Long-Term Protection of Integrity and Confidentiality

Security Foundations and System Constructions

Vom Fachbereich Informatik der
Technischen Universität Darmstadt genehmigte

Dissertation

zur Erlangung des Grades
Doktor rerum naturalium (Dr. rer. nat.)
von

Matthias Carl Geihs

geboren in Heppenheim an der Bergstraße.

Referenten: Prof. Dr. Johannes Buchmann
Prof. Dr. Ahto Buldas

Darmstadt, Germany, 2018

Dissertation von Matthias Geihs:

Long-Term Protection of Integrity and Confidentiality – Security Foundations and System Constructions

Technische Universität Darmstadt, Darmstadt, Germany
Tag der mündlichen Prüfung: 12.09.2018
Jahr der Veröffentlichung: 2018

Abstract

Huge amounts of information today are stored digitally and a significant amount of this information (e.g., health records) must be kept unaltered and confidential over long periods of time (i.e., decades or centuries). Consequently, there is a high demand for protection schemes that can ensure integrity and confidentiality over such long time periods. The cryptographic schemes used today for protecting integrity and confidentiality (e.g., RSA signatures and AES encryption), however, are not designed to provide long-term protection as their security relies on computational assumptions (e.g., that factoring large integers is infeasible) and trust assumptions (e.g., that a secret key is not compromised) which cannot be guaranteed over such long time periods. To achieve long-term integrity protection Bayer, Haber, and Stornetta proposed a method for prolonging the validity of digital signatures by using cryptographic timestamping. The security of this method, however, is unclear as no precise security analysis has been performed. To achieve long-term confidentiality protection there exist information-theoretically secure schemes (e.g., Quantum Key Distribution, One-Time-Pad Encryption, or Secret Sharing) whose security does not depend on computational assumptions. However, so far it is unclear whether information-theoretic confidentiality protection can be combined with prolongable integrity protection.

This thesis answers both of these research questions. In the first part, we develop the first formal security models and proofs for several long-term integrity protection schemes that are derived from the ideas of Bayer, Haber, and Stornetta. We first develop a novel computational model that captures long-lived adversaries whose computational power increases over time. Then, using this model, we show that signature-based long-term integrity protection can be constructed from short-term unforgeable signature schemes and that hash-based long-term integrity protection can be constructed from short-term preimage-aware hash functions. We also propose a new cryptographic primitive called long-term commitment, which is crucial for the second part of this thesis. In the second part we then present the first storage system that combines information-theoretic confidentiality protection with prolongable integrity protection. We also propose two extensions of this system, where the first enables long-term access pattern hiding security (i.e., it remains secret which data items are accessed by the user at which times) and the second improves the efficiency when storing large complex datasets.

Publications

Publications used in this thesis

[G1] **Matthias Geihs**, Denise Demirel, Johannes Buchmann: *A Security Analysis of Techniques for Long-term Integrity Protection*, Conference on Privacy, Security and Trust (PST), 2016.

[G2] Johannes Braun, Johannes Buchmann, Denise Demirel, **Matthias Geihs**, Mikio Fujiwara, Shiho Moriai, Masahide Sasaki, Atsushi Waseda: *LINCOS: A Storage System Providing Long-Term Integrity, Authenticity, and Confidentiality*, ACM Asia Conference on Computer and Communications Security (AsiaCCS), 2017.

[G3] Ahto Buldas, **Matthias Geihs**, Johannes Buchmann: *Long-Term Secure Commitments via Extractable-Binding Commitments*, Australasian Conference on Information Security and Privacy (ACISP), 2017.

[G4] Ahto Buldas, **Matthias Geihs**, Johannes Buchmann: *Long-Term Secure Time-Stamping under the Preimage Awareness Assumption*, International Conference on Provable Security (ProvSec), 2017.

[G5] **Matthias Geihs**, Nikolaos Karvelas, Stefan Katzenbeisser, Johannes Buchmann: *PROPYLA: Privacy Preserving Long-Term Secure Storage*, International Workshop on Security in Cloud Computing (SCC@AsiaCCS), 2018.

[G6] **Matthias Geihs**, Johannes Buchmann: *ELSA: Efficient Long-Term Secure Storage of Large Datasets*, to be submitted to International Conference on Information Security and Cryptology (ICISC), 2018.

Other publications

[G7] **Matthias Geihs**, Daniel Cabarcas: *Efficient Integer Encoding for Homomorphic Encryption via Ring Isomorphisms*, International Conference on Cryptology and Information Security in Latin America (LATINCRYPT), 2014.

[G8] **Matthias Geihs**, Denise Demirel, Johannes Buchmann: *On the Security of Long-Lived Archiving Systems Based on the Evidence Record Syntax*, International Conference on Codes, Cryptology, and Information Security (C2SI), 2015.

[G9] Christian Weinert, Denise Demirel, Martin Vigil, **Matthias Geihs**, Johannes Buchmann: *MOPS: A Modular Protection Scheme for Long-Term Storage*, ACM Asia Conference on Computer and Communications Security (AsiaCCS), 2017.

[G10] Johannes Buchmann, **Matthias Geihs**, Kay Hamacher, Stefan Katzenbeisser, Sebastian Stammler: *Long-Term Integrity Protection of Genomic Data*, International Workshop on Genome Privacy and Security (GenoPri), 2017.

[G11] Rachid El Bansarkhani, **Matthias Geihs**, Johannes Buchmann: *PQChain: An Authenticated Blockchain Protocol based on Post-Quantum Assumptions*, to appear in IEEE Journal on Security & Privacy – Special Issue on Blockchain, 2018.

[G12] **Matthias Geihs**, Oleg Nikiforov, Denise Demirel, Alexander Sauer, Denis Butin, Felix Günther, Gernot Alber, Thomas Walther, Johannes Buchmann: *The Status of Quantum-Based Long-Term Secure Communication over the Internet*, preprint available at arXiv/1711.09793, submitted to IEEE Transactions on Sustainable Computing – Special Issue on Sustainable Information Security and Forensic Computing, 2018.

Contents

Introduction

1. Introduction

Today, huge amounts of information are generated, exchanged, and stored digitally. A significant amount of this information is *long-lived* and *sensitive* (e.g., health records, governmental documents, enterprise documents, land registries, tax declarations) and requires protection of integrity and confidentiality for decades or even centuries.

1.1. Background

Digitally stored information today is commonly protected using digital signature schemes (e.g., RSA [RSA78]) to ensure integrity and authenticity, and encryption schemes (e.g., AES [Nat01]) to ensure confidentiality (see also [Buc16]). The commonly used signature and encryption schemes rely on computational assumptions to be secure. For example, many schemes rely on the assumption that finding the prime factors of large integers requires an infeasible amount of computing power using current computing technology or an infeasible amount of time. Such a computational assumption crucially depends on the exact meaning of *large* and *infeasible*. In fact, as computing technology is continuously being improved, what appears computationally infeasible today, may appear feasible at some point in the future. Such developments have severe consequences for the security of cryptographic schemes that rely on computational assumptions. Historically, we observe that, for the mentioned reasons, DES encryption [Nat77] was replaced by AES encryption [Nat01] and SHA-1 hashing [Nat95] was replaced by SHA-2 hashing [Nat02]. Moreover, we know that the Diffie-Hellman Key Exchange Protocol [DH76] and the RSA Signature Scheme [RSA78], which are allegedly the most widely used cryptographic schemes today, are prone to attacks based on quantum computing [Sho99] and will need to be replaced by quantum-secure alternatives before sufficiently powerful quantum computers are available (cf. [Nat17]). In summary, we observe that the commonly used *computationally secure* cryptographic schemes have a limited lifetime and appear insufficient to provide long-term protection (over decades or centuries).

A number of schemes have been proposed that mitigate the risks described above. In the early 1990s, Bayer, Haber, and Stornetta proposed a method for prolonging the validity of a digital signature beyond the validity of the corresponding digital signature scheme [BHS93]. Their method uses digital time-stamping and is the basis for several long-term integrity schemes that have been proposed afterwards (cf.

[VBC+15]). While it appears possible to prolong integrity and authenticity protection, it seems impossible to prolong confidentiality protection in a similar way. The reason is that once sensitive information is leaked to an adversarial entity, we usually cannot make that entity forget the information afterwards. From an information theoretic point of view, ciphertexts generated using a computationally secure encryption scheme (e.g., AES) contain almost as much sensitive information as the plaintexts. In the short-term this information may not be decryptable by a computationally limited adversary, but as soon as the encryption is breakable, knowing the ciphertexts is almost as good as knowing the plaintexts. As a consequence, computationally secure encryption algorithms, ultimately, appear unsuitable for long-term confidentiality protection. Suitable candidates, instead, are *information-theoretically secure* schemes, which do not require any computational assumptions. Examples for information-theoretically secure schemes are quantum key distribution and one-time pad encryption for confidentiality protection of data in transit, and proactive secret sharing for confidentiality protection of data at rest (cf. [BBMW14]).

1.2. Open challenges

As explained in section 1.1, the signature renewal method of [BHS93] is the basis for a number of schemes that are designed to provide long-term integrity protection. Even though an earlier version of this method described in [HS91] was found to be insecure, neither the revised version of the method, nor the schemes based on it are provided with a comprehensive security analysis. Without such an analysis, however, it remains unclear what are precisely the computational assumptions required and security guarantees provided by these schemes.

When taking a closer look at modeling computational security of long-lived systems it becomes evident that this task, in fact, requires a whole new computational model. A cryptographic scheme is traditionally considered computationally secure if any efficient strategy for breaking the scheme can be turned into an efficient strategy for solving some computational problem. Then, if there is no feasible strategy for solving the computational problem, it follows that there is no feasible strategy for breaking the security of the scheme. As pointed out in section 1.1, what is computationally infeasible today, might become feasible at some point in the future. However, such long-term computational aspects are typically not reflected within the computational models used for analyzing the security of cryptographic schemes. Canetti et al. made a step into this direction by proposing a computational framework that allows for expressing time-related computational constraints [CCK+08]. However, their framework neither allows for modeling computational entities with increasing computational power, nor for expressing exact security levels, while both aspects are crucial when analyzing the security of long-term protection schemes.

Consequently, to understand what are the security guarantees provided by long-term integrity schemes, a new computational model is needed and, based on this, precise security definitions and proofs must be developed.

Besides the lack of a precise understanding of the security of long-term integrity schemes, there is also no solution for combining long-term integrity with long-term confidentiality protection. On the one hand, existing long-term integrity schemes leak information to third-party timestamp services. On the other hand, existing long-term confidentiality schemes do not provide renewable integrity protection. To meet the protection requirements of sensitive long-lived data, new schemes must be developed that support simultaneous long-term protection of integrity *and* confidentiality.

1.3. Contributions

1.3.1. Security analysis of long-term protection schemes

The first contribution of this thesis is a rigorous security analysis of long-term integrity protection schemes (Part I).

As a basis, in chapter 3, we provide a novel computational model for long-lived systems. The model features a notion of time and allows for expressing time-related computational constraints. It captures adversaries who have a potentially unlimited running time, increase their computation rate over time, and change their computational architecture (e.g., from classical computers to quantum computers).

In chapter 4, we then present a formal security analysis of the renewal method described in [BHS93]. We first describe a long-term integrity scheme based on this method which uses signature-based timestamps for protection renewal. Then we analyze the security of the described scheme and show that it can be reduced to the security of the employed signature schemes within their validity period. In particular, we prove a lower bound on the security level of the long-term integrity scheme, where the bound is a function of the number of the employed cryptographic components and of their security level within their validity period. This lower bound can be used as a tool for estimating the security level of a given instantiation of the long-term integrity scheme at a given point in time.

Next, in chapter 5, we present a formal security analysis of long-term timestamp schemes that use a trusted repository. We prove the security of these schemes in the ideal primitive model using preimage-aware hash functions [DRS09]. In particular, we prove a lower bound on the security level of such a scheme, where the lower bound is a function of the number of repository queries, the number of data items allowed per timestamp, and the security level of the used cryptographic hash functions within their validity period.

Then, in chapter 6, we propose a novel cryptographic primitive called long-term commitment. While no commitment scheme can be unconditionally hiding and unconditionally binding at the same time, a long-term commitment is unconditional hiding and long-term binding (i.e., the binding security can be prolonged). On the negative side, we argue why it is unlikely that long-term commitments can be constructed from standard cryptographic primitives (e.g., one-way functions) using only black-box techniques. On the positive side, we show how to construct long-term commitments from unconditionally hiding and extractable-binding commitments. We also show that extractable-binding commitments can be constructed from extractable and collision-resistant hash functions [BCC+17].

1.3.2. Constructions of long-term secure storage systems providing integrity and confidentiality protection

The second contribution of this thesis is the development of various long-term secure storage systems that simultaneously provide integrity and confidentiality protection (Part II).

In chapter 7 we present LINCOS, which is the first storage system that simultaneously provides renewable integrity protection and information-theoretic confidentiality protection. On the way, we construct a novel long-term integrity scheme based on long-term commitments (cf. chapter 6) that does not leak any information about the protected data. We then combine this scheme with statistically secure secret sharing to obtain LINCOS. Afterwards, we present an experimental evaluation of LINCOS where long-term secure channels are implemented using QKD technology within the Tokyo QKD Network [SFI+11]. This is the first work that shows the feasibility of combined long-term integrity, authenticity, confidentiality protection.

Next, in chapter 8 we present PROPYLA, which is an extension of LINCOS that additionally provides long-term access-pattern-hiding security. This is relevant in scenarios where the storage provider has additional information about the structure of the stored data. Then, even if the storage provider cannot see the data content, it may still learn sensitive information by observing which data items are accessed at which point in times. Access-pattern-hiding security ensures that storage providers cannot learn anything from analyzing the access patterns. This property in addition to data integrity and confidentiality is achieved by using the components of LINCOS in combination with statistically secure ORAM schemes [CLP14]. In a performance analysis we show show that the communication, computation, and storage overhead of PROPYLA compared to LINCOS is poly-logarithmic in the number of stored data items.

Finally, in chapter 9 we present ELSA, which is an optimization of LINCOS for large datasets that contain relatively small data items. Such datasets often occur in practice (e.g, medical record databases) and ELSA drastically reduces the number

of timestamps required for protecting them. The performance improvements are achieved by using extractable-binding and statistically hiding vector commitment schemes. We first construct an extractable-binding and statistically hiding vector commitment scheme that is secure under selective decommitment. We then combine this scheme with renewable timestamps and secret sharing to obtain the storage architecture ELSA. We also experimentally evaluate the performance of ELSA in a simulated scenario where a dataset consisting of 12 000 data items of size 10 kB is stored, protected, and verified over a time span of 100 years. Our evaluation shows that ELSA completes this scenario an order of magnitude faster than LINCOS.

2. Preliminaries

2.1. Notation

We denote the set of natural numbers by $\mathbb{N}$ and the set of real numbers by $\mathbb{R}$. For $r_1, r_2 \in \mathbb{R}$, we define $\mathbb{R}_{[r_1,r_2]} = \{r \in \mathbb{R} : r_1 \leq r \leq r_2\}$. For $n \in \mathbb{N}$, by $[n]$ we denote the set $\{1, \ldots, n\}$. For vector $V = (v_1, \ldots, v_n)$, $n \in \mathbb{N}$, and set $I \subseteq [n]$, define $V_I := (v_i)_{i \in I}$, and for $i \in [n]$, we may also write $V_i := v_i$. For a probabilistic algorithm $\mathcal{A}$ and input x, we write $\mathcal{A}(x) \to_r y$ to denote that $\mathcal{A}$ on input x produces y using random coins r. For a pair of random variables (A, B) over a finite outcome set X, we define the statistical distance of A and B as $\Delta(A, B) := \sum_{x \in X} |\Pr_A(x) - \Pr_B(x)|$. We use the following square bracket notation for denoting lists. By [] we denote an empty list. For a list $L = [x_1, \ldots, x_n]$ and $i \in [n]$, we denote by $L[i :]$ the sublist $[x_i, \ldots, x_n]$, by $L[: -i]$ the sublist $[x_1, \ldots, x_{n-i+1}]$, and by $L[-i]$ the element x_{n-i+1}. For a list L and an element x, we write $L \mathrel{+}= x$ to denote that x is appended to L.

2.2. Computational security

A cryptographic primitive is *computationally secure* if for any probabilistic adversary given a large amount of computational resources the probability to break the security of the scheme is negligible. Such an adversary can be thought of as a program that runs on some kind of computing machine, which is modeled, for example, by a Turing Machine [Tur37] or a Quantum Turing Machine [Deu85]. The resources of such an adversary are measured in terms of the number of unitary operations performed by the machine. For $p \in \mathbb{N}$, by a p-step adversary we mean a program that halts after at most p operations when run on the considered computing machine model.

Definition 2.1. *Let $\epsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$ be a function and $\mathcal{P}$ be a cryptographic primitive. We say $\mathcal{P}$ is ϵ-secure, if any p-step adversary breaks $\mathcal{P}$ with probability at most $\epsilon(p)$.*

We remark that in the traditional view of cryptography the step count of an adversary is sometimes also referred to as the computation time of the adversary. In particular, no distinction between real time and computation steps is usually made. Furthermore, the class of computing machines considered is usually assumed to be the class of Turing Machines. We will see in chapter 3 that for analyzing long-term security of cryptographic schemes it is useful to distinguish between real time and

computational steps and to consider that the computational technology used by an adversary may change over time.

Security reductions

Complex cryptographic protocols (such as timestamp schemes) are often constructed from cryptographic primitives (such as hash functions or digital signature schemes). If $\mathcal{P}$ is a cryptographic protocol that uses a primitive $\mathcal{Q}$ as a building block, a typical security proof goes as follows. Assuming the existence of a p-step adversary $\mathcal{A}$ that breaks the protocol $\mathcal{P}$ with probability at least $\epsilon_{\mathcal{A}}(p)$, a p'-step adversary $\mathcal{B}$ is constructed that uses $\mathcal{A}$ and breaks the primitive $\mathcal{Q}$ with success probability at least $\epsilon_{\mathcal{B}}(p')$, where $p' = f(p)$ for some function f. We then have a statement that if $\mathcal{Q}$ is ϵ-secure, then $\mathcal{P}$ is ϵ'-secure with $\epsilon'(p) = \epsilon(f(p))$. The slower the growth of $\frac{\epsilon'(p)}{\epsilon(p)}$, the *tighter* the reduction is, i.e., the less *security loss* it has.

2.3. Cryptographic primitives

We describe several cryptographic primitives that are relevant in the context of long-term protection schemes.

2.3.1. Hash functions

A hash function H maps input strings of arbitrary length to output strings of fixed length. The security properties that are required from a hash function depend on the application [RS04]. For example, a hash function H may be required to be collision resistant, which means that it must be infeasible to find a pair of strings (x, x') such that $H(x) = H(x')$ and $x \neq x'$. Depending on the security model and use case, hash functions may also be keyed [BCK96]. In this case, the hash function is associated with a key generation algorithm that generates a hashing key. Before the hash algorithm is used, a key is generated and the hash algorithm then takes as input the key and a message.

