiation, and proof of existence of the document d. More precisely, she verifies at the current time $t \in [t_k, t_{k+1}]$ that the most recent signature s_k is valid. The validity of this signature is sufficient to guarantee the mentioned protection goals because the retriever trusts each notary involved. This is because she sees that the notaries attested the previous signatures. In particular, the first notary attested the signature s on the document d and the time-stamp T, which together provide authenticity, non-repudiation, and proof of existence for d.

3.3 Replication schemes

As we have seen in Sections 3.1 and 3.2, time-stamping and notarial schemes generate evidence of authenticity, non-repudiation, and proof of existence using hash functions and digital signatures. In contrast, replication is a different approach where no evidence is generated. In this section we first explain this approach and then present a replication scheme.

Replication guarantees only the long-term integrity of documents. The idea is to replicate the documents and distribute them to several peers. Copies of the same document are compared on a regular basis and integrity is proved by means of a majority vote.

Lots of Copies Keep Stuff Safe (LOCKSS) [35] is currently the only known solution

based on replication and it works as follows.

First, an archivist distributes the documents in his archive to a number of peers. Contrary to time-stamping and notarial schemes, in LOCKSS the archivist verifies integrity and not the retrievers. In order to do this for a document d, the archivist invites peers $p_1, \ldots, p_q$ to participate in a poll. The number q of peers is at least 3. The archivist selects a hash function h, and does the following for $1 \leq j \leq q$. He generates random strings r_j and sends messages $\{h, r_j\}$ to the peers p_j. Each peer p_j computes the hash $y_j = h(r_j||d)$ using its local copy of d and sends y_j to the archivist. He compares y_j with $h(r_j||d)$, where d is his copy of the document. If y_j equals $h(r_j||d)$ for more than half of the peers, then the archivist believes in the integrity of d. Otherwise, he iterates the following. He replaces his current copy of the document d by a copy from a disagreeing peer and compares votes y_j with $h(r_j||d)$ again. This procedure terminates and provides the uncorrupted document d as long as the majority of the peers is honest.

3.4 Schemes that fail to provide long-term protection

There are several solutions designed to provide the protection goals integrity, authenticity, non-repudiation, and proof of existence for archived documents. However, many of them fail in achieving these goals in the long term. We present some of these schemes that are based on inspiring ideas and show why they fail to provide long-term protection.

The State of Victoria in Australia developed the Victorian Electronic Record Strategy (VERS) to provide evidence of authenticity and non-repudiation for documents [52, 59]. This evidence consists of signatures created by authorized VERS users. Their public keys are certified by the archivist instead of a certification authority (CA), because the authors of VERS argue that CAs are likely to disappear during the archival period of documents. An authorized VERS user can re-sign an archived document, but not the previous signatures. Therefore, authenticity and non-repudiation are lost as any signature becomes insecure over time.

Certified Accountable Tamper-Evident Storage (CATS) [62] is a network storage system where users can modify stored documents. CATS generates evidence of authenticity, non-repudiation, and proof of existence for the history of changes on each document. Moreover, CATS provides proof of non-existence for a document, which means a date when the document was not present in archive. This evidence also guarantees accountability since it prevents CATS and users from denying their actions. The evidence consists of authenticated search trees, signatures, and widely

visible media. However, because CATS does not address the aging of cryptography, the protection goals and accountability cannot be ensured in the long term.

Auditing Control Environment (ACE) is a scheme by Song and JáJá [46] that provides proof of existence for documents. These documents can be submitted to the archive sequentially. ACE build two types of Merkle trees to be used as evidence. The first type of tree is built from the hashes of the new documents to be archived. The second type of Merkle tree is built from the roots of the first type of Merkle trees. The roots of the second type of Merkle trees are published on widely visible media. However, when the used hash function needs to be replaced, ACE rebuilds only the first type of Merkle tree. Therefore, roots published on widely visible media can no longer be used as evidence after the hash function becomes insecure. Consequently, the proof of existence of the archived documents is lost.

Oprea and Bowers [41] design a scheme to provide proof of existence and non-existence for documents. The authors propose a data structure that realizes an append-only, persistent, authenticated dictionary by combining Merkle trees with Patricia trees [28]. The scheme also uses signed time-stamps but the authors do not address the aging of hash functions and signatures. Therefore, the scheme can only guarantee proof of existence and non-existence for a limited time.

4 Trustworthiness analysis

In this chapter, we analyze the trustworthiness of the long-term protection schemes presented in Chapter 3. We start with a qualitative analysis which presents the trust assumptions necessary for each protection scheme. Next, we show how to quantify the trustworthiness of the evidence generated by these schemes. More precisely, we estimate how likely evidence is to indeed provide the promised protection goals (integrity, authenticity, non-repudiation, and proof of existence). To this end, we use *belief trust models*, which have already been used to quantify the trustworthiness of public keys. However, these trust models need as input the reputation of the involved parties, namely time-stamp authorities or notaries. To address this issue, we propose a reputation system for long-term archiving schemes that allows to determine the required data. Finally, we demonstrate that if a document is archived for several years, involving many parties that renew the evidence for the document, then its trustworthiness degrades over time. The content of this chapter was published as parts of [55, 58].

4.1 A qualitative analysis based on trust assumptions

We presented in Chapter 3 the long-term protection schemes *Advanced Electronic Signatures* (AdES), *Content Integrity Service* (CIS), *Evidence Record Syntax* (ERS), *Cumulative Notarizations* (CN), and *Lots of Copies Keep Stuff Safe* (LOCKSS). In this section, we first present the trust assumptions that are common for long-term archiving applications. Next, we compare the schemes based on the assumptions they require.

Evidence initialization trust In some schemes, it is necessary to trust the archivist to initialize evidence properly. The illegal reinitialization of evidence allows for tampering with non-repudiation as follows. The archivist can change the archived document and the date since when it existed by adapting the document, requesting a time-stamp on the new document, and replacing the existing document and sequence

of time-stamps by the new document and the new time-stamp. Furthermore, the archivist can also change only the date since when a document existed to the current date. To do this he requests a fresh time-stamp on a document and replaces the existing time-stamp sequence by the new time-stamp.

Crypto trust In all schemes that use cryptographic components, such as hash functions and signatures, these components are renewed on a regular basis. This is because in the long term key lengths become too small to guarantee the authenticity of signatures, signature keys may be compromised, and signature schemes or hash functions may become insecure. However, despite this renewal the problem of *sudden break of cryptography* remains. This problem refers to the situation where cryptographic keys or schemes become insecure before they are renewed. This may happen for several reasons. For example, a device that contains a secret key may be compromised or cryptanalytic progress may happen unexpectedly. In such situations, later replacement of cryptographic components is of no use. For example, suppose that a secret signature key is compromised before signatures that have been issued with this key are renewed. Then an adversary who knows the secret key can replace the respective signatures without retrievers being able to notice this. Therefore, an important assumption that is required in several schemes is that cryptographic components are not broken before they are renewed.

Widely visible media trust Trustworthy widely visible media (WVM) are necessary for schemes that rely on WVM-based time-stamps. Trustworthy WVM require that their content is uncorrupted and that the issuer of the WVM provided the correct date.

CA trust In several long-term schemes, signatures must be verified to check authenticity, non-repudiation, and proof of existence of a document. However, signature verification relies on the authenticity of the public verification key. This authenticity is deduced by the signature verifier from a valid certificate chain. The validity of the certificate chain relies on all certification authorities (CAs), in particular the trust anchor, to be trustworthy. That is, for every certificate a CA issues, the CA is trusted to correctly identify the entity owning the corresponding secret key, to correctly issue the certificate to this entity, and to operate a reliable revocation system. For example, signature verifiers can decide whether a CA is trustworthy by evaluating its certification practice statement[3].

[3] A document where a CA declares how it issues certificates. This document presents, for example, how the CA authenticates public key owners before signing their certificates. For details see [11].

TSA trust There are two types of time-stamps. The first type is the signature-based time-stamps and it needs TSA trust. TSA trust means that TSAs are trusted to include the time when the time-stamp was issued in their time-stamps. The second type is WVM-based time-stamps and it requires only widely visible media trust.

Notary trust Notarial schemes rely on trustworthy notaries to obtain attestations. The archivist sends signatures to notaries and they are trusted to attest that the received signatures were in fact valid. Additionally, notaries must be trusted when they attest that received hash functions were indeed secure.

Replication trust In replication schemes, a document is distributed to several peers and its integrity is checked by using majority voting. For replication to guarantee integrity, the archivist is trusted to find enough peers so that the necessary number of votes can be obtained. Moreover, the majority of these peers is trusted to compute their votes properly and to provide the correct copy of the document when requested. Also, the majority of peers is trusted to have uncorrupted copies of the document.

Integrity verification trust In schemes that produce no evidence of integrity, retrievers of documents cannot verify integrity by themselves. Therefore, integrity verification is delegated to archivists, who are trusted to carry it out correctly.

We now compare the schemes with respect to the trusted assumptions presented above. This comparison is summarized in Table 4.1.

To achieve the protection goals, most of the schemes make assumptions with respect to the archivist. The time-stamping schemes AdES, ERS, and CIS require that the archivist does not reinitialize evidence (evidence initialization trust). In contrast, in the notarial scheme CN this assumption is not required because notaries check evidence and therefore can notice whether it was reinitialized. The same holds true for LOCKSS since this scheme does not request time-stamps from third parties.

However, in LOCKSS integrity of a document depends on the archivist and the peers that store copies of the document. More precisely, LOCKSS assumes that the archivist can find enough peers which are honest and hold uncorrupted copies of the document. Thus, LOCKSS requires replication trust. Furthermore, LOCKSS requires integrity verification trust, which means that it is assumed that the archivist correctly verifies the integrity of the archived documents.

Table 4.1: The required trust assumptions for the protection schemes. Optional items are marked with *.

	Time-stamping			Notarial	Replication
Trust assumptions	AdES	ERS	CIS	CN	LOCKSS
Evidence initialization	✓	✓	✓	✗	✗
Crypto	✓	✓	✓	✓	✗
Replication	✗	✗	✗	✗	✓
Integrity verification	✗	✗	✗	✗	✓
TSA	✓	✓	✗	✓	✗
CA	✓	✓	✓*	✓	✗
Widely visible media	✗	✗	✓	✗	✗
Notary	✗	✗	✗	✓	✗

Since LOCKSS generates no cryptographic evidence of integrity, it is not vulnerable to the sudden compromise of cryptographic algorithms or their parameters. Therefore, LOCKSS needs no crypto trust. On the other hand, time-stamping and notarial schemes guarantee proof of existence and where applicable also authenticity and non-repudiation by generating cryptographic evidence[4]. Because the used cryptography can be suddenly compromised, all time-stamping and notarial schemes need crypto trust.

Time-stamps can be based on either signatures or widely visible media. Signature-based time-stamps require TSAs to include the correct time in their time-stamps. The schemes AdES and ERS use signature-based time-stamps and therefore require TSA trust. WVM-based time-stamps need no TSAs because time is inherited from the WVM. That is, the time associated with a WVM-based time-stamp is the time when the WVM was issue. CIS uses such time-stamps and therefore requires WVM trust instead of TSA trust.

The notarial scheme CN provides authenticity, non-repudiation, and proof of existence by means of notarial attestations. These attestations are created (i.e. signed) and renewed (i.e. verified and re-signed) by notaries. Therefore, CN requires notary trust, that is, notaries are trusted to verify old attestations properly before re-signing them.

Certification authorities are needed to authenticate TSAs' or notaries' verification public keys. Thus, the time-stamping schemes AdES and ERS and the notarial scheme CN require CA trust. Note that this trust assumption is optional for CIS

[4]Note that this is also the case of CIS, since in CIS hash functions are used to produce the WVM-based time-stamps.

because it uses WVM-based time-stamps. Such time-stamps use no digital signatures and, therefore, require no authenticated public keys. (The authenticity of these time-stamps relies on WVM trust.) CIS requires CA trust if CIS is used to achieve authenticity and non-repudiation for documents signed by their originators.

We now compare the schemes with respect to the required trust assumptions. LOCKSS appears to be the least trustworthy scheme. This is because LOCKSS is the only scheme that produces no evidence and therefore retrievers cannot verify by themselves whether integrity of documents is guaranteed. Thus, retrievers rely on the archivist for integrity verification, which seems to be the strongest trust assumption found in Table 4.1.

Next comes the notarial scheme CN, followed by the time-stamping schemes. Time-stamping schemes are likely to be the most trustworthy schemes because their trust assumptions are weaker than the notarial assumptions. More precisely, TSA trust is weaker than notary trust. This is because TSAs are trusted to only provide the correct time while retrievers can verify all generated evidence. However, in the notarial scheme the task of verifying evidence is delegated to notaries. Therefore, retrievers must trust the notaries to execute this task correctly.

However, with respect to several time-stamping schemes, it is still unclear whether WVM-based time-stamps should be preferred over signature-based time-stamps or the other way around. It should be noted that WVM make only sense if realized as hard copies because electronic copies would require evidence of proof of existence themselves. On the other hand, retrievers may be unable to verify the authenticity of a hard copy of WVM by themselves. Thus, it is likely that retrievers decide to believe in WVM only if they trust the archivist who preserves the hard copy. In this case, the archived documents and the hard copies of WVM should not be preserved by the same archivist. Otherwise, he could tamper with the documents and forge WVM-based evidence without being noticed. Therefore, although we can find TSAs issuing WVM-based time-stamps, more research is needed to understand WVM requirements in the long term and to judge how this approach compares with signature-based time-stamps.

Nonetheless, our qualitative comparison has the some limitations. First, some assumptions are hard to be used to compare schemes. For example, can we say that time-stamping schemes are more trustworthy than the notarial scheme even though the additional assumption evidence initialization is required? Second, the trustworthiness of each individual TSA or notary was not taken into account. TSAs and notaries can differ in trustworthiness and such differences could be observed, for example, in the reputation that these parties build over time. Thus, even if TSA trust is weaker than notary trust, retrievers may prefer the notarial scheme over a

time-stamping scheme if the notarial scheme uses notaries with higher reputation than the TSAs in the time-stamping scheme.

In the next section we will analyze trust by looking at the evidence that the schemes generate. This analysis is not intended to compare schemes as here, but rather to estimate how likely their evidence is to guarantee the promised protection goals. This approach will take into account the reputation of involved parties, which is a limitation of the qualitative approach.

4.2 Quantifying the trustworthiness of evidence

This section presents two contributions. First, it describes how to use a computational trust model and the reputation of involved parties to estimate the trustworthiness of evidence for an archived document. The trust model outputs probabilities which estimate how likely the evidence generated by a protection scheme indeed guarantees the promised protection goals. Like in the earlier sections, these goals include integrity, authenticity, non-repudiation, and proof of existence. Here, we deal only with time-stamping and notarial schemes because LOCKSS produces no evidence. Moreover, we assume that evidence is signature-based because this approach has been largely adopted. Also, widely visible media-based evidence is not covered here because it is unclear whether such media can be used for the long term. Next, we present an overview of our proposal by introducing the involved participants, the computed probabilities, and identified limitations. However, to be able to use a computational trust model we need the respective reputation data. Thus, the second contribution of this section is to show how to realize a reputation system that provides reputation information as input to the trust model.

4.2.1 Overview

In the long-term protection schemes for digital archives usually the following entities participate. First, there are archivists, who request evidence, e.g. time-stamps, for documents stored in their archives. Second, there are *evidence generators*, which generate evidence for documents on archivists' requests. Evidence generators can be time-stamp authorities (TSAs) or notaries. Certification authorities that issue certificates for TSAs, notaries or document signers are also evidence generators. Third, there are retrievers who obtain documents and the corresponding evidence from archives. Retrievers verify whether the evidence for a document, e.g. a time-stamp sequence, is cryptographically correct by checking, for example, the signatures contained in the evidence.

However, verified evidence is not necessarily trustworthy because the protection schemes require certain trust assumptions. Thus, we propose a reputation system to provide information that helps to approximate the trustworthiness of such evidence. This information is generated by three types of participants (retrievers, archivists, and evidence generators). First, after the evidence for a document has been generated, they verify it and rate the evidence positively if they think it is correct; otherwise negatively. Thus, these parties need to verify not only whether the evidence is cryptographically correct, but also whether it guarantees the promised protection goals. For example, if evidence is provided in the form of a time-stamp, it is checked whether the time-stamp signature is valid and the time contained in the evidence is feasible. Archivists can verify whether the contained time is feasible as soon as they request a new time-stamp. However, a retriever may verify time-stamps long after they were created. Therefore, we cannot expect that archivists and retrievers verify proof of existence with the same accuracy. Moreover, an evidence generator must be prevented from rating his or her own evidence. These two issues will be addressed in Section 4.2.3.

The positive or negative ratings given by retrievers, archivists, and evidence generators are named *experiences* and are stored in the reputation system. From the experiences we then compute the reputation of a generator and two probabilities that approximate the trustworthiness of the evidence. The first probability is called *trust score of evidence* and approximates how likely the evidence of a specific document is to be correct, that is, to guarantee the promised protection goals. The second probability is named *trust score of generators* and it estimates how likely the involved generators create evidence correctly. The trust score of a generator is derived from his or her reputation and takes into account the reported experiences on all evidences that he or she has ever produced. The reason why we use two distinct probabilities is that we take into account not only the trust in the evidence for a specific document (trust score of evidence), but also the up-to-date reputation of the corresponding evidence generators (trust score of generators).

We illustrate our approach with an example. We start with the *trust score of evidence*. For $k > 0$, assume there are an archived document d and pieces of evidence $E_1, \ldots, E_k$, e.g. time-stamps, created at times $t_1, \ldots, t_k$. These pieces are used to convince retrievers at the current time $t > t_k$ that d has the promised protection goals. Now assume that $E_1, \ldots, E_k$ have been verified by retrievers, archivists, and evidence generators. Then, for $1 \leq j \leq k$, the experiences of those who verified evidence E_j are used to compute the probability ω_{E_j} that E_j is correct. Next, the *trust score of evidence* is computed from the probabilities $\omega_{E_1}, \ldots, \omega_{E_k}$. The computed trust score is the probability that pieces $E_1, \ldots, E_k$ are all correct. We

provide further details how the probabilities and the trust score are computed in Section 2.11.

We now illustrate the *trust score of generators* using the same archived document d and pieces of evidence $E_1, \ldots, E_k$ from the previous example. For $1 \leq j \leq k$, assume generator G_j produced evidence E_j for document d at time t_j. Also, G_j may have produced further evidence for other archived documents. These further pieces of evidence are also verified and rated. The experiences on all evidence that G_j has ever created constitute the up-to-date reputation of G_j. All these experiences are then used to compute a probability ω_{G_j} that G_j produces correct evidences. Next, the *trust score of generators* is computed from the probabilities $\omega_{G_1}, \ldots, \omega_{G_k}$. The *trust score of generators* is the probability that generators $G_1, \ldots, G_k$ are all trustworthy.

Note that the two proposed trust scores approximate the trustworthiness at distinct points in time. While the *trust score of evidence* indicates how likely the evidence of a document was generated correctly **at the time when it was generated**, the trust score of the generators reflects the **current** *reputation* they have built over time. This value can also be used by the archivists to avoid untrustworthy generators when requesting new evidence.

On the other hand, the *trust score of the evidence* is needed to prevent retrievers from trusting in an untrustworthy evidence generated by a trustworthy party. Assume a generator has high reputation, but one day he or she suffers an attack and generates a false evidence. Since retrievers may trust an incorrect evidence because it was issued by a reputable generator, a second trust value is needed that evaluates the trustworthiness of the evidence itself.

The approach we just presented has some limitations. For now, we neither determine nor consider trust values for the involved archivists, although they participate in the creation of evidence. The reason is that their role in a protection scheme is quite different from the role of the generators; therefore, distinct mechanisms to compute the trust put in archivists are necessary. Moreover, most of the parties cannot easily verify whether archivists perform their role correctly. The individual who could identify misbehavior from an archivist is the owner of the archived document. He or she knows the protection goals of the document and could notice whether the archivist tampers with them. However, this individual has limited lifetime and cannot observe the archivist's behavior indefinitely. Also, some documents may have no owners. Therefore, for now we assume that the archivist is trusted to generate evidence correctly.

As seen, our approach allows to approximate the trustworthiness of evidence generated with protection schemes that rely on time-stamps, notarial attestations, and

certificates. However, as mentioned before a reputation system is needed that allows us to determine the respective reputation data. For CAs, we can use the reputation systems designed by Braun et al. [5]. Additionally, the reputation of CAs could be derived from their certification policies as Wazan et al. [60] propose. These two solutions for CAs are presented in Section 4.2.4. For time-stamp authorities and notaries we will present a reputation system in Section 4.2.3.

We next present our approach in the following order. First, Section 4.2.2 shows how to compute the trust scores introduced above. Next, Section 4.2.3 presents how to realize a reputation system that provides input for computing trust. In Section 4.2.4 we examine alternative approaches for approximating the trustworthiness of evidence. Finally, we demonstrate in Section 4.2.5 that, if the number of involved generators increases in the long term, then the trustworthiness of evidence tends to decay.

4.2.2 Computing trust scores

This section presents how to approximate the trustworthiness of evidence produced by time-stamping or notarial schemes. More precisely, we detail how to compute two probabilities that estimate how likely a given time-stamp sequence or attestation sequence is to guarantee the promised protection goals of a specific document. The first probability is called *trust score of evidence* and the second is named *trust score of evidence generators*. To this end, we are going to use the trust model called *Subjective Logic*. As justified in Section 2.11, we choose this model because it is well-established and uses mathematics which is easy for the reader to comprehend. However, Subjective Logic can be replaced by the *CertainTrust* trust model, because both models are interchangeable. Together with the selected model, we will use the reputation scores from the parties involved in the creation of evidence. Such reputation scores can be obtained using a reputation system as explained in Section 4.2.3.

We start by explaining how a retriever computes the *trust score of evidence* (ω_E). For $k > 1$, the retriever first obtains at the current time $t > t_k$ a document d and the corresponding evidences $E_1, \ldots, E_k$ which is either a time-stamp sequence or attestation sequence. In addition, for each $1 \leq j \leq k$ she obtains for evidence E_j three reputation scores from the reputation system. The first reputation score is $\vec{X}^A_{E_j} = (r^A_{E_j}, s^A_{E_j})$, where $r^A_{E_j}$ is the sum of the positive experiences that archivists had on E_j, $s^A_{E_j}$ is the sum of the negative experiences from archivists had on the same E_j. Similarly, the second and third reputation scores $\vec{X}^G_{E_j} = (r^G_{E_j}, s^G_{E_j})$ and $\vec{X}^R_{E_j} = (r^R_{E_j}, s^R_{E_j})$ are computed from the evidence generators' and retrievers' experiences on

E_j, respectively. The three reputation scores are computed in the same manner but differ in the type of participants.

From these reputation scores, the retriever computes the weighted average reputation score $\vec{X}_{E_j}$ of each evidence E_j. To this end, she uses non-negative weights w_A, w_G, and w_R for the reputation scores from the archivists, evidence generators, and retrievers experiences, respectively. These weights are parameters of the reputation system and represent how much each type of participant is trusted to report their experiences correctly. As we will see in Section 4.2.3, archivists and evidence generators can check an evidence as soon as it is generated whereas retrievers may check it much later. Thus, archivists and evidence generators can provide a more reliable opinion about the protection goals (e.g. proof of existence) than retrievers. Also, over time the number of retrievers that rate evidence may be larger than the number of archivists and evidence generators. Therefore, we suggest that $w_A \approx w_G > w_R$. Next, the retriever computes $\vec{X}_{E_j}$ using the weighted average operator presented in Section 2.11, Equation 2.1. That is, $\vec{X}_{E_j} = (r_{E_j}, s_{E_j})$ such that

$$r_{E_j} = \frac{w_A \cdot r^A_{E_j} + w_G \cdot r^G_{E_j} + w_R \cdot r^R_{E_j}}{w_A + w_G + w_R}, s_{E_j} = \frac{w_A \cdot s^A_{E_j} + w_G \cdot s^G_{E_j} + w_R \cdot s^R_{E_j}}{w_A + w_G + w_R}.$$

The final steps concern the trust model. The retriever maps each weighted average reputation score $\vec{X}_{E_j}$ into an opinion ω_{E_j} using the mapping function shown in Section 2.11, Equation 2.3. That is, $\omega_{E_j} = (b, d, u)$ such that

$$b = \frac{r_{E_j}}{r_{E_j} + s_{E_j} + 2}, d = \frac{s_{E_j}}{r_{E_j} + s_{E_j} + 2}, u = \frac{2}{r_{E_j} + s_{E_j} + 2}.$$

Finally, she calculates the trust that all evidences E_j are correct based on the corresponding opinions ω_{E_j}. This trust is the opinion ω_E and is computed using the conjunction operator explained in Section 2.11, Equation 2.4. That is,

$$\omega_E = \omega_{E_1} \wedge \ldots \wedge \omega_{E_k}.$$

We now describe how to compute the *trust score of evidence generators* (ω_G) given the evidence $E_1, \ldots, E_k$ of document d. Initially, the retriever identifies the distinct generators $g_1, \ldots, g_l$ of each piece of evidence in $E_1, \ldots, E_k$, where $l \leq k$. Note that the condition $l \leq k$ is necessary because one generator can create one or more pieces of evidence for the same document. Next, for every generator g_j ($1 \leq j \leq l$) she obtains from the reputation system three reputation scores. The first reputation score is $\vec{X}^A_{g_j}$ which means the addition of archivists' experiences on any evidence that g_j has created so far. The second and the third reputation scores are $\vec{X}^G_{g_j}$ and

$\vec{X}^R_{g_j}$ which correspond to evidence generators and retrievers, respectively, and are analogous to the reputation score $\vec{X}^A_{g_j}$.