Preimage-aware hash functions

Informally, a hash function H is called *preimage aware* (PrA) if whenever somebody first outputs as hash value y and later comes up with a preimage x, $H(x) = y$, then it must have known x when outputting y. The notion of preimage aware hash functions was formalized by Dodis et al. [DRS09] for hash functions H^P that use an ideal primitive P. The primitive is ideal in the sense that it can only be called via an oracle P that records all calls made to P in an advice string adv. More formally, for H^P to be preimage aware, there must exist an efficient algorithm $\mathcal{E}$ (the so-called

extractor) which when given y and the list adv of calls to the ideal primitive P, outputs x such that $H^P(x) = y$, or $\perp$ if the extraction failed. An adversary against the preimage awareness property of a hash function H tries to find x and y so that $\mathcal{E}(\mathsf{adv}, y) \neq x$ and $y = H^P(x)$.

Definition 2.2 (Preimage-aware function)**.** *Let $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$ be a function. A function H^P is ϵ-secure* preimage aware *(*PrA*) if for every $p_{\mathcal{E}}$, p_A, and q, there is a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every p_A-step adversary $\mathcal{A}$ that makes at most q calls to* Ex*,*

$$\mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{A}, \mathcal{E}) = \Pr\left[\mathbf{Exp}^{\mathrm{PrA}}_{P,H}(\mathcal{A}, \mathcal{E}) = 1\right] \leq \epsilon(p_{\mathcal{E}}, p_A, q) \ ,$$

where $\mathbf{Exp}^{\mathrm{PrA}}_{P,H}$ is defined in Listing 2.1.

Listing 2.1: PrA Security Experiment, $\mathbf{Exp}^{\mathrm{PrA}}_{P,H}(\mathcal{A}, \mathcal{E})$

$x \leftarrow \mathcal{A}^{\mathsf{P},\mathsf{Ex}}$;
$y \leftarrow H^P(x)$;
return 1 **if** $y \in \mathtt{Q} \wedge \mathtt{V}[y] \neq x$ **else** 0;

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
adv += (m, z);
return z;

oracle $\mathsf{Ex}(y)$:
$x \leftarrow \mathcal{E}(y, \mathsf{adv})$;
$\mathtt{Q} \leftarrow \mathtt{Q} \cup \{y\}$;
$\mathtt{V}[y] \leftarrow x$;
return x;

Hash chains

By a *hash chain* c, we mean a sequence $c[1], c[2], \ldots, c[m]$ of $2n$-bit strings and an m-bit string ι that is called the *shape* of c. The bits of ι are denoted by $\iota[1], \ldots, \iota[m]$. Every $c[k]$ consists of two n-bit halfs denoted by $c[k]_0$ and $c[k]_1$. By $x \overset{c}{\rightsquigarrow} r$ we mean that:

- $H(c[1]) = r$
- $H(c[k+1]) = c[k]_{\iota[k]}$, for every $k \in \{1, \ldots, m-1\}$
- $H(x) = c[m]_{\iota[m]}$

For example, a hash chain can be seen as a path through a hash tree [Mer90] from a leaf to the root (Figure 2.1).

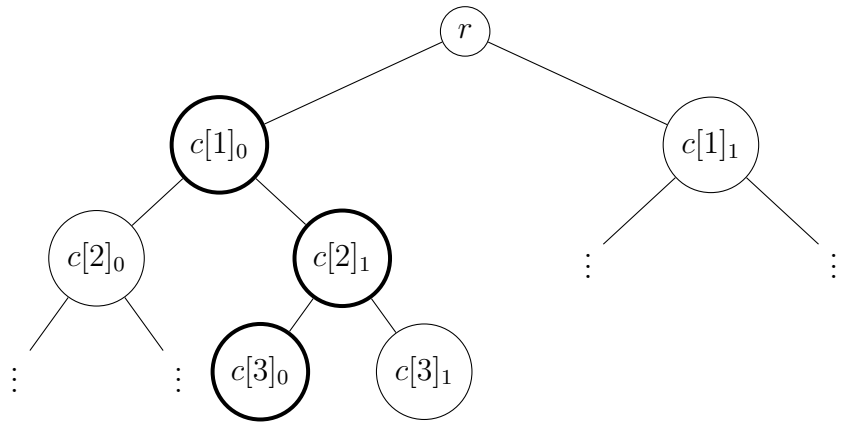

Figure 2.1.: A hash tree with hash chain $c = [c[1]_0 \| c[1]_1, c[2]_0 \| c[2]_1, c[3]_0 \| c[3]_1]$ of shape $\iota = [0, 1, 0]$, where for any x with $H(x) = c[3]_0$, $x \overset{c}{\rightsquigarrow} r$.

2.3.2. Digital signature schemes

A digital signature scheme SIG is defined by a tuple $(\mathcal{M}, \mathsf{Setup}, \mathsf{Sign}, \mathsf{Verify})$, where $\mathcal{M}$ is the message space, and Setup, Sign, Verify are algorithms with the following properties.

$\mathsf{Setup}() \to (sk, pk)$**:** This algorithm generates a secret signing key sk and a public verification key pk.

$\mathsf{Sign}(sk, m) \to s$**:** This algorithm gets as input a secret key sk and a message $m \in \mathcal{M}$. It outputs a signature s.

$\mathsf{Verify}(pk, m, s) \to b$**:** This algorithm gets as input a public key pk, a message m, and a signature s. It outputs $b = 1$, if the signature is valid, and 0, if it is invalid.

A signature scheme is considered ϵ-secure if for any t-bounded algorithm $\mathcal{A}$,

$$\Pr_{\mathsf{Setup} \to (sk,pk)} \begin{bmatrix} \mathcal{A}^O(pk) \to (m, s) : \\ \mathsf{Verify}(pk, m, s) = 1 \wedge m \notin Q \end{bmatrix} \leq \epsilon(t),$$

where $O(m) = \{Q \mathrel{+}= m; \mathsf{Sign}(sk, m) \to s; \text{return } s;\}$ [GMR88].

2.3.3. Timestamp schemes

A timestamp scheme involves a client and a timestamp service. The timestamp service initializes itself using algorithm Setup. The client uses protocol Stamp to request a timestamp from the timestamp service. Furthermore, there exists an

algorithm Verify that allows anybody to verify the validity of a message-timestamp-tuple.

Timestamp schemes can be realized in different ways (e.g., using a widely visible medium or trusted timestamp services [HS91]). For the most part, we consider timestamp schemes based on timestamp services and digital signatures. These work as follows. On initialization, a timestamp service chooses a signature scheme SIG and runs the setup algorithm $\mathsf{SIG.Setup} \rightarrow (sk, pk)$. It publishes the public key pk. A client obtains a timestamp for a message m, as follows. First, it sends the message to the timestamp service. Then, the timestamp service reads the current time t and creates a signature on m and t by running $\mathsf{SIG.Sign}(sk, [m, t]) \rightarrow s$. It then sends the signature s and the time t back to the client. Anybody can verify the validity of a timestamp (t, s) for a message m by checking $\mathsf{SIG.Verify}(pk, [m, t], s) = 1$. In addition, the authenticity of the public key is checked using a trust anchor that contains root certificates and revocation lists (cf. [BKW13]). The security of signature-based timestamp schemes is based on the security of the used signature schemes.

2.3.4. Commitment schemes

A commitment scheme can be thought of as a cryptographic version of a sealed envelope. It allows one party to fix a chosen message m without revealing it to other parties. At a later point, the party can decide to reveal the message m in a way that everybody is convinced that m is indeed the message that was previously fixed. More formally, a commitment scheme is defined as follows.

Definition 2.3 (Commitment scheme)**.** *A (non-interactive) commitment scheme* COM *is defined by a tuple* $(\mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$*, where* $\mathcal{M}$ *is the message space, and* Setup*,* Commit*,* Verify *are algorithms with the following properties.*

$\mathsf{Setup}() \rightarrow pk$**:** *This algorithm generates a public commitment key* pk*.*

$\mathsf{Commit}(pk, m) \rightarrow (c, d)$**:** *This algorithm gets as input a public key* pk *and a message* $m \in \mathcal{M}$*. It outputs a commitment* c *and a decommitment* d*.*

$\mathsf{Verify}(pk, m, c, d) \rightarrow b$**:** *This algorithm gets as input a public key* pk*, a message* m*, a commitment* c*, and a decommitment* d*. It outputs* $b = 1$*, if the decommitment is valid, and* 0*, if it is invalid.*

For a commitment scheme to be secure, it must satisfy *hiding* and *binding* security. These are commonly defined as follows. For a commitment scheme to be hiding secure, it is required that the commitment receiver does not learn the message during the commitment phase. If this property holds for commitment receivers with unlimited computational resources, then we say the scheme is unconditionally hiding.

Definition 2.4 (Statistically Hiding)**.** *Let* $\mathsf{CS} = (\mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Verify})$ *be a commitment scheme. For any public key* $k \in \mathsf{Setup}$ *and message* $m \in \mathcal{M}$*, define* $C_k(m)$ *as the random variable that has the distribution of* c *when sampling* $\mathsf{Commit}(k, m) \to (c, d)$*. A commitment scheme is* ϵ*-statistically-hiding if for any* $k \in \mathsf{Setup}$*, and any pair* $(m_1, m_2) \in \mathcal{M}^2$*, we have that* $\Delta(C_k(m_1), C_k(m_2)) \leq \epsilon$*.*

A commitment scheme is considered binding secure if a committer cannot change his mind about the committed message. Here, we consider committers with limited computational resources.

Definition 2.5 (Classical binding)**.** *Let* $\epsilon : \mathbb{N} \to \mathbb{R}_{[0,1]}$*. A commitment scheme* CS *is* ϵ*-classical-binding if for every integer* p*, for every* p*-step adversary* $\mathcal{A}$*:*

$$\mathbf{Adv}^{\mathrm{Bind}}_{\mathsf{CS}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\mathrm{Bind}}_{\mathsf{CS}}(\mathcal{A}) = 1\right] \leq \epsilon(p) \ .$$

Listing 2.2: The binding experiment $\mathbf{Exp}^{\mathrm{Bind}}_{\mathsf{CS}}(\mathcal{A})$.

```
ck ← CS.Setup;
(c, m, w, m', w') ← A(ck);
if CS.Verify(ck, m, c, w) = CS.Verify(ck, m', c, w') = 1 and m ≠ m' then
    return 1;
else
    return 0;
end
```

2.3.5. Secret sharing schemes

A secret sharing scheme allows a data owner to share a secret data object among a set of shareholders such that only specified subsets of the shareholders can reconstruct the secret, while all the other subsets of the shareholders have no information about the secret. Here, we consider threshold secret sharing schemes [Sha79], for which there exists a threshold parameter t (chosen by the data owner) such that any set of t shareholders can reconstruct the secret, but any set of less than t shareholders has no information about the secret. A secret sharing scheme has a protocol Setup for generating the sharing parameters, a protocol Share for sharing a data object, and a protocol Reconstruct for reconstructing a data object from a given set of shares. In addition to standard secret sharing schemes, proactive secret sharing schemes additionally provide a protocol Reshare for protection against so called mobile adversaries [HJKY95]. The protocol Reshare is an interactive protocol between the shareholders after which all the stored shares are refreshed so that they no longer

can be combined with the old shares for reconstruction. This protects against adversaries who gradually corrupt an increasing number of shareholders over the course of time.

2.3.6. Oblivious RAM

An oblivious random access machine (ORAM) [Gol87, GO96] allows a client to access a remotely stored database such that the storage server does not learn which data items are of current interest to the client. Here we assume that the client's data consists of N blocks of equal size and each block is associated with a unique identifier $\mathsf{id} \in \{1, \ldots, N\}$. The server holds a database of size $M > N$ blocks and each block in the server database is identified by a location $i \in \{1, \ldots, M\}$. A (stash-free) ORAM is defined by algorithms Setup, GenAP, and GetId with the following properties.

$\mathsf{Setup}(N)$**:** This algorithm takes as input the client database size N and generates a client local state s and a server database size M.

$\mathsf{GenAP}(s, \mathsf{id})$**:** This algorithm gets as input a client state s and a client block identifier $\mathsf{id} \in \{1, \ldots, N\}$, and outputs an access pattern $P \in \{1, \ldots, N\}^{2\times n}$, which is a sequence of server block location pairs, and a new client local state s'.

$\mathsf{GetId}(s, i)$**:** This algorithm gets as input a client state s and a server block location $i \in \{1, \ldots, M\}$, and outputs the corresponding client block identifier $\mathsf{id} \in \{1, \ldots, N\}$.

An ORAM is used as follows to store and access a database of size N. First, the client runs algorithm $\mathsf{Setup}(N) \to (s, M)$ and initializes the server database with M data blocks. To access (i.e., read or write) block $\mathsf{id} \in \{1, \ldots, N\}$ at the server, the client first computes $\mathsf{GenAP}(s, \mathsf{id}) \to (P, s')$ and updates its local state $s \leftarrow s'$. Then it accesses the server database according to the access pattern $P = [(i_1, j_1), \ldots, (i_n, j_n)]$, as follows. For every block location pair $(i, j) \in P$, it first retrieves block i. Then, it checks if $\mathsf{GetId}(s, i) = \mathsf{id}$ and if this is the case, processes the data. Afterwards, it stores the block at the new location j.

An ORAM is secure if the access patterns generated by GenAP are indindistinguishable from each other. In the security experiment (Listing 2.3), an adversary can instruct the client to access blocks of its choice and then sees the induced access patterns. In order to break the security, the adversary has to distinguish the access patterns of two access instructions of its choice.

Definition 2.6 (ORAM Security)**.** *An ORAM scheme* ORAM *is information theoretically secure if for any* $N \in \mathbb{N}$ *and probabilistic algorithm* $\mathcal{A}$*,*

$$\Pr[\mathbf{Exp}^{\mathrm{APH}}_{\mathit{ORAM},N}(\mathcal{A}) = 1] = \frac{1}{2} \ .$$

Listing 2.3: The ORAM access pattern hiding experiment $\mathbf{Exp}^{\mathrm{APH}}_{\mathtt{ORAM},N}(\mathcal{A})$.

$(s, M) \leftarrow \mathtt{ORAM}.\mathsf{Setup}(N)$;
$(\mathsf{id}_1, \mathsf{id}_2) \leftarrow \mathcal{A}^{\mathsf{Client}}(M)$;
$b \leftarrow_{\$} \{1, 2\}$;
$P \leftarrow \mathsf{Client}(\mathsf{id}_b)$;
$b' \leftarrow \mathcal{A}^{\mathsf{Client}}(P)$;
if b=b' **then**
  return 1;
else
  return 0;
end

oracle $\mathsf{Client}(\mathsf{id})$:
$(s, P) \leftarrow \mathtt{ORAM}.\mathsf{GenAP}(s, \mathsf{id})$;
return P;

Part I.

Analysis

3. A Computational Model for Long-Lived Systems

When analyzing the security of long-term protection schemes, attackers must be considered whose lifetime is as long as the lifetime of the protected data, i.e., decades or even centuries. During such long time periods computational assumptions, which cryptographic schemes rely on, are often invalidated and the corresponding schemes must be replaced by new schemes based on stronger computational assumptions. We observe the following two developments with respect to computational technology. Firstly, new algorithms are developed that break computationally secure schemes more efficiently. Secondly, new computational technologies are developed that allow for computing algorithms faster. The computational model typically used in cryptography, however, does not reflect such technological advancements. In order to analyze the security of long-lived systems, a new computational model must be developed that captures these developments.

Contribution. In this chapter we first discuss existing computational models and their shortcomings when used for analyzing the security of long-lived systems. We then present a novel computational model that is adequate for analyzing the security of long-lived cryptographic systems. In particular, our model allows for capturing that new algorithms are found and new computational technologies become available.

Publications. This chapter is based on publications [G1], [G4], and [G3].

3.1. Discussion of existing computational models

The most widely used computational model in cryptography is the Turing Machine Model [Tur37]. This model was proposed in 1936 by Alan Turing and it is widely accepted as an adequate model to express what is computable by the computing technology that we use today. In addition, it allows for measuring the efficiency of an algorithm [Coo71]. Moreover, it has been shown that several other important computation models (e.g., the Random Access Machine Model [CR72]) can be efficiently simulated in the Turing Machine Model.

The Turing Machine Model has been validated as an adequate model for computation on classical information using boolean operations. However, in the 1980s the concept of quantum computing was proposed [Ben80, Fey82, Deu85], which is computation on quantum information using quantum operations. Quantum computation is captured, for example, by the Quantum Turing Machine Model [Deu85] and the Quantum Random Access Machine Model [Kni96]. While it is still unclear whether quantum computers are inherently more powerful than classical computers, current results suggest that they are [Sho99]. In the future, even more advanced computational technology may be developed. Thus, our goal is to consider the most general class of computing machines possible and we want to be able to express that computational technology and, in particular, the computational technology is improving over time. Time-related computational constraints, however, are rarely considered in cryptographic security models, with the following exceptions:

In [CCK+08], Canetti et al. propose a framework for modeling computational security in long-lived systems. They first observe that in virtually all existing cryptographic security models the adversaries are assumed to be polynomially bounded while long-lived systems may be running for exponential time. To resolve this imbalance, they propose a computational model for long-lived systems where computation time and real time are distinguished. This allows them to model adversaries that are computationally bounded per unit of time while their overall running time is unlimited. As an example use of their framework, they study the security of a long-term integrity scheme. We also used their framework to analyze the security of a different long-term integrity scheme [G8]. However, while their framework allows for modeling long-lived computationally bounded adversaries, it does not consider that computational technology is advancing. Also, it does not allow for analyzing concrete security levels of cryptographic schemes which is crucial when analyzing the security of long-term integrity schemes, as we will see in chapter 4, chapter 5, and chapter 6.

In [Sch14], Schwenk proposes a formalism for modeling time in cryptographic systems and uses it for analyzing the security of authenticated key exchange protocols. The basic idea is to augment the security experiment with a global clock that is incremented whenever a special oracle call is made. However, while Schwenk uses the power of this time formalism to capture certain aspects of the functionality of the analyzed key exchange protocols, he does not use it to express time-related computational bounds for adversaries.

3.2. The computational model used in this thesis

We now present the computational model that we will use in this thesis for analyzing the security of long-lived systems. It is based on the notion of a computing machine, which is a general model of computation that captures classical computers as well

as quantum computers, and potentially other computational technologies that may be developed in the future. We then augment this model with a notion of real-time and describe a formalism for modeling computing machines whose computational power increases over time.

Computing machines with time-related computational bounds

A computing machine M describes a probabilistic input-output relation. For an input x and an output y, we write $\Pr[M(x) = y]$ to denote the probability that on input x machine M outputs y. The process of producing an output is called a computation of the machine. A computing machine has an internal state which is transformed consequently over the course of a computation by applying a certain set of unitary operations. The number of operations performed is also referred to by the step count of the machine.

We assume that at every point in time t, there is a certain class of computing machines $\mathcal{M}_t$ available. Furthermore, we assume that the class $\mathcal{M}_t$ of computing machines available at time t widens when t increases, i.e., $\mathcal{M}_t \subseteq \mathcal{M}_{t'}$ for $t < t'$. This captures that more powerful computing architectures are developed over the course of time, e.g., $\mathcal{M}_1$ may represent classical computers and $\mathcal{M}_2$ may represent quantum computers. We remark that quantum communication is not considered in this model.

Model of time. We model real time using a global clock time formalism similar to [Sch14]. That is, we augment our security experiments with a global clock Clock which is associated with a discrete state $\mathsf{time} \in \mathbb{N}$ that globally determines what is the current time in the experiment. The adversary is given the power to advance time by calling $\mathsf{Clock}(t)$, which sets $\mathsf{time} = t$ if $t > \mathsf{time}$.

Long-lived adversaries. We model long-lived adversaries with increasing computational power over time as follows. A long-lived adversary $\mathcal{A}$ is a sequence of machines $(\mathcal{A}_{(0)}, \mathcal{A}_{(1)}, \mathcal{A}_{(2)}, \ldots)$, where $\mathcal{A}_{(t)} \in \mathcal{M}_t$, and is associated with a global clock Clock. When $\mathcal{A}^{\mathsf{Clock}}$ is started, then actually the component $\mathcal{A}_{(0)}^{\mathsf{Clock}}$ is run. Whenever a component $\mathcal{A}_{(t)}^{\mathsf{Clock}}$ calls the clock oracle to set a new time t', then component $\mathcal{A}_{(t)}^{\mathsf{Clock}}$ is stopped and the component $\mathcal{A}_{(t')}^{\mathsf{Clock}}$ is run with input the internal state of $\mathcal{A}_{(t)}^{\mathsf{Clock}}$. For functions $\rho : \mathbb{N} \to \mathbb{N}$ and $q : \mathbb{N} \to \mathbb{N}$, we say $\mathcal{A}$ is *ρ-bounded* if for every time t, the aggregated step count of the machine components of $\mathcal{A}$ until time t is at most $\rho(t)$. We say $\mathcal{A}$ is *q-call-bounded* if for every time t, it has done at most $q(t)$ oracle calls until time t.

Notation. Let $\mathcal{P}$ be a cryptographic primitive and $\mathcal{M}$ be a machine class. We say $\mathcal{P}$ is ϵ-secure for adversaries of $\mathcal{M}$ if any p-step adversary of class $\mathcal{M}$ breaks $\mathcal{P}$ with probability at most $\epsilon(p)$.

4. Signature-based Long-Term Integrity Protection

In the early 1990s, Haber and Stornetta proposed a method for prolonging the validity of digital signatures [HS91]. While obvious flaws in their first proposal had been fixed later [BHS93], their method was not formally proven secure and the precise security guarantees remained unclear. Nevertheless, many schemes and standards for long-term integrity protection have been proposed based on this method over the last decades (see [VBC+15] for an overview), but comprehensive security models and proofs are still missing.

Contribution. In this chapter we provide the first formal security analysis of long-term integrity schemes. In particular, we formally define long-term integrity schemes and their security using the computational model described in chapter 3. We then describe a long-term integrity scheme based on the signature renewal method described in [BHS93] and analyze its security. The result of our security analysis is a lower bound on the security level of the described scheme, where the bound is a function of the short-term security levels of the used cryptographic components.

Publications. This chapter is based on publication [G1].

4.1. Discussion of long-term integrity schemes

This section informally discusses the functionality and security requirements of long-term integrity protection schemes that are derived from the ideas of Bayer, Haber, and Stornetta [BHS93].

4.1.1. Functionality

Suppose a user has a signed data object $D = (d, s)$, where d is the data object and s is the signature. Integrity and authenticity protection of d is guaranteed by the digital signature s. However, a digital signature has a limited lifetime whose length depends on the choice of the signature scheme and the choice of the signature scheme parameters [Len04]. It is also influenced by the possibility that the private signing key can leak to an attacker, which would enable the attacker to forge signatures. In

the following, we describe a protection method based on the ideas of [BHS93] for protecting integrity and authenticity of d over long time periods.

Initiating protection

At first, the user initiates the long-term integrity and authenticity protection of d by sending $D = (d, s)$ to a certified time-stamping authority (TSA) [ACPZ01]. The TSA creates a signature-based timestamp by reading the current time t and computing a signature s_0 on $D \| t_0$. The TSA then sends the timestamp (t_0, s_0) back to the user. The idea is that the timestamp can later prove that the signature was generated at a point in time when the corresponding signature scheme was still considered secure.

Prolonging protection

The security of the initial integrity protection relies on the security of the digital signature scheme used by the TSA, which is also limited. To prolong the validity of an initial integrity proof (t_0, s_0) for a data object D beyond the lifetime of the signature s_0, a new timestamp (t_1, s_1) is retrieved for (D, t_0, s_0) from a TSA that uses a stronger signature scheme. The tuple (t_0, s_0, t_1, s_1) then constitutes an integrity proof for D with an extended lifetime. In particular, it allows for proving that the signatures s_1, s_0, and s have been generated when the corresponding signature scheme instances were considered secure.

The validity of an integrity proof can be prolonged repeatedly. This is done by time-stamping the current integrity proof as described above. Whenever the validity of an integrity proof is prolonged, a new timestamp signature is added. Hence, the integrity proof can be described by a sequence of timestamp signatures, denoted as $t_0, s_0, \ldots, t_n, s_n$. We visualize the procedure in Figure 4.1.

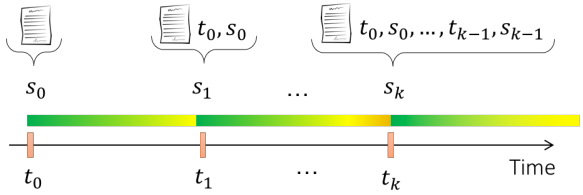

Figure 4.1.: Generation of a long-term integrity proof. The colored bars visualize the security of the signature schemes.

Verification

To verify the integrity and datedness of a data object $D = (d, s)$ with respect to time t_0 using integrity proof $t_0, s_0, \ldots, t_n, s_n$, it needs to be checked that each signature

s_i is valid for $D\|t_0\|s_0\|\ldots\|t_{i-1}\|s_{i-1}\|t_i$ at time t_{i+1}. To verify the authenticity of d, it needs to be checked that the signature s is valid for d at time t_0.

4.1.2. Requirements for security

In the following we discuss the requirements for the security of long-term integrity schemes. Intuitively, a long-term integrity scheme is secure if it is infeasible to produce a valid integrity proof p for a data object d and time t if d did not exist at time t. In the following we list the trust assumptions, cryptographic assumptions, and computational assumptions that are commonly considered when using long-term integrity schemes.

Trust assumptions. TSAs need to be trusted that they read and sign the correct time when generating timestamps.

Cryptographic assumptions. The proposed long-term integrity schemes rely on computationally secure cryptographic primitives, such as digital signature schemes and hash functions. A computationally secure cryptographic primitive is usually associated with a validity period that depends on the chosen scheme and the scheme parameters. It is essential that the used cryptographic scheme instances are secure within their validity period, e.g., that a signature scheme is unforgeable (cf. subsection 2.3.2) or a hash function is collision-resistant (cf. subsection 2.3.1).

Computational assumptions. Adversaries are long-lived and may increase their computational power over time (cf. chapter 3).

4.1.3. Related approaches

A variety of schemes for long-term integrity protection have been proposed that rely on the method proposed in [BHS93] and described in subsection 4.1.1. In the following we give a brief overview of the different approaches. A more detailed overview can be found, for example, in [VBC+15].

The method described in subsection 4.1.1 uses a sequence of timestamps to preserve the validity of a digital signature. Several schemes have been proposed that use a similar approach, e.g., Advanced Electronic Signatures [ETS10b, ETS10a], Content Integrity Service [Hab06], and the Evidence Record Syntax [GBP07, BSG11]. They mostly differ in how the sequence of timestamps to a document is generated and verified. Evidence Record Syntax, for instance, is a format which uses Merkle Trees [Mer90] and allows to protect a set of documents with a single timestamp.

For our description of the long-term protection method in subsection 4.1.1 we have considered signature-based time-stamping. An alternative method for time-

stamping is time-stamping based on public repositories (PR, also known as widely-visible medium, WVM). The idea behind PR-based time-stamping is to publish a hash of the data on a widely visible medium that has a time reference (e.g., a newspaper). Due to the public release of the hash value and the security properties of the used hash function, the data cannot be altered afterwards. This approach assumes that the PR will be available in the future. Furthermore, this approach asks for the existence of witnesses, i.e., people that saw the original data and can testify that the data has not been changed since. We analyze PR-based long-term integrity protection in chapter 5.

Another approach to long-term integrity protection uses highly trusted notaries. Such notaries can certify the validity of digital signatures and thereby enable schemes where old attestations can be removed once their validity has been certified within a new attestation. Thus, with notary-based schemes an integrity proof only contains a single digital signature. Notary-based long-term protection schemes are, for example, Cumulative Notarization [LG04] and Attested Certificates [VCBH13]. We remark that notary-based schemes require highly trustworthy notaries which the verifier trusts to properly verify and certify attestations. This is necessary because the verifier cannot later check whether this process has been carried out correctly because old attestations are deleted and only the most recent one is kept. While the trust assumptions for notary-based schemes are stronger than for TSA-based schemes, from a cryptographic viewpoint the constructions are similar (i.e., security is prolonged by signature renewal). While our analysis focuses on TSA-based schemes, it appears likely that our techniques can also be applied for analyzing notary-based schemes.

4.2. Formalization of long-term integrity schemes

In this section, we provide a formal security analysis of long-term integrity schemes. Our security analysis is based on the computational model described in chapter 3 and the observations made in section 4.1.

We start by defining the syntax of long-term integrity schemes and describe the construction of a long-term integrity scheme, referred to by TS-LTIS, within that syntax. The construction is based on the description given in subsection 4.1.1 and lies at the core of many proposed long-term integrity schemes (e.g., [GBP07, BSG11, Hab06]). Afterwards, we define what is the correct functionality of long-term integrity schemes and show that TS-LTIS functions correctly. Next, we define a security model for long-term integrity schemes via a game in which an adversary is asked to forge an integrity proof for a document that was not known by the adversary at the claimed point in time. Finally, we prove the security of TS-LTIS based on the proposed model.

4.2.1. Scheme definition and construction

A long-term integrity scheme consists of three algorithms, one for generating an initial integrity proof, one for prolonging the validity of an integrity proof, and one for verifying an integrity proof. The following definition captures this more formally.

Definition 4.1 (Long-term integrity scheme)**.** *A long-term integrity scheme is a tuple of algorithms* (Protect, Prolong, Verify) *with the following properties.*

- $\mathsf{Protect}(d) \rightarrow (P, t)$*: On input of data object* d*, this algorithm generates an integrity proof* P *and additionally outputs the time* t *that* P *refers to.*
- $\mathsf{Prolong}(d, P) \rightarrow P'$*: On input of data object* d *and integrity proof* P*, this algorithm generates a new integrity proof* P'*.*
- $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t, P) \rightarrow b$*: On input of trust anchor* TA*, current verification time* t_{ver}*, data object* d*, time* t*, and integrity proof* P*, this algorithm outputs* $b = 1$ *if* P *is a valid integrity proof for data object* d *and time* t *under trust anchor* TA *and given that the current time is* t_{ver}*. If* P *is not valid under these constraints, it outputs* $b = 0$*.*

Next, we describe the construction of a long-term integrity scheme based on the description given in subsection 4.1.1. We refer to subsection 2.3.3 for a more detailed description of the functionality of signature-based TSAs.

Construction 4.1 (TS-LTIS)**.** *The long-term integrity scheme TS-LTIS consists of the algorithms* (Protect, Prolong, Verify) *as defined below.*

- $\mathsf{Protect}(d)$*: Select a TSA and request a timestamp* τ *on* d *from the TSA. Output the timestamp* τ*, which is the initial integrity proof, and the time* $\tau.t$*,which is provided by the TSA with the timestamp* τ*.*
- $\mathsf{Prolong}(d, P)$*: When the signature scheme used to generate the latest time-stamp included in* P *is about to become insecure, do the following: Select a TSA and request a timestamp* τ *on* (d, P) *from the TSA. Output the pair* (P, τ)*, which is the new integrity proof.*
- $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t, P)$*: Let* $P = (\tau_1, \ldots, \tau_n)$ *and* $t_{n+1} = t_{\mathsf{ver}}$*, and for* $i \in \{1, \ldots, n\}$*, let* t_i *be the time associated with* τ_i*. For* $i \in \{1, \ldots, n\}$*, check that the timestamp* τ_i *was valid for data* $d \| \tau_1 \| \ldots \| \tau_{i-1}$ *and time* t_i *at time* t_{i+1} *by using the trust anchor* TA*. Then check that* $t_0 = t$*. If all checks go through output* $b = 1$*, otherwise output* $b = 0$*.*

  (We remark that the trust anchor TA *must contain the data that is needed for verifying timestamp signatures, e.g., public key certificates and revocation lists.*

Public key certificates come with expiry dates which also determine the validity of the used cryptographic algorithms. If the cryptography scheme described in the certificate is broken before the expiration date, then the certificate must be revoked. We refer to [BKW13] for more details on public key certificates.)

4.2.2. Correctness

A long-term integrity scheme must fulfill the following correctness requirements. They define what is the intended functionality of a long-term integrity scheme when no adversary is considered and all cryptographic services are assumed to be secure. Afterwards we show that TS-LTIS satisfies this correctness definition.

Definition 4.2 (Correctness)**.** *Let* (Protect, Prolong, Verify) *be a long-term integrity scheme and n be an integer. A proof of integrity P_n for a data object d is generated as follows.*

At time t_0, the algorithm Protect *is executed with input d. Let integrity proof P_0 and reference time t be the output of that execution. Then, for $i = 1, \ldots, n$, at time $t_i > t_{i-1}$, the algorithm* Prolong *is executed with input data object d and integrity proof P_{i-1} at a point in time when the cryptographic primitives used to generate P_{i-1} are still valid. Let integrity proof P_i be the output of that execution. Finally, at some point in time t_{ver}, the verification algorithm* Verify *is executed. Let* TA *denote the trust anchor provided by the environment at that time.*

For a long-term integrity scheme (Protect, Prolong, Verify) *to be correct we require that for any data object d if proof P_n is generated as described above, then the equation* $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t_0, P_n) = 1$ *must hold.*

The next theorem states that our construction TS-LTIS presented in subsection 4.2.1 functions correctly.

Theorem 4.1 (Correctness of TS-LTIS)**.** *The long-term integrity scheme TS-LTIS is correct.*

Proof. Let d be a data object and assume that P_n is an integrity proof generated as follows. First, algorithm Protect is executed on input data object d and outputs an integrity proof P_0 and a time t_0. Then, for $i = 1, \ldots, n$, algorithm Prolong is executed with P_{i-1} as input returning a new proof P_i. For correctness it is required that if the used TSAs are secure within their usage period, then it must hold that $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t_0, P_n) = 1$, where TA is the trust anchor at time t_{ver}.

Indeed, proof P_n generated as described above verifies for d and t_0, which can be seen as follows. We observe that the certificate validity checks in Verify succeed because we assume that the cryptographic services were chosen correctly. Furthermore, we observe that the signature checks during verification also succeed because P_n is constructed by executing algorithm Protect followed by multiple executions of algorithm Prolong, and thus, the data structure of the proof has the correct format. □

4.2.3. Security

Next, we define a security model for long-term integrity schemes. Our security definition is based on a probabilistic experiment in which an adversary can interact with cryptographic components in its environment and is then asked to output a forged integrity proof. We first describe the adversary model. Then we state the security definition. Finally, we show that our construction TS-LTIS satisfies that definition.

Adversary model

We consider adversaries with potentially unlimited running time that are computationally bounded per unit of time as described in chapter 3. An adversary may also interact with its environment, for example, by requesting timestamps from trusted time-stamping authorities. In the following, we first describe which queries an adversary can make to interact with its environment.

Set time. As described in chapter 3, we consider in our long-term security experiments a global clock that determines the global time in the experiment (for example, used by TSAs). The adversary is given the power to advance the time shown on the clock. Formally, this is modeled as follows. The environment has a global variable $t_{\mathsf{cur}} \in \mathbb{N}_0$. At initialization, the time is set to zero, i.e., $t_{\mathsf{cur}} := 0$. The adversary may advance time via the following query.

- $\mathsf{SetTime}(t)$: If time t is larger than the current time t_{cur} (i.e., $t > t_{\mathsf{cur}}$), t_{cur} is set to t (i.e., $t_{\mathsf{cur}} := t$).

We remark that because the computational power of the adversary is bounded with respect to time, the adversary is eventually forced to advance time via a SetTime query in order to continue its operation (cf. chapter 3).

Request timestamps. In the environment of the adversary TSAs are active. Each TSA is associated with a signature scheme S and a validity period, that is, a time interval $[t_1, t_2]$. At the beginning of the validity period, at time t_1, the TSA generates a key pair, $S.\mathsf{KeyGen}() \to (sk, pk)$ and the public key and the validity times are given to the adversary.

An adversary can request timestamps from TSAs in its environment. In addition, after the validity period of a TSA has passed, it can compromise the TSA. In this case, the adversary obtains the private signing key. This is modeled by allowing the adversary to make the following queries.

- $\mathsf{Timestamp}(i, x)$: The adversary makes a timestamp query Timestamp with input the identifier of a TSA i and a data object x. If the current time t_{cur} lies

within the validity period of TSA i, then a signature σ on the current time t_{cur} and the data object x is generated, i.e., $\mathsf{Sign}(t_{\mathsf{cur}} \| x) \to \sigma$. Afterwards, the tuple $(t_{\mathsf{cur}}, \sigma)$ is returned to the adversary.

- $\mathsf{CompromiseTSA}(i)$: When the adversary makes a query $\mathsf{CompromiseTSA}$ with input the identifier of a TSA i, it is checked whether the validity period of TSA i is over, and if this is the case, the private signing key of TSA i is given to the adversary.

We remark that our model of time-stamping is based purely on signatures. In other approaches, e.g., [ACPZ01], the data is first hashed and then signed. We analyze hash-based time-stamping in chapter 5.

Security definition

We now define unforgeability security of long-term integrity schemes.