The next steps are similar to the previously computed trust score of evidence. For every generator g_j and the corresponding reputation scores $\vec{X}^A_{g_j} = (r^A_{g_j}, s^A_{g_j})$, $\vec{X}^G_{g_j} = (r^G_{g_j}, s^G_{g_j})$, and $\vec{X}^R_{g_j} = (r^R_{g_j}, s^R_{g_j})$, the retriever computes the weighted average reputation score $\vec{X}_{g_j}$. Again, she uses weights w_A, w_G, and w_R which represents how much archivists, evidence generators, and retrievers are trusted to report their experiences correctly. The average reputation scores $\vec{X}_{g_j}$ are computed with the weighted average operator. That is, $\vec{X}_{g_j} = (r_{g_j}, s_{g_j})$ such that

$$r_{g_j} = \frac{w_A \cdot r^A_{g_j} + w_G \cdot r^G_{g_j} + w_R \cdot r^R_{g_j}}{w_A + w_G + w_R}, s_{g_j} = \frac{w_A \cdot s^A_{g_j} + w_G \cdot s^G_{g_j} + w_R \cdot s^R_{g_j}}{w_A + w_G + w_R}.$$

Next, the retriever maps every $\vec{X}_{g_j}$ to an opinion ω_{g_j} using the mapping function. That is, $\omega_{g_j} = (b, d, u)$ such that

$$b = \frac{r_{g_j}}{r_{g_j} + s_{g_j} + 2}, d = \frac{s_{g_j}}{r_{g_j} + s_{g_j} + 2}, u = \frac{2}{r_{g_j} + s_{g_j} + 2}.$$

The computed opinion ω_{g_j} approximates the trustworthiness of generator g_j. From these computed opinions, the retriever finally calculates the opinion ω_G that the l generators are trustworthy using the conjunction operator. That is, she computes

$$\omega_G = \omega_{g_1} \wedge \ldots \wedge \omega_{g_l}.$$

As seen, we assume that the retriever obtains reputation scores from a reputation system. In the next section, we design such a reputation system to provide the required information.

4.2.3 Realizing a reputation system

In this section, we describe a reputation system where participants report their experiences on the evidence for archived documents. From these experiences, the reputation system computes the reputation scores needed as input for the trust model presented in Section 4.2.2.

Participants

As mentioned before, there are three types of participants in the reputation system. The first type is *evidence generators*, which include time-stamp authorities and notaries that generate evidence on archivists' requests. The second type are archivists

and the third type are retrievers of documents from archives. Archivists and retrievers verify generated evidence and rate it by reporting their experiences, i.e. whether they think evidence is cryptographically correct and indeed guarantees the promised protection goals. An experience is realized as a boolean, i.e. 1 refers to a positive experience whereas 0 to a negative one. Evidence generators also verify evidence generated by other generators and report their experiences to the reputation system. Participants and their tasks are illustrated in Figure 4.1.

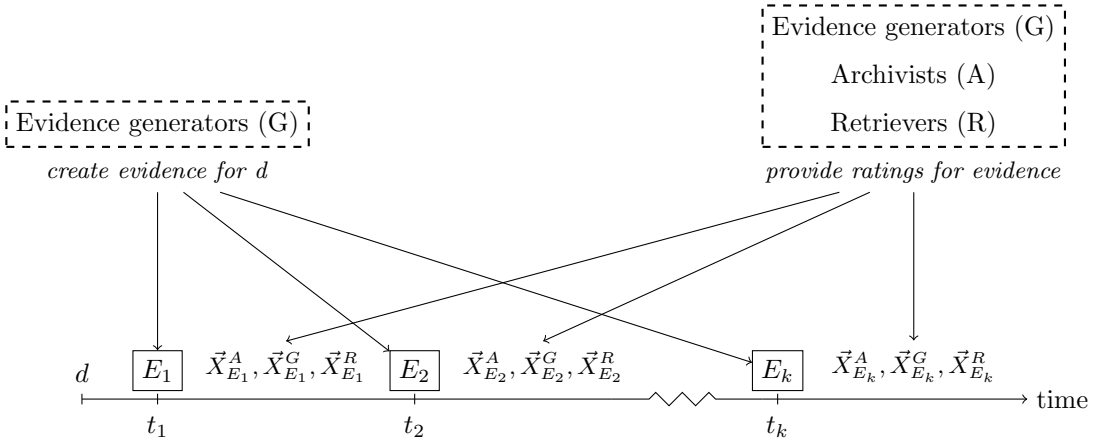

Figure 4.1: The evidence E_k for document d being created at the time t_k and then rated at the time $t > t_k$, $k > 0$. The ratings provided by evidence generators, archivists, and retrievers are represented by the reputation scores $\vec{X}^G_{E_k}$, $\vec{X}^A_{E_k}$, and $\vec{X}^R_{E_k}$, respectively.

Reporting experiences

The reputation system writes the reported experiences on an online accessible bulletin board. A bulletin board is a write-only medium where written data cannot be deleted and can be read consistently by anyone (for details, see [24]). Such a medium is desired because experiences on long-term evidence must be stored indefinitely for future retrievers. Another reason to use a bulletin board is that it allows the participants of the reputation system to audit all stored experiences. As we will show later, this allows a retriever to check whether the reputation system computes trust scores from the stored experiences correctly.

Together with every reported experience, the reputation system also publishes the type of participant that reported the experience, the corresponding evidence, and the name of the generator of this particular evidence on the bulletin board. Distinguishing the type of the participant (archivists, evidence generators, or retrievers) is important because their experiences may not be equally reliable. As we will see

next, this is because participants are likely to check evidence at different times. In order to distinguish the type of the participants and also to prevent sybil attacks[5] against the reputation system, participants are required to authenticate themselves before reporting experiences.

Recall that an experience conveys a participant's judgment whether a piece of evidence for a document indeed guarantees the promised protection goals of this document. This judgment is based on the verification of the piece of evidence. We explain this verification for a time-stamp and attestation. If the piece of evidence is a signed time-stamp, the participants first check that the time-stamp signature is valid. Next, they check that the time in the time-stamp is recent. How far the time included in the time-stamp is allowed to deviate from the time when the participants perform the verification is a parameter of the reputation system. Note that this verification is only possible if it happens soon after the time-stamp is created. Afterwards, participants must use other methods to verify the time contained in the time-stamp. For example, they can check archival logs recording the date when the time-stamp was stored together with the document in the archive.

The verification of evidence is more evolved when it comes to a notarial attestation. In this case, the participants verify that not only the time of the attestation is recent but also that the attested data has the promised protection goals. Therefore, the participants must be provided with the new attestation, the previous attestations, and the corresponding protected document. Note that the verification needs also to take place soon after the attestation was created so that proof of existence can be checked.

As seen, the time when participants check evidence is quite important. This is because the sooner evidence is checked, the more likely wrongly generated evidence can be detected. Thus, the reputation system needs to classify experiences (i.e. the verification results) with respect to when they were reported. This classification can be done by using the type of the participant that reports an experience as follows. Archivists who request evidence can verify it as soon as they receive new evidence. Evidence generators can also verify evidence soon after its creation if the archivists provide the generators with new evidence timely. In contrast, retrievers are likely to access evidence long after it was created; therefore, they may be unable to check evidence as accurately as archivists and evidence generators.

[5]Malicious users use false identities to subvert a reputation system.

Creation, verification, and rating of evidence

For the reputation system to ensure that participants provide their experiences timely, we now present a protocol for the creation, verification, and rating of evidence.

1. At an initial time t_k, an archivist selects an evidence generator g_k and requests new evidence from g_k for a document d.

2. Generator g_k produces new evidence E_k and returns it to the archivist.

3. The archivist verifies E_k as explained before and submits his experience together with E_k and the identification of g_k to the reputation system.

4. The reputation system publishes the archivist's experience together with g_k, E_k, and the participant type A on the bulletin board, where A refers to archivists.

5. The reputation system randomly selects $r > 0$ generators other than g_k that will be allowed to report their experiences on E_k. We assume that r generators can always be found and that they are selected truly at random to reduce the change of collusion against or in favor of evidence generator g_k. The reputation system sends the identification of the selected generators to the archivist and sets a time $t_r > t_k$. In the time period $[t_k, t_r]$, the randomly selected generators can report their experiences; after t_r, their experiences will be rejected by the reputation system. The goal is to guarantee that proof of existence is verified with a minimum accuracy of $t_r - t_k$ units of time.

6. The archivist sends the new evidence E_k to the r selected generators. If E_k is a notarial attestation, then the archivist also sends previous attestations and document d.

7. The selected generators verify E_k as described before and submit their experiences on E_k to the reputation system.

8. The reputation system publishes each submitted experience together with g_k, E_k, and the participant type G on the bulletin board, where G refers to generators.

9. After time t_r, the reputation system allows only retrievers to submit their experiences on E_k. In this case, retrievers obtain E_k from the archive and verify it. If they verify E_k soon after it was generated, then they verify E_k

like archivists and evidence generators; otherwise, retrievers need additional methods to check E_k for proof of existence.

10. Retrievers submit their experiences together with evidence E_k and identification g_k to reputation system.

11. The reputation system publishes each retriever's experience together with g_k, E_k, and the participant type R on the bulletin board, where R refers to retrievers.

Computation of reputation scores

We now explain how the reputation system computes the two types of reputation scores. As explained in Section 2.11, a reputation score of a party or information z is a vector $\vec{X}_z^C = (r_z^C, s_z^C)$, where r_z^C is the sum of all positive experiences that a community C had on z and s_z^C is the sum of all negative experiences from C on z.

The first type of reputation score is the *reputation scores of an evidence* E, where E and the corresponding experiences on E have been published on a bulletin board. The reputation scores of E are $\vec{X}_E^A = (r_E^A, s_E^A)$ computed from archivists' experiences, $\vec{X}_E^G = (r_E^G, s_E^G)$ calculated from evidence generators' experiences, and $\vec{X}_E^R = (r_E^R, s_E^R)$ (retrievers' experiences). To obtain such scores, a retriever submits E to the reputation system. The reputation system computes $\vec{X}_E^A$, $\vec{X}_E^G$, and $\vec{X}_E^R$ and returns them to the retriever.

The second type of reputation score is the *reputation scores of an evidence generator* g. Assume for $k > 0$ that the generator identified by g has produced evidences $E_1, \ldots, E_k$ so far. Then, for $1 \leq j \leq k$, the bulletin board first computes the three reputation scores $\vec{X}_{E_j}^A$, $\vec{X}_{E_j}^G$, and $\vec{X}_{E_j}^R$ of each evidence E_j as explained above. Finally, the bulletin board adds the reputation scores from the same type of participants. That is, it computes the reputation scores $\vec{X}_g^A$ (type archivist), $\vec{X}_g^G$ (type evidence generators), and $\vec{X}_g^R$ (type retrievers) such that

$$\vec{X}_g^A = (\sum_{j=1}^{k} r_{E_j}^A, \sum_{j=1}^{k} s_{E_j}^A)$$

$$\vec{X}_g^G = (\sum_{j=1}^{k} r_{E_j}^G, \sum_{j=1}^{k} s_{E_j}^G)$$

$$\vec{X}_g^R = (\sum_{j=1}^{k} r_{E_j}^R, \sum_{j=1}^{k} s_{E_j}^R),$$

as described in Section 2.11, Equation 2.2. To obtain such scores, a retriever submits g to the reputation system. The reputation system computes $\vec{X}_g^A$, $\vec{X}_g^G$, and $\vec{X}_g^R$ and returns them to the retriever.

Benefits, trust assumptions, and performance

We finally analyze the benefits of using the reputation system, identify the required trust assumptions, and indicate performance improvements. A benefit of the reputation system is that it can help to raise the trustworthiness of the evidence generators. For example, an archivist can select evidence generators with good reputation over evidence generators with bad reputation. This raises the trustworthiness of the generated evidences. Furthermore, evidence generators such as time-stamp authorities and notaries are more likely to produce correct evidences and build good reputation, if they compete with other evidence generators to be selected by archivists.

The use of a reputation system also introduces further trust assumptions. The reputation system is trusted to publish any reported experience together with the corresponding type of the reporter participant, verified evidence, and evidence creator correctly on the bulletin board. In order to guarantee the correct type of participants, the reputation system is also trusted to authenticate participants properly. Moreover, proof of existence must be verified with a minimum accuracy. Such an accuracy cannot be guaranteed if generators check an evidence long after it was created. To prevent this, the reputation system is trusted to not accept experiences submitted after an established deadline (see time t_r presented above). Note that generators are also trusted to receive evidences timely so that they can submit their experiences before the deadline.

However, to some extent the correct behavior of the reputation system can be verified. Archivists, retrievers, and evidence generators can check whether the values reported are published correctly. Furthermore, the trust assumptions could be softened if participants sign the experiences they report.

Note that the reputation system need not be trusted to compute reputation scores. This is because third parties can verify the reputation scores by retrieving the experiences from the bulletin board and computing the reputation scores by themselves. The reason for computing these scores on the reputation system's side is to reduce the communication among the bulletin board and retrievers.

Finally, the performance of the bulletin board could be improved by using cryptographic tools. For example, the bulletin board could store the hash of each evidence instead of the evidence itself. However, this would require a protection scheme to address the aging of the used hash function. We leave this challenge for future work.

4.2.4 Alternative approaches

This section presents alternative approaches for approximating the trustworthiness of the evidence for digital documents. We briefly describe theses approaches and analyze whether they can be used together with long-term protection schemes.

Trust in evidence that contains digital signatures can be derived from the trust we put in the authenticity of the public keys needed to verify these signatures. Assessing the trust in public keys authenticity is a built-in feature of the so-called PGP Web of Trust [63]. PGP is a decentralized PKI where users quantify their trust in the keys they know as *ultimate*, *completely*, *marginal*, *none*, and *unknown*. When a user receives a new key and this key has been signed (i.e. certified) by a key the user already knows, he or she can deduce his or her trust in the new key from the trust in the known key. Trust can also be deduced if there is a chain of certifications between the new key and a known one. However, these chains have a maximum length which users define. Over time, such chains tend to become too long. Moreover, users may find no chains from a known key to a very old key. Therefore, PGP Web of Trust is not recommended to be used for the long-term protection of documents.

An alternative approach is to use a centralized PKI, such as X.509. In this approach, the trust in the authenticity of public keys is derived from the trust in the certification authorities (CAs) that certify the public keys. Such PKIs provide no built-in trust assessment. To address this issue, Wazan et al. [60] propose a formal trust model to compute trust in a CA from its certification policies. A certification policy describes the policy and practices of a CA, such as how key subjects are authenticated. The approach of Wazan et al. needs an expert party that users fully trust to analyze the certification policies and compute trust in CAs. This approach could be used to provide extra input for our reputation system and, therefore, help to approximate the trustworthiness of evidence with more confidence. For example, such extra input is desired when the reputation system is initialized and contains no experiences yet.

Another solution to compute the trust in CAs is proposed by Braun et al. [5]. Instead of trusted experts, users evaluate by themselves the CAs present in the certificate chains needed to establish Transport Layer Security (TLS) [15] connections. More precisely, if a user connects to the expected website successfully over TLS, he or she stores in his or her local database that the involved CAs are indeed the subjects owning the public keys and that the CAs issued trustworthy certificates. Additionally, there is a reputation system where users can submit their experiences or obtain other users' experiences. This solution could be used together with our approach if retrievers report their experiences on certificates used as evidence for

documents.

4.2.5 The trustworthiness of evidence over time

Time-stamping and notarial schemes produce evidence to convince retrievers of the protection goals of a document. The creation of this evidence involves one or more evidence generators (viz. time-stamp or notaries), each of which is trusted to produce evidence correctly.

However, usually there is no full trust in the evidence generators and correspondingly in the evidence itself. There is always a small probability that generators act improperly. Since retrievers are required to trust every evidence generator and the number of involved evidence generators increases over time, the trustworthiness of evidence degrades as time goes by. Intuitively speaking, the longer a document is archived and the more evidence generators are involved in the evidence generation process, the more likely it is that one generator was malicious and compromised evidence. This can also be seen from the operator used to compute the conjunction between opinions about the trustworthiness of generators or evidence (see Section 2.11, Equation 2.4). If the belief b in, say, one evidence generator is $1 - \epsilon$, then the belief in k generators is $(1 - \epsilon)^k$ which converges to zero as k goes to infinity.

Dealing with trustworthiness decay of evidence is a challenge for which we see two possible approaches. The first approach is to mitigate trustworthiness decay by setting incentives for evidence generators to build good reputation and then allowing archivists to select the generators that have the highest reputations. The reputation system proposed is a first step in this direction. The second approach is to reset the trustworthiness of evidence. In this case, audit methods to reassure that the protection goals of documents are still preserved may be used. However, this is left for future work.

5 Performance analysis

In this chapter we analyze and compare the performance of several protection schemes. They are analyzed with respect to their space, time, and communication complexity. Space complexity refers to the size of the evidence generated by the schemes. Time complexity refers to the time necessary to create, update, and verify evidence. Communication complexity measures the sizes of the messages exchanged to create or update evidence. We start by assessing performance analytically, that is, without considering the cryptographic primitives used. Then, for a more realistic comparison, we implement the protection schemes using existing cryptographic primitives and carry out two experiments. In the first experiment we analyze how the schemes compare with respect to space, time, communication. In the second experiment we evaluate how distinct signature lifetimes affect the performance of the schemes. Furthermore, we use the results to verify the predictions drawn from our analytical evaluation. Finally, we show how the performance of the schemes could be improved. The content of this chapter was published as parts of [56, 58].

5.1 Analytical evaluation

In this section we provide an analytical evaluation that allows to predict and interpret the performance of several protection schemes without taking specific cryptographic primitives or hardware into account. Such an evaluation is important because we cannot estimate how future primitives and hardware will affect the performance of protection schemes.

For this evaluation, we consider the notarial scheme Cumulative Notarizations (CN) and the time-stamping schemes Advanced Electronic Signatures (AdES), Content Integrity Service (CIS), and Evidence Record Syntax (ERS). These schemes are described in Sections 3.1 and 3.2. The analysis does not include LOCKSS because the functionality of this scheme is very different from the functionalities of the other schemes. It only guarantees integrity of documents and produces no evidence. Also,

evidence is signature-based because it is still unclear whether widely visible media-based evidence can be used for the long term.

Based on the proposed evaluation, we compare notarial and time-stamping schemes with respect to their performance. To this end, we use the following conventions that make the schemes comparable:

- There are m documents of the same size in the archive. There is also a signature on each of these documents.
- All time-stamps are signed.
- All schemes proceed iteratively. In the first iteration, initial evidence for the individual protection goals is created. In the later iterations, the evidence is updated.
- Keys used to update evidence are renewed in every iteration.
- Hash functions are only renewed after r iterations, because for security reasons the lifetimes of signature keys are expected to be shorter. (In practice, the lifetime of hash functions lasts around 80 years while the lifetime of signatures around two to four years.) More precisely, since attackers can tamper with the protection goals if they compromise a signature key before the signature lifetime ends, shorter lifetimes are needed that the attackers have shorter time to succeed in their attack. Therefore, in the kth iteration we define the number of hash function renewals as $p = \lfloor k/r \rfloor$.
- Verification information for a signature contains a certificate chain that allows to reduce the trust in the public verification key to the trust in a trust anchor. It also contains the revocation information for the certificates in this chain. This verification information is assumed to be of constant size.

We analyze the 1) time complexity, 2) space complexity, and 3) communication complexity of the different schemes. For time complexity we distinguish between the time necessary 1.1) for an archivist to initialize evidence, 1.2) for the archivist to update evidence, and 1.3) for retrievers to verify evidence. Space complexity refers to the size of the evidence that is generated to establish the protection goals while communication complexity measures the size of the messages exchanged to create evidence. For the sake of simplicity, we assume that $m = 2^l$ for a positive integer l and that in ERS all the m documents are initially time-stamped using the same Merkle tree.

Moreover, we need to distinguish between two types of objects in our analysis. The first type is hashes which are the bit strings produced by hash functions. The second type is large objects whose sizes are typically bigger than the sizes of hashes. Large objects are documents, signatures, verification information, time-stamps, and attestations. This distinction is necessary to compare the times needed to hash these objects because the running times of typical hash functions is proportional to the length of the hashed objects (for example, the *sponge hash functions* [2]). Moreover, the distinction helps analyze space and communication complexities because they depend also on the length of the stored or exchanged objects.

1.1) Performance of evidence initialization: As a first step, we compare the time complexities of the creation of initial evidence for the m objects. In all schemes, a considerable amount of work is spent for hashing documents, signatures, verification information, and hashes or concatenations of such objects. As the running time of typical hash functions is proportional to the length of the hashed objects, we approximate the running time of the initialization by counting the number of hashed objects. This is done in Table 5.1.

Table 5.1: The numbers of objects being hashed while generating initial evidence for m documents.

Scheme	Documents	Signatures	Ver. info	Hashes
CN	m	m	0	0
AdES	m	m	m	0
CIS	m	m	m	0
ERS	m	m	m	$2(m-1)$

In our analysis, CN is the fastest scheme because it hashes only the documents and signatures. Next are AdES and CIS. ERS is the slowest scheme because in addition to what AdES and CIS need to hash, ERS creates the initial Merkle tree by hashing $m-1$ concatenations of node pairs.

1.2) Performance of evidence renewal: Next, we quantify the times needed for evidence creation in the kth iteration, for $k \geq 2$. Recall that m is the number of documents in the archive and that $m = 2^l$ for some positive integer l. Also, $p = \lfloor k/r \rfloor$ is the number of hash function renewals, where r is the number of iterations after which the hash function is renewed. Again, we count the number of hashed objects. Table 5.2 shows this comparison.

Table 5.2: The numbers of objects being hashed while updating the evidence for $m = 2^l$ documents in the kth iteration. Here $p = \lfloor k/r \rfloor$ is the number of hash function renewals, where r is the number of iterations after which the hash function is renewed.

Scheme	Documents	Signatures	Time-stamps	Ver. info	Hashes
CN	0	0	0	0	0
AdES	m	m	$m(k-1)$	mk	0
CIS	m	m	$m(k-1)$	mk	$2m(k-1)$
ERS-O	0	0	1	1	0
ERS-N	m	m	$m(k-1)$	mk	$mlp + 2(m-1)$

The notarial scheme CN is the fastest scheme because it hashes no objects when updating evidence. To compare the time-stamping schemes, we distinguish two situations for ERS. In the first situation (ERS-O), the archivist selects the previous hash function and hashes only the most recent time-stamp together with the corresponding verification information. In the second situation (ERS-N), the archivist selects a new hash function and builds a new Merkle tree. The leaves of the new tree are computed by hashing all documents, signatures, time-stamps, verification information, and authentication paths.

When no new hash function is selected, ERS-O is the fastest time-stamping scheme since compared with AdES and CIS only very few objects are hashed. Also, AdES is faster than CIS since in addition to the objects hashed by AdES, the CIS scheme hashes $2m(k-1)$ hashes. However, hashes are small compared with the other hashed objects. So the time difference can be expected to be small. If a new hash function is used, then in our analysis AdES is the fastest time-stamping scheme. The difference between CIS and ERS-N depends on the parameters l and p.

1.3) Performance of evidence verification: Next, we turn to approximating the time complexity of evidence verification for individual documents. Again, m is the number of documents in the archive and $m = 2^l$ for some positive integer l. Also, the number of hash function renewals is defined as $p = \lfloor k/r \rfloor$, where r is the number of iterations after which the hash function is renewed. The time in evidence verification is mainly spent for hashing data objects and verifying signatures on documents, time-stamps, or attestations. Therefore, this is what we count. By a signature verification we mean that the verification algorithm is applied to the signature and the verification information is checked. The latter means that the certificate chain from the certificate for the signature verification key to the trust anchor is validated. The trust anchor is not included in the chain and, therefore, is

not validated.

Table 5.3 compares the number of signature verifications and Table 5.4 compares the number of hashed objects required by the individual schemes to verify the evidence for one document after the kth iteration in the evidence creation.

Table 5.3: The numbers of signature verifications required while verifying the kth evidence for one document.

Scheme	Signature verifications
CN	1
AdES	$k+1$
CIS	$k+1$
ERS	$k+1$

Table 5.4: The numbers of objects being hashed while verifying the kth evidence for one document. Here $p = \lfloor k/r \rfloor$ is the number of hash function renewals, where r is the number of iterations after which the hash function is renewed. Also, there are $m = 2^l$ documents in the archive.

Scheme	Documents	Signatures	Time-stamps	Ver. info	Hashes
CN	0	0	0	0	0
AdES	k	k	$(k^2-k)/2$	$(k^2+k)/2$	0
CIS	k	k	$(k^2-k)/2$	$(k^2+k)/2$	k^2-k
ERS	$p+1$	$p+1$	$r(p^2+p)/2-2p+k-1$	$r(p^2+p)/2-p+k$	$(lp^2+5lp)/2+2l$

With respect to verification, CN is expected to be the fastest scheme since it verifies fewer signatures and hashes fewer objects than the time-stamping schemes. Comparing the time-stamping schemes is more difficult. AdES, CIS, and ERS require the same number of signature verifications. But when comparing the number of hashed objects, ERS has the advantage of hashing fewer objects than AdES and CIS. More precisely, since $p+1 \approx k/r$ we can see from the expressions that ERS hashes approximately $1/r$ as many documents, signatures, time-stamps, and verification information as AdES or CIS. On the other hand, ERS must compute $p+1 \approx k/r$ Merkle tree roots using authentication paths. However, the time for doing this can be expected not to compensate the advantage of ERS since this computation requires hashing pairs of hashes which are small compared to the other objects. Therefore, in our analysis ERS is the fastest time-stamping scheme with respect to verification of evidence. Next is AdES which has a slight advantage over CIS because CIS hashes k^2-k more hashes than AdES.

2) Space complexity: Table 5.5 compares the size of the evidence for the m documents. The evidence is stored in the archives and consists of signatures, time-stamps, individual certificates, verification information, and hashes. All schemes, except for ERS, keep separate evidence for each document. In ERS, sets of documents share the same time-stamps and the same verification information for the time-stamps. In our case, the m documents share the same time-stamps and time-stamp verification information. However, each document has its own signature, signature verification information, and authentication paths.

Table 5.5: The numbers of objects stored as the kth evidence for $m = 2^l$ documents. Here we define $p = \lfloor k/r \rfloor$ as the number of hash function renewals, where r is the number of iterations after which the hash function is renewed.

Scheme	Signatures	Time-stamps	Certificates	Ver. info	Hashes
CN	$m(k+1)$	m	m	0	0
AdES	m	mk	0	mk	0
CIS	m	mk	0	mk	0
ERS	m	k	0	$m+k-1$	$ml(p+1)$

Table 5.5 shows that in ERS the number of large objects (viz. signatures, time-stamps, certificates, and verification information) is proportional to k and m, while the number of hashes is proportional to mlp. Note that mlp is approximately mk/r and is expected to be small because the lifetime r of hash functions is usually long. This happens because hash functions are expected to be used for decades whereas signatures only for few years (e.g. between two and four years) due to security reasons.

As for CN, AdES, and CIS, evidence size grows proportionally to mk. Thus, the expressions in Table 5.5 suggest that for increasing k, the ERS scheme is the most space efficient scheme. Next comes CN, followed by AdES and CIS. The reasons for this order are as follows. In CN, only the number of document signatures is proportional to mk while in AdES and CIS the number of time-stamps and verification information is proportional to mk. Since verification information is usually larger than a single document signature, AdES and CIS should consume more space than CN. Note that although Table 5.5 shows no certificates for AdES, CIS, and ERS, these schemes store more certificates than CN. The reason is that every verification information contains a chain of certificates.