Definition 4.3 (Unforgeability security)**.** *Let* $\mathsf{LTIS} = (\mathsf{Protect}, \mathsf{Prolong}, \mathsf{Verify})$ *be a long-term integrity scheme,* $\mathcal{A}$ *be an adversary as defined in chapter 3, and* Env *be an environment as described in section 4.2.3.*

The forgery game is defined as follows. The adversary $\mathcal{A}$ *interacts with its environment* Env *and, at some point in time* t_{ver}*, outputs* (d, t, P)*. Let* TA *denote the trust anchor (i.e., public keys of TSAs and scheme validity times) provided by the environment at time* t_{ver}*. The adversary wins the game if* $\mathsf{Verify}(\mathsf{TA}, t_{\mathsf{ver}}; d, t, P) = 1$ *and no timestamp service was queried with* d *at time* t*. That is, the adversary is able to output a valid integrity proof for data object* d *and time* t*, without knowing* d *at time* t*. We denote the winning probability of* $\mathcal{A}$ *in this game by* $\mathsf{Adv}^{\mathsf{integrity}}_{\mathsf{LTIS},\mathcal{A},\mathsf{Env},t_{\mathsf{ver}}}$.

We say a long-term integrity scheme LTIS *is* ϵ*-secure for environment* Env *and computing machine model* $\mathcal{M}$ *if for any* ρ*-bounded adversary* $\mathcal{A} \in \mathcal{M}$*, for any point in time* t_{ver}*,*

$$\mathsf{Adv}^{\mathsf{integrity}}_{\mathsf{LTIS},\mathcal{A},\mathsf{Env},t_{\mathsf{ver}}} \leq \epsilon(\rho, t_{\mathsf{ver}}) \ .$$

The following theorem states that the security level of the long-term integrity scheme $\mathsf{TS\text{-}LTIS}$ can be reduced to the security levels of the used cryptographic components within their validity period.

Theorem 4.2. *Let* Env *be an environment with timestamp services* $\mathcal{T} = \{\mathsf{TS}_i\}_i$ *and for all* i*, let* $[\mathsf{TS}_i.t_1, \mathsf{TS}_i.t_2]$ *be the validity period of* TS_i*. If for all* i*,* TS_i *is* ϵ_{TS_i}*-secure against adversaries of* $\mathcal{M}_{\mathsf{TS}_i.t_2}$*, then the long-term integrity scheme TS-LTIS is* ϵ*-secure for* Env *and* $\mathcal{M}$ *with*

$$\epsilon(\rho, t_{\mathsf{ver}}) = \sum_{\mathsf{TS}_i \in \mathcal{T}_{t_{\mathsf{ver}}}} \epsilon_{\mathsf{TS}_i}(\rho(\mathsf{TS}_i.t_2)) \ ,$$

where $\mathcal{T}_{t_{\mathsf{ver}}}$ *is the set of timestamp services available until time* t_{ver}*.*

Proof. Let $\mathcal{A} \in \mathcal{M}$ be a ρ-bounded adversary and t_{ver} be a point in time. For every timestamp service TS_i, we construct from $\mathcal{A}$ an adversary $\mathcal{B}_i$ against the signature scheme corresponding to TS_i, and show that if $\mathcal{A}$ breaks the security of $\mathsf{TS\text{-}LTIS}$, then at least one of the $\mathcal{B}_i$ breaks the security of TS_i within its valitidy period (i.e., before time $\mathsf{TS}_i.t_2$).

We describe the adversary $\mathcal{B}_i$ against the timestamp service TS_i. The adversary $\mathcal{B}_i$ runs $\mathcal{A}$ until time $\mathsf{TS}_i.t_2$ and simulates the environment of $\mathcal{A}$ as described in the game above with the following exceptions. When $\mathcal{A}$ asks for the signature verification key of authority i, $\mathcal{B}_i$ provides the public verification key of its forgery game, and when $\mathcal{A}$ asks for a timestamp of that authority, then $\mathcal{B}_i$ creates that timestamp using its signing oracle. Algorithm $\mathcal{B}_i$ constantly scans the communication of $\mathcal{A}$ with its environment for a pair (x, τ), where x is some data object and τ is a timestamp corresponding to TS_i for which $\mathcal{A}$ has not requested a timestamp yet. If $\mathcal{B}_i$ finds such a tuple, it outputs that tuple as the $(x \| \tau.t, \tau.\sigma)$, where $\tau.t$ is the timestamp time and $\tau.\sigma$ is the timestamp signature, as a forgery for its signature forgery game.

We observe that we can bound the probability that $\mathcal{A}$ outputs a data object d with a valid integrity proof $P = (\tau_1, \ldots, \tau_n)$ for time t by the sum of the probability that one of the timestamps in P was forged by $\mathcal{A}$ while the corresponding timestamp service was valid. This can be seen, as follows. Suppose a timestamp for some data object x' and time t' is not forged. Then a timestamp service was queried with x' at time t'. We observe that the verification procedure of $\mathsf{TS\text{-}LTIS}$ guarantees that τ_i is a valid timestamp for data $d, \tau_1, \ldots, \tau_i$ and time t_i at time t_{i+1}. It also guarantees that the timestamp service corresponding to τ_i is valid at the time of the next timestamp t_{i+1}. It follows that for $i \in \{1, \ldots, n\}$, the timestamp service $\mathsf{TS}_{\tau_i.\mathsf{TS}}$ was queried with $(d, \tau_1, \ldots, \tau_{i-1})$ at time $\tau_i.t$. Thus, whenever $\mathcal{A}$ outputs (d, t, P) such that P is valid for d and time t but no timestamp service was queried with d at time t, then one of the adversaries $\mathcal{B}_i$ finds a forged timestamp signature while scanning the queries of $\mathcal{A}$ before the corresponding timestamp service becomes invalid.

In summary, we can bound the probability that $\mathcal{A}$ forges a long-term integrity proof by the sum over the probability that the security of at least one of the timestamp services is broken within its validity period, i.e.,

$$\mathsf{Adv}^{\mathsf{integrity}}_{\mathsf{LTIS},\mathcal{A},\mathsf{Env},t_{\mathsf{ver}}} \leq \sum_{\mathsf{TS}_i \in \mathcal{T}_{t_{\mathsf{ver}}}} \epsilon_{\mathsf{TS}_i}(\rho(\mathsf{TS}_i.t_2)) \ .$$

□

We remark that this reduction-based security proof is substantially different from typical reduction-based security proofs in the sense that the reduction algorithm is required to be successful within the validity period of the respective cryptographic component, which is in most cases before the long-term adversary has finished its computation.

Table 4.1.: Security of long-term integrity protection as a function of the security of cryptographic services.

n	L_{comp}	L_{int}
2^{16}	2^{-80}	2^{-64}
	2^{-128}	2^{-112}
	2^{-192}	2^{-174}

Table 4.2.: Security of long-term integrity protection as a function of the length parameter.

L_{comp}	n	L_{int}
2^{-128}	2^{8}	2^{-120}
	2^{16}	2^{-112}
	2^{32}	2^{-96}

4.3. Evaluation of security level

Given the security analysis in section 4.2 and the result of Theorem 4.2, we evaluate the security level L_{int} of the long-term integrity scheme TS-LTIS. Here, by security level we mean the probability that an adversary of the considered type breaks the scheme.

We observe that we can analyze the security level based on two parameters. The first parameter L_{comp} is a common upper bound on the failure probability of each of the used cryptographic components (i.e., signature schemes). For example, $L_{\mathsf{comp}} = 2^{-80}$ means that any component has at least 80 bit security (i.e., a failure probability of at most 2^{-80}) within its validity period against the considered class of adversaries. The second parameter n corresponds to the total number of components available. According to Theorem 4.2, it is $L_{\mathsf{int}} \leq n \cdot L_{\mathsf{comp}}$.

In Table 4.1 we show the security level L_{int} of TS-LTIS for $n = 2^{16}$ and different values of L_{comp}. We observe that there is a constant factor of security loss between the common security level of the cryptographic components L_{comp} and the security level of the long-term integrity scheme L_{int}.

In Table 4.2 we show the security level L_{int} of TS-LTIS for $L_{\mathsf{comp}} = 2^{-128}$ and different values of n. We observe that the security loss increases when n increases. The magnitude of the security loss depends on the number of involved cryptographic components (i.e., signature schemes) and seems unavoidable, as discussed in [BJLS16].

5. Long-Term Secure Time-Stamping from Preimage-Aware Hashing

Hash-based time-stamping is a viable alternative to signature-based time-stamping (cf. chapter 4) whose security within short-lived systems has been studied extensively [BS04, BLSW05, BL07, BN08, BN10, BL13]. However, the security of hash-based time-stamping in long-lived systems, and, in particular, when used in combination with the renewal method of [BHS93], has not been studied thus far.

Contribution. In this chapter we prove the security of hash-based long-term time-stamping in the ideal primitive model by using preimage-aware hash functions. In order to achieve this, we first establish the notion of extractable security for short-term timestamp schemes and show that hash-based time-stamping is extractable secure if preimage-aware hash functions are used. Afterwards, we introduce the notion of extractable security to long-term time-stamping and show that long-term time-stamping is extractable secure if extractable secure short-term time-stamping is used. In particular, we provide formulas for estimating concrete security levels. We also apply our results and show what is the security level achieved by hash-based long-term time-stamping for different parameters and protection periods.

Publications. This chapter is based on publication [G4].

5.1. Background

5.1.1. Hash-based time-stamping

Digital time-stamping was first proposed by Haber and Stornetta [HS91]. It is used to prove that a given data object existed at a certain point in time. In the following we describe a hash-based timestamp scheme based on their ideas [HS91, BHS93] and survey various security models that have been proposed for such a scheme.

Scheme description

The following hash-based timestamp scheme is associated with a hash function H and a set of allowed hash chain shapes $\mathcal{S}$. It uses a trusted repository Rep that accepts hash value queries. If Rep receives a hash value query r, it publishes r so that everybody can verify that r existed at this point in time t.

The time-stamping procedure is divided into rounds. During each round, a time-stamp server receives a set of bitstrings $\{x_1, \ldots, x_n\}$ from clients. At the end of each round it runs algorithm Stamp to generate timestamps for these bitstrings and returns the timestamps to the clients. Algorithm Verify is used to verify timestamps.

Stamp: On input of bitstrings $x_1, \ldots, x_n$ ($n \leq |\mathcal{S}|$), a hash tree [Mer90] is computed from leaves $x_1, \ldots, x_n$. Let r be the root of that hash tree and c_i be the hash chain corresponding to the path from leaf x_i to the root r (cf. Figure 2.1). The timestamp server publishes the root hash r at the repository Rep and for $i \in \{1, \ldots, n\}$, sends c_i as the response to request x_i. Hash chain c_i is also called a *timestamp* for bitstring x_i.

Verify: On input x, hash chain c, and a hash value r published at the repository, it is checked that c has allowed shape, $\mathsf{shape}(c) \in \mathcal{S}$, and c is a hash chain from x to r, $x \overset{c}{\rightsquigarrow} r$. The algorithm outputs 1 if these conditions hold, otherwise the algorithm outputs 0.

Security model

Intuitively, security of a timestamp scheme means that an adversary cannot *backdate* any x, i.e., generate an x and a hash chain c such that $\mathsf{Verify}(x, c, r) = 1$ for an r published at Rep before the generation of x. Such a condition is formalized, for example, in [BS04, BL06], where a two-stage adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is considered. At the first phase of the attack, $\mathcal{A}_1$ stores hashes into the repository Rep in an arbitrary way. At the second stage, $\mathcal{A}_2$ presents an *unpredictable* x, a hash chain c, and selects an r published at Rep so that $\mathsf{Verify}(x, c, r) = 1$. The unpredictability of x is essential as otherwise x could have been pre-computed by $\mathcal{A}_1$ before r is published and hence x could in fact be older than r.

The security definitions of [BS04, BL06] model the future as a computationally efficient stochastic process which may not be the case in the real world. There are no arguments against the future documents having arbitrary distributions. Additionally, the success of $\mathcal{A}$ is defined as the average over such a distribution and it might still be easy to backdate fixed documents. Having such arguments in mind, extraction-based security definitions for time-stamping have been explored in [BL07]. Intuitively, such conditions say that whenever $\mathcal{A}_1$ publishes a hash r to Rep and later $\mathcal{A}_2$ outputs a document x and a hash chain c with $\mathsf{Verify}(x, c, r) = 1$, then x must have been "known" by $\mathcal{A}_1$ when r was stored in Rep. Formally, this is expressed

by assuming the existence of an extraction algorithm $\mathcal{E}$ that depends on $\mathcal{A}_1$ and outputs a set of bitstrings X such that if $\mathcal{A}_2$ outputs (x, c) with $\mathsf{Verify}(x, c, r) = 1$ for some $r \in \mathsf{Rep}$, then $x \in X$ with overwhelming probability.

In section 5.2 and section 5.3 we propose extraction-based security definitions for timestamp schemes and long-term timestamp schemes in the ideal primitive model. We then analyze the security of the hash-based long-term timestamp scheme described in subsection 5.1.2 based on these definitions.

5.1.2. Hash-based long-term time-stamping

Next, we describe a hash-based long-term timestamp scheme based on the idea of Bayer et al. for extending the lifetime of a digital signature [BHS93]. Such a scheme has the property that the validity of existing timestamps can be prolonged by obtaining additional timestamps. In the following we assume that the short-term timestamp schemes available are described by the set $\mathcal{TS} = \{\mathsf{TS}_i\}_i$. Each timestamp scheme $\mathsf{TS}_i \in \mathcal{TS}$ is associated with a start time t_i^s and a breakage time t_i^b. The start time defines when the scheme becomes available and the breakage time defines after which time the timestamps created using this scheme are not considered valid anymore. Additionally, we assume the existence of a repository Rep that is used for publishing root hash values. The long-term timestamp scheme is defined by algorithm Stamp for creating an initial timestamp, algorithm Renew for renewing a timestamp, and algorithm Verify for verifying a timestamp.

Stamp: This algorithm gets as input a timestamp scheme identifier i and a sequence of bitstrings $x_1, \ldots, x_n$. It creates timestamps for the bitstrings using scheme TS_i by computing $(r, c_1, \ldots, c_n) \leftarrow \mathsf{TS}_i.\mathsf{Stamp}(x_1, \ldots, x_n)$. Then, the root hash r is published together with identifier i at the repository Rep. Let t be the time when r was published. For $j \in \{1, \ldots, n\}$, the algorithm responds to request x_j with timestamp $T_j = [(i, c_j, r, t)]$.

Renew: This algorithm gets as input a timestamp scheme identifier i' and a sequence of bitstrings and timestamps $(x_1, T_1), \ldots, (x_n, T_n)$. The algorithm renews the timestamps using scheme $\mathsf{TS}_{i'}$ as follows. First, it computes new timestamps $(r, c_1, \ldots, c_n) \leftarrow \mathsf{TS}_{i'}.\mathsf{Stamp}(x_1 \| T_1, \ldots, x_n \| T_n)$. Then, it publishes the root hash r together with the timestamp scheme identifier i' at the repository Rep. Let t be the time when r is published. For $j \in \{1, \ldots, n\}$, the algorithm sends (i', c_j, r, t) as the response to request (x_j, T_j). The client receiving (i', c_j, r, t) updates its timestamp by appending (i', c_j, r, t) to T_j.

Verify: This algorithm takes as input a bitstring x, a long-term timestamp $T = (C_1 \| \ldots \| C_n)$, where $C_j = (i_j, c_j, r_j, t_j)$, a time t, a reference $\mathtt{R}$ to the trusted repository Rep, and a set of admissible timestamp schemes $\mathcal{TS} = \{\mathsf{TS}_i\}_i$. For $j \in \{1, \ldots, n\}$, it is verified that $\mathsf{TS}_{i_j}.\mathsf{Verify}((x \| C_1 \| \ldots \| C_{j-1}), c_j, r_j) = 1$,

$(i_j, r_j) \in \mathtt{R}[t_j]$, and $t^b_{i_j} > t_{j+1}$. The algorithm outputs 1 if these conditions hold, otherwise it outputs 0.

5.2. Extractable time-stamping

In this section we define extractable time-stamping for (short-term) timestamp schemes (cf. subsection 5.1.1). Informally, extractability of a timestamp scheme TS means that if a root hash r is published at the repository at time t and later someone comes up with a bitstring x and a hash chain c such that $\mathsf{TS.Verify}(x, c, r) = 1$, then x must have been known at time t. Our notion of extractable time-stamping is reminiscent of PrA hash functions [DRS09] and knowledge-binding commitments [BL07]. After giving the definition of an extractable timestamp scheme, we analyze the security of the timestamp scheme construction described in subsection 5.1.1 when instantiated with a PrA hash function.

5.2.1. Security definition

More formally, extractability of a timestamp scheme TS^P with an ideal primitive P is defined using an experiment $\mathbf{Exp}^{\text{ExTs}}$ (Listing 5.1). In this experiment an adversary $\mathcal{A}$ publishes root hashes at the repository. The adversary also uses the ideal primitive P and queries to P are recorded in an advice string adv. The definition of extractable time-stamping requires the existence of an extractor $\mathcal{E}$ with the following properties. Whenever $\mathcal{A}$ publishes a root hash r, the extractor $\mathcal{E}$ extracts from r and the advice adv, a set of supposedly timestamped bitstrings X. At the end, $\mathcal{A}$ outputs a bitstring x, a timestamp c, and a root hash r. It wins if c is valid for x and r, r was published, and x was not extracted.

Definition 5.1 (Extractable time-stamping)**.** *Let $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$. A timestamp scheme TS^P using ideal primitive P is ϵ-secure* extractable *(ExTs) if for all integers $p_{\mathcal{E}}$, $p_{\mathcal{A}}$, and $q_{\mathcal{E}}$, there is a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every $p_{\mathcal{A}}$-step adversary $\mathcal{A}$ that makes at most q calls to* Rep*,*

$$\mathbf{Adv}^{\text{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{E}) = \Pr\left[\mathbf{Exp}^{\text{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{E}) = 1\right] \leq \epsilon(p_{\mathcal{E}}, p_{\mathcal{A}}, q) \ .$$

5.2.2. Security analysis

We analyze the security of the hash-based timestamp scheme from subsection 5.1.1. We first recall various properties of hash chain shapes [BL13] which are useful for analyzing the security of hash-based timestamp schemes.

Definition 5.2. *We say that a timestamp scheme associated with allowed shapes $\mathcal{S}$ is N-bounded if $|\mathcal{S}| \leq N$.*

Listing 5.1: The extractable time-stamping experiment $\mathbf{Exp}^{\text{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{E})$.