3) Communication complexity: Finally, Table 5.6 presents the communication complexity of the different schemes. Here we count the number of objects sent or

received by an archivist from or to notaries or time-stamp authorities. These objects are documents, signatures, time-stamps, notarial attestations (i.e. signatures), certificates, and hashes. Again, there are m documents in the archive. Also, $p = \lfloor k/r \rfloor$ is the number of hash function renewals, where r is the number of iterations after which the hash function is renewed.

Table 5.6: The numbers of objects being exchanged between an archivist and notaries or time-stamp authorities while creating the evidence for m documents in the kth iteration. Here $p = \lfloor k/r \rfloor$ is the number of hash function renewals, where r is the number of iterations after which the hash function is renewed.

Scheme	Documents	Signatures	Time-stamps	Certificates	Hashes
CN ($k = 1$)	m	$2m$	$2m$	m	m
CN ($k > 1$)	m	$m(k+1)$	m	m	0
AdES	0	0	m	0	m
CIS	0	0	m	0	m
ERS	0	0	1	0	1

The notarial scheme CN has the highest communication complexity. In CN, the archivist requests notary signatures for the m documents by sending the documents, their signatures, and all previous notary signatures to a notary. In turn, the notary returns one notary signature for every document. Since the communication complexity is proportional to km, CN should have by far the highest communication complexity when k grows.

AdES and CIS are much more efficient than CN because the archivist sends m hashes and receives m time-stamps. ERS is the most efficient scheme because the archivist sends one hash (a Merkle tree root or the hash of a time-stamp) and receives a time-stamp for it.

5.2 Experiments

The previous section presented a performance evaluation which can be used to compare schemes analytically. In this comparison we considered neither specific cryptographic primitives nor their parameters, such as signature algorithms and signature key sizes.

Therefore, in this section we select existing hash functions, signature algorithms, and signature key sizes to compare the protection schemes in a more realistic scenario. We start by describing how we implemented Java prototypes for the protection schemes. Next, we present two experiments. In the first experiment we use

the implementations to measure the time, space, and communication of schemes by generating evidence based on RSA signatures that are renewed every five years. We also identify the most time-consuming operations in the implementation as they are executed in this experiment. In the second experiment we evaluate how the implementations perform when larger keys for RSA signatures are used.

As before, we do not include LOCKSS into our comparison because it provides no cryptographic evidence.

5.2.1 Implementations design

We implemented prototypes for the long-term protection schemes AdES, CIS, ERS, and CN using Java 7 and the corresponding descriptions in Chapter 3. The prototypes generate and verify evidence for the protection goals authenticity, non-repudiation, and proof of existence. The evidence is stored in the XML format.

The implementations use the xades4j library [16] to create XML signatures. In turn, xades4j uses the Bouncy Castle crypto provider [29]. When the implementations use a digital signature, the corresponding public verification key is available in the form of an X.509 certificate.

To make the schemes comparable, each of these certificates is the last element c_3 of a certificate chain c_1, c_2, c_3. The certificates c_1, c_2, and c_3 are issued by certification authorities A_1, A_2, and A_3, respectively. Certificate c_2 certifies the key for verifying the signature on c_3. Likewise, c_1 certifies the key for verifying the signature on c_2. The key to verify the signature on c_1 is a trust anchor and we assume verifiers know this key. Our implementations also deal with revocation. However, for simplicity, we assume that none of the certificates used in the experiments is revoked. Therefore, we use an empty certificate revocation list for each of the certificates.

The implementations ran on Solaris 11 using an Intel i5 M560 2.67 GHz processor and 4 GB RAM. We used no code optimization (i.e. *just-in-time-compilation*) because we cannot guarantee it equally improves the performance of all prototypes.

5.2.2 Comparing schemes in the long term

In this section we use our prototypes to compare the schemes with respect to performance. Here, the prototypes use RSA keys that are expected to be secure for at least five years, which means evidence needs to be updated every five years. As we will show, the results are comparable to the predictions based on the analytical evaluation presented in Section 5.1. Next, we first design the experiment and then analyze the results.

Experiment design

Our experiment is designed as follows. There are 128 documents, each of size 30720 bytes, in the archive. Also, each document is signed using a 1478-bit RSA key the hash function SHA-256. The goal of the experiment is to protect these documents for 100 years starting in the year 2013. We therefore create initial evidence in the year 2013. Then we update the evidence every five years, i.e. in the years 2018, 2023, 2028, 2033, 2038, 2043, 2048, 2053, 2058, 2063, 2068, 2073, 2078, 2083, 2088, 2093, 2098, 2103, and 2108.

For generating evidence, we choose the RSA scheme and a hash function SHA. For signing we select the RSA key lengths according to the conservative predictions by Lenstra [31] such that they remain secure for at least five years, i.e. until the next evidence update happens. The selected hash functions also follow the same conservative predictions. Since Lenstra predict that the hash function SHA-256 remains secure until year 2090, we select SHA-256 from 2013 to 2083 and then SHA-384 from 2088 onwards. This also enables us to evaluate both the performance of evidence updates in which the hash function remains the same and of evidence updates in which the hash function is replaced. The selected key lengths and hash functions are found in Table 5.7.

We use the prototypes to measure time, space, and communication as follows. Time is measure by counting the running times for each prototype to generate or verify evidence. We exclude the times spent on reading or writing files in file system. For evidence generation, we also exclude the times needed to communicate with time-stamp authorities or notaries and the time they need to respond requests.

Space is measured by counting the sizes of the evidence produced by each prototype. Communication is measured by counting the sizes of the data exchanged between each prototype and the trusted third parties when evidence is generated.

As for running times, potential increase of computer speed has to be taken into account. This is typically done using Moore's law, which predicts that processor speed doubles every 18 months. However, such predictions should be taken with necessary care, since for physical reasons Moore's law is only expected to be valid until 2020 [51]. Therefore, we provide not only the running times measured in the experiment, but also the correction factors in accordance with Moore's law. These factors are $2^{(y_p - y_s)12/18}$, where y_s is the year when the implementation is supposed to be executed and y_p is the year when the used processor was launched. We used an Intel i5 M560 2.67 GHz processor launched in 2010. By multiplying a provided running time (measured while using our processor) and a correction factor computed for the year y_s, the reader can estimate how fast the implementation would perform

Table 5.7: The key and hash sizes used to generate evidence.

Year when evidence is generated	Key size (bits)	Hash size (bits)
2013	1478	
2018	1708	
2023	1958	
2028	2228	
2033	2521	
2038	2835	
2043	3172	
2048	3532	256
2053	3916	
2058	4325	
2063	4758	
2068	5217	
2073	5701	
2078	6213	
2083	6751	
2088	7317	
2093	7910	
2098	8533	384
2103	9184	
2108	9865	

if using a processor launched in year y_s.

Results

We start by comparing the schemes with respect to the running times required by the archivist to generate evidence. The running times are found in Table 5.8. We will see that our predictions in Section 5.1 are quite good. However, they only take hashing and signature verification into account. This explains certain deviations.

We compare Table 5.8 with our predictions in Tables 5.1 and 5.2. First, we consider the initialization step. Table 5.8 confirms the predictions found in Table 5.1. As our experiments show, CN is in fact the fastest scheme. Next are AdES and CIS, followed by ERS. As expected, ERS is the slowest scheme.

Next, we turn to evidence update. The only year in which the hash function is updated is 2088. In all other iterations the hash function from the previous iteration

Table 5.8: The times (seconds) needed to generate evidence for 128 signed documents.

Year	AdES	CIS	ERS	CN	Moore's Law factor
2013	0.18	0.18	0.24	0.07	$2^{-2.00}$
2018	0.45	0.46	0.24	0.00	$2^{-5.33}$
2023	0.72	0.74	0.26	0.00	$2^{-8.67}$
2028	0.96	1.02	0.29	0.00	$2^{-12.00}$
2033	1.26	1.35	0.30	0.00	$2^{-15.33}$
2038	1.58	1.67	0.32	0.00	$2^{-18.67}$
2043	1.91	2.02	0.35	0.00	$2^{-22.00}$
2048	2.25	2.39	0.38	0.00	$2^{-25.33}$
2053	2.63	2.76	0.40	0.00	$2^{-28.67}$
2058	3.03	3.22	0.41	0.00	$2^{-32.00}$
2063	3.47	3.62	0.43	0.00	$2^{-35.33}$
2068	3.91	4.15	0.45	0.00	$2^{-38.67}$
2073	4.45	4.64	0.48	0.00	$2^{-42.00}$
2078	4.92	5.15	0.50	0.00	$2^{-45.33}$
2083	5.50	5.73	0.52	0.00	$2^{-48.67}$
2088	6.21	6.50	6.33	0.00	$2^{-52.00}$
2093	6.85	7.13	0.68	0.00	$2^{-55.33}$
2098	7.61	7.89	0.70	0.00	$2^{-58.67}$
2103	8.30	8.61	0.73	0.00	$2^{-62.00}$
2108	9.08	9.47	0.75	0.00	$2^{-65.33}$

is used. Table 5.2 predicts that in iterations where the hash functions is not updated, CN should use no time, ERS should be the fastest time-stamping scheme, followed by AdES and CIS. This is again confirmed by Table 5.8. In fact, we see that ERS is significantly faster than the other two time-stamping schemes. In 2088 the hash function SHA-256 is replaced by SHA-384. In this case, AdES is faster than ERS and CIS as predicted. Furthermore, we see that ERS is slightly faster than CIS because ERS hashes fewer hashes. This difference becomes plausible when we plug in the parameters from our experiment for the year 2088 in Table 5.1: $k = 16$, $m = 128$, $p = 1$, authentication path of length $l = 7$. In this case, ERS hashes 1150 hashes while CIS hashes 3840 hashes.

Next, Table 5.9 shows the times required by retrievers to verify evidence. As predicted and shown in Tables 5.3 and 5.4, CN is the fastest scheme. The time-stamping schemes are almost identical in performance. This is mainly due to the fact that according to Table 5.3 they require the same number of signature verifications.

The number of hashes appears to play an insignificant role. This situation will be further investigated in Section 5.2.2.

Table 5.9: The times (seconds) needed to verify evidence for one document.

Year	AdES	CIS	ERS	CN	Moore's Law factor
2013	0.25	0.25	0.25	0.03	$2^{-2.00}$
2018	0.31	0.31	0.31	0.03	$2^{-5.33}$
2023	0.37	0.36	0.36	0.03	$2^{-8.67}$
2028	0.43	0.43	0.43	0.03	$2^{-12.00}$
2033	0.51	0.51	0.50	0.03	$2^{-15.33}$
2038	0.59	0.60	0.59	0.03	$2^{-18.67}$
2043	0.69	0.69	0.69	0.03	$2^{-22.00}$
2048	0.80	0.80	0.79	0.03	$2^{-25.33}$
2053	0.92	0.92	0.91	0.04	$2^{-28.67}$
2058	1.05	1.06	1.04	0.04	$2^{-32.00}$
2063	1.20	1.21	1.19	0.04	$2^{-35.33}$
2068	1.36	1.37	1.34	0.04	$2^{-38.67}$
2073	1.54	1.55	1.53	0.04	$2^{-42.00}$
2078	1.74	1.75	1.72	0.05	$2^{-45.33}$
2083	1.96	1.98	1.94	0.05	$2^{-48.67}$
2088	2.21	2.21	2.18	0.06	$2^{-52.00}$
2093	2.48	2.49	2.46	0.06	$2^{-55.33}$
2098	3.56	3.56	3.50	0.99	$2^{-58.67}$
2103	4.74	4.77	4.69	1.13	$2^{-62.00}$
2108	6.12	6.13	6.04	1.31	$2^{-65.33}$
2113	6.12	6.13	6.04	1.31	$2^{-68.67}$

We now compare the schemes with respect to space. Figure 5.1 presents the total size of the evidence for all 128 documents created by the individual schemes. As predicted and shown in Table 5.5, from 2018 onwards, ERS is the most space efficient scheme, followed by CN and the least space efficient schemes are AdES and CIS. In 2113, the differences are quite dramatic.

Finally, we compare how schemes differ in communication. Figure 5.2 illustrates the sizes of the messages exchanged between the implementations and the trusted third parties when evidence is created or updated. As predicted and shown in Table 5.6, the communication complexity for CN is the highest. The communication complexity of the time-stamping schemes are much lower with the communication complexity for ERS being almost zero.

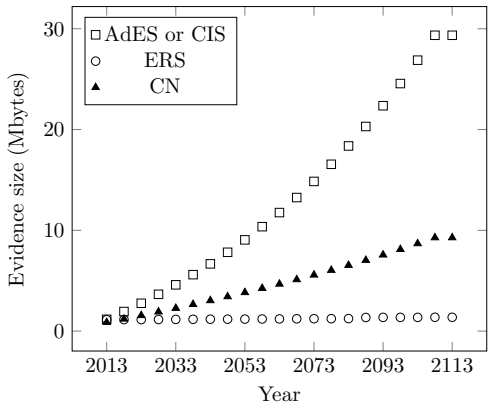

Figure 5.1: The sizes of evidence for 128 signed documents.

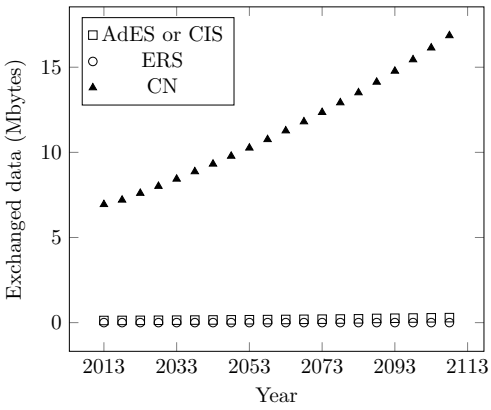

Figure 5.2: The sizes of data exchanged between the archivist and the trusted third parties while generating evidence for 128 documents.

Profiling evidence verification

The analytical evaluation in Section 5.1 indicates that ERS allows for the most efficient evidence verification, followed by the other time-stamping schemes AdES and CIS. However, in the experiments these schemes performed similarly, suggesting that hashing plays an insignificant role for the used parameters. To verify whether this is true, we used a profiler software that allows to identify which Java procedures are executed and how much time they consume while the prototypes check evidence.

By using the profiler we confirmed that hashing consumes almost no time when compared to other procedures. Among these procedures, the most time-consuming is the verification of certificates. This scenario is also true for the notarial scheme. For example, when checking evidence in year 2113, certificate verification consumes 73% of the running time of the time-stamping schemes and 83% of the running time

of notarial scheme. The remaining time is mainly spent on overhead.

Certificates verification is a significant *hot spot*, i.e. an operation that requires considerable running time while an implementation runs, in all schemes because of the used parameters. More precisely, in our experiment we use a chain of three certificates together with three CRLs as the verification information for each signature on time-stamp or attestation. Since every certificate or CRL is individually signed, certificate chains contain many more signatures to be verified than time-stamps or attestations contain. Since RSA verification is a very time-consuming operation, checking certificate chains consumes considerable time when evidence is verified.

Another used parameter that influences the results for this scenario is the size of the selected documents to be protected. Their small sizes (30720 bytes) leads to the irrelevant hashing times. For larger documents we can expect appreciable hashing times for AdES, CIS, and CN but not necessarily for ERS. This is because AdES, CIS, and CN need to hash the document when verifying every time-stamp or attestation, whereas ERS has to hash the document only when the hash function is replaced.

5.2.3 Comparing schemes for distinct signature lifetimes

In the analytical evaluation and the previous experiment we did not take distinct signature lifetimes into account. However, this is an important parameter since it has a high impact on the performance of the protection schemes. In this section we analyze the performance of the schemes with respect to this parameter. Note that we are aware of the fact that it is in practice recommended to generate new signature keys around, depending on the application, every two to four years. Nevertheless, we think that it is an interesting contribution to analyze performance with respect to distinct signature lifetimes. Thus, we first explain how they are expected to affect performance and why they need to be analyzed with the help of implementations. Then, we describe our experiment and present the results.

The predicted lifetimes of signatures determine how often evidence needs to be updated (see Section 3.1.2). This update frequency implicates in how much evidence will be created. More precisely, the longer the lifetimes, the less evidence is created. It follows that also less evidence needs to be verified. Moreover, the lifetimes of signatures are directly related to the sizes of the signature keys used to generate evidence. Longer lifetimes require bigger key sizes and signatures generated with bigger key sizes need more time to be verified individually.

Therefore, selecting bigger signature keys can be expected to improve the performance of time-stamping and notarial schemes with respect to communication, space,

and evidence generation time but not necessarily verification time. Communication performance is improved because when using bigger keys archivists request new evidence fewer times from time-stamp authorities or notaries. Space performance becomes also better because less generated evidence needs to be stored. Furthermore, generation time is reduced because if less evidence is generated, less data needs to be hashed in the next update. (Note that bigger keys does not affect the generation time for the notarial scheme CN because archivists hash no data upon requesting new evidence.)

As said, using bigger signature keys not necessarily improves performance with respect to verification time. This is because of a trade-off between the number of signatures to verify and the time needed to check each signature. On the one hand, the use of bigger signature keys reduces the time needed to verify evidence because there are fewer signatures to be checked. On the other hand, the time needed to check evidence is increased because signatures created with bigger keys are slower to be verified.

Note that it is hard to take this trade-off into account in the analytical evaluation, because the times needed to verify signatures together or individually depend on the used signature algorithms and their concrete implementations. Therefore, prototypes of the protection schemes are necessary to analyze this trade-off.

Next we design an experiment to analyze how distinct RSA signature key sizes affect the times that our implementations need to verify evidence. Since communication, space, and update time are known to be improved by using bigger keys, we do not measure these parameters in this experiment.

Experiment design

Part of this experiment is similar to the previous experiment found in Section 5.2.2. We use the same 128 documents of 30720 bytes. Each document has a 1478-bit RSA signature generated using hash function SHA-256. The protection goals of these documents are guaranteed from 2013 to 2113 by evidence generated using RSA keys. This evidence is updated every five years.

However, in the new experiment we generate two additional types of evidence for the documents from 2013 to 2113 using RSA keys. One type of evidence is updated every 15 years and another type every 25 years. Thus, to generate evidence that is updated every 5, 15, or 25 years we need to select the appropriate RSA key sizes such that they remain secure until the next evidence update happens. Such sizes were selected using the conservative predictions by Lenstra [31] and are found in Table 5.10. Since the 15- and 25-year lifetimes require fewer evidence updates, in

Table 5.10: The key sizes used to generate evidence. A dash shows that no new evidence is generated in that year.

Year	5-year key sizes	15-year key sizes	25-year key sizes
2013	1478	1958	2521
2018	1708	-	-
2023	1958	-	-
2028	2228	2835	-
2033	2521	-	-
2038	2835	-	4325
2043	3172	3916	-
2048	3532	-	-
2053	3916	-	-
2058	4325	5217	-
2063	4758	-	6751
2068	5217	-	-
2073	5701	6751	-
2078	6213	-	-
2083	6751	-	-
2088	7317	8533	9865
2093	7910	-	-
2098	8533	-	-
2103	9184	9865	-
2108	9865	-	-

some years no update takes place. Such years are marked with a dash in Table 5.7.

Although no hash functions are found in Table 5.10, these cryptographic primitives are still necessary to generate evidence. As in the previous experiment, we use the hash function SHA-256 from 2013 to 2083 and SHA-384 from 2088 for the three types of evidence.

Results

We now compare how the three predicted signature lifetimes (5, 15 and 25 years) affect the verification times. For this, we compute the difference between the times the prototypes spent on verifying the evidence generated with the different signature key sizes. In the following we will use the terms 5-year signature, 15-year signature and 25-year signature for signatures generated with keys that are predicted to hold

Table 5.11: The ratios between verification times when using 5-, 15-, and 25-year keys.

Year	AdES			CIS			ERS			CN		
	5/15	5/25	15/25	5/15	5/25	15/25	5/15	5/25	15/25	5/15	5/25	15/25
2013	0.94	0.82	0.88	0.93	0.83	0.90	0.92	0.81	0.88	1.01	1.02	1.01
2018	2.02	1.77	0.88	2.00	1.79	0.90	1.96	1.74	0.88	1.03	1.04	1.01
2023	3.27	2.88	0.88	3.08	2.76	0.90	2.99	2.64	0.88	1.08	1.09	1.01
2028	1.92	3.98	2.07	1.92	3.96	2.06	1.90	3.73	1.97	0.95	1.10	1.16
2033	2.60	5.39	2.07	2.58	5.33	2.06	2.53	4.97	1.97	0.99	1.15	1.16
2038	3.28	2.63	0.80	3.30	2.63	0.80	3.27	2.64	0.81	1.01	0.80	0.79
2043	2.39	3.28	1.37	2.38	3.26	1.37	2.36	3.31	1.41	0.94	0.84	0.90
2048	2.92	4.01	1.37	2.90	3.98	1.37	2.87	4.03	1.41	0.96	0.87	0.90
2053	3.51	4.83	1.37	3.48	4.78	1.37	3.45	4.85	1.41	1.04	0.94	0.90
2058	2.53	5.71	2.26	2.58	5.69	2.21	2.54	5.73	2.25	0.92	1.00	1.10
2063	2.96	3.34	1.13	3.05	3.34	1.10	2.99	3.11	1.04	0.95	0.77	0.81
2068	3.45	3.90	1.13	3.53	3.87	1.10	3.47	3.60	1.04	1.01	0.82	0.81
2073	2.73	4.49	1.64	2.79	4.46	1.60	2.69	4.18	1.55	0.88	0.87	0.99
2078	3.14	5.16	1.64	3.21	5.14	1.60	3.09	4.80	1.55	0.96	0.94	0.99
2083	3.59	5.90	1.64	3.68	5.89	1.60	3.53	5.48	1.55	1.00	0.98	0.99
2088	1.35	1.27	0.94	1.37	1.27	0.93	1.35	1.24	0.92	0.06	0.05	0.76
2093	1.54	1.44	0.94	1.55	1.44	0.93	1.53	1.41	0.92	0.06	0.05	0.76
2098	2.26	2.12	0.94	2.28	2.12	0.93	2.24	2.06	0.92	1.04	0.79	0.76
2103	1.64	2.87	1.75	1.65	2.88	1.74	1.63	2.80	1.72	0.90	0.90	1.01
2108	2.14	3.74	1.75	2.14	3.74	1.74	2.12	3.65	1.72	1.04	1.04	1.01
2113	2.14	3.74	1.75	2.14	3.74	1.74	2.12	3.65	1.72	1.04	1.04	1.01
Average	2.49	3.49	1.39	2.50	3.47	1.38	2.46	3.35	1.36	0.90	0.86	0.94

for 5, 15, and 25 years respectively. The difference is presented as the ratio computed by dividing the verification time for evidence E_1 by the verification time for evidence E_2, where E_1 and E_2 are verified in the same year but E_2 uses a bigger signature key than E_1. A ratio smaller than one indicates that a smaller key allows for a faster verification of evidence whereas a ratio greater than one indicates that a smaller key allows for a slower verification.

The computed ratios are found in Table 5.11. They are used to compare the results for the predicted lifetimes of 5- and 15-year signatures, 5- and 25-year signatures, and 15- and 25-year signatures. These comparisons are found in the columns labeled with "5/15", "5/25", and "15/25". Moreover, for each of these columns, the average ratio from 2013 to 2113 is presented.

Table 5.11 shows that the prototypes for the time-stamping schemes AdES, CIS, and ERS tend to perform worse with respect to evidence verification if evidence uses smaller signature keys. This tendency can be seen in the average ratios computed from 2013 to 2113, which show that verification is slower for smaller keys. For

example, the verification of 5-year AdES evidence needs 2.49 as many time as the verification of 15-year AdES evidence. This indicates that when using time-stamping schemes bigger keys are preferred because the time needed to verify fewer time-stamps tends to compensate the longer times spent on verifying each time-stamp.

In contrast, the implementation for the notarial scheme CN tends to perform slightly better with smaller signature keys. This tendency is indicated by the average ratios in Table 5.11. For example, the verification of 5-year CN evidence requires 0.86 as many time as checking 25-year CN evidence. Also, when comparing 5- with 15-year signatures and 15- with 25-year signatures, we see no significant differences. This shows that the smaller the keys are, the faster CN performs.

The result that CN performs faster with smaller signature keys is not surprising because only the most recent signature is verified. Therefore, whether the used signature key sizes reduce or increase the number of generated signatures is irrelevant whereas the size of the used keys is important. Since smaller key sizes allow for a faster verification of each signature, CN performs faster with smaller keys.

However, there are some years when the above tendencies are not observed in Table 5.11. We identify four situations where tendencies are different and explain why. The first situation happens in the initial years when the notarial scheme CN perform equally for 5-, 15-, and 25-year signatures. The reason is that the verification times are too short and there is a deviation when measuring them from the prototypes.

The second situation happens in some later years when CN performs again equally for distinct signature key sizes. For example, in 2023, 2038, 2053, 2068, 2083, 2098, and 2108. The reason now is that in these years two or more predicted signature lifetimes require the same key sizes (see Table 5.10). Thus, the times needed to verify CN evidence are comparable although the predicted signature lifetimes differ.

The third situation occurs in years 2013, 2018, 2023, and 2038 when the time-stamping prototypes perform faster for smaller signature keys (see ratios smaller than one). This is because in these years using 5-, 15-, or 25-year signatures produces the same numbers of time-stamps. Since the numbers of time-stamps to be verified are the same, only the size of used keys affects the verification time. Thus, the prototypes perform better with smaller key sizes since such sizes allow to verify each signature faster.

The final situation is observed in 2088, 2093, and 2098 when the time-stamping schemes perform again faster for smaller signature keys (see the ratios 0.94, 0.93, and 0.92). Also, in 2088 and 2093 the notarial scheme CN presents extremely low ratios (0.06 and 0.05). The reason seems to be that the verification of each signature using certain key sizes is very inefficient in our prototypes. More precisely, the key sizes that are equal or greater than 8533 bits, which are used from 2098 onwards

(see Table 5.10), are very inefficient. To illustrate this situation, Figure 5.3 shows the times needed to verify AdES and CN evidence when evidence is updated every five years. Note that the times needed to verify evidence grows faster from 2098 onwards. Verification for CIS and ERS are not illustrated since it presents quite similar run times.

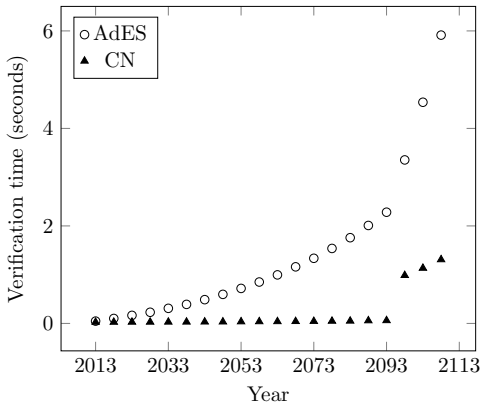

Figure 5.3: The time needed to verify evidence using AdES or CN and 5-year signatures. Note that the verification becomes very time-consuming from 2098 onwards.

Thus, the controversy tendencies are because in 2088, 2093, and 2098 the 15- and 25-year signatures require those particular key sizes but the 5-year signatures do not. Therefore, when using 15- and 25-year signatures, the time-stamping prototypes need extra time because of the inefficient key sizes, which compensates the advantage of verifying fewer signatures than when using 5-year signatures. For the CN prototype, the extremely low ratios indicates that 5-year signatures can be verified much faster than the 15- and 25-year signatures.

5.3 Final comparisons and possible improvements

In this section we first provide a final comparison of the protection schemes by summarizing the results from Sections 5.1 and 5.2. Based on this comparison, we then suggest how to improve the performance of these schemes.

The notarial scheme CN is more efficient than the time-stamping schemes with respect to space and time but at the cost of higher communication complexity. CN requires less space because it stores no verification information (e.g. certificates and revocation information) for the notarial attestations whereas the time-stamping schemes store verification information for each generated time-stamp. CN is also

faster because no data is hashed when updating evidence and only the most recent attestation is checked when verifying evidence. In contrast, the time-stamping schemes need to hash data upon updating evidence and check all generated time-stamps when verifying evidence. However, communication complexity is higher for CN because archivists need to send documents and the previous attestations to request new evidence while in the time-stamping schemes archivists only send hashes.

When comparing the time-stamping schemes, ERS is the most efficient with respect to communication and space. This is because in ERS an archivist requests and stores only one new time-stamp for a set of documents, whereas in AdES and CIS one new time-stamp is needed for each document in the set.

However, ERS is not necessarily the best scheme when it comes to evidence generation or verification. In regard to evidence generation, ERS is the most time-efficient scheme except in the few cases when the evidence is initialized or the hash function is replaced. In the initialization, ERS is the slowest scheme. When the hash function is replaced ERS is comparable to the other time-stamping schemes. As for evidence verification, all time-stampings schemes are comparable. Their differences in hashing become irrelevant since the verification time is mostly consumed by certificate verifications. For larger documents, however, hashing differences can be appreciable when using AdES, CIS, or CN.

The performance of time-stamping and notarial schemes also depends on the key sizes of the used signatures. Using bigger keys reduces space, communication, and update times since archivists need to update evidence fewer times. However, the same is not always true for evidence verification times. The time-stamping schemes tend to perform better when using bigger signature keys because fewer time-stamps and less verification information need to be checked. However, since the notarial scheme CN verifies only the most recent attestation, reducing the number of generated attestations by using bigger keys is of no use. Thus, using smaller keys is desirable because they allow for faster signature verifications.

We now suggest improvements for the protection schemes. As said, the notarial scheme is the worst scheme with respect to communication because an archivist has to send the document and previous signatures to notaries. To address the issue, he could send hashes computed from the document and from previous signatures instead of sending the document and signatures themselves. This proposal will be discussed in Chapter 6.

The time-stamping schemes could be improved with respect to time. AdES and CIS could be changed such that hashing is similar to ERS hashing. More precisely, when updating evidence, AdES and CIS could hash the document and the existing evidence only if it is necessary to replace the latest hash function. This change will

be incorporated in the new version of CIS proposed in Chapter 7.

Certificate chain verifications are the most time-consuming operation observed in all schemes. To mitigate this issue, we identify three methods. The first method is using larger signature keys for time-stamping as shown before but with necessary care. This is because it will impact on the performance of time-stamp authorities when they sign time-stamps. Also, because an attacker has a higher chance to get access to the signature keys during their longer lifetimes and to forge signatures.

The second method is to skip the verification of certain time-stamps if the validities of several time-stamps in a sequence overlap. This particular situation may happen when documents are submitted to an archive and time-stamp sequentially. This can also happen when archived documents are frequently changed or the format of a document is migrated. In these two cases, one time-stamp might be generated soon after an evidence updated occurred. Such a method will be used in the new version of CIS proposed in Chapter 7.

The third method refers to changing the used PKIs such that certificate chains become shorter and therefore contain fewer certificates and certificate revocation lists. This approach will be discussed in Chapter 8.

6 A new notarial scheme

In this chapter we propose a new notarial scheme named *Attested Certificates* (AC) and show that it improves the existing notarial scheme *Cumulative Notarizations* (CN) such that less communication and space are used. We reduce communication by leaving the verification of document signatures to retrievers. In this case the signed documents do not have to be sent to the notaries. The notaries only attest the validity of certificates before attesting them. We reduce space by generating evidence without the use of time-stamps. While in CN time-stamps are requested from time-stamp authorities and added to the evidence, in AC smaller attestations are generated. Furthermore, in AC only the latest attestation is kept instead of generating a sequence of several attestations as in CN. To show that AC outperforms CN, we provide an analytical evaluation and carry out experiments using prototypical implementations. We show that AC is more efficient than CN with respect to space and communication. Moreover, although AC leaves the verification of document signatures to retrievers, the experiments showed that the running times needed to verify AC or CN evidence are comparable. Furthermore, we also compare AC with the time-stamping schemes. Finally, we compare AC and CN with respect to trustworthiness. The comparison is based on the required trust assumptions and shows that AC can be expected to be equivalent to CN. The content of this chapter was published in [54, 55].

6.1 Improving Cumulative Notarizations

In Chapter 5 we compared the time-stamping schemes and the notarial scheme Cumulative Notarizations (CN) with respect to performance. This comparison showed that CN has the highest communication complexity. CN requires more data to be exchanged than the time-stamping schemes because CN notaries verify old evidence before generating new evidence. Therefore, in addition to the old evidence, notaries need to receive the document while for time-stamp authorities the corresponding

hashes are sufficient. Since the document and the CN evidence are usually larger than hashes, CN has the worst the communication complexity.

In this section we describe the new scheme AC, which has a lower communication and space complexities than CN. Communication is reduced by removing the need for sending the document and the *whole* evidence to notaries. More precisely, in CN an archivist sends the document d, the signature s of d, the certificate c needed to verify s, the time-stamp T on d and s, and the notary signatures (i.e. attestations) $s_1, \ldots, s_k$ to a notary. In our new scheme AC, d and s are no longer sent; instead, the archivist sends hashes computed from d and s. Also, AC needs neither the time-stamp T nor the old attestations $s_1, \ldots, s_{k-1}$. Therefore, T and $s_1, \ldots, s_{k-1}$ are no longer sent. This has also a positive effect on the space AC needs. Since T and $s_1, \ldots, s_{k-1}$ are no longer needed, the archivist does not have to store them.

We summarize the differences between AC and CN with respect to communication and space in Table 6.1. This table presents for each notarial scheme which objects need to be sent and stored.

Table 6.1: The objects that the notarial schemes CN and AC send or store.

Objects	Sent		Stored	
	CN	AC	CN	AC
Document	✓	✗	✓	✓
Document signature	✓	✗	✓	✓
Certificate	✓	✓	✓	✓
Time-stamp	✓	✗	✓	✗
Last attestation (s_k)	✓	✓	✓	✓
Old attestations ($s_1, \ldots, s_{k-1}$)	✓	✗	✓	✗
Hashes	✗	✓	✗	✓

Next, we present the protocol for the new notarial scheme AC in Section 6.1.1. Then, we justify the design of the presented protocol and explain why it works.

6.1.1 Attested Certificates

In this section we present the protocol for the new scheme AC. We start by presenting the set-up, i.e. which data is required before generating evidence. Next we present the subprotocols to initialize and to update the evidence for documents. This evidence consists of distinct objects, which are also presented. Finally, we show how a retriever verifies the generated evidence

Set-up

Initially, an archivist receives the following objects to be archived.

- A document d.
- The signature s on d.
- The certificate c required to verify s.

The archivist applies the notarial scheme AC to generate evidence that guarantees the protection goals authenticity, non-repudiation, and proof of existence for d.

Evidence generation

The evidence generation is divided into two procedures. The first procedure is to initialize evidence for the document d at time t and the second one to update the evidence. The first procedure has the following steps.

1. The archivist selects a hash function h_1, computes hash $y_1 = h_1(d||s)$, and sends h_1, y_1, and c to a notary.
2. The notary checks that c is valid at the current time t by performing the following steps.
   a) He obtains the certificate chain from c until a trust anchor.
   b) He collects revocation information for all certificates.
   c) He executes the verification algorithm for c as described in Section 2.7.
3. The notary checks that h_1 is secure at the current time t.
4. The notary creates a signature s_1 on $c||t||h_1||y_1$ and returns t and s_1 to the archivist.
5. The archivist stores t, s_1, h_1, and y_1 together with d, s, and c.

The signature s_1 is a notarial attestation used to inform retrievers that: a) c is a valid certificate to verify the document signature s at time t, and b) document d and signature s existed at time t (proof of existence).

The second procedure helps to update evidence and is necessary to address the aging of cryptography. Thus, the second procedure must be executed before the

lifetime of the latest used signature or hash function ends. We first explain how evidence is updated when the signature lifetime is about to end.

Assume for $1 \leq j \leq k$ that the archivist has the document d, the signature s on d, the certificate c required to verify s, the used hash functions $h_1, \ldots, h_j$, the computed hashes $y_1, \ldots, y_j$, the last signature s_k, and the time t when the first signature s_1 was generated. Then, before the lifetime of signature s_k ends at a time $t_{k+1} > t_k \geq t$, the following steps are executed.

1. The archivist sends $c, t, h_1, \ldots, h_j, y_1, \ldots, y_j$, and s_k to a notary.
2. The notary checks that the signature s_k is valid and that the hash function h_j is secure at time t_{k+1}.
3. The notary generates a new signature s_{k+1} on $c||t||h_1|| \ldots ||h_j||y_1|| \ldots ||y_j$.
4. The notary returns s_{k+1} to the archivist.
5. The archivist stores s_{k+1} together with $d, s, c, t, h_1, \ldots, h_j$, and $y_1, \ldots, y_j$.
6. The archivist deletes the previous signature s_k.

Now assume for $1 \leq j \leq k$ that at time t_{k+1} the lifetime of the last used hash function h_j is about to end. Then, the following steps are performed.

1. The archivist selects a new hash function h_{j+1} and computes $y_{j+1} = h_{j+1}(d||s)$.
2. The archivist sends c, t, $h_1, \ldots, h_{j+1}$, $y_1, \ldots, y_{j+1}$, and s_k to a notary.
3. The notary checks that the signature s_k is valid and that the hash functions h_j and h_{j+1} are secure at time t_{k+1}.
4. The notary generates a new signature s_{k+1} on $c||t||h_1|| \ldots ||h_{j+1}||y_1|| \ldots ||y_{j+1}$.
5. The notary returns s_{k+1} to the archivist.
6. The archivist stores s_{k+1}, h_{j+1}, and y_{j+1} together with d, s, c, t, $h_1, \ldots, h_j$, and $y_1, \ldots, y_j$.
7. The archivist deletes the previous signature s_k.

Note that for $k > 1$ the signature s_k generated at time t_k is used to attest that the previous signature s_{k-1} was valid at time t_k. Moreover, since s_k is also created on certificate c together with time t and all hashes computed from the document d together with signature s, s_k is used to re-attest the proof of existence of d and s. As we will discuss later, the re-attestation requires trusting all involved notaries.

Stored evidence

For $k \geq 1$, at the current time $t' \in [t_k, t_{k+1}]$ the archivist stores the following objects as evidence for the document d.

- The signature s of d.
- The certificate c required to verify s.
- The time t when the first signature s_1 was generated.
- The hash functions h_j and hashes $y_j = h_j(d||s)$, for each $1 \leq j \leq k$.
- The last signature s_k.

Evidence verification

A retriever verifies the AC evidence for a document d as follows. First, assume for $1 \leq j \leq k$ that the retriever has obtained document d, the signature s of d, the time t when evidence for d was initialized, hash functions $h_1, \ldots, h_j$, and the last signature s_k from the archive. Next, the retriever executes the following steps.

1. For each $1 \leq j \leq k$, she computes the hash $y_j = h_j(d||s)$.
2. She checks that s_k is a valid signature on $c||t||h_1|| \ldots ||h_j||y_1|| \ldots ||y_j$.
3. She verifies that s is a valid signature on document d. For this verification she uses the public key contained in certificate c.

6.1.2 Retrievers check document signatures by themselves

Our experiments in Chapter 5 showed that the notarial scheme CN has the worst communication complexity because a notary needs to receive a document in order to verify and attest the signature on this document. Thus, retrievers need not verify the document signature again and evidence verification becomes faster. However, we have also seen that for all protection schemes retrievers spend most of the time on checking certificates rather than document signatures. Thus, the experiments indicate that the higher communication complexity for CN is not really compensated by the performance improvements for retrievers.

Therefore, for the new notarial scheme AC, we propose that notaries receive no documents and attest only that certificate chains are indeed valid. The verification of document signatures is left to retrievers.

6.1.3 Proof of existence is provided only by notaries

In CN, proof of existence is provided by a time-stamp authority. The time-stamp authority provides proof of existence for document d and the signature s on d by issuing a single time-stamp T on $d||s$.

The use of time-stamps requires more data to be communicated between archivists and time-stamp authorities or notaries when evidence is initialized or updated. Furthermore, since archivists need to store the time-stamps requested for each signed document, the use of time-stamps also contributes to the size of stored evidence.

To reduce communication and space complexities, we propose for the new notarial scheme AC that proof of existence is provided by notaries instead of time-stamp authorities. As shown in Section 6.1.1, an archivist no longer requests the time-stamp T on the document d together with the document signature s. Instead, he sends a hash $y_1 = h_1(d||s)$ to a notary. The notary creates an initial attestation s_1 by signing the received hash y_1 together with the current date t. The initial attestation s_1 and date t work as a time-stamp. Also, notaries can re-attest the proof of existence of d and s by re-signing the hash y_1 and date t.

Therefore, communication is improved because no additional communication with time-stamp authorities is needed. Space is also improved because archivists need to store no time-stamps.

6.1.4 Addressing the aging of cryptography

As seen in Section 6.1.1, we propose a protocol to update AC evidence and address the aging of cryptography. In this protocol, a notary needs to re-sign all the hashes that an archivist has computed from the archived document together with the document signature. As we will show in this section, re-signing all these hashes is necessary to prevent an attack where a malicious archivist can find a collision for the archived document.

We start by describing how the malicious archivist can succeed in this attack if notaries do not sign all hashes computed from the archived document and document signature. Although in the presented protocol an object is signed by creating a signature on the object, in practice the object is first hashed and then a signature is generated on the computed hash. Since this practical procedure is necessary for the attack to succeed, assume the owner of a document d creates a signature s on the hash $h_0(d)$. Then, the owner sends d, s, h_0, and the certificate c required to verify s to the archivist.

Next, the archivist initializes AC evidence for d as proposed in Section 6.1.1.

First, he selects a hash function h_1. To simplify our description, assume h_1 has a longer lifetime that hash function h_0 used to sign document d. Next, he sends h_1, $y_1 = h_1(d||s)$, and c to a notary. The notary returns the initial signature s_1 on $c||t||h_1||y_1$ at time t. Since the attack is independent of the hash function a notary uses to sign, we do not identify it. The archivist stores d, s, c, t, h_0, h_1, y_1, and s_1.

Now assume that, contrary to the proposed protocol, only the most recent hash computed from the document d and signature s is signed when evidence is updated. More precisely, for $1 < j \leq k$, at time t_k instead of creating a new signature s_k on $c||t||h_1||\ldots||h_j||y_1||\ldots||y_j$ (proposed protocol), s_k is created on $c||t||h_j||y_j$, where $y_j = h_j(d||s)$ is the most recent hash the archivist computed.

Then, the archivist performs the following steps to carry out the attack.

1. At time $t_2 > t = t_1$, when h_0 is no longer secure and the lifetime of h_1 is about to end, he first finds a document $d' \neq d$ such that $h_0(d') = h_0(d)$. Then, he selects a new hash function h_2, computes a new hash $y_2 = h_2(d'||s)$, and sends h_1, h_2, y_1, y_2, c, t, and s_1 to a notary.

2. The notary first verifies that s_1 is valid and that h_1 and h_2 are still secure. Then he returns a new signature s_2 on $c||t||h_2||y_2$ to the archivist.

3. The archivist stores h_2, y_2, and s_2 together with d'. He deletes document d, the previous signature s_1, the hash function h_1, and hash y_1.

4. The archivist can use s, c, t, h_0, h_2, and s_2 to convince a retriever that d' was signed by the same signer of d at time t.

Note that the described attack is possible because the notaries have no access to document d. Therefore, they cannot guarantee that the submitted hashes y_1 and y_2 were computed from the same document.

However, a retriever could notice this attack as follows. First, she needs to obtain d, s, c, t, h_0, h_1, and s_1 from the archive before time t_2, when the archivist deletes d, h_1, and s_1. Then, she later obtains h_j and s_k ($1 < j \leq k$). She will notice the attack by checking that s_1 is valid signature on $c||t||h_1||h_1(d||s)$ but that s_k is an invalid signature on $c||t||h_j||h_j(d||s)$.

Note that the malicious archivist can still succeed in his attack even if the notaries sign more hashes but not all of them. In this case, however, the chance that the attack is detected are higher. For example, assume that notaries sign the two most recent hashes computed from the document d and signature s. Thus, in step 1 the archivist sends c, t, h_1, h_2, $y_1 = h_1(d||s)$, $y_2 = h_2(d'||s)$, and s_1 to the notary, who returns the signature s_2 on $c||t||h_1||h_2||y_1||y_2$ at time t_2. In this case, a retriever

can notice the attack easier than before. First, she obtains d, s, c, t, h_1, h_2, and s_2 from the archive at the current time t', where $t_2 \leq t' < t_3$. Then she will notice the attack by verifying that s_2 is an invalid signature on $c||t||h_1||h_2||h_1(d||s)||h_2(d||s)$.

However, in our example the attack may be no longer detected if evidence is updated again. Assume that at time t_3 the archivist sends $y_2 = h_2(d'||s)$ and $y_3 = h_3(d'||s)$ to a notary and obtains a signature s_3 on $c||t||h_2||h_3||y_2||y_3$. In this case, a retriever that obtains the collision document d' and the corresponding evidence after t_3 cannot notice the attack only by verifying s_3. This is because s_3 is a valid signature on $c||t||h_2||h_3||h_2(d'||s)||h_3(d'||s)$. As before, the retriever will only notice the attack if she has access to d before the archivist deletes it at time t_2.

To prevent the above attack, we proposed the protocols to update and verify AC evidence (see Section 6.1.1). For $1 \leq j \leq k$, in the update protocol the archivist sends all hashes $y_j = h_j(d||s)$ computed until time t_k when evidence is updated. The update protocol requires the notary to check that at least the two most recent hash functions h_{j-1} and h_j are secure at t_k. Also, the notary is required to compute s_k on all hashes $y_1, \ldots, y_j$. In the verification protocol, a retriever obtains at the current time $t' > t_k$ the document d, signature s on d, certificate c, hash functions h_j, and the most recent attestation s_k. She checks that the notary has signed all hashes $y_j = h_j(d||s)$ by verifying that s_k is a valid signature on $c||t||h_1||\ldots||h_j||h_1(d||s)||\ldots||h_j(d||s)$.

Finally, we show that the proposed protocol indeed prevents the described attack without relying on retrievers. For the malicious archivist to succeed at time t_k ($k \geq 1$), he needs to find a document $d' \neq d$ such that $y_j = h_j(d'||s) = h_j(d||s)$ for each $1 \leq j \leq k$ and request a new notarial signature by sending every h_j and y_j to a notary. Now, assume notaries are trusted to check the security of hash functions correctly and to only generate a new attestation s_k if the submitted hash functions h_j and h_{j-1} are secure at the current time t_k. For $k = 1$, the archivist can only request a signature from a notary at time t_1 if h_1 is secure; otherwise the notary rejects the request. Therefore, h_1 must be secure and the archivist cannot find a collision $h_1(d'||s) = h_1(d||s)$. For $1 < j \leq k$, at time t_k the archivist selects a new hash function h_j. He can only request a new signature from a notary if h_j and h_{j-1} are secure. Thus, he cannot find $d' \neq d$ such that $h_j(d'||s) = h_j(d||s)$ and $h_{j-1}(d'||s) = h_{j-1}(d||s)$. Therefore, the attack cannot succeed.

6.2 Performance evaluation

This section provides a performance evaluation for the new notarial scheme AC. We compare it with the existing notarial scheme CN to show that AC achieves a better performance as promised in Section 6.2.1.

As in Chapter 5, the evaluation consists of two steps. First, we analytically evaluate how AC differs in performance from CN and the time-stamping schemes. Second, we carry out an experiment where implementations for the protection schemes and available cryptographic primitives are used.

6.2.1 Analytical evaluation

In this section we perform a similar analytical evaluation as in Section 5.1 to predict the performance of the new notarial scheme AC. Based on these predictions, we compare AC with the existing notarial scheme CN and show that AC provides a better performance. Furthermore, we also compare AC with the time-stamping schemes AdES, CIS, and ERS.

Next, we first recapitulate the conventions and parameters used in the analytical evaluation. Then, we provide the expressions that predict the performance for AC and CN. Note that for CN and the time-stamping scheme the expressions are the same found in Section 5.1.

To make the schemes comparable, we make the following conventions.

- There are $m = 2^l$ documents of the same size in the archive, where l is a positive integer. The archive also contains a signature on each of these documents.
- Time-stamp authorities sign their time-stamps.
- Hash functions are only renewed after r iterations because their lifetimes are expected to longer than the lifetimes of signatures. As explained in Section 5.1, such differences are because of security reasons. Therefore, in the kth iteration we define the number of hash function renewals as $p = \lfloor k/r \rfloor$.
- Verification information for a signature contains a certificate chain that allows to reduce the trust in the public verification key to the trust in a trust anchor. It also contains the revocation information for the certificates in this chain. This verification information is assumed to be of constant size.

The performance parameters used to compare the schemes are again time complexity, space complexity, and communication complexity. For time complexity we

distinguish between three times: 1) the time an archivist needs to initialize evidence, 2) the time the archivist spends on updating evidence, and 3) the time necessary for a retriever to verify evidence. Space refers to the size of evidence that is generated to establish the protection goals authenticity, non-repudiation, and proof of existence. Communication is the amount of data exchanged between the archivist and notaries or time-stamp authorities when initializing or updating evidence.

As before, we need to distinguish between two types of objects in our analysis. The first type is hashes, i.e. the bit string a hash function outputs. The second type is objects whose sizes are typically larger than the sizes of hashes; namely, documents, signatures, verification information, time-stamps, and attestations. This distinction is necessary when predicting the time complexity since the time spent on hashing normally depends on the size of the hashed objects. Moreover, space and communication complexities also depend on the size of the used objects.

Performance of evidence initialization

To evaluate AC with respect to initialization time, we again assume that hashing is the most time-consuming operation. Thus, we count the objects that the archivist needs to hash when initializing evidence for m documents. The numbers of hashed objects are found in Table 6.2.

Table 6.2: The numbers of objects being hashed while generating initial the evidence for m documents.

Scheme	Documents	Signatures	Ver. info	Hashes
CN	m	m	0	0
AC	m	m	0	0

As Table 6.2 shows, CN and AC need to hash the same number of objects when initializing evidence for m documents. Therefore, both schemes are expected to perform similarly. Since CN outperforms the time-stamping schemes (see Section 5.2.2), they are also expected to be outperformed by AC.

Performance of evidence renewal

We now compare the times for evidence update in the kth iteration, $k > 1$. As before, we count the numbers of objects that the archivist hashes when updating evidence for m documents. These numbers are found in Table 6.3.

The notarial scheme AC provides two methods to update evidence, which must be taken into account when comparing AC and CN. The first method helps to update

Table 6.3: The numbers of objects being hashed while updating the evidence for $m = 2^l$ documents in the kth iteration.

Scheme	Documents	Signatures	Time-stamps	Ver. info	Hashes
CN	0	0	0	0	0
AC-O	0	0	0	0	0
AC-N	m	m	0	0	0

evidence when the lifetime of the last signature is close to end. We refer to this method as AC-O in Table 6.3. The second method (AC-N) is needed to update evidence before the lifetime of the last hash function ends.

In AC-O the archivist hashes no objects. Because the same is true for CN, AC-O and CN should perform similarly when updating evidence. However, AC-N and CN are expected to compare differently. In AC-N the archivist hashes m documents and m signatures, whereas in CN nothing is hashed. Therefore, CN should be faster than AC-N.

Although AC-N is slower than CN, the notarial scheme AC still outperforms all time-stamping schemes discussed in Chapter 3. This can be seen if we compare AC-O and AC-N with the time-stamping schemes in Table 5.1. Note that when comparing AC and ERS, we should compare AC-O with ERS-O and AC-N with ERS-N since AC-O and ERS-O are performed in the same iterations. The same is true for AC-N and ERS-N.

Performance of evidence verification

Here we compare the times necessary to verify evidence for individual documents after the kth iteration. As in the analytical evaluation, we count the number of signatures to be verified (Table 6.4) and the number of objects to be hashed (Table 6.5). However, now we must distinguish document signatures from notary signatures (i.e. attestations). The reason is that when a retriever checks a document signature in AC, she does not check the signer's certificate chain because it has already been checked by a notary. Nonetheless, when verifying a notary signature in AC or CN, the retriever needs to check the notary's certificate chain.

The scheme AC is expected to be slower than CN with respect to evidence verification as indicated by Tables 6.4 and 6.5. The first table shows that AC is slower because AC evidence contains one more signature to be verified than CN evidence. The second table shows that AC is also slower since it requires the retriever to hash $p + 1 \approx k/r$ objects whereas in CN nothing is hashed.

Table 6.4: The numbers of signature verifications required while verifying the kth evidence for one document.