$(x,c,r) \leftarrow \mathcal{A}^{\mathsf{P},\mathsf{Rep}}$;
return 1 **if** $\mathsf{TS.Verify}(x,c,r) = 1 \wedge r \in \mathtt{R} \wedge x \notin \mathtt{L}[r]$ **else** 0;

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
$\mathsf{adv} \mathrel{+}= (m,z)$;
return z;

oracle $\mathsf{Rep}(r)$:
$X \leftarrow \mathcal{E}(\mathsf{adv}, r)$;
$\mathtt{R} \leftarrow \mathtt{R} \cup \{r\}$;
$\mathtt{L}[r] \leftarrow X$;
return X;

Definition 5.3. *An N-bounded timestamp scheme is said to be* shape-compact*, if the length of allowed hash chains does not exceed $2\log_2 N$.*

Next we proof a bound on the security of the hash-based timestamp scheme from subsection 5.1.1 if instantiated as N-bounded and shape compact.

Theorem 5.1. *The timestamp scheme from subsection 5.1.1 instantiated as N-bounded and shape compact and with an ϵ-secure* PrA *hash function H^P is ϵ'-secure extractable with*

$$\epsilon'(p_{\mathcal{E}}, p_{\mathcal{A}}, q) = \epsilon\left(\frac{\alpha \cdot p_{\mathcal{E}}}{2N\log_2 N}, \beta\cdot(p_{\mathcal{A}} + 2qN\log_2 N), 2qN\log_2 N\right) ,$$

for some small constants α and β.

Proof. Let $\mathcal{S}$ be the set of allowed shapes associated with an N-bounded shape compact timestamp scheme TS and let $\mathcal{E}$ be an extractor for H^P that extracts a preimage given a hash value and the ideal primitive calls. Such an extractor exists if H^P is PrA-secure. Using $\mathcal{E}$ as a black box, we construct a list extractor $\mathcal{L}$ (Listing 5.2), which extracts timestamped bitstrings from published root hash values. Having as input a P-query string adv and a bitstring r (a hash value), the list extractor $\mathcal{L}$ extracts for every allowed hash-chain shape $\iota \in \mathcal{S}$, the corresponding hash chain in a top-down way, starting from r. Due to the shape-compactness, the step count of $\mathcal{L}$ is bounded by $\alpha \cdot p_{\mathcal{E}} \cdot N \cdot \max_{\iota\in\mathcal{S}} \mathsf{length}(\iota) \leq 2 \cdot \alpha \cdot p_{\mathcal{E}} \cdot N\log_2(N)$, for some small constant α and if $p_{\mathcal{E}}$ is a bound on the step count of $\mathcal{E}$.

Let $\mathcal{A}^{\mathsf{P},\mathsf{Rep}}$ be an adversary that participates in the ExTs-experiment denoted by $\mathbf{Exp}^{\text{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{L})$. We construct an adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ (Listing 5.3) that participates in the PrA-experiment $\mathbf{Exp}^{\text{PrA}}_{P,H}(\mathcal{B},\mathcal{E})$ as follows. The adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ runs algorithm $\mathcal{A}^{\mathsf{P},\mathsf{Rep}}$, where calls to the repository Rep are simulated using algorithm R^{Ex} (Listing 5.4).
Note that whenever $\mathcal{A}$ succeeds in the ExTs-experiment, it finds (x,c,r) such that $x \overset{c}{\rightsquigarrow} r$, $x \in \mathtt{R}$, and $x \notin \mathtt{L}[r]$. This means there must be k such that $H^P(c[k+1]) =$

Listing 5.2: List extractor $\mathcal{L}(\mathsf{adv}, r)$.

for $\iota \in \mathcal{S}$ **do**
  $x \leftarrow \mathcal{E}(\mathsf{adv}, r)$, $c \leftarrow []$, $k \leftarrow 0$;
  while $x \neq \perp$ ***and*** $k < \mathsf{length}(\iota)$ **do**
    $k \leftarrow k + 1$;
    $x \leftarrow \mathcal{E}(\mathsf{adv}, x_{\iota[k]})$ (where $x = x_0 \| x_1$);
  end
  $L[r] \leftarrow L[r] \cup \{x\}$;
end
return L;

Listing 5.3: PrA adversary $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$.

$(x, c, r) \leftarrow \mathcal{A}^{\mathsf{P},R^{\mathsf{Ex}}}$;
$\iota \leftarrow \mathsf{shape}(c)$;
if $\exists k : \mathtt{R}[c[k]_{\iota[k]}] = 1, \mathtt{V}[c[k]_{\iota[k]}] \neq c[k+1]$ **then**
  return $c[k+1]$;
else
  return x;
end

Listing 5.4: Rep simulator $R^{\mathsf{Ex}}(r)$.

for $\iota \in \mathcal{S}$ **do**
  $x \leftarrow \mathsf{Ex}(r)$, $k \leftarrow 0$;
  while $x \neq \perp$ ***and*** $k < \mathsf{length}(\iota)$ **do**
    $k \leftarrow k + 1$;
    $x \leftarrow \mathsf{Ex}(x_{\iota[k]})$;
  end
  $\underline{\mathtt{L}}[r] \leftarrow \underline{\mathtt{L}}[r] \cup \{x\}$;
end
return $\underline{\mathtt{L}}[r]$;

$c[k]_{\iota[k]}$ (where ι is the shape of c), but the extractor $\mathcal{E}$ failed to extract $c[k+1]$ from $c[k]_{\iota[k]}$. Therefore, after simulating $\mathcal{A}$, the adversary $\mathcal{B}$ obtains the hash chain c and finds the smallest k, such that $\mathtt{R}[c[k]_{\iota[k]}] = 1$ and $\mathtt{V}[c[k]_{\iota[k]}] \neq c[k+1]$, and outputs $c[k+1]$, or, if no such k exists, outputs x. We observe that $\mathcal{B}$ succeeds in the PrA experiment whenever $\mathcal{A}$ succeeds in the ExTs experiment,

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{L}) \leq \mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{B},\mathcal{E}) \ .$$

The step count of $\mathcal{B}^{\mathsf{P},\mathsf{Ex}}$ is $\mathcal{O}(p_{\mathcal{A}} + 2q_R N \log_2 N)$, where $p_{\mathcal{A}}$ is the step count of $\mathcal{A}$ and q_R is the number of calls to Rep. It follows that

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A},\mathcal{L}) \leq \mathbf{Adv}^{\mathrm{PrA}}_{P,H}(\mathcal{B},\mathcal{E}) \leq \epsilon(p_{\mathcal{E}}, \alpha \cdot (p_{\mathcal{A}} + 2q_R N \log_2 N), 2q_R N \log_2 N) \ ,$$

for some small constant α and where $p_{\mathcal{E}}$ is a bound on the step count of $\mathcal{E}$. The step count of $\mathcal{L}$ is $p_{\mathcal{L}} = \mathcal{O}(2p_{\mathcal{E}} N \log_2 N)$. We obtain that the timestamp scheme constructed from H^P is ϵ'-secure extractable with

$$\epsilon'(p_{\mathcal{L}}, p_{\mathcal{A}}, q_R) = \epsilon\left(\frac{\beta \cdot p_{\mathcal{L}}}{2N \log_2 N}, \alpha \cdot (p_{\mathcal{A}} + 2q_R N \log_2 N), 2q_R N \log_2 N\right) \ ,$$

for some small constant β. □

The $\log_2 N$ term can be eliminated if a more efficient tree-extractor is used.

Theorem 5.2. *The timestamp scheme from subsection 5.1.1 instantiated as N-bounded and shape compact and with an ϵ-secure* PrA *hash function H^P is ϵ'-secure extractable with*

$$\epsilon'(p_{\mathcal{E}}, p_{\mathcal{A}}, q) = \epsilon\left(\frac{\alpha \cdot p_{\mathcal{E}}}{2N}, \beta \cdot (p_{\mathcal{A}} + 2Nq), 2Nq\right) \ ,$$

for some small constants α and β.

Sketch. The list extractor used in the proof of Theorem 5.1, extracts a separate hash chain for each leaf of the hash tree. This means that each of the inner nodes of the tree are extracted many times. The efficiency of the extraction can be improved by avoiding redundant extraction of hash chains that partially overlap. This is what the tree extractor does and it leads to the improved security bound. □

5.3. Extractable long-term time-stamping

In this section we propose a security model for the hash-based long-term timestamp scheme described in subsection 5.1.2. First, we define what it means for a long-term timestamp scheme to be extractable secure. Then, we prove a lower bound on the security level of the long-term timestamp scheme construction described in subsection 5.1.2 based on the security level of the timestamp schemes used for timestamp renewal.

5.3.1. Security definition

We define extractable security of hash-based long-term timestamp schemes via experiment $\mathbf{Exp}^{\text{ExLTs}}$ (Listing 5.5). Similar to the definitions of PrA hash functions and extractable time-stamping without renewal, our definition of extractable long-term time-stamping uses an ideal primitive P which can only be called via an oracle P that records all calls to P in an advice string adv. The experiment also involves a global clock $\mathtt{Clock}$ as described in chapter 3.

The experiment involves an adversary $\mathcal{A}$ and an extractor $\mathcal{E}$. The adversary $\mathcal{A}$ may publish root hash values r at the repository Rep at any time by calling $\mathsf{Rep}(r)$. When Rep is called with root hash r at time t, it records r associated with t in a global table $\mathtt{R}$ (i.e., $\mathtt{R}[t] \leftarrow \mathtt{R}[t] \cup \{r\}$, where initially $\mathtt{R}[t] = \{\}$). Additionally, the extractor $\mathcal{E}$ on input adv and r extracts a set of bitstrings X that is stored associated with time t in a table $\mathtt{L}$ (i.e., $\mathtt{L}[t] \leftarrow \mathtt{L}[t] \cup X$, where initially $\mathtt{L}[t] = \{\}$). The goal of the adversary $\mathcal{A}$ is to produce (x, T, t) such that T is a valid long-term timestamp for bitstring x and time t, and x was not extracted at time t (i.e., $\mathsf{Verify}(x, c, t, \mathtt{R}) = 1$ and $x \notin \mathtt{L}[t]$).

Definition 5.4 (Extractable long-term time-stamping)**.** *Let $\mathcal{M}$ describe the available machines classes and $\mathcal{TS}$ describe the available timestamp schemes. Let $\epsilon : \mathbb{N}^4 \to \mathbb{R}_{[0,1]}$. A long-term timestamp scheme LTS^P, which uses an ideal primitive P, is ϵ-secure extractable (for $\mathcal{M}$ and $\mathcal{TS}$) if for all bounds $\rho_{\mathcal{E}}$, ρ_A, and q, there is a $\rho_{\mathcal{E}}$-bounded extractor $\mathcal{E} \in \mathcal{M}$, such that for every $\rho_{\mathcal{A}}$-step-bounded and q-call-bounded adversary $\mathcal{A} \in \mathcal{M}$, and for every time t:*

$$\mathbf{Adv}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathcal{TS}}(\mathcal{A}, \mathcal{E}, t) = \Pr\left[\mathbf{Exp}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathcal{TS}}(\mathcal{A}, \mathcal{E}, t) = 1\right] \leq \epsilon(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) \ .$$

Listing 5.5: Extractable long-term time-stamping, $\mathbf{Exp}^{\text{ExLTs}}_{P,\mathsf{LTS},\mathcal{TS}}(\mathcal{A}, \mathcal{E}, t^*)$.

$(x, T, t) \leftarrow \mathcal{A}^{\mathtt{Clock},\mathsf{P},\mathsf{Rep}}$;
return 1 **if** $\mathsf{LTS}.\mathsf{Verify}(x, T, t, \mathtt{R}, \mathcal{TS}) = 1 \wedge x \notin \mathtt{L}[t] \wedge \mathtt{time} \leq t^*$ **else** 0;

oracle $\mathtt{Clock}(t)$:
if $t > \mathtt{time}$ **then**
  $\mathtt{time} \leftarrow t$;
end

oracle $\mathsf{P}(m)$:
$z \leftarrow P(m)$;
$\mathsf{adv} \leftarrow \mathsf{adv}||(m, z)$;
return z;

oracle $\mathsf{Rep}(r)$:
$X \leftarrow \mathcal{E}(\mathsf{adv}, r)$;
$t \leftarrow \mathtt{time}$;
$\mathtt{R}[t] \leftarrow \mathtt{R}[t]||r$;
$\mathtt{L}[t] \leftarrow \mathtt{L}[t]||X$;
return X;

5.3.2. Security analysis

Before we analyze the security of the long-term timestamp scheme described in subsection 5.1.2, we adapt the notion of extractable time-stamping from section 5.2

to the long-term setting where the class of computing machines available changes over time.

Definition 5.5 (Extractable time-stamping (refined))**.** *Let $\mathcal{M}_{\mathcal{E}}$ and $\mathcal{M}_A$ be classes of machines and $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$. We say a non-renewable timestamp scheme TS is ϵ-secure extractable for adversaries of $\mathcal{M}_{\mathcal{A}}$ and extractors of $\mathcal{M}_{\mathcal{E}}$ if for all integers $p_{\mathcal{E}}$, $p_{\mathcal{A}}$, and $q_{\mathcal{E}}$, there exists a $p_{\mathcal{E}}$-step extractor $\mathcal{E} \in \mathcal{M}_{\mathcal{E}}$, such that for every $p_{\mathcal{A}}$-step adversary $\mathcal{A} \in \mathcal{M}_{\mathcal{A}}$ that makes at most q calls to* Rep*:*

$$\mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}}(\mathcal{A}, \mathcal{E}) \leq \epsilon(p_{\mathcal{E}}, p_{\mathcal{A}}, q) \ .$$

We now prove a bound on the security level of the long-term timestamp scheme described in subsection 5.1.2 in terms of the security level of the available timestamp schemes.

Theorem 5.3. *Let $\mathcal{M}$ describe the available computing machine classes and $\mathcal{TS} = \{\mathsf{TS}_i^P\}_i$ describe the available timestamp schemes, which use an ideal primitive P. If for every i, TS_i^P is ϵ_i-secure extractable for adversaries of $\mathcal{M}_{t_i^b}$ and extractors of $\mathcal{M}_{t_i^s}$, then the long-term timestamp scheme described in subsection 5.1.2 is ϵ-secure extractable with*

$$\epsilon(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) = \sum_{i \in \{i : t_i^b \leq t\}} \epsilon_i \left(\alpha \cdot \rho_{\mathcal{E}}(t_i^b), \beta \cdot \left(\rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b) \right), q(t_i^b) \right) \ ,$$

for some small constants α and β.

Proof. Let $\mathcal{L}_i \in \mathcal{M}_{t_i^s}$ be the list extractor for TS_i that exists because TS_i is ϵ_i-secure extractable for adversaries of $\mathcal{M}_{t_i^b}$ and extractors of $\mathcal{M}_{t_i^s}$. Using the $\mathcal{L}_i$'s as black boxes, we construct a long-term list extractor $\mathcal{L} \in \mathcal{M}$ in the following way (Listing 5.6). Having as input an ideal primitive advice string adv and a bitstring r, the list extractor $\mathcal{L}$ decomposes r into a timestamp scheme identifier i and a hash value r', checks if scheme i can be used at the current time, and if so, extracts a set of bitstrings X from r' using list extractor $\mathcal{L}_i$. The step count of $\mathcal{L}$ in this run is at most the step count of $\mathcal{L}_i$ (plus a small simulation overhead).

Listing 5.6: Long-term extractor $\mathcal{L}(\mathsf{adv}, r)$

```
r = (i, r');
if t_i^s ≤ time < t_i^b then
    return L_i(adv, r');
else
    return ⊥;
end
```

Let $\mathcal{A}$ be any adversary that participates in the ExLTs experiment. For every i, we construct an adversary $\mathcal{B}_{(i)}$ (Listing 5.7) that participates in the ExTs experiment as follows.[1] The adversary $\mathcal{B}_{(i)}^{\mathsf{P},\mathsf{Rep}}$ simluates a run of $\mathcal{A}^{\mathtt{Clock},\mathsf{P},R}$ where the clock `Clock` is simulated using Listing 5.9 and the repository R is simulated using Listing 5.8. The simulation is performed only while `time` does not exceed the breakage time t_i^b. The adversary $\mathcal{B}_{(i)}$ tries to find a moment $t'' < t_i^b$ when $\mathcal{A}$ submits (j, r'') to the repository to renew a tuple (x', i, c', r', t') valid with scheme TS_i and $\mathcal{A}$ submitted (i, r') to the repository, but x' has not been extracted. This violates the ExLTs-condition for TS_i.

Listing 5.7: ExLTs adversary $\mathcal{B}_{(i)}^{\mathsf{P},\mathsf{Rep}}$.

$(x, c, t) \leftarrow \mathcal{A}^{\mathtt{Clock},\mathsf{P},R}$ **while** $\mathtt{time} < t_i^b$
until $\exists j, x', c', r', t', r'', t''$ **such that**

- $(j, r'') \in \underline{\mathtt{R}}[t'']$ **and** $t_j^s < t'' < t_i^b, t_j^b$,
- $(x', i, c', r', t') \in \underline{\mathtt{L}}[t'']$,
- $S_i.\mathsf{Verify}(x', c', r') = 1$,
- $(i, r') \in \underline{\mathtt{R}}[t']$ **and** $t_i^s < t' < t_i^b$,
- $x' \notin \underline{\mathtt{L}}[t']$;

if $\exists$ ***such*** $(j, x', c', r', t', r'', t'')$ **then**
  return x';
else
  return $\perp$;
end

Define $I_t = \{i : t_i^b \leq t\}$. We observe that whenever $\mathcal{A}$ is successful until time t, one of $\{\mathcal{B}_i : t_i^b \leq t\}$ is successful,

$$\mathbf{Adv}_{P,\mathsf{LTS}}^{\mathrm{ExLTs}}(\mathcal{A}, \mathcal{L}, t) \leq \sum_{i \in I_t} \mathbf{Adv}_{P,\mathsf{TS}_i}^{\mathrm{ExTs}}(\mathcal{B}_{(i)}, \mathcal{L}_i) \ .$$

[1] We use the notation $\mathcal{B}_{(i)}$ to distinguish between $\mathcal{B}_{(i)}$ and the time-components $\mathcal{B} = (\mathcal{B}_0, \mathcal{B}_1, \ldots)$ of an adversary.