Scheme	Document signatures	Notary signatures
CN	0	1
AC	1	1

Table 6.5: The numbers of objects being hashed while verifying the kth evidence for one document. Here $p = \lfloor k/r \rfloor$ is the number of hash function renewals and r is the number of iterations after which the hash function is renewed.

Scheme	Documents	Signatures
CN	0	0
AC	$p+1$	$p+1$

However, this difference in verification times between CN and AC is expected to become insignificant in the long term. The reason is that the verification of the notary signatures tends to become the most time-consuming operation when verifying CN or AC evidence since bigger keys are needed to generate new notary signatures. The same is not true for the document signature, which is never renewed. Therefore, the time that AC needs to verify the document signature tends to become insignificant if compared with the time AC or CN requires when checking the notary signatures.

Compared with time-stamping schemes (Table 5.2), the new notarial scheme AC should be faster when verifying evidence. This is because in AC the retriever verifies fewer signatures and certificate chains. Also, she needs to hash fewer objects.

Space complexity

We compare the notarial schemes with respect to the space needed to store evidence for m documents. For this, we count again the objects stored for each schemes (see Table 6.6). We distinguish between two types of objects: large objects (viz. documents, signatures, time-stamps, certificates, and verification information) and hashes (i.e. short bit strings generated by hash functions). In the table, we set the number m of documents to $m = 2^l$ for some positive integer l. Also, the number of hash function renewals is determined by $p = \lfloor k/r \rfloor$, where r is the number of iterations after which the hash function is renewed.

Table 6.6 shows that for AC evidence the number of large objects is proportional

Table 6.6: The numbers of objects stored as the kth evidence for $m = 2^l$ documents. Here the number of hash function renewals is defined by $p = \lfloor k/r \rfloor$, where r is the number of iterations after which the hash function is renewed.

Scheme	Signatures	Time-stamps	Certificates	Ver. info	Hashes
CN	$(k+1)m$	m	m	0	0
AC	$2m$	0	m	0	$(p+1)m$

to m but independent of the iteration number k. Also, the number of hashes is proportional to mp which is approximately mk/r. Because signatures usually have a shorter lifetime than hash functions, mk/r can be expected to be small. For CN evidence, its size grows proportionally to mk, suggesting that AC needs less space than CN for increasing k.

When comparing AC with the time-stamping schemes in Table 5.5, we also see that AC is more space-efficient. The difference is because of the sizes of the stored objects. Since large objects such as time-stamps and verification information must be stored in the time-stamping schemes but not in AC, it has a lower space complexity.

Communication complexity

We compare CN and AC with respect to their communication complexities. As in Section 5.1, we count the number of objects that are exchanged between an archivist and notaries or time-stamp authorities. These objects are documents, signatures, time-stamps, notarial attestations (i.e. signatures), certificates, and hashes. They are counted in Table 6.7. Again, there are m documents in the archive, and the number of hash function renewals is $p = \lfloor k/r \rfloor$, where r is the number of iterations after which the hash function is renewed. Also, for each scheme we distinguish between the situations where evidence is initialized ($k = 1$) and where evidence is updated ($k > 1$).

Table 6.7 shows that AC requires less communication than CN when initializing or updating evidence. To initialize CN evidence ($k = 1$), the archivist first needs to request m time-stamps by sending m hashes to the time-stamp authorities which return m time-stamps. Then, he requests m attestations by sending m documents, m document signatures, m time-stamps, and m certificates to notaries which return m signatures. By contrast, to initialize AC evidence for m documents, he requests no time-stamps and sends neither documents nor signatures to notaries. Therefore, AC needs less communication to initialize evidence.

Table 6.7: The numbers of objects being exchanged between the archivist and notaries or time-stamp authorities while creating the evidence for m documents in the kth iteration. Here, the number of hash function renewals is determined by $p = \lfloor k/r \rfloor$, where r is the number of iterations after which the hash function is renewed.

Scheme	Documents	Signatures	Time-stamps	Certificates	Hashes
CN ($k = 1$)	m	$2m$	$2m$	m	m
AC ($k = 1$)	0	m	0	m	m
CN ($k > 1$)	m	$(k + 1)m$	m	m	0
AC ($k > 1$)	0	$2m$	0	m	$(p + 1)m$

When updating evidence ($k > 1$), AC is again more efficient because neither documents nor time-stamps are sent or received. Also, the number of exchanged signatures in AC is independent of the iteration k, whereas the same is not true for CN. Thus, for increasing k AC should communicate less data than CN. Note that mk/r hashes are exchanged in AC but not in CN. However, these hashes are not expected to compensate the advantage of AC over CN. This is because mk/r is expected to be small since the lifetimes of signatures are usually shorter than the lifetimes of hash functions. Furthermore, the size of these hashes is small if compared with the other exchanged objects.

Finally, note that the notarial schemes AC and CN still have worse communication complexities than the time-stamping schemes. The reason is that large objects (e.g. signatures and certificates) are exchanged in the notarial schemes whereas only hashes are exchanged in the time-stamping schemes.

6.2.2 Experiment

We have compared the performance of the new notarial scheme AC with the performance of the existing schemes CN, AdES, CIS, and ERS analytically. We now provide a more realistic comparison which takes available cryptographic primitives into account. For this end, we first design a prototype for the new notarial scheme AC such that it can be compared with the prototypes described in Chapter 5. Next, we use the AC prototype to simulate the protection of documents for which new evidence is generated every five years for 100 years. Using the results, we finally show that AC performs as predicted.

Implementation design

The prototype for AC was implemented using the same technologies as for the other schemes. That is, we used Java 7, the library xades4j to generate XML signatures, and the Bouncy Castle cryptographic provider for cryptographic operations. As before, the AC prototype uses certificate chains containing three X.509 certificates and for each certificate there is an empty certificate revocation list. We executed the prototype also on Solaris 11 using an Intel i5 M560 2.67 GHz processor and 4 GB RAM.

Experiment design

We repeat the experiment described in Section 5.2.2, where we use the prototypes to generate and verify evidence for 128 signed documents from 2013 to 2113. The documents have the same size as before (30720 bytes) and are signed using 1478-RSA keys and the hash function SHA-256. Evidence is first initialized in 2013 and updated every five years until 2113 using the appropriate key sizes and hash functions. These keys and hash functions were selected following Lenstra's conservative predictions and are found in Table 5.7.

Results

In this section we discuss the results of our experiment. The results include the running times needed to generate or verify evidence, the sizes of evidence generated, and sizes of objects exchanged with trusted parties. We use the results to confirm the performance predictions in Section 6.2.1 and to compare AC with the schemes AdES, CIS, ERS, and CN in practice. The results for these scheme are the same found in Chapter 5.

We first compare AC and CN with respect to the running times needed to generate evidence. The running times are presented in Figure 6.1. Also, because AC and CN are expected to be faster than all time-stamping schemes, the figure also compares AC with the most efficient time-stamping schemes, namely ERS. The presented times have not been corrected with Moore's law, so that the reader can compare them visually.

We compare the figure with the predictions for evidence initialization (Table 5.1) and evidence update (Table 6.3) for AC and CN. For evidence initialization, the analytical evaluation indicates that the running times of AC and CN should be comparable. This is confirmed by the similar running times measured for AC and CN which are observed in year 2013.

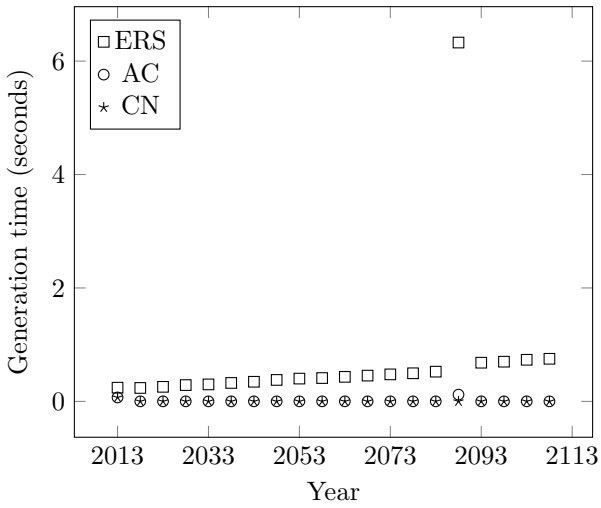

Figure 6.1: The times needed to generate evidence for 128 signed documents. The peak in ERS times occurs because all documents and their evidences are hashed to build a new Merkle tree.

For evidence update, the predictions suggest that AC and CN are also comparable except when AC replaces the latest used hash function (see AC-N in Table 6.3). Our experiments confirm that AC and CN are comparable from 2018 until 2108. However, in year when the hash function is replaced (2088) the schemes are also comparable since the advantage of CN over AC is too small.

When comparing AC and the most efficient time-stamping scheme (ERS) with respect to evidence generation, the running times measured also confirm the prediction that AC is faster. It follows that AC also outperforms the time-stamps schemes AdES and CIS.

Note that the peak in ERS times occurs because a new Merkle tree is built. This requires hashing the 128 documents, their corresponding signatures and time-stamps, and the verification information for each signature or time-stamp. Additional hash evaluations are also needed to compute the leaves and the root of the tree. This process is executed whenever a new hash function needs to be used.

Next, Figure 6.2 shows the running times that AC and CN need to verify the evidence for one document. Since the notarial schemes are expected to outperform all time-stamping schemes, this figure also compares AC with the most efficient time-stamping scheme (ERS). The illustrated times have not been corrected with Moore's law.

The predictions in Table 6.4 and 6.5 indicate that AC is slower than CN. However, Figure 6.2 shows that the time differences between the prototypes are very small.

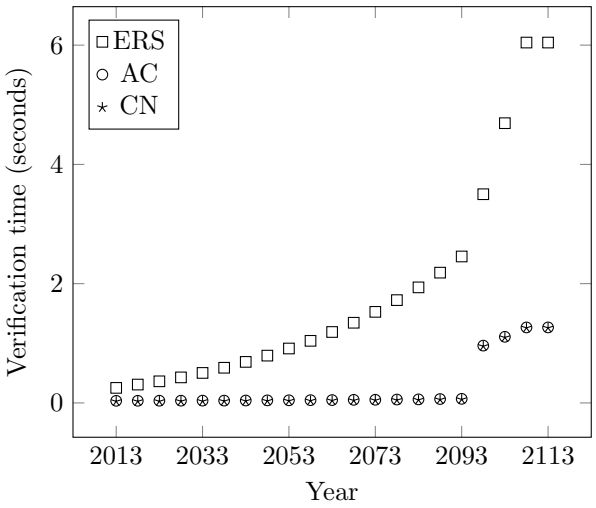

Figure 6.2: The times needed to verify evidence for one signed document.

It follows that AC and CN are in practice comparable even though AC requires retrievers to execute more operations when verifying evidence. Recall that AC leaves the verification of the document signature to the retrievers, who also need to compute hashes from the document and its signature.

Figure 6.2 also confirms the predictions that AC is faster than the time-stamping scheme ERS. It follows that AC also outperforms the other time-stamping schemes, which are all slower than ERS. The confirmed predictions can be explained by comparing the numbers of signatures to be verified (Tables 5.3 and 6.4) and by comparing the objects to be hashed (Tables 5.4 and 6.4). Although not explicitly shown in these tables, AC is also predicted to be faster because retrievers need to check fewer certificate chains. Recall that the verification of chains are left to AC notaries and that certificate verification has been shown to need significant running time (see Section 5.2.2).

Next, Figure 6.3 presents the total size of the evidence for all 128 documents created by the individual schemes AC, CN, and ERS. Again, ERS is selected because it is the most space-efficient time-stamping scheme.

As shown in Figure 6.3, AC needs by far less space than CN. Thus, we confirm the predictions found in Table 6.6. Furthermore, we can see in the figure that AC also needs less space than ERS, as predicted in Section 6.2.1. Since ERS is the most space-efficient time-stamping scheme, it is also true that AC is more space-efficient than the other time-stamping schemes described in Chapter 3.

We finally compare the communication complexities for the individual schemes, that is, the sizes of the messages exchanged between the archivist and the trusted

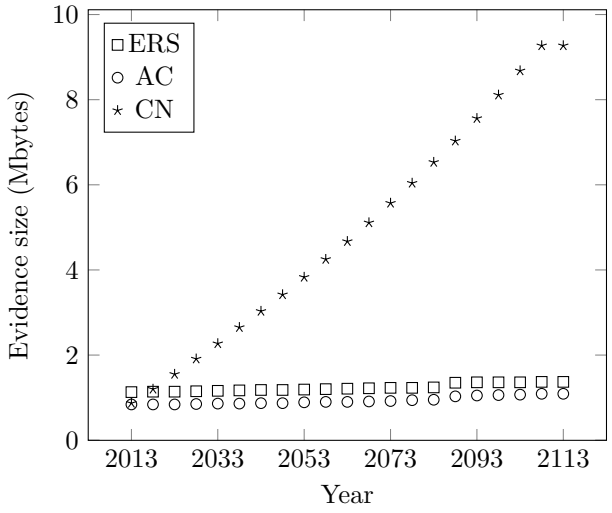

Figure 6.3: The sizes of evidence for 128 signed documents.

third parties when evidence is generated. Figure 6.4 illustrates the communication complexities for AC and CN. Since the notarial schemes are predicted to be less efficient than any time-stamping scheme, Figure 6.4 also illustrates the communication complexity for the least efficient time-stamping schemes AdES and CIS.

As Figure 6.4 shows, AC requires by far less data to be communicated than CN. Thus, we confirm the prediction in Table 6.7. However, the figure also confirms that AC is still inferior to all time-stamping schemes.

6.3 Trustworthiness analysis

As shown in Section 6.1, we design the new notarial scheme AC by improving the existing notarial scheme CN. Since the changes affect the roles of the trusted parties (e.g. notaries), in this section we compare AC and CN with respect to trustworthiness. For this, we identify the necessary trust assumptions and then compare AC and CN with respect to the identified assumptions.

AC and CN require the assumptions *crypto trust* and *CA trust* (see Chapter 4) because both schemes use digital signatures. As explained before, crypto trust refers to the requirement that neither cryptographic primitives nor secret keys used to generate evidence are compromised before evidence is updated. CA trust means that certification authorities are trusted to correctly identify the entity that owns the corresponding secret key, to correctly issue the certificate to this entity, and to operate a reliable revocation system.

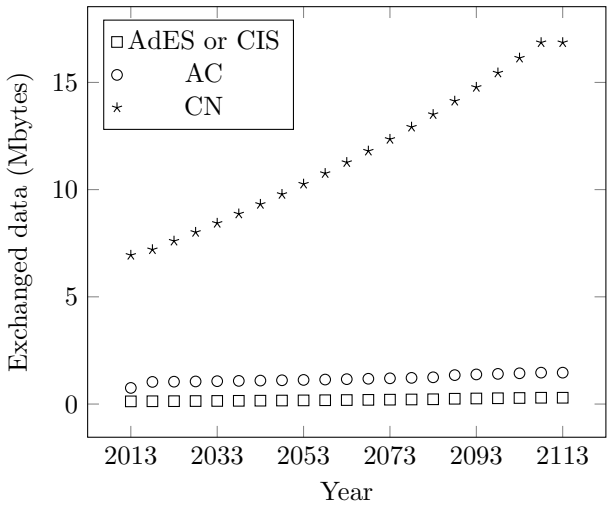

Figure 6.4: The sizes of data exchanged between the archivist and notaries or time-stamping authorities while generating evidence for 128 documents.

CN and AC differ in *TSA trust* and *notary trust*. In CN, a time-stamp authority is needed that provides a time-stamp as proof of existence. Therefore, CN assumes TSA trust which means that the time-stamp authority is trusted to generate a time-stamp containing the time when the time-stamp was generated. In contrast, in AC no time-stamps are needed and notaries are trusted to provide proof of existence. Thus, AC requires only notary trust.

However, the assumption notary trust defined earlier is too broad to allow us to compare CN and AC. The reason is that AC and CN trust notaries to carry out distinct procedures. Therefore, we next identify the individual procedures and compare AC and CN as to trustworthiness. We summarize our findings in Table 6.8.

Table 6.8: The procedures that notaries are trusted to carry out. The procedures that require no trusted entity are marked with a dash.

Procedure	Trusted CN entity	Trusted AC entity
Verify notary signatures	Notary	Notary
Verify certificates	Notary	Notary
Verify crypto. security	Notary	Notary
Verify document signatures	Notary	-
Maintain list of hashes	-	Notary
Provide proof of existence	Time-stamp authority	Notary

The first three procedures are verifying notary signatures (i.e. attestations), checking certificates, and verifying the security of cryptographic algorithms. They are common for AC notaries and CN notaries.

The following three procedures are different for AC notaries and CN notaries. The first of these procedures is the verification of document signatures. As described before, CN notaries are trusted to verify document signatures whereas in AC this verification is left to retrievers. Thus, relying on notaries to execute this procedure seems to be a disadvantage of CN.

The second procedure is maintaining the list of hashes properly. More precisely, AC notaries are trusted to not remove hashes from the list. This list is necessary to ensure the integrity of the archived document. Depending on notaries to keep this list seems to be a disadvantage of AC.

The third procedure is providing proof of existence. AC notaries are trusted to provide proof of existence correctly. That is, they must provide the current time in the notarial attestation when initializing evidence for one document (see Section 6.1.1). In CN, this procedure is delegated to time-stamp authorities.

When trusting a time-stamp authority or notary, there are disadvantages which are related to the number of involved parties. By trusting a time-stamp authority, more parties are involved in evidence generation and trust in the generated evidence tends to be lower (see Section 4.2.5). On the other hand, when trusting a notary, attackers need to compromise fewer entities to succeed in tampering with the protection goals of a protected document.

Furthermore, regardless of the entities trusted to provide proof of existence, it can be compromised by a malicious notary. In CN, time-stamps authorities are trusted to provide correct time-stamps but a malicious notary can still use an invalid time-stamp when generating evidence. In AC, notaries are trusted to guarantee proof of existence and can also attest a wrong time when generating evidence. However, in both cases the archivist can detected such malicious notaries.

As seen, AC and CN notaries are trusted to carry out three distinct procedures. Each procedure has disadvantages but in the end they seem to be balanced between AC and CN. Thus, the assumption notary trust is expected to be similar in both schemes. It follows that AC and CN are expected to be equally trustworthy.

7 A new time-stamping scheme

This chapter presents a new time-stamping scheme called *Content Integrity Service with Skip Lists* (CISS). The goal of this scheme is to provide an efficient solution for the following scenario. Assume a protection scheme is used to protect many documents, which are not submitted and time-stamped together as a set (as required for ERS), but sequentially. In this case, it would be still more efficient if only one proof of existence (i.e. in the form of a time-stamp sequence) has to be generated to protect these documents, since each proof must be protected from becoming insecure over time. CIS allows for generating only one time-stamp sequence for a set of documents, but has three shortcomings addressed by CISS. First, although all documents in the set share the same time-stamp sequence, to generate or verify the time-stamp sequence for one document only this document is needed. In comparison, using CIS all documents in the set need to be hashed. Second, a new time-stamp is appended to the time-stamp sequence each time a new document is added to the set protected by this sequence. It follows that the validities of several time-stamps may overlap and consequently not all time-stamps need to be checked during verification. CISS allows to skip during verification the time-stamps that are not needed to guarantee proof of existence. We show that this can be done by using a data structure called *skip list* to generate and select the time-stamps. Third, the time-stamps are generated using signature schemes and hash functions. Since these two cryptographic primitives usually have different lifetimes, we distinguish between two types of time-stamps to be created. One type is to address the aging of signatures and requires hashing only the previous time-stamp in the sequence. The other type is to be used before the last used hash function becomes insecure and requires building a Merkle tree from all previous time-stamps. Since signatures are expected to have shorter lifetimes than hash functions, using the appropriate type of time-stamps prevents the retriever from hashing not necessary data; hence, hashing times are reduced. We provide a performance analysis for CIS and CISS. First, we compare the schemes analytically with respect to time and space. Then, we run an experiment and compare the results with the analytical evaluation. The

analysis reveals that CISS provides a significantly faster evidence verification. The content of this chapter was published as parts of [57].

7.1 Improving the performance of CIS

The time-stamping scheme CIS described in Chapter 5 is a promising scheme. It allows an archivist to time-stamp documents at distinct moments using a single time-stamp sequence. That is, instead of generating and maintaining a time-stamp sequence for each time-stamped document, he keeps only one time-stamp sequence which is updated when a further document needs to be time-stamped. Thus, the documents share the same time-stamp sequence as evidence.

This feature of CIS is desired, for example, in two situations. First, when the format or the content of a time-stamped document must be changed frequently. In this case, CIS can be used to time-stamp the history of changes of this document. Second, when a set of documents are submitted to the archive and time-stamped sequentially, CIS allows for generating a single time-stamp sequence for this set. Thus, the archivist needs to address the aging of cryptography of only one time-stamp sequence.

Although CIS provides this important feature for archives, Chapter 5 revealed performance issues. In this section we first discuss four issues that affects the performance of CIS. Then, we show how they are addressed in the new time-stamp scheme time-stamping scheme Content Integrity Service with Skip Lists (CISS).

The first issue is that, even if a retriever wants to verify the protection goals of only one document from a set of documents by checking the time-stamp sequence, she must hash all the whole set.

The second performance issue is that the retriever may need to verify redundant time-stamps, i.e. time-stamps that are not needed to guarantee the protections goals of the document she wants to verify. These time-stamps are generated not to address the aging of signatures or hash functions, but to provide the protection goals for new documents added to the set of documents. Note that verifying time-stamps requires checking certificates, which Chapter 5 revealed to be very time-consuming.

The third issue is the hashing of time-stamps that are not needed for verification. More precisely, to verify every time-stamp in the sequence, the retriever must compute one hash from the previous time-stamps. This is required when verifying time-stamps generated to address the aging of hash functions. However, this is not the case of time-stamps created to cope with the aging of signatures. (In this case, hashing only the previous time-stamp is sufficient.) Since signatures have usually a

shorter lifetime than hash functions, most of the time-stamps in the sequence are to address the aging of signatures. Therefore, the verification of the sequence may potentially require hashing many times time-stamps which are not necessary.

The fourth issue is that, even when all previous time-stamps must be hashed, this is done inefficiently in CIS. More precisely, a single hash is computed recursively, where in each step a hash is computed from a part of the time-stamp sequence together with the hash computed in the previous step. This procedure turns out to be inefficient because it hashes extra data, namely it computes hashes from hashes (see Chapter 5, Table 5.2).

Note that the first and second issues happen when one sequence of time-stamps is used as evidence for a set documents in the archive. The third and fourth issue happen even when using CIS to generate a time-stamp sequence for only one document.

Therefore, we propose the new time-stamping scheme CISS to address the above issues. We first solve the issues of hashing all documents (first issue) and verifying redundant time-stamps (second issue) by combining skip lists with time-stamps. This is shown in Section 7.1.1. Next, we address the issue of hashing not necessary time-stamps (third issue) by distinguishing between two types of time-stamps. One type for addressing the aging of hash functions and another for the aging of signatures. This is show in Section 7.1.2. Furthermore, the proposed solution is not affected by the CIS issue of hashing time-stamps recursively (fourth issue).

7.1.1 Combining skip lists with time-stamps

In this section we describe how to combine skip lists with time-stamps to provide long-term proof of existence for documents. First, we present skip lists. Next, we present existing hash-based skip lists and their limitations. Following, we describe our proposal of a hash-based skip list which uses time-stamps. The proposal is intended to not only provide the promised protection goals but also allow for their efficient verification.

Skip lists

As shown in Section 2.12, a skip list is a data structure that can be used to store a sequence elements. The skip list contains several linked lists which store the elements added to the skip list. In a linked list, every element has a link to the next element. The stored elements and the internal linked lists are ordered such that we can search for an element without visiting every element, i.e. we can *skip* elements.

Existing skip lists based on hash links

Maniatis and Baker [34] design the first skip lists to be used as integrity evidence for documents. In their skip lists, the internal links are realized as hashes. We provide a simplified description of the construction of these hashes. For $i \geq 0$, assume that d_i and d_{i+1} are the $(i+1)$th and $(i+2)$th documents added to a skip list. An internal link l from d_i to d_{i+1} is computed by $l = h(d_i||d_{i+1})$, where h is a hash function and $||$ is concatenation.

The integrity evidence for document d_i consists of the hash link l and document d_{i+1}. To verify integrity, a retriever recomputes the hash link as before and checks whether it matches l.

However, these skip lists have three issues. First, when verifying the integrity of one document, the retriever needs to hash additional documents (e.g. to verify d_i she must hash d_{i+1}). This issue is also found in CIS. Second, Maniatis and Baker provide no proof of existence for the documents in the skip lists[6]. Third, the aging of the used hash functions is not addressed, therefore long-term integrity cannot be achieved.

The new time-stamping scheme that we propose extends these skip lists such that only the document to be verified is hashed and long-term proof of existence is achieved efficiently.

Allowing for verifying integrity without hashing additional documents

Our first improvement of CIS is to allow a retriever to verify the integrity of a document without hashing additional documents. For this, we propose to store document hashes instead of the documents themselves in the hash-based skip lists.

Let us provide an example to illustrate our proposal when integrity evidence is provided. (Proof of existence will be addressed in the next section.) Assume that the internal hash links of the skip list are created between two documents hashes. Thus, for $i \geq 0$ when the skip list already contains document hash $h(d_i)$ and $h(d_{i+1})$ is added, a link l from $h(d_i)$ to $h(d_{i+1})$ is computed by $l = h(h(d_i)||h(d_{i+1}))$.

In the above example, the integrity evidence for document d_i consists now of hash link l and document hash d_{i+1}. Note that, contrary to the skip lists proposed by Maniatis and Baker, document d_{i+1} is no longer needed; hence, the size of evidence is reduced. To verify integrity, the retriever first computes $h(d_i)$. Then she recomputes the link as before and compares whether it matches l.

[6] A hash link can be used as a proof of existence which uses no date. Such sort of proof of existence is not sufficient to achieve our protection goals because dates are needed that allow retrievers to verify whether cryptographic primitives were used before they became insecure.

This example does not allow for proof of existence. To provide this protection goal, we will see later that the computation of l needs to be slightly changed.

Providing proof of existence for hash-based skip list

The next step in designing our new time-stamp scheme is to provide proof of existence for the documents hashes and hash links in the skip list. For this, we create a sequence of time-stamps to be used together with the hash links as evidence of proof of existence for a set of archived documents. As we will see, the verification of this time-stamp sequence has some performance issues that will be addressed later.

Since the set of archived documents share the same time-stamp sequence, the archivist must update the sequence when a new document is received or a document already in the set is changed. A procedure for this is to time-stamp every new document together with the most recent time-stamp in the sequence. This procedure is available in CIS but here we realize it more efficiently by using the internal hash links of the skip lists as follows.

Assume d_0 is the first document in the set to be protected. The archivist computes the hash $h(d_0)$ and adds it to an empty skip list. Since $h(d_0)$ is the only element in the skip list, he creates no internal hash links in the linked list L_0 in level 0. Next, he requests a time-stamp T_0 on $h||h(d_0)$.

Now assume d_1 is the second document in the set to be protected. The archivist computes the hash $h(d_1)$, adds it to the skip list, and creates the hash link l_1^0 from T_0 to d_1 in the linked list L_0 such that $l_1^0 = h(T_0||h(d_1))$. Next, he creates a time-stamp T_1 on $h||l_1^0$.

In general, for $i \geq 0$, the archivist adds the hash $h(d_i)$ of the new document d_i to the skip list, generates the hash link $l_i^0 = h(T_{i-1}||h(d_i))$ in the linked list L_0, and requests a time-stamp T_i on $h||l_i^0$.

Note that by adding a new time-stamp T_i to the sequence, the archivist not only establishes proof of existence for the new document d_i, but also extends the validity of the previous time-stamp T_{i-1}. (It may happen that the validity T_{i-1} is about to expire but there is no document hash $h(d_i)$ to be added to the skip list. We will deal with this situation and especially with the aging of hash functions in Section 7.1.2.) This is why he can use a single time-stamp sequence as evidence for documents that are time-stamped at distinct times.

We now identify performance issues in this time-stamping procedure. When adding a time-stamp T_i to the sequence, the archivist extends the validity of T_{i-1} even though if T_{i-1} is far from being expired. In this situation, when verifying the protection goals for a document d_j $(0 \leq j < i)$, a retriever needs to check

time-stamps $T_j, \ldots, T_{i-1}$ but not necessarily T_i. She can skip T_i if T_{i-1} is still valid.

An example of the above situation is found in Figure 7.1. The document hashes $h(d_0)$, $h(d_1)$, $h(d_2)$, and $h(d_3)$ were added to the skip list and time-stamped at times t_0, t_1, t_2, and t_3 using time-stamps T_0, T_1, T_2, and T_3, respectively. When these time-stamps are created ("•"), the validity of the previous time-stamp is far from being expired ("○"). A retriever verifies the proof of existence of document d_0 at time t_4, when T_2 and T_3 are still valid. In this case, she checks time-stamps T_0, T_1, and T_2 but skips T_3.

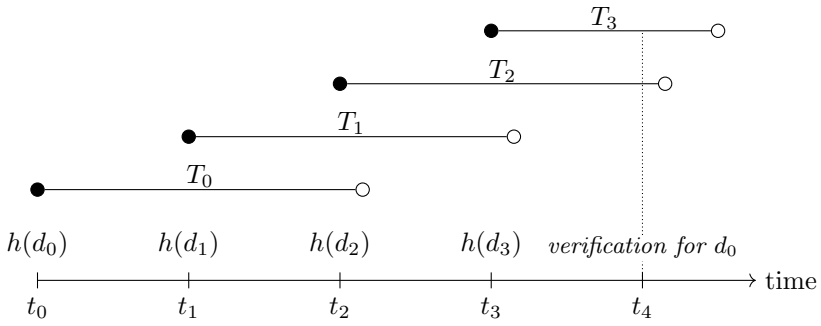

Figure 7.1: The time-stamps T_0, T_1, T_2, and T_3 provide proof of existence for the document hashes $h(d_0)$, $h(d_1)$, $h(d_2)$, and $h(d_3)$ at the times t_0, t_1, t_2, and t_3. Every time-stamp extends the validity of the previous time-stamp. The beginning and end of a validity are indicated by "•" and "○", respectively. At the time t_4 a retriever checks the proof of existence of d_0.

Although the retriever can skip the most recent time-stamp, she must verify the other time-stamps in the sequence. Assume that the archivist adds a new time-stamp T_{i+1} before T_i and T_{i-1} expire. Next, assume that the retriever verifies the time-stamp sequence for d_j when T_i and T_{i-1} have expired and T_{i+1} is still valid. In this case, both T_i and T_{i-1} are redundant but they must be verified because each time-stamp is applied on a hash (i.e. a hash link) from the previous time-stamp in the sequence.

Therefore, it may happen that the addition of new document hashes to the skip list generates time-stamps that are not necessary to guarantee the proof of existence of previous documents. As a consequence, performance can be significantly affected because checking time-stamps includes the time-consuming verification of certificates (see Chapter 5). This issue will be address in the next section.

Verifying time-stamps sequences efficiently

As discussed before, some time-stamps in the sequence may be not needed to guarantee the protections goals of a document. However, these time-stamps must be checked because of the construction of the time-stamp sequence. (In the sequence, each time-stamp is created on a hash link computed from the previous time-stamp.) Our next step is to allow retrievers to skip the verification of such not necessary time-stamps. Thus, we first present the idea behind skipping time-stamps and then we show how to realize it. Afterwards, we provide an example where we combine hash-based skip lists and time-stamps.

To allow for skipping time-stamps, we propose a new construction of hash links as follows. As explained before, when an archivist adds a new document hash to the skip list, he creates a hash link from the most recent time-stamp to the new document hash. However, in the new construction if there exist older time-stamps which have not expired, he also creates hash links from these time-stamps to the new document hash. Finally, he requests a new time-stamp on the hash links he has just created.

The result of this new construction is twofold. First, proof of existence for the new document is provided as before. Second, there can be more than one time-stamp in the sequence that extends the validity of an earlier time-stamp.

When verifying the time-stamp sequence for one document, a retriever needs to check only a subsequence of the time-stamp sequence. The first time-stamp in the subsequence is the time-stamp created when the hash of the document was added to the skip list. The last time-stamp is the most recent time-stamp. The intermediary time-stamps are those applied on hash links that make up the shortest path from the first to the last time-stamp in the skip list.

For example, assume that the archivist added document hashes $h(d_0)$, $h(d_1)$, and $h(d_2)$ to a skip list and requested time-stamps T_0, T_1, and T_2 at times t_0, t_1, and t_2, respectively. In this time-stamp sequence, T_2 provides proof of existence for d_2 and extends the validity of T_1. T_1 provides proof of existence for d_1 and extends the validity of T_0. T_0 guarantees proof of existence of d_0. Figure 7.2 depicts the document hashes and time-stamps over a time-line. The black circles indicate when the time-stamps are created. The white circles show when the time-stamps expire, i.e. when the signatures on the time-stamps expire.

Now assume that the archivist adds document hash $h(d_3)$ at time t_3, when the time-stamps T_1 and T_2 have not expired yet. Then, he creates the hash link l_3^0 from the most recent time-stamp T_2 to $h(d_3)$ such that $l_3^0 = h(T_2||h(d_3))$. Furthermore, he creates a hash link l_3^1 from the unexpired time-stamp T_1 to $h(d_3)$ such that

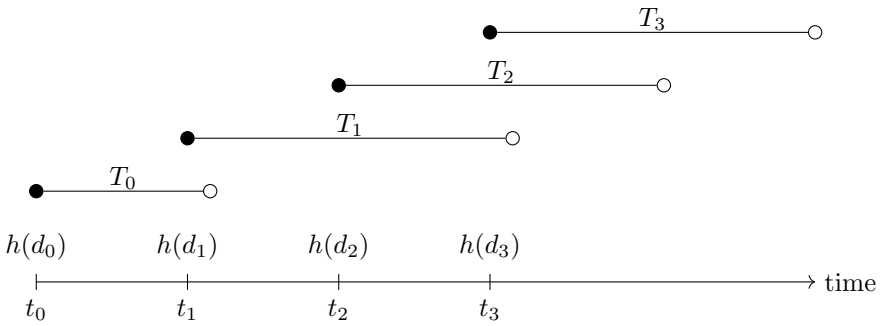

Figure 7.2: The time-stamps T_0, T_1, T_2, and T_3 provide proof of existence for the document hashes $h(d_0)$, $h(d_1)$, $h(d_2)$, and $h(d_3)$ at the times t_0, t_1, t_2, and t_3. Each time-stamp also extends the validity of the previous time-stamp. The beginning and end of a validity are marked with "•" and "◦", respectively.

$l_3^1 = h(T_1||h(d_3))$. Finally, he generates a time-stamp T_3 on $h||h(l_3^0||l_3^1)$. Thus, T_3 extends the validities of time-stamp T_2 on $h(d_2)$ and time-stamp T_1 on $h(d_1)$.

In this example, when a retriever verifies the proof of existence of d_0, she checks only the time-stamps T_0, T_1, and T_3. She can skip T_2 since T_3 extends the validity of the signature on T_1.

However, when adding a new document hash, the more time-stamps have not expired, the more hash links can be generated. To avoid that creating links consumes considerable space and time, we limit the number of links to $g+1$, where g depends on the position of the document hash in the skip list and can be calculated with Equation 2.5 presented in Section 2.12.

Next, we show how to realize our skip list taking the limit $g+1$ into account. We also consider that verification information V is needed to verify time-stamp T and that skip lists have several linked lists $L_0, L_1, \ldots$. Furthermore, we use indices to identify the used hash functions because they must be replaced over time. Their replacement is discussed in Section 7.1.2.

Let d_0 be the first document for which the archivist generates evidence at time t_0. He executes the following steps.

1. Select a hash function h_1 and initialize an empty skip list.
2. Compute hash $h_1(d_0)$ and add it to skip list at position 0.
3. Request a time-stamp T_0 on $h_1||h_1(d_0)$.

Now, assume d_1 is the next document to be protected at time $t_1 > t_0$, when the signature on time-stamp T_0 has not expired and the hash function h_1 is still secure.

The archivist proceeds as follows.

4. Compute hash $h_1(d_1)$ and add it to skip list at position 1.
5. Obtain verification information V_0 showing that the signature on T_0 is valid at t_1.
6. Create a link l_1^0 from $T_0||V_0$ to hash $h_1(d_1)$ in the linked list L_0 such that $l_1^0 = h_1(T_0||V_0||h_1(d_1))$.
7. Request a time-stamp T_1 on $h_1||h_1(l_1^0)$.

Now assume at time $t_2 > t_1$ the signature on T_1 has not expired, h_1 is still secure, and d_2 is the next document to be protected. The archivist executes the following steps.

8. Compute hash $h_1(d_2)$ and add it to skip list at position 2.
9. Obtain verification information V_1 showing that the signature on T_1 is valid at t_2.
10. Create a link l_2^0 from $T_1||V_1$ to hash $h_1(d_2)$ in the linked list L_0 such that $l_2^0 = h_1(T_1||V_1||h_1(d_2))$.
11. If time-stamp T_0 has not expired at time t_2,
    a) then
        i. Create a link l_2^1 from $T_0||V_0$ to $h_1(d_2)$ in linked list L_1 such that $l_2^1 = h_1(T_0||V_0||h_1(d_2))$.
    b) else
        i. Create empty string named l_2^1.
12. Request a time-stamp T_1 on $h_1||h_1(l_2^0||l_2^1)$.

Assume for $i > 2$ that the archivist has added document hashes $h_1(d_0), \ldots, h_1(d_{i-1})$ to the skip list, and that he has generated time-stamps $T_0, \ldots, T_{i-1}$. At time t_i, when h_1 is still secure and at least the signature on T_{i-1} has not expired, the archivist generates evidence for document d_i as follows.

13. Compute hash $h_1(d_i)$ and add it to skip list at position i.
14. Compute g using i as input for Equation 2.5 presented in Section 2.12.

15. For every $0 \leq j \leq g$, if there exists an unexpired signature on a time-stamp T_k such that $k = min\{x \in \mathbb{Z}^* | i - 2^j \leq x < i - 2^{j-1}\}$,
    a) then
        i. Collect verification information V_k showing that T_k had a valid signature at time t_{k+1}.
        ii. Create a link l_i^j from $T_k||V_k$ to $h_1(d_i)$ in the linked list L_j such that $l_i^j = h_1(T_k||V_k||h_1(d_i))$.
    b) else
        i. Create an empty string named l_i^j.
16. Request a time-stamp T_i on $h_1||h_1(l_i^0||\ldots||l_i^g)$.

Finally, an example of such a skip list is given in Figure 7.3. The figure illustrates the following scenario. The archivist has added document hashes $h_1(d_0)$, $h_1(d_1)$, $h_1(d_2)$, and $h_1(d_3)$ to the skip list. Also, he has generated the time-stamps T_0, T_1, T_2, and T_3 on these hashes at times t_0, t_1, t_2 and t_3. Next, the archivist adds hash $h_1(d_4)$ at time t_4, when only the signature on T_0 has already expired. He computes $g = 2$ by plug in the parameters $i = 2$ and $m = 1$ in $g = \log_2(i/m)$ (see Section 2.12, Equation 2.5). Then, he creates $g+1$ links: (1) l_4^0 from time-stamp T_3 and verification information V_3 to $h_1(d_4)$, (2) l_4^1 from T_2 and V_2 to $h_1(d_4)$, and (3) l_4^2 from T_1 and V_1 to $h_1(d_4)$. Note that he does not generate the link l_4^2 from T_0 and V_0 because the signature on T_0 has already expired at time t_4.

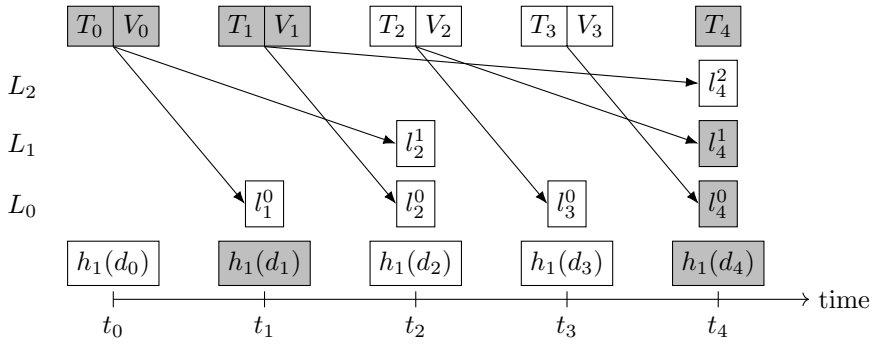

Figure 7.3: Using a hash-based skip list together with time-stamps to guarantee to the protection goals of the documents d_0 to d_4. The hash links l are constructed with the hash function h_1. An arrow from a time-stamp T and its verification information V to a link l indicates that T and V are hashed to produce l. The evidence for the document d_0 at the time t_4 consists of the colored objects.

Evidence for retrievers and its verification

The archivist provides retrievers with the evidence extracted from a skip list as follows. Assume for $n > 0$ that he has added documents $d_0, \ldots, d_n$ to the skip list sequentially. Thus, the evidence for document d_i $(0 \leq i \leq n)$ consists of hashes, time-stamps, and verification information. These hashes are the document hashes and hash links extracted from the shortest path from the hash of d_i (position i) to the most recent time-stamp in the skip list (position n). This sequence includes no hash which retrievers can compute by themselves. The time-stamps are also extracted from the shortest path. If $i > 0$, then the time-stamp and verification information at position $i - 1$ is also extracted.

We provide two examples using Figure 7.3. First, we present the evidence for the first document in the skip list, i.e. d_0, at time $t > t_4$. Its evidence consists of hashes $h_1(d_1), h_1(d_4), l_4^0, l_4^1$, time-stamps T_0, T_1, T_4, and verification information V_0, V_1. Note that the evidence does not include, for example, the hash link l_4^2, which retrievers can compute from $h_1(d_4)$, T_1, and V_1 (i.e. $l_4^2 = h_1(T_1||V_1||h_1(d_4))$). Furthermore, retrievers must collect verification information for time-stamp V_4 by themselves.

The next example is for the evidence for the second document, i.e. d_1, also at time $t > t_4$. Its evidence consists of hashes $h_1(d_4), l_4^0, l_4^1$, time-stamps T_0, T_1, T_4, and verification information V_0, V_1. Note that although a retriever does not check the signature on T_0, she needs T_0 and V_0 to compute the hash link $l_1^0 = h_1(T_0||V_0||h_1(d_1))$.

Provided an archived document and the evidence, a retriever verifies the protection goals of the document as follows.

1. Compute the hash of the document.
2. Compute the hash links which were not provided.
3. Verify the signature on each time-stamp.

Steps 1 is needed to verify the integrity of the document. Step 2 is required to check the integrity of time-stamps and their verification information. Step 3 is necessary to verify the authenticity of time-stamps and the proof of existence of the document.

7.1.2 Addressing the aging of cryptography

An issue identified in CIS is that the aging of cryptography is addressed inefficiently. This is because CIS provides a single procedure to extend the lifetime of signatures and the lifetime of hash functions. To solve this issue in our new scheme CISS, we

distinguish between the aging of signatures and the aging of hash functions. Thus, in this section we first explain a procedure to be executed before the latest used signature algorithm or keys become insecure (signature aging). Then, we describe a procedure to be run before the latest used hash functions becomes insecure (hash function aging). Finally, we show how to adapt the evidence such that a retriever can verify whether the aging of cryptography has been properly addressed.

As explained before, when an archivist adds the hash of a new document to the skip list, he generates hash links and request a new time-stamp on these hash links. The hash links are computed from the time-stamps that have not expired together with the new document hash. Afterwards, a new time-stamp is requested on these hash links. Therefore, when the archivist adds a new document hash to the skip list, he is already addressing the aging of the signatures on the unexpired time-stamps.

However, a question that remains is how to address such aging when there is no new document hash to be added to the skip list for a long period. To solve this issue, we propose that the archivist simply adds an arbitrary hash (e.g. a string of zeros) to the skip list, generates the necessary hash links, and time-stamps the generated links.

Nonetheless, the above procedure does not address the aging of the used hash function h_1. It is necessary to generate new evidence showing that the document hashes, the hash links, the time-stamps, and the verification information already existed before h_1 became insecure. For this, the archivist needs to select a new hash function h_2, compute a hash from all these objects, and apply a new time-stamp on the computed hash. We detail these procedures in the following.

To compute a hash from the above objects efficiently, we suggest to use a binary hash tree as proposed by Blazic et al. [3]. This can be, for instance, a Merkle tree (see Section 2.4). Assume for $i \in [1, n]$ that document hashes $h_1(d_i)$ have been added to the skip list and that T_n is the most recent time-stamp. Evidence E_i for document d_i contains the shortest path from $h_1(d_i)$ to T_n, i.e. the links that a retriever cannot compute by herself, the preceding time-stamp if necessary, time-stamps for the links, and the corresponding verification information. The archivist computes hashes $h_2(d_i||E_i)$ as the leaves of a new binary hash tree. Next, he calculates every internal node and the Merkle tree root r_1 as described in Section 2.4. Furthermore, he computes the authentication path $A_{i,1}$ from each leaf $h_2(d_i||E_i)$ to the root r_1.

To time-stamp the computed hash, i.e. the Merkle tree root r_1, the archivist uses a new skip list. He initializes an empty skip list and adds r_1 as the first element. Next, he requests a time-stamp on r_1 as for the first document hash in the first skip list.

As we described before, the evidence contains the hashes in the shortest path

from the document hash to the most recent time-stamp in the skip list. Now, we need to extend this evidence such that retrievers can check whether the archivist has addressed the aging of the used cryptography. For the aging of signatures nothing changes in the evidence. This is because the archivist simply added an arbitrary hash instead of a document hash to the skip list. For the aging of hash functions, the evidence E_i for a document d_i is augmented by the authentication path $A_{i,1}$ from d_i to the Merkle tree root r_1.

A retriever verifies the extended evidence for document d_i as follows. She generates the missing hash links and verifies the time-stamps as before. The exception is when checking the time-stamp on the Merkle tree root r_1. In this case, she first selects the hash function h_2 used to generate r_1. Next, she recomputes the Merkle tree root r_1 from d_i, its evidence E_i, and the corresponding authentication path $A_{i,1}$. Finally, she verifies the signature on the time-stamp using the recomputed root r_1 and the needed verification information.

Note that after the Merkle tree root r_1 was time-stamped, new documents can also be time-stamped. In this case, they must be hashed using the latest hash function h_2 and the computed hashes must be added to the new skip list. For the documents time-stamped before r_1 was generated, the evidence must also include the shortest path from r_1 to the most recent time-stamp.

An example of the above situation is given in Figure 7.4, where two skip lists are illustrated. The first skip list is initialized at time t_0 and contains document hashes, hash links, and time-stamps generated until time t_4 using hash function h_1. The second skip list is initialized at time t_5, when h_1 is replaced by hash function h_2. The second skip list contains a Merkle tree root r_1 computed from the objects stored in the first skip list. Also, it contains the document hashes added after t_5. The set of hashes and time-stamps in the shortest path from an added hash to the last time-stamp in a skip list is denoted by S. Thus, at time $t > t_8$, the evidence E_0 for document d_0 contains S_1, the authentication path from d_0 to r_1, and S_5. For document d_6, the evidence contains only S_6.

Moreover, the archivist must generate a new Merkle tree root whenever the latest used hash function is about to become insecure. However, when computing the leaves of the new Merkle tree, the authentication paths in the previous trees must also be hashed. More precisely, for $j > 0$ assume that the evidence E for document d was initialized using hash function h_j. Now, for $k > j+1$ assume that the archivist selects a new hash function h_k and that he computes a new Merkle tree root r_k. For this, he computes the leaf $h_k(d||E)$ and E must include the authentication paths from document d to the previous roots $r_j, \ldots, r_{k-2}$.

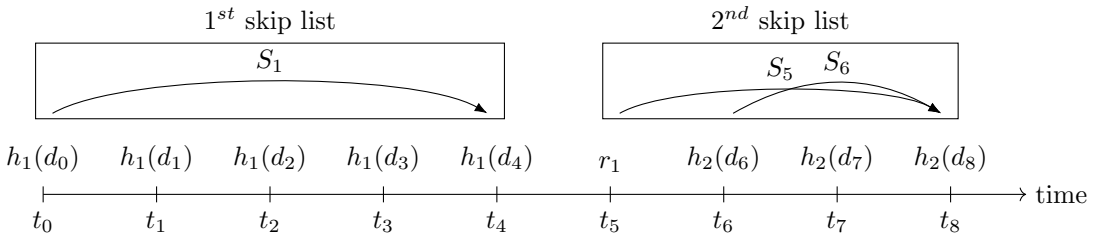

Figure 7.4: Two skip lists. The first skip list is used to provide proof of existence for the documents d_0 to d_4 until time t_4 using the hash function h_1. The second skip list contains the Merkle tree root r_1 computed from the objects in the first skip list using the hash function h_2. It also contains the hashes of the objects d_6 to d_8. The shortest path from an added hash to the last time-stamp in a skip list is denoted by S.

7.2 Performance analysis

This section provides a performance analysis for CIS and CISS while they are used to generate and verify evidence for a set of documents in an archive. The generated evidence is a single sequence of time-stamps which is shared by these documents. The documents are signed by their owners, and new documents can be added to the set and time-stamped after the evidence was initialized. We do not consider other schemes from previous chapters because they were not designed for the described scenario. Moreover, we do not repeat the analysis from Chapter 5 for CISS, where the number of documents in the set is fixed. For this scenario we have already shown that the time-stamping scheme ERS and the notarial schemes are the best options.

As in Chapter 5, we first present an analytical evaluation. Next, we run experiments using prototypical implementations and analyze the obtained results. In the evaluation and experiments, we compare the schemes with respect to time and space. Time refers to the times needed to generate and verify evidence. Space refers to the size of the generated evidence. We do not evaluate the communication complexity of these schemes because they are expected to use the same number of time-stamps.

7.2.1 Analytical evaluation

In this section we analyze how CIS and CISS differ in performance without taking any specific cryptographic primitives into account. As before, we assume that the most time-consuming operations in these schemes are hashing and signature verification. Therefore, we use the numbers of hashed objects and verified signatures to compare the schemes.

In contrast to previous analytical evaluations, we cannot count the hashed objects and verified signatures as a function of the iteration in which evidence is updated or verified. The reason is that the evidence must be updated when new documents are time-stamped, however we cannot predict in which iteration the next document will be time-stamped. Therefore, instead of iterations we will use the length k of the time-stamp sequence and the number f of documents in the set, where $f = k = 0$ before the first document is time-stamped, and $1 \leq f \leq k$ before the $(f + 1)$th document is time-stamped.

Moreover, for this analysis we assume that the documents are signed by their owners before being time-stamped.

Performance of evidence initialization

We start with the performance of evidence initialization, which takes place when an archivist time-stamps the first signed document. In this evaluation we count the objects the archivist must hash in each scheme. In CIS and CISS, he hashes the same objects: the initial document, the signature on the document, and the verification information for the signature. Therefore, CIS and CISS are expected to perform similarly.

Performance of evidence update

We now analyze the performance of updating an initialized sequence of k time-stamps for f signed documents, where $1 \leq f \leq k$. We count the objects the archivist must hash when requesting the next time-stamp. However, here we need to distinguish between three situations: when a new signed document is time-stamped, when the signature lifetime of the last time-stamp is about to end, and when the lifetime of the last used hash function is close to expire. We next analyze each situation individually.

We first analyze the situation where a new signed document is time-stamped. For this situation, we refer to CIS as CIS-F and to CISS as CISS-F in Table 7.1. Note that for CISS-F the number of hashed time-stamps and verification information depends on the hash links to be created from the unexpired time-stamps. Since we cannot predict how many time-stamps have not expired when the next document is time-stamped, we assume the worst-case scenario for CISS, where the maximum number $g+1$ of hash links are generated. Recall that $g = \log_2(k/m)$, where $m \in [1, k]$ is odd (see Equation 2.5 in Section 2.12).

The table shows that the number of objects to be hashed in CIS-F is proportional to k and f whereas in CISS-F it is proportional to $g + 1 \approx \log_2 k$ in the worst-case

Table 7.1: The numbers of objects being hashed while updating a sequence of k time-stamps which is used as evidence for f documents, where $1 \leq f \leq k$, $g = \log_2(k/m)$, and $m \in [1, k]$ is odd.

Scheme	Documents	Signatures	Time-stamps	Ver. info	Hashes
CIS	f	f	k	$k+f$	$2k+f-1$
CIS-F	$f+1$	$f+1$	k	$k+f+1$	$2k+f$
CISS-F (worst-case)	1	1	$g+1$	$g+2$	$2g+2$
CISS-O	0	0	1	1	2
CIS-N	k	k	k	$2k$	$3k-1$
CISS-N (best-case)	k	k	$3k-2$	$4k-2$	$5k-3$

scenario. It follows that for larger k and f CISS-F is expected to outperform CIS.