Listing 5.8: Repository simulator $R(r)$.

```
r = (j, r');
t ← time;
if t_j^s ≤ t < t_j^b then
    if j=i then
        X ← Rep(r');
    else
        X ← L_j(adv, r');
    end
else
    X ← ⊥;
end
R[t] ← R[t]‖r;
L[t] ← L[t]‖X;
return X;
```

Listing 5.9: Clock simulator $\texttt{Clock}(t)$.

```
if t > time then
    time ← t;
end
```

Assume $\mathcal{A}$ is $\rho_{\mathcal{A}}$-step-bounded and q-call-bounded, and assume $\mathcal{L}$ is $\rho_{\mathcal{L}}$-bounded. Then, for each i, the step count of $\mathcal{B}_{(i)}$ is $\mathcal{O}(\rho_{\mathcal{B}}(t_i^b) + q(t_i^b)\rho_{\mathcal{L}}(t_i^b))$. We obtain that for every t,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{ExLTs}}_{P,\mathsf{LTS}}(\mathcal{A},\mathcal{L},t) &\leq \sum_{i\in I_t} \mathbf{Adv}^{\mathrm{ExTs}}_{P,\mathsf{TS}_i}(\mathcal{B}_{(i)},\mathcal{L}_i) \\ &\leq \sum_{i\in I_t} \epsilon_i\left(\alpha\cdot\rho_{\mathcal{L}}(t_i^b), \beta\cdot(\rho_{\mathcal{B}}(t_i^b) + q(t_i^b)\rho_{\mathcal{L}}(t_i^b)), q(t_i^b)\right) \ ,\end{aligned}$$

for some small constants α and β. □

5.4. Evaluation

We evaluate which protection level the long-term timestamp scheme described in subsection 5.1.2 provides in a practical scenario. For our evaluation we consider a scenario where data is protected over a time period of Y years. The security level of the long-term timestamp scheme is evaluated in terms of the security level of the hash functions that are used to instantiate the available timestamp schemes. Here, we assume that all used hash functions have the same security level during their validity period, where by security level we mean a bound on the success probability of an adversary.

5.4.1. Scenario

We assume that a set $\mathcal{TS} = \{\mathsf{TS}_i^P\}_i$ of available hash-based timestamp schemes, where for each i, H_i^P is the hash function used by TS_i^P. We assume that the PrA-security of a hash function derives from the ratio of the adversary power p_A and the extractor power $p_{\mathcal{E}}$, and is influenced by the number of repository calls q and a base security level δ. Concretely, we assume that each hash function H_i is ϵ-secure PrA until its breakage time t_i^b with $\epsilon(p_{\mathcal{E}}, p_A, q) = \frac{p_A}{p_{\mathcal{E}}} q\delta$. Furthermore, we assume that each timestamp scheme TS_i is N-bounded and shape compact, which means that each timestamp round up to N timestamps are generated. For our practical security analysis we neglect the constants α and β derived in Theorem 5.2 and Theorem 5.3 as they are in most cases close to 1.

By Theorem 5.2 we obtain that each timestamp scheme TS_i is ϵ'-secure extractable until time t_i^b with

$$\epsilon'(p_{\mathcal{E}}, p_{\mathcal{A}}, q) \leq \epsilon\left(\frac{p_{\mathcal{E}}}{2N}, p_{\mathcal{A}} + 2Nq, 2Nq\right) = \frac{p_{\mathcal{A}} + 2Nq}{p_{\mathcal{E}}}(2N)^2 q\delta \ .$$

Furthermore, by Theorem 5.3 we obtain that the long-term timestamp scheme is ϵ''-secure long-term extractable with

$$\begin{aligned}\epsilon''(\rho_{\mathcal{E}}, \rho_{\mathcal{A}}, q, t) &\leq \sum_{i \in I_t} \epsilon' \left(\rho_{\mathcal{E}}(t_i^b), \rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b), q(t_i^b)\right) \\ &\leq \sum_{i \in I_t} \frac{\rho_{\mathcal{A}}(t_i^b) + q(t_i^b)\rho_{\mathcal{E}}(t_i^b) + 2Nq(t_i^b)}{\rho_{\mathcal{E}}(t_i^b)} (2N)^2 q(t_i^b)\delta \\ &\leq \sum_{i \in I_t} \left(\frac{\rho_{\mathcal{A}}(t_i^b)}{\rho_{\mathcal{E}}(t_i^b)} + \left(\frac{2N}{\rho_{\mathcal{E}}(t_i^b)} + 1\right) q(t_i^b)\right) (2N)^2 q(t_i^b)\delta \ .\end{aligned}$$

We assume that the adversary and the extractor have the same computation power, i.e., $\frac{\rho_{\mathcal{A}}(t)}{\rho_{\mathcal{E}}(t)} = 1$, and we observe that any reasonable extractor $\mathcal{E}$ extracts at least $2N$ bitstrings before the breakage time of a scheme, i.e., $\rho_{\mathcal{E}}(t_i^b) \geq 2N$. Let time be denoted in years and assume that each year at most L new timestamp schemes become available, and at most R root hashes are published at the repository, i.e., $|I_t| = |\{i : t_i^b \leq t\}| \leq tL$ and $q(t) \leq tR$. We obtain the following bound on the security level of the long-term timestamp scheme:

$$\epsilon''(\rho_{\mathcal{E}}, \rho_B, q, t) \leq 12t^3(NR)^2 L\delta \ .$$

5.4.2. Results

In Figure 5.1 we show the security level of the long-term timestamp scheme for different time spans Y and parameters N, L, R, and δ. The default parameters are $Y = 100$, $L = 10$, $N = 2^{32}$, $R = 365$, and $\delta = 2^{-192}$.

By the upper left graph of Figure 5.1, we observe a cubic security loss over time for the case that the security level of the used hash function remains constant. Using a base security level of $\delta = 2^{-192}$ for the hash function, the security level of the long-term timestamp scheme after 1 year is 2^{-104}, after 10 years it drops to 2^{-94}, and after 100 years it drops to 2^{-84}. There is a linear loss in security for increasing the number L of short-term timestamp schemes used per year, as depicted in the upper right graph. Allowing a larger number R of hash values to be published at the repository results in a quadratic security loss, as can be seen in the middle left graph. The number N of timestamps that can be issued per root hash is an important factor because in practice it may be large. The security level decreases quadratically when N is increased, as can be seen in the middle right graph. Using $N = 2^{16}$ results in security level 2^{-116} after 100 years, while using $N = 2^{48}$ results in security level 2^{-52}. Finally, the bottom graph shows how the security of the long-term timestamp scheme depends linear on the base security level δ of the used hash functions.

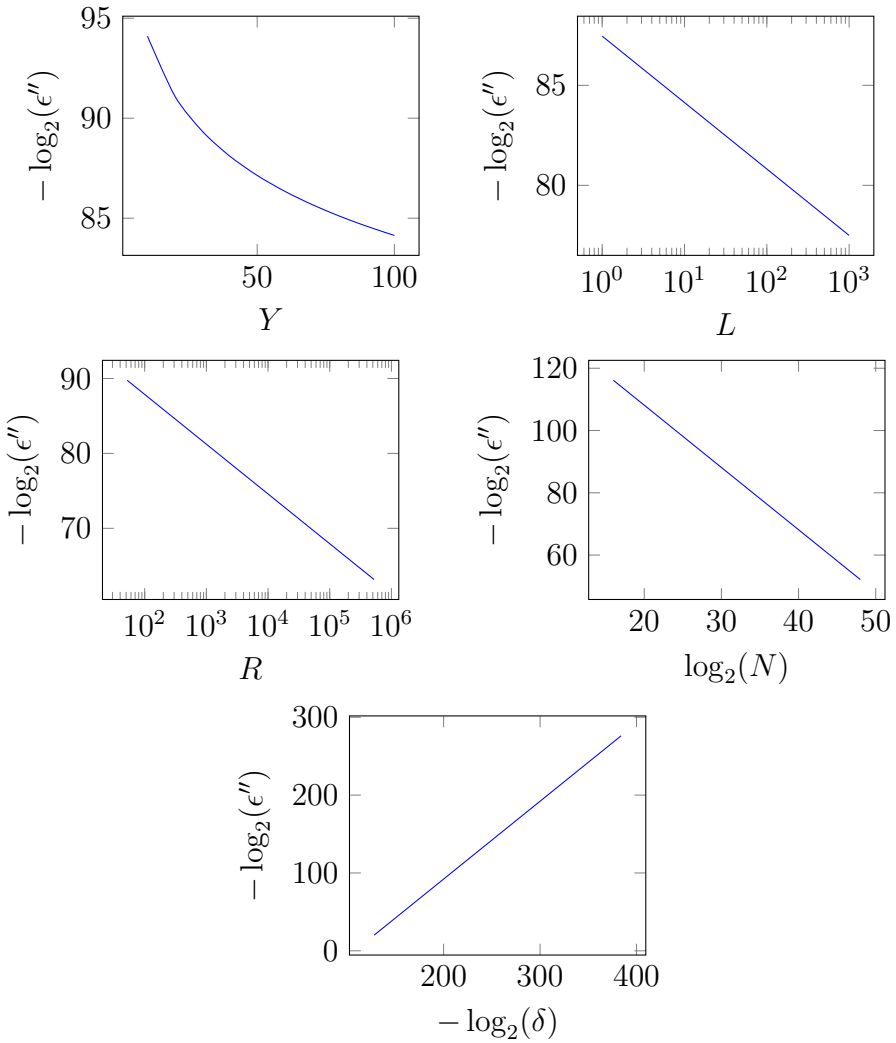

Figure 5.1.: Evaluation of the security level ϵ'' of long-term time-stamping when run for Y years, and with parameters L, R, N, and δ. Here, L is the number of new short-term timestamp schemes per year, R is the number of published root hashes per short-term scheme, N is the number of documents covered by one root hash, and δ is the base security level of the hash functions.

6. Long-Term Commitments via Extractable-Binding Commitments

Cryptographic commitments are either unconditionally hiding or unconditionally binding, but cannot be both. As a consequence, the security of commonly used commitment schemes is threatened in the long-term, when adversaries become computationally much more powerful.

Contribution. We improve over this situation by putting forward a new notion of commitment schemes, so called *long-term commitment schemes*. These schemes allow for adjusting the protection level after the initial commitment. On the way to constructing long-term commitment schemes, we find that such a construction seems impossible solely based on classically binding commitments. We thus propose the notion of extractable-binding commitments, which requires that the committer "knows" the committed message at the time of the commitment. We then present a construction of an extractable-binding long-term commitment scheme based on extractable-binding (short-term) commitments. Finally, we prove the security of our construction using the computational model described in chapter 3.

Publications. This chapter is based on publication [G3].

6.1. On the (im)possibility of constructing long-term secure commitments

Commitment schemes are important building blocks in many cryptographic protocols and also important mechanisms in secure electronic archival storage (see Part II). They enable a party to commit to (potentially) secret data, so that the commitment process reveals no information (the *hiding property*) and the committing party cannot afterwards deny or modify the data (the *binding property*).

The classical binding definition provided in subsection 2.3.4 is widely used, but it also appears unsuitable in certain scenarios. For example, it cannot be achieved in the standard Universal Composability Framework [CF01] and it seems insufficient for commitments in the presence of quantum computers [Unr16]. In the following, we will discuss why it also appears insufficient for constructing long-term commitments.

It is a well-known fact that no commitment scheme can be simultaneously hiding and binding against an adversary with unlimited computational resources [May97, BCMS97]. A different approach to enabling long-term security (in contrast to unconditional security) is to employ a renewal technique such as the one by Bayer, Haber, and Stornetta for prolonging the validity of a digital signature [HS91, BHS93]. We adapt their idea to commitments as follows. A document X and its commitment $c = C(X)$ are renewed (at time t) by creating a new commitment $c^* = C^*(X, c)$. If later (say, at $t' > t$) the cryptographic mechanisms of C are broken but those of C^* are still secure, and it is believed that the mechanisms in C were secure at t, then after opening the *renewed commitment* (c, c^*) (and seeing X), it is still reasonable to believe that X was indeed the committed message.

In order to prove the security of such a construction, our goal is to reduce its long-term security to the short-term security of the individual commitments. If we rely on the classical binding property, however, such a reduction seems impossible to obtain (at least using black-box techniques). Indeed, we would have to show that if there exists an efficient adversary $\mathcal{A}$ that creates an ambiguous opening of (c, c^*) at time t', then there exists an efficient adversary $\mathcal{A}'$ that successfully creates an ambiguous opening of C at time $t < t'$. But such an adversary cannot exist because C is still binding secure at time t and $\mathcal{A}$ does provide not produce any useful outputs before time $t' > t$.

We solve this problem by using an *extraction-based* binding property, which requires that for every efficient committer $\mathcal{A}_1$, there exists an efficient extractor $\mathcal{E}$, such that $\mathcal{E}$, on input the random coins of $\mathcal{A}_1$, predicts the message decommitted by any efficient $\mathcal{A}_2$. This allows us to extract information from the long-term adversary before it finishes its computation.

6.2. Extractable-binding commitments

For extractable-binding commitments we require that the committer, who produces a commitment and later opens the commitment to some message, must already know the message at the time of the commitment. As we will see in section 6.4, extractable-binding commitments are sufficient to construct long-term commitment schemes.

6.2.1. Definition

Extractable-binding is defined in experiment $\mathbf{Exp}^{\mathrm{ExtBind}}$ with a two staged adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and an extractor $\mathcal{E}$. Here, $\mathcal{A}_1$ is the committing algorithm who outputs a commitment c and an advice string s that contains information how to open c. The extractor $\mathcal{E}$ gets as input the random coins ω of $\mathcal{A}_1$ and the advice string s. It outputs a message m^* which is the extracted message. Afterwards, the second stage

adversary $\mathcal{A}_2$ is run on input of advice string s and outputs a commitment opening (m, w). A commitment scheme is extractable-binding if there exists an extractor such that for any commitment opening, the extracted message equals the opened message.

Definition 6.1 (Extractable binding)**.** *Let $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$. A (non-interactive) commitment scheme* CS *is ϵ-extractable-binding, if for every integers p_1 and p_2, for every p_1-step adversary $\mathcal{A}_1$, there exists a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every p_2-step adversary $\mathcal{A}_2$:*

$$\mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) = \Pr\left[\mathbf{Exp}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) = 1\right] \leq \epsilon(p_1, p_{\mathcal{E}}, p_2) \ .$$

Listing 6.1: $\mathbf{Exp}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2)$ (Extractable-Binding Experiment)

$ck \leftarrow \mathsf{CS.Setup}$;
$(s, c) \xleftarrow{\omega} \mathcal{A}_1(ck)$;
$m^* \leftarrow \mathcal{E}(ck, \omega)$;
$(m, w) \leftarrow \mathcal{A}_2(s)$;
return 1 ***if*** $\mathsf{CS.Verify}(ck, m, c, w) = 1 \wedge m \neq m^*$ ***else*** 0;

6.2.2. Relation to other cryptographic notions

In the following we discuss the relation of extractable-binding commitments to other cryptographic notions.

Classical-binding commitments

The following theorem shows that every extractable binding commitment scheme is classical-binding.

Theorem 6.1 (EB⇒CB)**.** *If* CS *is ϵ-extractable binding, then* CS *is ϵ'-classical-binding, where $\epsilon'(p) = \inf\{2 \cdot \epsilon(p, p_{\mathcal{E}}) : p_{\mathcal{E}} \in \mathbb{N}\}$.*

Proof. Let $\mathcal{A}$ be a p-step Bind-adversary and let $\mathcal{E}$ be any extractor guaranteed by the assumption. We construct an ExtBind adversary $(\mathcal{A}_1, \mathcal{A}_2)$ as follows. The first stage $\mathcal{A}_1(ck)$ runs $\mathcal{A}(ck)$ (with random string ω) to obtain (c, m, w, m', w') and returns (s, c), where $s = (ck, \omega, m, w, m', w')$. The second stage $\mathcal{A}_2(s)$ parses s to obtain $(ck, \omega, m, w, m', w')$, tosses a coin $b \leftarrow \{0, 1\}$, and outputs (m, w) if $b = 0$, and otherwise outputs (m', w'). If $\mathcal{A}$ is successful in $\mathbf{Exp}^{\mathrm{Bind}}_{\mathsf{CS}}$, then $(\mathcal{A}_1, \mathcal{A}_2)$ is successful in $\mathbf{Exp}^{\mathrm{ExtBind}}_{\mathsf{CS}}$ with probability $\frac{1}{2}$ independent of the extractor $\mathcal{E}$. As the running time of $(\mathcal{A}_1, \mathcal{A}_2)$ is about p ($\mathcal{A}_2$ just parses and tosses a coin), we have $\mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) \geq \frac{1}{2}\mathbf{Adv}^{\mathrm{Bind}}_{\mathsf{CS}}(\mathcal{A})$ and hence,

$$\mathbf{Adv}^{\mathrm{Bind}}_{\mathsf{CS}}(\mathcal{A}) \leq 2 \cdot \mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) \leq 2 \cdot \epsilon(p, p_{\mathcal{E}}) \ ,$$

where $p_{\mathcal{E}}$ is the step count of $\mathcal{E}$. □

Knowledge-binding commitments

Knowledge-binding commitments were proposed by Buldas and Laur [BL07] as a new security notion for time-stamping. In comparison to extractable-binding, knowledge-binding is defined for multi-message commitments (e.g., list commitments or set commitments) and the extractor depends on the second stage adversary A_2, but A_2 gets an additional advice string which is not available to the extractor. Theorem 6.2 implies that the single-message variant of knowledge-binding (Definition 6.2) implies extractable-binding up to a small security loss due to the reduction.

Definition 6.2 (Knowledge binding (single-message))**.** *Let $\epsilon : \mathbb{N}^2 \to \mathbb{R}_{[0,1]}$. A commitment scheme CS is ϵ-knowledge-binding, if for all integers p and $p_{\mathcal{E}}$, for every p-step adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, there exists a $p_{\mathcal{E}}$-step extractor $\mathcal{E}$, such that for every advice string $a \in \{0,1\}^p$:*

$$\mathbf{Adv}^{\mathrm{KBind}}_{\mathsf{CS}}(\mathcal{A}, \mathcal{E}, a) = \Pr\left[\mathbf{Exp}^{\mathrm{KBind}}_{\mathsf{CS}}(\mathcal{A}, \mathcal{E}, a) = 1\right] \leq \epsilon(p, p_{\mathcal{E}}) \ ,$$

where $\mathbf{Exp}^{\mathrm{KBind}}_{\mathsf{CS}}$ is defined in Listing 6.2.

Listing 6.2: Knowledge-binding experiment $\mathbf{Exp}^{\mathrm{KBind}}_{\mathsf{CS}}(\mathcal{A}, \mathcal{E}, a)$.

```
ck ← CS.Setup;
(s, c) ←^{ω1} A1(ck);
m' ← E(ck, ω1);
(m, w) ← A2(a, s);
if CS.Verify(ck, m, c, w) = 1 ∧ m ≠ m' then
    return 1;
else
    return 0;
end
```

Theorem 6.2 (KB⇒EB)**.** *If CS is ϵ-knowledge-binding, then CS is ϵ'-extractable-binding with $\epsilon'(p_1, p_{\mathcal{E}}, p_2) = \epsilon(p_1 + \alpha \cdot p_2 \log p_2, p_{\mathcal{E}})$, for some constant α.*

Proof. Consider an arbitrary p_1-step adversary $\mathcal{A}_1$ that acts in terms of the extractable binding experiment. We define $\mathcal{A}'_2(a, s)$ as an universal probabilistic Turing machine that uses the first argument a as a program for an arbitrary p_2-step machine $\mathcal{A}_2(s)$. According to [AB09], $\mathcal{A}'_2$ runs in $O(p_2 \log p_2)$ steps, i.e., $\alpha \cdot p_2 \log p_2$ for some constant α. Consider the KBind-adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}'_2)$ that runs in $(p_1 + \alpha \cdot p_2 \log p_2)$ steps and let $\mathcal{E}$ be a $p_{\mathcal{E}}$-step KBind-extractor. Note that $\mathcal{E}$ can also be considered as an ExtBind-extractor. We observe that $\mathcal{A}$ succeeds in the

KBind-experiment if and only if $(\mathcal{A}_1, \mathcal{A}_2)$ succeeds in the ExtBind-experiment and hence,

$$\mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) = \mathbf{Adv}^{\mathrm{KBind}}_{\mathsf{CS}}(\mathcal{A}, \mathcal{E}, a) \leq \epsilon(p_1 + \alpha \cdot p_2 \log p_2, p_{\mathcal{E}}) \ .$$

□

Extractable collision-resistant hash functions

In [BCC+17], Bitansky et al. propose the notion of an extractable and collision-resistant hash function (ECRH). They show how to construct an ECRH from several different knowledge assumptions (i.e., Knowledge of Exponent and Knowledge of Knapsack). In the following we show that if there exists an ECRH, then there exists an extractable-binding (and statistically hiding) commitment scheme. In particular, we show that the commitment scheme by Halevi and Micali [HM96], from here on referred to by HM96, instantiated with an ECRH is extractable-binding.

In the following, we first provide the relevant definitions, then prove that extractable collision-resistant hash functions are extractable-binding, and finally prove that HM96 instantiated with an extractable-binding hash function is extractable-binding.

Definition 6.3 (Collision-resistance)**.** *We say a keyed function (K, F) is ϵ-collision resistant if for all $t \in \mathbb{N}$, for all t-bounded algorithms $\mathcal{A}$:*

$$\mathbf{Adv}^{(\mathrm{K,F})}_{\mathsf{CR}}(\mathcal{A}) = \Pr\left[\begin{matrix} k \leftarrow K, (x, x') \leftarrow \mathcal{A}(k) : \\ F(k, x) = F(k, x') \end{matrix}\right] \leq \epsilon(t) \ .$$

Definition 6.4 (Extractability)**.** *We say a keyed function (K, F) is ϵ-extractable if for all $t_{\mathcal{A}}, t_{\mathcal{E}} \in \mathbb{N}$, for all $t_{\mathcal{A}}$-bounded algorithms $\mathcal{A}$, there exists a $t_{\mathcal{E}}$-bounded algorithm $\mathcal{E}$, such that:*

$$\mathbf{Adv}^{(\mathrm{K,F})}_{\mathsf{EX}}(\mathcal{A}, \mathcal{E}) = \Pr\left[\begin{matrix} k \leftarrow K, y \xleftarrow{r} \mathcal{A}(k), x \leftarrow \mathcal{E}(k, r) : \\ y \in \mathsf{Image}(F(k, \cdot)) \wedge F(k, x) \neq y \end{matrix}\right] \leq \epsilon(t_{\mathcal{A}}, t_{\mathcal{E}}) \ .$$

Definition 6.5 (Extractable-binding hash function)**.** *A keyed hash function (K, H) is ϵ-extractable-binding if for all $t_1, t_2, t_{\mathcal{E}} \in \mathbb{N}$, it holds that for every t_1-bounded algorithm $\mathcal{A}_1$, there exists a $t_{\mathcal{E}}$-bounded algorithm $\mathcal{E}$, such that for all t_2-bounded algorithms $\mathcal{A}_2$:*

$$\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}) = \Pr\left[\begin{matrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}(k, r) : \\ H(k, x) = y \wedge x \neq x' \end{matrix}\right] \leq \epsilon(t_1, t_2, t_{\mathcal{E}}) \ .$$

Theorem 6.3 (ECRH⇒EBHF)**.** *Let (K, H) be a keyed hash function that is ϵ_{EX}-extractable and ϵ_{CR}-collision-resistant. There exist constants α and β such that (K, H) is ϵ_{EB}-extractable-binding for*

$$\epsilon_{\mathsf{EB}}(t_1, t_2, t_{\mathcal{E}}) = \epsilon_{\mathsf{EX}}(\alpha \cdot t_1, t_{\mathcal{E}}) + \epsilon_{\mathsf{CR}}(\beta \cdot (t_1 + t_2 + t_{\mathcal{E}}))$$

Proof. Let (K, H) be a keyed hash function that is ϵ_{CR}-collision-resistant and ϵ_{EX}-extractable. Fix any constants t_1, t_2, and $t_{\mathcal{E}}$, t_1-bounded algorithm $\mathcal{A}_1$, and t_2-bounded algorithm $\mathcal{A}_2$.

Let $\mathcal{B}_1$ be an algorithm that on input k, runs $\mathcal{A}_1(k) \to (y, s)$ and outputs y. Define $t_{\mathcal{B}_1}$ as the maximum over the computational resources required to evaluate $\mathcal{B}_1$ on input k for $k \in K$.

For any t, let $\mathcal{E}_{\mathcal{B}_1,t}$ be a t-bounded algorithm such that $\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EX}}(\mathcal{B}_1, \mathcal{E}_{\mathcal{B}_1,t}) \leq \epsilon_{\mathsf{EX}}(t_{\mathcal{B}_1}, t)$. Such an algorithm exists because (K, H) is ϵ_{EX}-extractable.

Let t' be the maximum constant such that $\mathcal{E}_{\mathcal{B}_1,t'}$ is $t_{\mathcal{E}}$-bounded for all inputs (k, r), where $k \in K$ and r is a random string produced by $\mathcal{A}_1(k)$. Define $\mathcal{E}_{\mathcal{A}_1} = \mathcal{E}_{\mathcal{B}_1,t'}$.

Let $\mathcal{B}_2$ be an algorithm that on input k, runs $\mathcal{A}_1(k) \xrightarrow{r} (y, s)$, $\mathcal{A}_2(s) \to x$, and $\mathcal{E}_{\mathcal{A}_1}(k, r) \to x'$, and outputs (x, x').

We now derive an upper bound on the probability that $\mathcal{A}_2$ outputs a preimage and $\mathcal{E}_{\mathcal{A}_1}$ fails to extract a preimage or outputs a different preimage. We observe that

$$\begin{aligned}
\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}_{\mathcal{A}_1}) &= \Pr\begin{bmatrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ H(k, x) = y \wedge x \neq x' \end{bmatrix} \\
&= \Pr\begin{bmatrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ H(k, x) = y \wedge x \neq x' \wedge H(x') = y \end{bmatrix} \\
&\quad + \Pr\begin{bmatrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ H(k, x) = y \wedge x \neq x' \wedge H(x') \neq y \end{bmatrix} .
\end{aligned}$$

Furthermore, we observe that

$$\begin{aligned}
\Pr\begin{bmatrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ H(k, x) = y \wedge x \neq x' \wedge H(x') = y \end{bmatrix} &= \Pr\begin{bmatrix} k \leftarrow K, (x, x') \leftarrow \mathcal{B}_2(k) : \\ H(k, x) = H(k, x') \wedge x \neq x' \end{bmatrix} \\
&= \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{CR}}(\mathcal{B}_2) ,
\end{aligned}$$

and

$$\begin{aligned}
\Pr\begin{bmatrix} k \leftarrow K, (y, s) \xleftarrow{r} \mathcal{A}_1(k), \\ x \leftarrow \mathcal{A}_2(s), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ H(k, x) = y \wedge x \neq x' \wedge H(x') \neq y \end{bmatrix} &\leq \Pr\begin{bmatrix} k \leftarrow K, y \xleftarrow{r} \mathcal{B}_1(k), x' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k, r) : \\ y \in \mathsf{Image}(H(k, \cdot)) \wedge H(x') \neq y \end{bmatrix} \\
&= \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EX}}(\mathcal{B}_1, \mathcal{E}_{\mathcal{A}_1}) .
\end{aligned}$$

It follows that

$$\begin{aligned}&\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}_{\mathcal{A}_1}) \\ &\leq \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EX}}(\mathcal{B}_1, \mathcal{E}_{\mathcal{A}_1}) + \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{CR}}(\mathcal{B}_2) \\ &\leq \epsilon_{\mathsf{EX}}(t_{\mathcal{B}_1}, t_{\mathcal{E}}) + \epsilon_{\mathsf{CR}}(t_{\mathcal{B}_2}) \ .\end{aligned}$$

We conclude that there exist constants α and β such that (K, H) is ϵ_{EB}-extractable-binding with

$$\epsilon_{\mathsf{EB}}(t_1, t_2, t_{\mathcal{E}}) = \epsilon_{\mathsf{EX}}(\alpha \cdot t_1, t_{\mathcal{E}}) + \epsilon_{\mathsf{CR}}(\beta \cdot (t_1 + t_2 + t_{\mathcal{E}})) \ .$$

□

We briefly state the commitment scheme described in [HM96] with a keyed hash function.

Construction 6.1 (HM96)**.** *Let $M, N \in \mathbb{N}$, (K, H) be a keyed hash function with domain $\{0,1\}^*$ and codomain $\{0,1\}^N$, and* UHF *be a universal hash function with domain $\{0,1\}^M$ and codomain $\{0,1\}^N$. The commitment scheme* HM96 *proposed in [HM96] associated with a keyed hash function (K, H) is constituted by algorithms* Setup*,* Commit*, and* Verify *defined as follows.*

Setup$() \to k$**:** *Sample $K \to k$ and output k.*

Commit$(k, m) \to (c, d)$**:** *Sample $\{0,1\}^M \to d$. Compute $H(k, m) \to s$ and $H(k, d) = y$. Sample* UHF $\to h$ *with $h(d) = s$. Set $c = (y, h)$.*

Verify$(k, m, c, d) \to b$**:** *Let $c = (y, h)$. If $H(k, m) = h(d)$ and $H(k, d) = y$, set $b = 1$. Otherwise set $b = 0$.*

The next theorem states that HM96 is extractable-binding if the used hash function (K, H) is extractable-binding.

Theorem 6.4 (EBHF⇒EBCOM)**.** *Let (K, H) be an ϵ_{H}-extractable-binding hash function. There exist constants α and β such that the commitment scheme* HM96 *instantiated with (K, H) is ϵ_{C}-extractable-binding for*

$$\epsilon_C(t_1, t_2, t_{\mathcal{E}}) = 2 \cdot \epsilon_H(\alpha \cdot (t_1 + t_{\mathcal{E}}), \alpha \cdot t_2, \beta \cdot t_{\mathcal{E}}) \ .$$

Proof. Let (K, H) be an ϵ_{H}-extractable-binding hash function. Fix any constants t_1, t_2, and $t_{\mathcal{E}}$, t_1-bounded algorithm $\mathcal{A}_1$, and t_2-bounded algorithm $\mathcal{A}_2$.

Let $\mathcal{B}_1$ be an algorithm that on input k, runs $\mathcal{A}_1(k) \to ((y, h), s)$ and outputs (y, s). Define $t_{\mathcal{B}_1}$ as the maximum over the computational resources required to evaluate $\mathcal{B}_1$ on input k for $k \in K$.

For any t and t', let $\mathcal{E}_{\mathcal{B}_1,t,t'}$ be a t'-bounded algorithm such that for all for t-bounded algorithms $\mathcal{B}_2$, $\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{B}_1,\mathcal{B}_2,\mathcal{E}_{\mathcal{B}_1,t,t'}) \leq \epsilon_{\mathsf{H}}(t_{\mathcal{B}_1},t,t')$. Such an algorithm exists because (K,H) is ϵ_{H}-extractable-binding.

Let $\mathcal{B}_2$ be an algorithm that on input s, computes $\mathcal{A}_2(s) \to (m,x)$ and outputs x. Define $t_{\mathcal{B}_2}$ as the maximum over the computational resources required to evaluate $\mathcal{B}_2$ on input s for $k \in K$ and $((y,h),s) \in \mathcal{A}_1(k)$.

For any t and t', let $\mathcal{C}_{1,t,t'}$ be an algorithm that on input k, computes $\mathcal{A}_1(k) \xrightarrow{r} ((y,h),s)$, $\mathcal{E}_{\mathcal{B}_1,t,t'}(r,k) \to x'$, $h(x') \to z'$, and outputs (z',s). Define $t_{\mathcal{C}_{1,t,t'}}$ as the maximum over the computational resources required to evaluate $\mathcal{C}_{1,t,t'}$ on input k for $k \in K$.

Let $\mathcal{C}_2$ be an algorithm that on input s, computes $\mathcal{A}_2(s) \to (m,x)$ and outputs m. Define $t_{\mathcal{C}_2}$ as the maximum over the computational resources required to evaluate $\mathcal{C}_2$ on input s for $k \in K$ and $((y,h),s) \in \mathcal{A}_1(k)$.

For any t_3,t_4,t_5,t_6, let $\mathcal{E}_{\mathcal{C}_{1,(t_3,t_4)},(t_5,t_6)}$ be a t_5-bounded algorithm such that for all t_6-bounded algorithms $\mathcal{C}_2$, it is $\mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{C}_{1,(t_3,t_4)},\mathcal{C}_2,\mathcal{E}_{\mathcal{C}_{1,(t_3,t_4)},(t_5,t_6)}) \leq \epsilon_{\mathsf{H}}(t_{\mathcal{C}_{1,t_3,t_4}},t_6,t_5)$. Such an algorithm exists because (K,H) is ϵ_{H}-extractable-binding.

For any t_3,t_4,t_5,t_6, let $\mathcal{E}_{\mathcal{A}_1,(t_3,t_4,t_5,t_6)}$ be an algorithm that on input (k,r), computes $\mathcal{E}_{\mathcal{B}_1,t_3,t_4}(k,r) \xrightarrow{r'} x'$, $\mathcal{E}_{\mathcal{C}_{1,(t_3,t_4)},(t_5,t_6)}(k,r\|r') \to m'$, and outputs m'.

Let t' be the maximum constant such that $\mathcal{E}_{\mathcal{A}_1,t',t_4,t',t_6}$ is $t_{\mathcal{E}}$-bounded for all inputs (k,r), where $k \in K$ and $((h,y),s) \xleftarrow{r} \mathcal{A}_1(k)$. Define $\mathcal{E}_{\mathcal{A}_1} = \mathcal{E}_{\mathcal{A}_1,t',t_{\mathcal{B}_2},t',t_{\mathcal{C}_2}}$, $\mathcal{E}_{\mathcal{B}_1} = \mathcal{E}_{\mathcal{B}_1,t',t_{\mathcal{B}_2}}$, and $\mathcal{E}_{\mathcal{C}_1} = \mathcal{E}_{\mathcal{C}_{1,(t',t_{\mathcal{B}_2})},(t',t_{\mathcal{C}_2})}$.

We now prove an upper bound on the probability that for a commitment produced by $\mathcal{A}_1$, algorithm $\mathcal{A}_2$ finds an opening for a message different from the one extracted by $\mathcal{E}_{\mathcal{A}_1}$. We observe that

$$
\begin{aligned}
&\mathbf{Adv}^{\mathsf{HM96}}_{\mathsf{EB}}(\mathcal{A}_1,\mathcal{A}_2,\mathcal{E}_{\mathcal{A}_1}) \\
&= \Pr\left[\begin{array}{c} k \leftarrow \mathsf{HM96}.K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ m' \leftarrow \mathcal{E}_{\mathcal{A}_1}(k,r) : \\ \mathsf{HM96}.V(k,m,c,d) \wedge m \neq m' \end{array}\right] \\
&= \Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k,r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m \neq m' \end{array}\right] \\
&= \Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k,r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m \neq m' \wedge x \neq x' \end{array}\right] \\
&\quad + \Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k,r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m \neq m' \wedge x = x' \end{array}\right]
\end{aligned}
$$

by the definitions of $\mathsf{HM96}$ and $\mathcal{E}_{\mathcal{A}_1}$ and the law of total probability. Next, we observe that

$$\begin{aligned}
&\Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k, r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m{\neq}m' \wedge x \neq x' \end{array}\right] \\
&= \Pr\left[\begin{array}{c} k \leftarrow K, (y,s) \xleftarrow{r} \mathcal{B}_1(k), x \leftarrow \mathcal{B}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k, r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m{\neq}m' \wedge x \neq x' \end{array}\right] \\
&\leq \Pr\left[\begin{array}{c} k \leftarrow K, (y,s) \xleftarrow{r} \mathcal{B}_1(k), x \leftarrow \mathcal{B}_2(s), \\ x' \leftarrow \mathcal{E}_{\mathcal{B}_1}(k,r) : \\ H(k,x) = y \wedge x \neq x' \end{array}\right] \\
&= \mathbf{Adv}^{(\mathrm{K,F})}_{\mathsf{EB}}(\mathcal{B}_1, \mathcal{B}_2, \mathcal{E}_{\mathcal{B}_1})
\end{aligned}$$

and

$$\begin{aligned}
&\Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k, r\|r') : \\ H(k,x) = y \wedge H(k,m) = h(x) \wedge m{\neq}m' \wedge x \neq x' \end{array}\right] \\
&\leq \Pr\left[\begin{array}{c} k \leftarrow K, ((h,y),s) \xleftarrow{r} \mathcal{A}_1(k), (m,x) \leftarrow \mathcal{A}_2(s), \\ x' \xleftarrow{r'} \mathcal{E}_{\mathcal{B}_1}(k,r), m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k, r\|r') : \\ H(k,m) = h(x) \wedge m{\neq}m' \end{array}\right] \\
&= \Pr\left[\begin{array}{c} k \leftarrow K, (z,s) \xleftarrow{r} \mathcal{C}_1(k), x \leftarrow \mathcal{C}_2(s), \\ m' \leftarrow \mathcal{E}_{\mathcal{C}_1}(k,r) : \\ H(k,m) = z \wedge m{\neq}m' \end{array}\right] \\
&= \mathbf{Adv}^{(\mathrm{K,F})}_{\mathsf{EB}}(\mathcal{C}_1, \mathcal{C}_2, \mathcal{E}_{\mathcal{C}_1})
\end{aligned}$$

by the definitions of $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{C}_1$, and $\mathcal{C}_2$. In combination, we obtain

$$\begin{aligned}
&\mathbf{Adv}^{\mathsf{HM96}}_{\mathsf{EB}}(\mathcal{A}_1, \mathcal{A}_2, \mathcal{E}_{\mathcal{A}_1}) \\
&\leq \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{B}_1, \mathcal{B}_2, \mathcal{E}_{\mathcal{B}_1}) + \mathbf{Adv}^{(\mathrm{K,H})}_{\mathsf{EB}}(\mathcal{C}_1, \mathcal{C}_2, \mathcal{E}_{\mathcal{C}_1}) \\
&\leq \epsilon_{\mathsf{H}}(t_{\mathcal{B}_1}, t_{\mathcal{B}_2}, t_{\mathcal{E}_{\mathcal{B}_1}}) + \epsilon_{\mathsf{H}}(t_{\mathcal{C}_1}, t_{\mathcal{C}_2}, t_{\mathcal{E}_{\mathcal{C}_1}}) \ .
\end{aligned}$$

We conclude that there exist constants α and β, such that the commitment scheme $\mathsf{HM96}$ instantiated with an ϵ_H-extractable-binding hash function (K, H) is ϵ_C-extractable-binding for

$$\epsilon_C(t_1, t_2, t_{\mathcal{E}}) = 2 \cdot \epsilon_H(\alpha \cdot (t_1 + t_{\mathcal{E}}), \alpha \cdot t_2, \beta \cdot t_{\mathcal{E}}) \ .$$

□

Statistical hiding security of $\mathsf{HM96}$ is proven in [HM96]. It follows that if there exists an extractable-binding hash function, then there exists a statistically hiding and extractable-binding commitment scheme.