Next, we compare CIS and CISS when the next time-stamp is needed to address the aging of the last time-stamp signature. The number of hashed objects for CIS and CISS corresponds to CIS and CISS-O in the table. In this case, CISS-O is independent of k and f. Therefore, when k and f grows CISS-O is expected to be faster than CIS.

Finally, we compare the schemes as they address the aging of the last used hash function. Let us first explain how we count the objects that must be hashed to compute a Merkle tree in CISS. As explained in Section 7.1.2, these objects are every document and its evidence, which contains the hash links, time-stamps, and the corresponding verification information in the shortest path from the document time-stamp to the last time-stamp in the skip list. However, since we cannot predict the shortest path, we make two simplifications. First, we assume $1 \leq f = k$, i.e. new time-stamps were only created when a new document was time-stamped. This simplification is needed because, if we also consider the time-stamps generated to address the aging of cryptography, we cannot predict whether a document hash was added to the skip list before addressing the aging of cryptography.

In the second simplification, we assume the best-case scenario for CISS, where the shortest path from each document hash to the last time-stamp contains only a single hash link. Thus, the evidence for each document is minimal and contains one hash link and up to three time-stamps: 1) the time-stamp generated before the document was time-stamped, 2) the time-stamp on the document, and 3) and the last time-stamp. Also, the evidence contains a document signature and all verification information needed.

In the table, CIS-N and CISS-N stand for the numbers of hashed objects for CIS and CISS when the schemes replace the last used hash function. As can be seen

from the expression, for large k CISS-N should be slower since it hashes more time-stamps, verification information, and hashes. Since CISS-N is slower than CIS in the best-case scenario, for any scenario CIS should outperform CISS-N.

Performance of evidence verification

To analyze the performance of evidence verification, we compare CIS and CISS with respect to only the number of signatures to be verified. Although in Chapter 5 we have also counted the objects to be hashed, the experiments presented so far have shown that hashing is insignificant during evidence verification.

As before, we assume that there exists a sequence of k time-stamps which is used as evidence for a set of f signed documents, where $1 < f \leq k$. Moreover, we assume a retriever wants to verify the first document time-stamped in the set (the worst-case scenario). For CIS, this verification requires the retriever to check the k time-stamps (i.e. k signatures). In CISS, she can check fewer time-stamps because of the hash links between nonconsecutive time-stamps in the skip list. These links are generated when creating the $f - 1$ time-stamps on the other documents in the set. (Whether the links are created depends on the position of the new time-stamp in the skip list and the number of unexpired time-stamps.) If these links exist in the skip list, then the retriever skips the verification of the time-stamps between any two linked time-stamps. Consequently, she verifies less than k time-stamps. It follows that CISS is expected to outperform CIS.

Space complexity

To analyze how CIS and CISS differ in space complexity, we count the objects stored as evidence. In CIS, the document signatures, time-stamps, and verification information are stored. In addition to what is stored in CIS, in CISS hashes are stored, such as hash links and authentication paths. Since these hashes are significantly smaller than the other objects, CIS is expected to require slightly less space than CISS.

7.2.2 Experiment

This section provides an experiment that was carried out to compare the performance of CIS and CISS in practice. First, we present the prototypical implementation for CISS. Next, we describe the experiment. Finally, we discuss the results comparing them with the predictions presented in the analytical evaluation.

Implementation design

To allow for comparing CISS with CIS in practice, we implemented a prototype for CISS using the same technologies mentioned in Chapter 5. Furthermore, we executed the prototypes for CIS and CISS on a personal computer with an Intel i5 M560 2.67 GHz processor and 4 GB RAM running Solaris 11. The executions used no code optimization (i.e. *just-in-time-compilation*) because we cannot guarantee it equally improves the performance of all prototypes.

As before, the public keys needed to verify signatures are distributed in the form of X.509 certificates. These certificates are the last certificates c_3 of certificate chains c_1, c_2, c_3. For each certificate in the chains there is an empty certificate revocation list.

Experiment design

In this experiment, two sequences of time-stamps are generated. The first sequence contains the time-stamps created by the CIS prototype. The second sequence consists of the time-stamps generated by the CISS prototype. The time-stamps are generated and added to the sequences when a new signed document is time-stamped. Both sequences are initialized in 2013, when the first document and its signature are time-stamped. Then, in every year from 2014 to 2113, a new document and signature are time-stamped and the two time-stamp sequences are updated.

As in Sections 5.2.2 and 6.2.2, each received document is 30720 bytes in size and every generated time-stamp is expected to be valid for five years after its generation. The key sizes used to sign documents and time-stamps are selected in accordance to the conservative predictions by Lenstra [31]. For each generated signature, the selected key size is expected to be secure for five years after the signature generation. The selected key sizes are found in Table 5.7.

The hash function SHA-256 is also used to generate the signatures and skip list until 2085. Since SHA-256 is expected to be secure until 2090 and every time-stamp is expected to be valid for five years, we replace this hash function by SHA-384 in 2086. In this year a new time-stamp is generated for CIS and CISS but no signed document is received in the archive. For CISS, in this year a Merkle tree is built.

As said, a new time-stamp is added every year from 2013 to 2113 and each time-stamp is expected to be valid for five years. Thus, every time the time-stamp sequences are updated, there are up to four unexpired time-stamps. Therefore, depending on the position of a new time-stamp in a skip list, CISS can generate hash links from the unexpired time-stamps to the new time-stamp. When verifying the time-stamp sequence, such links allow for skipping up to three consecutive

time-stamps. Figure 7.5 illustrates this situation when the fifth time-stamp T_4 is generated and hash links from the unexpired time-stamp T_0, T_2, and T_3 to T_4 are created. The time-stamps T_1, T_2, and T_3 can be skipped upon verifying the time-stamp sequence for the first document d_0.

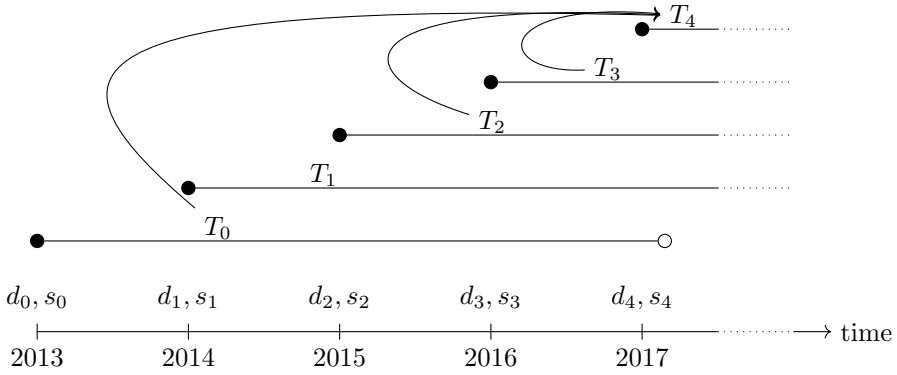

Figure 7.5: The time-stamp sequence $T_0, \ldots, T_4$ generated for the documents d_0 to d_4 and the corresponding signatures s_0 to s_4 in the experiment. "•" indicates when a time-stamp is generated, while "∘" shows when it becomes invalid. The arrows stand for the hash links in the skip list.

For each year, we measured the running times CIS and CISS used to update or verify their time-stamp sequences. Moreover, we measure the space that these schemes required to store the time-stamp sequences, verification information, and document signatures. For CISS, the space is also used to store hash links and authentication paths.

Results

In this section we discuss the results obtained from the experiment. We compare the measured performance with the predictions presented in Section 7.2.1. This comparison is done for the performance of evidence initialization, update, and verification and for space complexity. As we will see, some results differ from the predictions. In this case, we point out possible reasons for such differences.

Figure 7.6 compares the running times needed to generate evidence. To allow the reader to compare the measured times visually, we have not corrected them in accordance with Moore's law. In 2013 the schemes use approximately the same running times to initialize the time-stamp sequences. This confirms the predictions that the schemes should be comparable when initializing evidence.

Next is the performance of evidence update. In the years when a new document

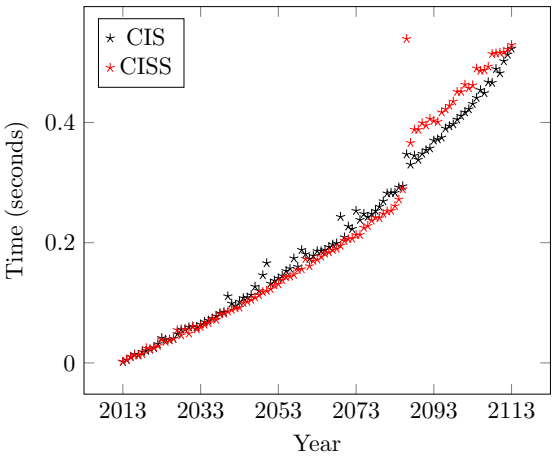

Figure 7.6: The times needed to generate a time-stamp sequence for a set of documents. The sequence is updated every year, when a new document is added to the set.

is time-stamped (2013 to 2095 and 2097 to 2113), we can see that the running times do not confirm the advantage of CISS over CIS predicted in Table 7.1. Instead, the schemes are comparable, being CISS is slightly faster from 2013 to 2095 and slower from 2097 to 2113.

This contradiction indicates that the analytical evaluation did not include the operations that compensate the shorter hashing times for CISS. By profiling the CISS prototype, two operations other than hashing seem to use non-negligible running times during evidence update. First is the translation of data between the formats used for the evidence files (viz. ASCII) and for the input and output of hash functions and signature schemes. Second is the parsing of the evidence files.

Next, we compare the schemes as they generate evidence and replace hash function SHA-256 by SHA-384 in 2086. In this year, CISS is slower than CIS as predicted in Table 7.1. Moreover, although the performance of CISS was expected to be worse, note that it is also affected by the two time-consuming operations mentioned above.

Figure 7.7 illustrates the running times measured when verifying the sequence of time-stamps for the initial document (the worst case). These running times have not been corrected with Moore's law. As predicted, CISS is faster than CIS because the hash links can allow for skipping the verification of time-stamps (depending on the hash link, up to three time-stamps can be skipped in the experiment). Also, not only time-stamps are skipped but also their certificate chains, whose verification is quite time-consuming. Moreover, note that the advantage of CISS increases significantly over time. This is because the longer the time-stamp sequence is, the more time-stamp and certificate chains can be skipped.

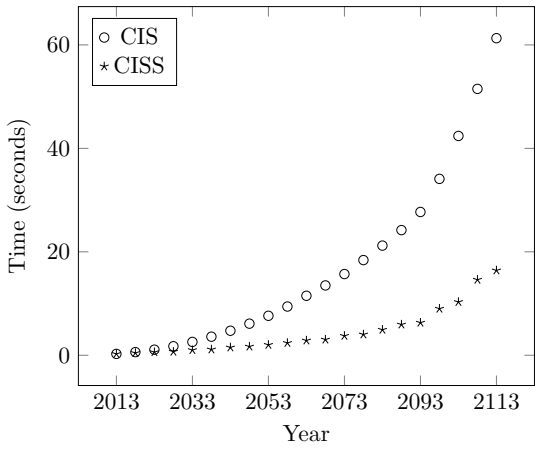

Figure 7.7: The times needed to verify a time-stamp sequence, which is used to protect a set of documents. The verification is done for the first document time-stamped (the worst case).

Finally, Figure 7.8 compares CIS and CISS with respect to their space complexities. As predicted, CIS wins with a small advantage over CISS. Note this advantage is larger after the hash function is replaced in 2086. The reason is that in CISS the authentication paths for 85 documents must be stored. Theses paths are needed when verifying the time-stamp created on 2086, i.e. the time-stamp on Merkle tree root, for the documents received until 2085.

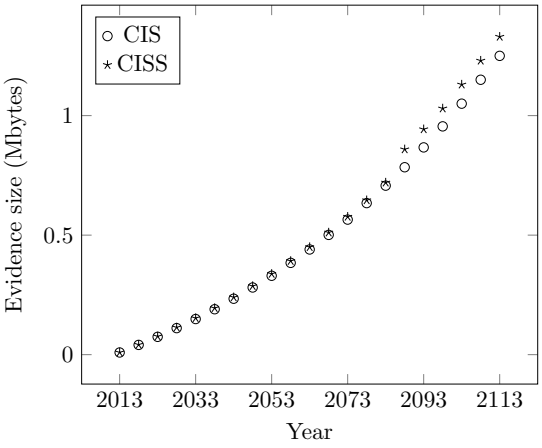

Figure 7.8: The sizes of evidence for a set documents. Every year a new document is added to the set.

8 A new public key certificate

In Chapter 5 we showed that when a retriever verifies the evidence generated by the long-term protection schemes, most of the time is spent on checking certificate chains. Moreover, this time is even longer using the time-stamping schemes, because they accumulate certificate chains when generating evidence. Thus, in this section we first revisit certificate chains, showing why their verification usually leads to performance issues. Next, we describe and analyze a solution for these issues proposed by Silva et al. [45]. Their solution provides a faster verification of time-stamps by replacing the certificate chain needed to verify a time-stamp by a small verification information produced by a root certification authority. It turns out that their solution is not efficient when the number of time-stamps issued by a TSA grows largely. Thus, afterwards we propose a new solution called *re-signed certificates* and provide a corresponding performance analysis confirming its efficiency. The content of this chapter was published as parts of [53].

8.1 Problem specification

When verifying the signatures contained in the evidence of a document, the length of the certificate chains plays an important role. More precisely, a public key is needed to verify a signature on a time-stamp. The link between the public key and the corresponding TSA is provided by a certificate, which we refer to as the *conventional certificate* c_n, $n > 1$. These certificates are signed by CAs which likewise have certificates that bound them to their public keys. This leads to a chain of certificates $c_1, \ldots, c_n$, where for each $1 \leq i < n$, the public key contained in c_i is used to verify the signature on c_{i+1}. The public key required to verify the signature on c_1 is unsigned or self-signed and bound to a special type of certification authorities (CAs) called root CAs. These CAs serve as trust anchor and are assumed to be fully trustworthy. Moreover, each CA provides revocation information for the certificates it issues. This information is usually signed. Since trust anchors are fully trustworthy, no revocation information is needed for them.

Thus, when verifying a time-stamp signature with the TSA public key certificate c_n, a retriever must also check that c_n is valid. More precisely, for each certificate $c \in \{c_1, \ldots, c_n\}$ she checks that c and its verification information contain valid signatures and that c is neither revoked nor expired. Therefore, since the verification of a signature is a time-consuming operation, when the retriever checks the time-stamp signature, she spends most of the time on verifying the validity c_n.

To mitigate this issue, our experiments indicated that longer signature lifetimes can be used to reduce the number of time-stamps. Since fewer time-stamps are used, when verifying them fewer certificate chains are necessary. However, this is only possible to a limited extent for two reasons. First, because chains still contain most of the signatures to be verified. Second, because the lifetime of the keys must be chosen in a way that forgery during their lifetime is unlikely.

Another solution for the same problem was proposed by Silva et al. [45]. It consists of replacing the certificate chains and revocation verification needed to verify time-stamps by a new, smaller verification information. In the next section we describe and analyze their approach.

8.1.1 Re-signed time-stamps

To reduce the long times spent on checking certificates, Silva et al. propose a new way of generating verification information for time-stamps which needs neither certificate chains nor revocation information. To generate such verification information, a root CA is used as follows. First, the root CA receives a time-stamp. Next, the root CA verifies that the signature on this time-stamp is valid. Then, the root CA re-signs the time-stamp, generating a new signature to be used as the verification information for this time-stamp. Thus, since the root CA's public key requires neither a certificate chain nor revocation information, when a retriever uses the new verification information to check the time-stamp, she verifies only the root CA signature.

Next, we detail the protocol and procedures needed to realize the above approach. More precisely, we first make clear the difference between the time-stamps used by Silva et al. and the time-stamps introduced in Section 2.8. Then, we describe how the root CA generates the new verification information for the time-stamps. Afterwards, we show how archivists generate evidence using the new verification information and how retrievers verify evidence. Finally, we analyze this approach and show where performance can be improved.

Signature-based time-stamps

The time-stamps used in this approach are constructed exactly as explained in Section 2.8. The only difference is that they contain additional data. More precisely, a TSA generates a time-stamp T by creating a signature s on $h||y||t$, where h identifies a hash function, $y = h(\cdot)$ is the hash of the object to be time-stamped, and t is the current time. However, T contains not only h, t, and s as before, but also y. As we will see later, including y allows to verify time-stamps without checking signature s.

Preparation

In this protocol, all time-stamps issued by a TSA are sent to a root CA. For performance reasons, the TSA sends only a compact representation (a Merkle tree root) of all time-stamps. The root CA then returns a signature on this compact representation.

Assume a TSA has issued time-stamps $T_1, \ldots, T_n$ using a hash function h. Then, the following steps are executed.

1. The TSA generates a Merkle tree root r from the time-stamps $T_1, \ldots, T_n$, where $h(T_1), \ldots, h(T_n)$ are the leaves of this tree.
2. The TSA generates a signature u on $h||r$.
3. The TSA sends u, h, r, and its certificate c to the root CA.
4. The root CA checks whether u is a valid signature on r using the public key in c. As explained before, this check requires verifying the certification chain from c to the trust anchor.
5. If u is invalid, the root CA returns an error message and the protocol is aborted.
6. The root CA creates a signature σ on $h||r$.
7. The root CA returns σ to the TSA.

This protocol is repeated when the TSA issues further time-stamps.

Evidence generation

We now explain how an archivist generates evidence for the documents in the archive. Assume he has an initialized sequence of time-stamp $T_1, \ldots, T_k$ for a document and the validity of T_k is about to end. Also, assume that the TSA U_k that issued T_k

has already executed the preparation protocol presented above. Thus, the archivist requests the next time-stamp T_{k+1} as follows.

1. The archivist selects the latest used hash function h_k.
2. The archivist obtains from the TSA U_k the authentication path A_k from the time-stamp hash $h_k(T_k)$ to the Merkle tree root r_k, and the root CA signature σ_k on $h_k||r_k$.
3. The archivist selects a hash function h_{k+1} (the selection of h_{k+1} is explained in Section 3.1).
4. The archivist computes the hash $y_{k+1} = h_{k+1}(T_k||A_k||\sigma_k)$ (depending on the time-stamp scheme, he also hashes other objects, e.g. the previous time-stamps).
5. The archivist request a new time-stamp T_{k+1} on $h_{k+1}||y_{k+1}$ from a distinct TSA U_{k+1} at time t_{k+1}.
6. The archivist stores A_k, σ_k, and T_{k+1} together with the document, $T_1, \ldots, T_k$, $A_1, \ldots, A_{k-1}$, and $\sigma_1, \ldots, \sigma_{k-1}$.

Evidence verification

We now explain how evidence is verified. Assume a retriever has the time-stamp sequence $T_1, \ldots, T_k$, the authentication paths $A_1, \ldots, A_{k-1}$, and the root CA signatures $\sigma_1, \ldots, \sigma_{k-1}$ as evidence for a document d. Then, she executes the following steps.

1. Request from the latest used TSA U_k the authentication A_k and the root CA signature σ_k needed to verify the most recent time-stamp T_k.
2. For each $T_i \in \{T_1, \ldots, T_k\}$:
   a) select the hash function h_i from T_i;
   b) compute the time-stamped hash y_i such that $y_i = h_i(d)$ if $i = 1$, otherwise $y_i = h_i(T_{i-1}||A_{i-1}||\sigma_{i-1})$ (see the generation of evidence, step 4);
   c) check that $y_i \in T_i$;
   d) compute the root r_i using the authentication path A_i and the hash $h_i(T_i)$;
   e) verify that the signature σ_i on $h_i||r_i$ is valid at the time $t_{i+1} \in T_{i+1}$.

Finally, note that although the signature s_i contained in time-stamp T_i is not verified, proof of existence, authenticity, and non-repudiation are still guaranteed. The proof of existence of the time-stamp T_{i-1} ($i > 1$) or the document d ($i = 1$) is guaranteed because the retriever computes the hash y_i from T_{i-1} or d (step b) and checks that $y_i \in T_i$ (step c). The authenticity and non-repudiation of T_i are ensured because she computes the Merkle tree root r_i (step d) and checks that the root CA signature σ_i on $h_i||r_i$ is valid (step e).

Performance issues

The presented approach addresses the performance problem of verifying TSA certificate chains because they are replaced by single root CA signatures. However, the approach raises other performance issues. More precisely, since a TSA can issue numerous time-stamps, we identify the following issues that can appear when the number of issued time-stamp grows significantly.

- A TSA needs to recompute the Merkle tree several times as new time-stamps are issued.
- Since root CAs are usually kept off-line, they have to be turned on several times to sign all Merkle tree roots.
- A TSA requires more time to compute a Merkle tree because of the large number of leaves.
- A retriever needs more time to verify the time-stamps because authentication paths become longer.
- More data must be exchanged between archivists, TSAs, and root CAs.

Next, we propose a solution that also addresses the problem of verifying certificate chains, but allows for better performance when the number of time-stamps grows largely.

8.2 Re-signed certificates

Similar to the approach presented before, we propose to use a trust anchor (not necessarily a root CA) to shorten the certificate chains and to eliminate the revocation information needed to verify time-stamps. Next, we first describe our approach and

then we present the protocols needed to realize it. We compare our approach and the previous solution with respect to performance in Section 8.3.

To shorten certificate chains and to eliminate the revocation information, a trust anchor can be used as follows. Assume that a TSA has signed time-stamps and its public key is certified by the certificate c_n in the chain $c_1, \ldots, c_n$, where $n > 1$. Then, before the TSA public key becomes invalid, every time-stamp sequence containing time-stamps from this TSA must be updated. To do this, first, the trust anchor verifies that c_n is valid at the current time by checking the chain and the corresponding revocation information. Afterwards, the trust anchor re-signs c_n, generating the re-signed certificate c'_n. This new certificate constitutes the verification information for any time-stamp issued by that particular TSA and can be added as verification information to the updated time-stamp sequences.

The verification of a time-stamp needs no additional checks in our approach. A retriever verifies the signature on the time-stamp as usual, using the TSA public on the re-signed certificate c'_n. Also, she verifies the trust anchor signature on c'_n. No revocation information is needed for c'_n, because the trust anchor has already checked it.

Next, we present the protocols and procedures needed to realize our approach.

Evidence generation

We now describe how an archivist generates evidence. Assume that he has an initialized time-stamp sequence $T_1, \ldots, T_k$ for a document d in the archive. Thus, when the most recent time-stamp T_k is about to become invalid, the archivist starts the following protocol.

1. He sends the certificate c_k needed to verify signature s_k on time-stamp T_k to the trust anchor.

2. The trust anchor checks whether c_k is valid at the current time. This verification requires checking the certificate chain and corresponding revocation information for c_k.

3. If c_k is invalid, the trust anchor returns an error message and the protocol is aborted.

4. The trust anchor re-signs the content of c_k, generating a new certificate c'_k.

5. The trust anchor returns c'_k.

6. The archivist requests a new time-stamp T_{k+1} on $h_{k+1}||h_{k+1}(T_k||c'_k)$ from a fresh TSA, where h_{k+1} is a hash function and c'_k the verification information for T_k. (The selection of h_{k+1} is explained in Section 3.1.2).

7. The archivist stores T_{k+1} and c'_k together with d, $T_1, \ldots, T_k$, and $c'_1, \ldots, c'_{k-1}$.

Note that the above protocol can be optimized when distinct archivists use time-stamps from the same TSA and when both put trust in the same trust anchor. In this case, only one archivist needs to submit the TSA certificate c_k to the trust anchor. Afterwards, c'_k can be published in a public repository where the next archivists can simply download the re-signed certificate without contacting the trust anchor.

Evidence verification

The verification of evidence remains the same described in Chapters 3 and 7. As we will see next, the difference is that now only for the most recent time-stamp a certificate chain and revocation information is necessary.

Assume a retriever obtains a time-stamp sequence $T_1, \ldots, T_k$ and the re-signed certificates $c'_1, \ldots, c'_{k-1}$ for a document d. Then, she executes the following steps.

1. Collect conventional verification information (i.e. certificate chain and revocation status) required to verify the most recent time-stamp T_k.

2. Verify that the signature on T_k is valid at the current time using the collected verification information.

3. For time-stamp $T_i \in \{T_1, \ldots, T_{k-1}\}$:

   a) verify that the signature on T_i is valid at time t_{i+1} using the public key in the re-signed TSA certificate c'_i;

   b) verify that c'_i is a valid certificate at time t_{i+1} using the trust anchor public key (no revocation information is checked).

Note that, contrary to the previous solution, we do not restrict our approach to root CAs. This is because in some contexts, trust anchor such as notaries could be employed. Nevertheless, we suggest selecting as trust anchor the same trust anchor needed to verify the TSA's conventional certificate (i.e. the root CA). The reason is that, since anchors are fully trustworthy but can be compromised, reducing the number of anchors is desired.

Another difference from the previous solution is that the number of time-stamps a TSA issues does not affect the effort to generate the re-signed certificate c'_n. Moreover, this certificate needs to be issued only once for each TSA, when its conventional certificate is about to expire. Note that this is the time when archivists must update the time-stamp sequences. In the next section we provide a more detailed performance analysis.

8.3 Performance analysis

This section provides a performance analysis where we compare the approaches re-signed certificates, re-signed time-stamps, and conventional certificates. As performance, we analyze the time and the communication complexities required to generate verification information for time-stamps. Moreover, we also analyze the times needed to verify time-stamps.

As in Chapter 5, we measure time and communication complexity analytically. More precisely, we approximate time by counting the signatures to be verified and created. If there are objects to be hashed, they are also counted. To quantify communication, we count the objects exchanged between the involved parties.

To make the approaches comparable, we assume the following.

- Conventional certificate chains contain $k > 1$ certificates.
- A revocation information is signed and required for each conventional certificate.
- A TSA issues $n = 2^l$ time-stamps during its lifetime, where l is a positive integer.

We next present our comparisons as follows. First, we compare the approaches as to the times TSAs and trust anchors need to generate verification information for time-stamps. The next comparison is with respect to times retrievers use to verify time-stamps. Finally, we compare the approaches with respect to their communication complexity.

Performance of TSAs Table 8.1 presents the numbers of objects and signatures a TSA must hash and create, respectively, when generating evidence for n time-stamps. Since in the approach using conventional certificates the TSAs do not generate verification information for their time-stamps, there are neither hashed objects nor verified signatures.

Table 8.1: The objects and signatures a TSA must hash and create, respectively, when generating verification information for $n = 2^l$ time-stamps.

Approach	Hashed objects		Created signatures
	Time-stamps	Hashes	
Conventional certificates	0	0	0
Re-signed time-stamps	n	$2n - 2$	1
Re-signed certificates	0	0	0

The table shows that when using re-signed time-stamps, the performance of the TSA depends on the number n of time-stamps the TSA has issued. This is because the TSA builds a Merkle tree from the n time-stamps and signs the root. In contrast, when using conventional and re-signed certificates, no operation is required from the TSA. Therefore, they allow for a better performance than when using re-signed time-stamps.

Performance of trust anchors Table 8.2 compares the approaches as to the performance of the used trust anchor. The performance is measured by counting the signatures to be verified and created when generating evidence for time-stamps. Since trust anchors generate no verification information using conventional certificates, signatures are neither verified nor created.

Table 8.2: The numbers of signatures a trust anchor needs to verify and create when it is used to generate verification information with neither certificate chains nor revocation information.

Approach	Verified signatures	Created signatures
Conventional certificates	0	0
Re-signed time-stamps	$1 + 2k$	1
Re-signed certificates	$2k$	1

From Table 8.2 we can see that the re-signed time-stamps and certificates are inferior to conventional certificates. However, re-signed certificates still allow for a better performance than re-signed time-stamps. The reason is that when using re-signed time-stamps the trust anchor not only checks the TSA certificate chain of length k, but also the signature on a Merkle tree root.

Performance of retrievers We analyze the performance of retrievers while they verify a single time-stamp. Their performance is approximated by the signatures and the objects to be verified and hashed in Tables 8.3 and 8.4, respectively. Since the number of hashed objects can differ from one protection scheme to another, we assume that the time-stamp to be verified was applied on the previous time-stamp in a time-stamp sequence together with the corresponding verification information.

Table 8.3: The numbers of signatures to be checked when verifying one time-stamp.

Approach	Signatures
Conventional certificates	$1 + 2k$
Re-signed time-stamps	1
Re-signed certificates	2

Table 8.4: The numbers of objects to be hashed when verifying one time-stamp.

Approach	Certificates	Revocation info.	Hashes	Signatures
Conventional certificates	k	k	0	0
Re-signed time-stamps	0	0	$2l$	1
Re-signed certificates	1	0	0	0

The tables show that the re-signed time-stamps and certificates are superior to conventional certificates. This is because, when using conventional certificates, retrievers are required to verify the TSA certificate chain of length k, whereas in the other two approaches this chain is not checked.

When comparing re-signed certificates and times-stamps, we need to take into account the number $n = 2^l$ of time-stamps a TSA issues during its lifetime. For a small n, the re-signed time-stamps are expected to be faster since they require checking one signature less. However, for a large n this advantage can be compensated by the time required to reconstruct the Merkle tree root from the authentication path of length l.

Communication complexity Table 8.5 presents the communication complexity for each approach. More precisely, we count the objects archivists, a TSA, and a trust anchor need to receive when generating $n = 2^l$ time-stamps and the corresponding verification information. For simplicity, we assume the trust anchor signs a Merkle tree root and the TSA certificate only once.

To compare the schemes in the table, we distinguish between small and large numbers $n = 2^l$ of time-stamps. For a small n, we count the number of large objects

Table 8.5: The numbers of objects exchanged between archivists, a TSA, and a trust anchor.

Approach	Certificates	Rev. info.	Hashes	Signatures
Conventional certificates	nk	nk	n	0
Re-signed time-stamps	k	k	$n + nl + 1$	$n + 2$
Re-signed certificates	$n + k$	k	n	0

(certificates and revocation information). We can see that re-signed time-stamps are the best approach (lowest communication complexity), followed by re-signed certificates and conventional certificates.

For a large n, we analyze the asymptotic behavior of the expressions. We can see that the communication complexity is proportional to nk in the conventional certificates, k and $nl = n \log_2(n)$ in the re-signed time-stamps, and n and k in re-signed certificates. Therefore, when n goes to infinity, re-signed certificates is the best approach, followed by conventional certificates and re-signed time-stamps.

9 Conclusions and future work

In this work we analyzed and improved solutions that provide one or more of the protection goals integrity, authenticity, non-repudiation, and proof of existence for archived documents. First, we surveyed the solutions that can provide protection as long as the documents remain in the archive. Since archival periods can be longer than the lifetime of cryptographic primitives, a requirement for these schemes is addressing the aging of cryptography properly. Interestingly, only few solutions fulfill this requirement. They are: the time-stamping schemes *Advanced Electronic Signatures* (AdES), *Content Integrity Service* (CIS), and *Evidence Record Syntax* (ERS); the notarial scheme *Cumulative Notarizations* (CN); and the replication scheme *Lots of Copies Keep Stuff Safe* (LOCKSS).

Next, we analyzed the above protection schemes as to trustworthiness and performance. The trustworthiness analysis was divided into two parts. In the first part, we analyzed trustworthiness qualitatively by evaluating the required trust assumptions. We showed that the time-stamping and notarial schemes are expected to be more trustworthy than LOCKSS because they provide evidence that can be used to verify the protection goals.

In the second part of our analysis, we showed how to use *a belief trust model* to approximate the trustworthiness of evidence generated by time-stamping and notarial schemes. More precisely, we used the trust model to compute the *trust* a user puts in the evidence given the reputation of the parties involved in the creation of this evidence. Since the reputation of these parties is needed, we also designed a reputation system. In this reputation system, participants verify evidence and use the verification results to evaluate the involved parties. Namely, notaries, time-stamp authorities, and certificate authorities.

The analysis of the trust model and reputation system led to two important conclusions. First, when a document is archived for several years and many parties are involved in generating evidence for this document, the trustworthiness of the evidence tends to degrade. Therefore, as important as addressing the aging of cryptography is coping with the decay of the trustworthiness of evidence. Second,

a solution to mitigate this decay is to raise the trustworthiness of involved parties. To this end, the reputation system can be used to realize incentives for these parties to build good reputation, and, therefore, become more trustworthy.

Afterwards, we evaluated the performance of the time-stamping and notarial schemes as to time, space, and communication complexities. This evaluation contained an analytical evaluation and experiments. In the analytical evaluation, we evaluated performance without taking specific cryptographic primitives into account. By contrast, in the experiments we analyzed the performance of prototypical implementations for the schemes using practical (state of the art) cryptographic primitives.

The experiments confirmed the analytical evaluation, although some deviations were expected. This is because the analytical evaluation does not consider all operations carried out by the prototypical implementations, such as the overhead of parsing evidence files. Moreover, the use of XML also impacts on the performance of the implementations, because they have to translate evidence from the text format (i.e. ASCII) to the format required as input for the cryptographic primitives.

The performance analysis showed that the notarial scheme CN is the fastest scheme, at the cost of the highest communication complexity. Next comes the time-stamping schemes ERS, followed by AdES and CIS. The analysis also demonstrated that the verification of certificate chains is the most time-consuming operation when verifying evidence. Furthermore, since the time-stamping schemes accumulate certificate chains, the experiments showed that longer signature lifetimes can be used to reduce the number of chains and improve verification times. In contrast, the notarial scheme does not benefit from this solution because it accumulates no chains.

Based on these findings, we then designed three solutions to improve the performance of notarial and time-stamping schemes. The first solution is named *Attested Certificates* (AC) and is based on the notarial scheme CN. AC requires significantly less space and communication than CN. Furthermore, both schemes are comparable with respect to time and trustworthiness.

The second solution is called *Content Integrity Service with Skip Lists* (CISS) and is based on the time-stamping scheme CIS. CIS is useful for archives where a set of documents uses a single time-stamp sequence as evidence for these documents. However, contrary to ERS, CIS allows these documents to be submitted to the archive sequentially and not as a set. But when submitting documents sequentially, several time-stamps may be created such that their validities overlap. We addressed this issue in CISS by using a data structure called *skip list* that reduces the number of time-stamps that must be checked during verification. Furthermore, CISS hashes less data than CIS when generating or verifying time-stamps used to address the

aging of cryptography.

To compare CIS and CISS, we provided a performance analysis. It showed that CISS is significantly faster than CIS when new documents are submitted and time-stamped on a regular basis. However, although CISS hashes less data than CIS when addressing the aging of cryptography, this led to no improvements with respect to the times required to generate and verify evidence. The reason seems to be that the differences between CIS and CISS in the hashing times are compensated by the extra overhead of more complex evidence files used for CISS. However, when protecting archived documents of larger sizes, the advantage of CISS in hashing is expected to magnify.

Finally, we designed a public key infrastructure-based solution to reduce the verification time for long certificate chains in time-stamping schemes. More precisely, we proposed to use a trust anchor to verify and attest the validity of time-stamp authority certificates. Compared with other solutions for this issue, our approach reduces the effort of a time-stamp authority when it issues numerous time-stamps during its lifetime.

9.1 Future work

In this work we analyzed the trustworthiness of long-term protection schemes qualitatively and quantitatively. In the qualitative analysis, we compared the required trust assumptions. However, not all trust assumptions evaluated are well understood yet. For instance, *widely visible media*, where it is assumed that public records are not corrupted and that they contain the dates when they were created. It has often been suggested to employ widely visible media for time-stamping, and some time-stamp authorities already do so by using newspapers. However, more work is necessary to evaluate whether users in the far future will trust this approach. This is because users may be unable to assess the integrity and proof of existence of old newspapers by themselves. In this case, it is unclear whether they will trust the archivist who preserves the newspapers.

To analyze trustworthiness quantitatively, we proposed to use a belief trust model together with a reputation system. This approach can be further extended. With respect to applying the trust model, it can be used to analyze the trustworthiness of more complex evidence. For example, when creating two redundant time-stamp sequences for the same document such that, if only one sequence is compromised, then the protection goals are not lost. As to the reputation system, it can also be extended to provide the reputation of the involved archivists, since they participate

in the generation of evidence.

The performance of the reputation system could be analyzed and improved. One interesting approach to improve performance is to use a compact representation of evidence instead of the evidence itself when submitting or requesting reputation in the system. This approach would require developing methods for the reputation system that provide long-term proof of existence for its internal data.

The migration of evidence between protection schemes raises interesting questions. More precisely, can an archivist change the used protection scheme after evidence was initialized? If the migration is between notarial and time-stamping schemes, how is trust assessed after migration?

Cloud and mobile computing are important technologies that could also be used for digital archiving. This leads to new performance questions. For example, can a document be replicated among several cloud providers whereas the evidence is generated and stored by a single provider? A challenge would be to design a long-term secure link between the replicas and the evidence. As to mobile computing and its performance constraints, can the verification of evidence be optimized, e.g. by outsourcing the verification with the help of verifiable computing?

The time-stamps presented in this work are based on calendar dates. Such time-stamps are used to determine when a document existed and whether its evidence was renewed timely, i.e. before the used cryptography became insecure. By contrast, time-stamps using relative dates can only establish whether one bit string existed before another bit string. Therefore, further research is needed to show how to use these time-stamps in the long term such that they also allow retrievers to check whether the aging of the used cryptography has been addressed timely.

Replacing the used hash function is a time-consuming operation in all presented schemes, because documents and their entire evidence need to be re-hashed. A challenging question is whether the replacement of hash functions in protection schemes can be done more efficiently.

The new time-stamping scheme Content Integrity Service with Skip Lists (CISS) allows for verifying the protection goals of one document from a set of time-stamped documents. More precisely, during the verification a retriever hashes no documents other than the document to be checked and verifies a minimal number of time-stamps. However, in some scenarios using evidence that requires hashing the whole document set may be useful to, say, guarantee that the archivist has not deleted any document. Thus, extending CISS to allow for verifying documents individually or as a set is a desirable feature.

The protection schemes presented in this work achieve the protection goals integrity, authenticity, non-repudiation, and proof of existence for documents in digi-

tal archives. More research is needed to evaluate how these schemes perform when combined with solutions that provide long-term confidentiality. This is necessary, for example, for digital archives storing sensitive documents such as testaments and health records. Moreover, designing a trust model that also approximates the trustworthiness of long-term confidentiality is to the best of our knowledge an open question.

Furthermore, realizing confidentiality requires an identity management system. To the best of our knowledge, designing such system with long-term capabilities is still a challenge. Also, an interesting question is how to deal with the confidentiality of a document when its owner passed away.

Finally, the presented solutions can further be analyzed and extended as to the life cycle of documents. For example, documents may require format migration and even destruction. Therefore, more research is needed to allow for generating evidence showing that the semantics of a document is preserved after format migration. As to the destruction of documents, schemes such as ERS, CIS, and CISS need to be extended to allow for the removal of documents and, optionally, their complete evidence from the archive.

Bibliography

[1] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro, editors, *Sequences II*, pages 329–334. Springer New York, 1993. Cited on pages 12 and 20.

[2] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Sponge functions. In *ECRYPT hash workshop*, 2007. Cited on page 49.

[3] Aleksej Jerman Blazic, Svetlana Saljic, and Tobias Gondrom. Extensible markup language evidence record syntax (xmlers). RFC 6283, July 2011. URL `https://tools.ietf.org/html/rfc6283`. Cited on pages 23 and 99.

[4] Johannes Braun, Johannes Buchmann, Ciaran Mullan, and Alexander Wiesmaier. Long term confidentiality: a survey. *Designs, Codes and Cryptography*, 71(3):459–478, 2014. Cited on page 3.

[5] Johannes Braun, Florian Volk, Jiska Classen, Johannes Buchmann, and Max Mühlhäuser. CA trust management for the web PKI. *IOS Press: Journal of Computer Security, 2014*, June 2014. Cited on pages 37 and 45.

[6] Johannes Buchmann. *Introduction to Cryptography*. Springer, 2002. Cited on pages 8 and 9.

[7] Johannes A. Buchmann, Evangelos Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer Berlin Heidelberg, 2013. Cited on pages 10 and 11.

[8] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen. *Bekanntmachung zur elektronischen Signatur nach dem Signaturgesetz und der Signaturverordnung (Übersicht über geeignete Algorithmen)*. 2015. URL `https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/ElekSignatur/Algorithmenkatalog_Entwurf_2015.pdf`. Cited on page 13.

[9] Judith Burns and Kara Scannell. Broadcom, sec settle backdating case, 2008. URL `http://www.wsj.com/articles/SB120890020229436085`. Cited on page 2.

[10] Stefania Cavallar, Bruce Dodson, Arjen K. Lenstra, Walter Lioen, Peter L. Montgomery, Brian Murphy, Herman te Riele, Karen Aardal, Jeff Gilchrist, Gérard Guillerm, Paul Leyland, Jöel Marchand, François Morain, Alec Muffett, ChrisandCraig Putnam, and Paul Zimmermann. Factorization of a 512-bit rsa modulus. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, volume 1807 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2000. Cited on page 2.

[11] S. Chokhani, W. Ford, R. Sabett, C. Merrill, and S. Wu. Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework. RFC 3647 (Informational), November 2003. URL `http://www.ietf.org/rfc/rfc3647.txt`. Cited on page 30.

[12] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard), May 2008. URL `http://www.ietf.org/rfc/rfc5280.txt`. Cited on page 11.

[13] Department of Health and The Rt Hon Jeremy Hunt. Nhs challenged to go paperless by 2018, 2013. URL `https://www.gov.uk/government/news/jeremy-hunt-challenges-nhs-to-go-paperless-by-2018--2`. Cited on page 1.

[14] DH/Digital Information Policy. Records management: Nhs code of practice (2nd edition), 2009. URL `http://systems.hscic.gov.uk/infogov/links/recordscop2.pdf`. Cited on page 1.

[15] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246 (Proposed Standard), January 1999. URL `http://www.ietf.org/rfc/rfc2246.txt`. Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507. Cited on page 45.

[16] Luís Felipe dos Santos Gonçalves. Xades4j: a java library for xades signature services, 2014. URL `https://code.google.com/p/xades4j/`. Cited on page 54.

[17] ETSI. *CMS Advanced Electronic Signatures (CAdES)*. Number TS 101 733. 1.7.4 edition, Jul 2010. Cited on page 21.

[18] European Telecommunications Standards Institute. *Electronic Signatures and Infrastructures (ESI); XML Advanced Electronic Signatures (XAdES)*. Number TS 101 903. 1.4.2 edition, dec 2010. URL `http://www.etsi.org/deliver/etsi_ts/101900_101999/101903/01.04.02_60/ts_101903v010402p.pdf`. Cited on page 21.

[19] Diego Gambetta. Can we trust trust? In Diego Gambetta, editor, *Trust: Making and Breaking Cooperative Relations*, pages 213–237. Blackwell, 1988. Cited on page 14.

[20] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Comput.*, 17 (2):281–308, 1988. doi: 10.1137/0217017. URL `http://dx.doi.org/10.1137/0217017`. Cited on page 13.

[21] T Gondrom, R Brandner, and U Pordesch. Evidence Record Syntax (ERS), 2007. URL `http://www.ietf.org/rfc/rfc4998.txt`. Cited on page 23.

[22] Simon Goodley. Property title fraud costs land registry £26m in compensation, 2011. URL `http://www.theguardian.com/money/2011/may/15/land-registry-title-fraud-compensation`. Cited on page 2.

[23] Stuart Haber and Pandurang Kamat. Content integrity service for long-term digital archives. In *Archiving Conference*, pages 159–164. Society for Imaging Science and Technology, 2006. Cited on page 22.

[24] James Heather and David Lundin. The append-only web bulletin board. In Pierpaolo Degano, Joshua Guttman, and Fabio Martinelli, editors, *Formal Aspects in Security and Trust*, volume 5491 of *Lecture Notes in Computer Science*, pages 242–256. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01464-2. doi: 10.1007/978-3-642-01465-9_16. URL `http://dx.doi.org/10.1007/978-3-642-01465-9_16`. Cited on page 40.

[25] Audun Jøsang, Ross Hayward, and Simon Pope. Trust network analysis with subjective logic. In *Computer Science 2006, Twenty-Nineth Australasian Computer Science Conference (ACSC2006)*, pages 85–94, 2006. Cited on pages 15 and 16.

[26] Dorel Kiik. Personal communication, mai 2014. Cited on page 1.

[27] David King. *The Commissar Vanishes: The Falsification of Photographs and Art in Stalin's Russia*. Metropolitan Books, 1997. Cited on page 2.

[28] Donald Ervin Knuth. *The art of computer programming*. Pearson Education, 2005. Cited on page 28.

[29] Legion of the Bouncy Castle Inc. The legion of the bouncy castle java cryptography apis, 2014. URL `https://www.bouncycastle.org`. Cited on page 54.

[30] Dimitrios Lekkas and Dimitris Gritzalis. Cumulative notarization for long-term preservation of digital signatures. *Computers & Security*, 23(5):413–424, 2004. Cited on page 25.

[31] Arjen K. Lenstra. Key lengths. In Hossein Bidgoli, editor, *Handbook of information security*, volume 2, pages 617–635. John Wiley, Hoboken, N.J, 2006. Cited on pages 13, 55, 61, 82, and 105.

[32] Arjen K. Lenstra, James P. Hughes, Maxime Augier, Joppe W. Bos, Thorsten Kleinjung, and Christophe Wachter. Public keys. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 626–642. Springer Berlin Heidelberg, 2012. Cited on page 9.

[33] Feng Luan, Mads Nygård, Lars Gaustad, and Gustavsen Inger-Mette. The challenges of migration as a long-term preservation strategy: The findings of team norway and longrec. In *InterPARES 3 International Symposium*, pages 279–300, jun 2009. Cited on page 1.

[34] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *11th USENIX Security Symposium*, pages 297–312, 2002. Cited on pages 11, 18, and 91.

[35] Petros Maniatis, Mema Roussopoulos, Thomas J. Giuli, David S. H. Rosenthal, and Mary Baker. The lockss peer-to-peer digital preservation system. *ACM Transactions on Computer Systems (TOCS)*, 23(1):2–50, 2005. Cited on page 26.

[36] D Harrison McKnight and Norman L Chervany. The meanings of trust. Technical Report MISRC Working Paper Series 96-04, University of Minnesota, 1996. Cited on page 14.

[37] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89*, volume 435 of *Lecture Notes in Computer Science*, pages 218–238. Springer New York, 1990. Cited on page 8.

[38] Gordon E. Moore. Cramming more components onto integrated circuits. *Proceedings of the IEEE*, 86(1):82–85, Jan 1998. Cited on page 13.

[39] National Institute of Standards and Technology. *Recommendation for Key Management – Part 1: General (Revised)*. 2007. URL `http://csrc.nist.gov/groups/ST/toolkit/documents/SP800-57Part1_3-8-07.pdf`. Cited on page 13.

[40] National Institute of Standards and Technology. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions, 2014. Cited on page 8.

[41] Alina Oprea and Kevin D. Bowers. Authentic time-stamps for archival storage. In *14th European Symposium on Research in Computer Security (ESORICS'09)*, pages 136–151, 2009. Cited on page 28.

[42] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Algorithms and Data Structures, Workshop WADS '89*, pages 437–449, 1989. Cited on page 16.

[43] Sebastian Ries. Certain trust: a trust model for users and agents. In *2007 ACM Symposium on Applied Computing*, pages 1599–1604, 2007. Cited on page 15.

[44] Jordan Robertson. Digital health records' risks emerge as deaths blamed on systems, 2013. URL `http://www.bloomberg.com/news/2013-06-25/digital-health-records-risks-emerge-as-deaths-blamed-on-systems.html`. Cited on page 1.

[45] Nelson da Silva, Thiago Acórdi Ramos, and Ricardo Custódio. Carimbos do tempo autenticados para a preservação por longo prazo de assinaturas digitais. In *XI Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 57–70, 2011. Cited on pages 5, 109, and 110.

[46] Sangchul Song and Joseph JáJá. Techniques to audit and certify the long-term integrity of digital archives. *International Journal on Digital Libraries*, 10(2–3): 123–131, 2009. Cited on page 28.

[47] Richard Stiennon. Trusted third party time stamps, October 2006. URL `http://www.zdnet.com/article/trusted-third-party-time-stamps`. Cited on page 13.

[48] Surety, LLC. Ensuring Record Integrity with Absolute Proof^{SM}. Technical whitepaper, 2003. Cited on page 12.

[49] Revenue Irish Tax and Customs. Annual report 2013, 2014. URL `http://www.revenue.ie/images/ar_13_en/annual_report_summary.pdf`. Cited on page 1.

[50] Vernon Turner, John F. Gantz, David Reinsel, and Stephen Minton. The digital universe of opportunities: Rich data and the increasing value of the internet of things. Technical Report #IDC_1672, 2014. URL `http://idcdocserv.com/1678`. Cited on page 1.

[51] Moshe Y. Vardi. Moore's law and the sand-heap paradox. *Communications of the ACM*, 57(5):5–5, May 2014. ISSN 0001-0782. Cited on page 55.

[52] Victoria - Public Record Office. *Victorian Electronic Records Strategy Final Report*. 4 edition, 1998. ISBN 0731155203. Cited on page 27.

[53] Martín Vigil and Ricardo Custódio. Cleaning up the PKI for Long-Term Signatures. In *XII Simpósio Brasileiro de Segurança da Informação e de Sistemas Computacionais*, pages 140–153, 2012. Cited on page 109.

[54] Martín Vigil, Cristian Thiago Moecke, Ricardo Felipe Custódio, and Melanie Volkamer. The notary based PKI - A lightweight PKI for long-term signatures on documents. In *Public Key Infrastructures, Services and Applications - 9th European Workshop, EuroPKI 2012, Pisa, Italy, September 13-14, 2012, Revised Selected Papers*, pages 85–97, 2012. Cited on page 68.

[55] Martín Vigil, Daniel Cabarcas, Johannes Buchmann, and Jingwei Huang. Assessing trust in the long-term protection of documents. In *2013 IEEE Symposium on Computers and Communications (ISCC 2013)*, pages 185–191, 2013. Cited on pages 29 and 68.

[56] Martín Vigil, Christian Weinert, Kjell Braden, Denise Demirel, and Johannes Buchmann. A performance analysis of long-term archiving techniques. In *2014 IEEE International Conference on High Performance Computing and Communications, 6th IEEE International Symposium on Cyberspace Safety and Security, 11th IEEE International Conference on Embedded Software and Systems*, pages 878–889, 2014. Cited on page 47.

[57] Martín Vigil, Christian Weinert, Denise Demirel, and Johannes Buchmann. An efficient time-stamping solution for long-term digital archiving. In *IEEE*

33rd International Performance Computing and Communications Conference (IPCCC 2014), pages 1–8, 2014. Cited on page 89.

[58] Martín Vigil, Johannes Buchmann, Daniel Cabarcas, Christian Weinert, and Alexander Wiesmaier. Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. *Computers & Security*, 50(0): 16–32, 2015. Cited on pages 19, 29, and 47.

[59] Andrew Waugh, Ross Wilkinson, Brendan Hills, and Jon Dell'oro. Preserving digital information forever. In *Fifth ACM Conference on Digital Libraries*, pages 175–184, 2000. Cited on page 27.

[60] Ahmad Samer Wazan, Romain Laborde, François Barrère, and Abdelmalek Benzekri. The X.509 trust model needs a technical and legal expert. In *IEEE International Conference on Communications (ICC 2012)*, pages 6895–6900, 2012. Cited on pages 37 and 45.

[61] Adam Wierzbicki. *Trust and Fairness in Open, Distributed Systems*, volume 298 of *Studies in Computational Intelligence*. Springer, 2010. Cited on page 14.

[62] Aydan R. Yumerefendi and Jeffrey S. Chase. Strong accountability for network storage. *ACM Transactions on Storage (TOS)*, 3(3), 2007. Cited on page 27.

[63] Philip R. Zimmermann. *The Official PGP User's Guide*. MIT Press, Cambridge, MA, USA, 1995. ISBN 0-262-74017-6. Cited on pages 10 and 45.

Wissenschaftlicher Werdegang

April 2011 - Juli 2015 Wissenschaftlicher Mitarbeiter und Doktorand in der Arbeitsgruppe von Professor Johannes Buchmann, Fachbereich Informatik, Fachgebiet Theoretische Informatik – Kryptographie und Computeralgebra an der Technischen Universität Darmstadt.

September 2007 - August 2010 Studium der Informatik (M.Sc) und wissenschaftlicher Mitarbeiter in der Arbeitsgruppe von Professor Ricardo Custódio, Fachbereich Informatik, Fachgebiet Computersicherheit an der Universidade Federal de Santa Catarina, Brasilien.

Januar 2003 - Juli 2007 Studium der Informatik (B.Sc) an der Universidade Federal de Santa Catarina, Brasilien.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Darmstadt, Juli 2015 _____