6.3. Long-term commitment schemes

6.3.1. Scheme definition

A long-term secure commitment scheme allows to generate commitments that remain binding for long periods of time, e.g., decades or even centuries. Such a commitment is generated in an initial commitment generation procedure and needs to be updated periodically in order to remain valid. For initial commitment and also for updating a commitment, a short-term secure commitment scheme is chosen whose security must be provided until the next update.

The following definition captures long-term commitment schemes more formally. Here, by a *a reference to a commitment function* we mean a pointer to the commitment algorithm of a chosen commitment scheme. A commitment function Com gets as input a message m and outputs a commitment c. By *a trusted commitment verification function* we mean a function that allows to verify commitments that have been generated in the past. A commitment verification function Ver gets as input a message m, a commitment c, a witness w, and a time t. It outputs 1 if (m, w) is a valid opening for commitment c at time t and it outputs 0 in any other case.

Definition 6.6 (Long-term commitment scheme)**.** *A long-term commitment scheme is a triple of algorithms* Commit, $\mathsf{Recommit}$, Verify, *where:*

- Commit *gets as input a reference to a commitment function* Com*, and a message* m*. It outputs a state* S*, a witness* W*, and a commitment* c*.*
- $\mathsf{Recommit}$ *gets as input a state* S *and a reference to a commitment function* Com*. It outputs a state* S'*, a renewed witness* W*, and a commitment* c*.*
- Verify *gets as input a reference to a trusted commitment verification function* Ver*, a message* m*, a list of commitments* C*, and a witness* W*. It outputs a boolean* b*, where* $b = 1$ *if* C *is a valid long-term commitment for* m *and* $b = 0$ *if* C *is invalid.*

A long-term commitment scheme $(\mathsf{Commit}, \mathsf{Recommit}, \mathsf{Verify})$ is used by a committer $\mathcal{A}$ for committing to a message m in the presence of a verifier $\mathcal{B}$, as follows. In the protocol, the verifier $\mathcal{B}$ maintains a list of commitment values C.

Initial commitment. The committer $\mathcal{A}$ chooses a secure commitment scheme CS, generates the initial commitment $(S, W, c) \leftarrow \mathsf{Commit}(\mathsf{CS.Com}, m)$ and sends the commitment c to the verifier $\mathcal{B}$. When $\mathcal{B}$ receives c, it reads the current time t and sets $C \leftarrow [(t, c)]$.

Recommitment. $\mathcal{A}$ chooses a new commitment scheme CS, generates a new commitment by running $(S, W, c) \leftarrow \mathsf{Recommit}(\mathsf{CS.Com}, S)$, and sends c to $\mathcal{B}$. When $\mathcal{B}$ receives c, it reads the current time t and appends (t, c) to C.

Verification. $\mathcal{A}$ sends the witness W to $\mathcal{B}$. When $\mathcal{B}$ receives W, it uses a trusted commitment verification function Ver and runs $b \leftarrow \mathsf{Verify}(\mathsf{Ver}, m, C, W)$. The verifier $\mathcal{B}$ accepts the commitment if $b = 1$, otherwise $\mathcal{B}$ rejects.

6.3.2. Security definition

In the following, we present definitions of binding and hiding for long-term commitment schemes.

Commitment scheme instances. In the following security definitions we define experiments that involve a set of commitment scheme instances $\mathcal{C} = \{\mathsf{CS}_1, \mathsf{CS}_2, \ldots\}$. Each instance CS_i is associated with a start time t_i^s and an end time t_i^b. At the start time, public commitment parameters are generated and after the end time, commitments generated using this instance are considered invalid (e.g. not considered secure anymore).

The following long-term experiments use the clock oracle described by Listing 6.3. This oracle, in addition to defining the time, also checks whether new commitment instances have become available and generates and outputs the public commitment parameters accordingly.

Listing 6.3: The clock oracle $\texttt{Clock}(t)$.

```
CK ← [];
if t > time then
    time ← t;
    forall i ∈ {j : t_j^s = t} do
        ck ← CS_i.Setup;
        CK[i] ← ck;
        CK ← CK‖(i, ck);
    end
end
return CK;
```

Hiding. The unconditionally hiding experiment $\mathbf{Exp}^{\mathrm{LtHide}}$ for long-term commitment schemes (Listing 6.4) is defined similar to the unconditionally hiding experiment $\mathbf{Exp}^{\mathrm{Hide}}$ for (short-term) commitment schemes. It considers an unbounded adversary $\mathcal{A}$ which is given access to oracle `Clock`. The adversary $\mathcal{A}$ generates two messages (m_0, m_1) and an advice string s. Then, a coin is flipped $\mathtt{b} \leftarrow_{\$} \{0, 1\}$ and the adversary may call oracles `Com` and `ReCom`. If the `Com` oracle is called, an initial long-term commitment to m_b is generated and returned to the adversary. When the `ReCom` oracle is called, a chosen long-term commitment is renewed and the adversary

gets the renewed commitment value. At some point in time, the adversary $\mathcal{A}$ guesses which message has been committed to by outputting a bit b' and wins if it guesses correctly, i.e. if $b' = \mathtt{b}$. Unconditional hiding for long-term commitment schemes is defined as follows.

Definition 6.7 (Long-term hiding)**.** *A long-term commitment scheme* LCS *is unconditionally hiding if for any set* $\mathcal{C}$ *of unconditionally hiding commitment schemes, for any adversary* $\mathcal{A}$*:*

$$\mathbf{Adv}^{\mathrm{LtHide}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A}) = \Pr\left[\mathbf{Exp}^{\mathrm{LtHide}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A}) = 1\right] = \frac{1}{2}\ .$$

Listing 6.4: The long-term hiding experiment $\mathbf{Exp}^{\mathrm{LtHide}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A})$.

$(m_0, m_1, s) \leftarrow \mathcal{A}^{\mathtt{Clock}}$;
$\mathtt{b} \leftarrow_{\$} \{0,1\}$;
$b' \leftarrow \mathcal{A}^{\mathtt{Clock},\mathtt{Com},\mathtt{Recom}}(s)$;
if $b' = \mathtt{b}$ **then**
  return 1;
else
  return 0;
end

oracle $\mathtt{Com}(i,j)$:
$(S_i, W_i, c) \leftarrow \mathsf{LCS.Commit}(\mathsf{Com}_j, m_{\mathtt{b}})$;
return c;

oracle $\mathtt{Recom}(i,j)$:
$(S_i, W_i, c) \leftarrow \mathsf{LCS.Recommit}(S_i, \mathsf{Com}_j)$;
return c;

Binding. The binding experiment $\mathbf{Exp}^{\mathrm{LtExtBind}}$ for long-term commitment schemes (Listing 6.5) considers a two-staged long-term adversary $(\mathcal{A}_1, \mathcal{A}_2)$, which is given access to an oracle `Clock`, and an extractor $\mathcal{E}$. The first-stage adversary $\mathcal{A}_1$ outputs an initial commitment c and an advice string s using random coins ω. The extractor $\mathcal{E}$ then gets the public commitment parameters `CK` and random coins ω and outputs an extracted message m'. The initial commitment c is recorded by running $\mathtt{Rec}(c)$. Afterwards, the second-stage adversary $\mathcal{A}_2$ gets the advice string s, runs the long-term commitment protocol (during which he may call `Rec` several times), and finally outputs a message m and a long-term witness W. The adversary wins if it is finished

early enough ($\texttt{time} \leq \tau$), (m, W) is a valid opening for the commitment sequence $\texttt{C}$, and m differs from the extracted message m'.

Definition 6.8 (Long-term binding)**.** *Let $\mathcal{M}$ describe the available machine classes and $\mathcal{C}$ describe the available commitment scheme instances. Let $\epsilon : \mathbb{N}^5 \to \mathbb{R}_{[0,1]}$. A long-term commitment scheme* LCS *is ϵ-binding (for $\mathcal{M}$ and $\mathcal{C}$) if for any bounds ρ_1, $\rho_{\mathcal{E}}$, ρ_2, and q, for any ρ_1-bounded deterministic adversary $\mathcal{A}_1 \in \mathcal{M}$, there exists a $\rho_{\mathcal{E}}$-bounded extractor $\mathcal{E} \in \mathcal{M}$, such that for any ρ_2-bounded $\mathcal{A}_2 \in \mathcal{M}$ that is q-call-bounded, and any time t:*

$$\mathbf{Adv}^{\mathrm{LtExtBind}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A}_1,\mathcal{E},\mathcal{A}_2,t) = \Pr\left[\mathbf{Exp}^{\mathrm{LtExtBind}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A}_1,\mathcal{E},\mathcal{A}_2,t) = 1\right] \leq \epsilon(\rho_1,\rho_{\mathcal{E}},\rho_2,q,t) \ .$$

Listing 6.5: Long-term extractable binding, $\mathbf{Exp}^{\mathrm{LtExtBind}}_{\mathsf{LCS},\mathcal{C}}(\mathcal{A}_1,\mathcal{E},\mathcal{A}_2,t)$.

$(s, c) \xleftarrow{\omega} \mathcal{A}_1^{\texttt{Clock}}$;
$m' \leftarrow \mathcal{E}(\texttt{CK}, \omega)$;
$\texttt{Rec}(c)$;
$(m, W) \leftarrow \mathcal{A}_2^{\texttt{Clock},\texttt{Rec}}(s)$;
if $\texttt{time} \leq t$ ***and*** $m \neq m'$ ***and***
$\mathsf{LCS}.\mathsf{Verify}(\texttt{Ver}, m, \texttt{C}, W) = 1$ **then**
  return 1;
else
  return 0;
end

oracle $\texttt{Rec}(c)$:
$t \leftarrow \texttt{time}$;
$\texttt{C} \leftarrow \texttt{C} \| (t, c)$;

function $\texttt{Ver}(m, c, w, t)$:
$c = (i, c')$;
if $t_i^s \leq \texttt{time}$ ***and*** $t < t_i^b$ **then**
  $b \leftarrow \mathsf{CS}_i.\mathsf{Verify}(\texttt{CK}[i], m, c, w)$;
  return b;
else
  return 0;
end

6.4. Construction and security analysis

In the following we describe a long-term commitment scheme construction and prove its security. The construction is based on the ideas of Bayer, Haber, and Stornetta [HS91, BHS93] for renewing timestamps. The main idea is that in order to renew

a commitment, a new commitment is given to the message and the opening value of the previous commitment. In the following, we refer to this construction by LtCom. We remark that we here consider commitment schemes that allow messages of arbitrary length, i.e., with message space $\{0,1\}^*$ (e.g., [HM96]).

Construction 6.2. *The algorithms* Commit, $\mathsf{Recommit}$, *and* Verify *of the long-term commitment scheme* LtCom *are defined as follows:*

- $\mathsf{Commit}(\mathsf{Com}, m)$*: Run* $(c, w) \leftarrow \mathsf{Com}(m)$*. Set* $W = [w]$ *and* $S = (m, W)$*. Output* S*,* W*, and* c*.*
- $\mathsf{Recommit}(S, \mathsf{Com})$*: Run* $(c, w) \leftarrow \mathsf{Com}(S)$*. Let* $S = (m, [w_1, \ldots, w_n])$*. Set* $W = [w_1, \ldots, w_n, w]$ *and* $S' = (m, W)$*. Output* S'*,* W*, and* c*.*
- $\mathsf{Verify}(\mathsf{Ver}, m, C, W)$*: Let* $C = [(c_1, t_1), \ldots, (c_n, t_n)]$ *and* $W = [w_1, \ldots, w_n]$*. Compute* $b \leftarrow \bigwedge_{i=1}^{n} \mathsf{Ver}((m, [w_1, \ldots, w_{i-1}]), c_i, w_i, t_{i+1})$*, where* t_{n+1} *is the current time. Output* b*.*

Now we analyze the security of LtCom. First, we note that LtCom is unconditionally hiding, if unconditionally hiding commitment schemes are used. This is because then the individual commitments are independent of each other and of the committed messages and decommitments. Next, we prove that LtCom is long-term binding, given that the accumulated security level of the used short-term commitment schemes is sufficiently small. For the long-term binding security analysis we first refine the extractable-binding definition for short-term commitment schemes to make it meaningful in the long-term security model. More specifically, we refine the definition such that the computational models of the adversary and the extractor may be restricted to a certain machine classes.

Definition 6.9 (Extractable-binding (refined))**.** *Let* $\mathcal{M}_{\mathcal{A}}$ *and* $\mathcal{M}_{\mathcal{E}}$ *be machine classes and* $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$*. We say a commitment scheme* CS *is* ϵ*-extractable-binding for adversaries of* $\mathcal{M}_{\mathcal{A}}$ *and extractors of* $\mathcal{M}_{\mathcal{E}}$ *if for every integers* p_1 *and* p_2*, for every* p_1*-step adversary* $\mathcal{A}_1 \in \mathcal{M}_{\mathcal{A}}$*, there exists a* $p_{\mathcal{E}}$*-step extractor* $\mathcal{E} \in \mathcal{M}_{\mathcal{E}}$*, such that for every* p_2*-step adversary* $\mathcal{A}_2 \in \mathcal{M}_{\mathcal{A}}$*:*

$$\mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2) \leq \epsilon(p_1, p_{\mathcal{E}}, p_2) \ .$$

The following theorem establishes a bound on the binding security of the long-term commitment scheme LtCom in terms of the binding security of each of the chosen short-term commitment schemes.

Theorem 6.5. *Let* $\mathcal{M}$ *describe the available machine classes and* $\mathcal{C} = \{\mathsf{CS}_i\}_i$ *describe the available commitment scheme instances. For every* i*, assume that the*

commitment scheme CS_i *is* ϵ_i*-extractable-binding for adversaries of* $\mathcal{M}_{t_i^b}$ *and extractors of* $\mathcal{M}_{t_i^s}$*. Then,* LtCom *is* ϵ*-binding with*

$$\epsilon(\rho_1, \rho_{\mathcal{E}}, \rho_2, q, t) = \sum_{i \in \{i : t_i^b \leq t\}} \epsilon_i \left(\rho_{\mathcal{A}}(t_i^b), \rho_{\mathcal{E}}(t_i^b), \alpha\right)$$

and $\rho_{\mathcal{A}}(t) = \rho_1(t) + \rho_2(t) + q(t) * \rho_{\mathcal{E}}(t)$*, for a constant* α*.*

Proof. To prove the theorem, we first describe the extractor algorithm that extracts the committed message from the first commitment of a long-term adversary in $\mathbf{Exp}^{\text{LtExtBind}}$. Then we describe a reduction from the security of the long-term commitment scheme to the aggregated security of the commitment schemes $\mathcal{C}$.

We start by describing the long-term extractor $\mathcal{E}$ (Listing 6.6) that we construct using the first-stage adversary $\mathcal{A}_1$ and the short-term extractors $\{\mathcal{E}_i\}_i$ corresponding to commitment schemes $\{\mathsf{CS}_i\}_i$. When the long-term extractor is called with input $(\mathtt{CK}, \omega)$, it runs $\mathcal{A}_1$ using commitment parameters $\mathtt{CK}$ and random coins ω for obtaining the commitment c. The extractor then decomposes c into a commitment scheme identifier i and a commitment value c'. Afterwards, it checks if scheme i is currently usable and if this is the case, it runs the extractor $\mathcal{E}_i$, corresponding to scheme i, with input the corresponding key $\mathtt{CK}[i]$ and random coins ω. The long-term extractor outputs the message m returned by the short-term extractor $\mathcal{E}_i$.

Listing 6.6: Long-term extractor $\mathcal{E}(\mathtt{CK}, \omega)$

```
Simulate A_1 using commitment parameters CK and random coins ω to
  obtain commitment c;
c = (i, c');
if t_i^s ≤ time < t_i^b then
    m ← E_i(CK[i], ω);
else
    m ←⊥;
end
return m;
```

Next, we describe the reduction from a successful long-term adversary to a set of short-term commitment adversaries, of which at least one is successful. Let $(\mathcal{A}_1, \mathcal{A}_2)$ be an adversary pair that participates in $\mathbf{Exp}^{\text{LtExtBind}}$. For each commitment scheme $\mathsf{CS}_{\underline{i}} \in \mathcal{C}$, we construct a corresponding short-term adversary pair $(\mathcal{B}_{\underline{i},1}, \mathcal{B}_{\underline{i},2})$ that participates in $\mathbf{Exp}^{\text{ExtBind}}$. The adversary $\mathcal{B}_{\underline{i},1}$ (Listing 6.7) simulates the experiment $\mathbf{Exp}^{\text{LtExtBind}}$ with $(\mathcal{A}_1, \mathcal{A}_2)$ until it successfully obtains two different message-witness pairs which are valid for the same commitment with respect to commitment scheme $\mathsf{CS}_{\underline{i}}$, or the lifetime of the commitment scheme $\mathsf{CS}_{\underline{i}}$ is over. It passes the two different message-witness pairs (m_0, w_0, m_1, w_1) to $\mathcal{B}_{\underline{i},2}$ and commits c. The adversary $\mathcal{B}_{\underline{i},2}$

(Listing 6.8) gets as input the message-witness pairs, flips a coin $b \leftarrow_\$ \{0,1\}$, and outputs (m_b, w_b).

For every time t, define $I_t = \{i : t_i^b \leq t\}$ as the set of indices of the schemes whose lifetime expires until time t. We observe that for every successful run of the long-term adversary $(\mathcal{A}_1, \mathcal{A}_2)$, there is $i \in I_t$ such that the run of the short-term adversary $(\mathcal{B}_{i,1}, \mathcal{B}_{i,2})$ is successful. It follows that

$$\mathbf{Adv}^{\mathrm{LtExtBind}}_{\mathsf{LtCom},\mathcal{C}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2, t) \leq \sum_{i \in I_t} \mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}_i}(\mathcal{B}_{i,1}, \mathcal{E}_i, \mathcal{B}_{i,2}) \ .$$

Assume $\mathcal{A}_1$ is ρ_1-bounded, $\mathcal{A}_2$ is ρ_2-bounded, and $\mathcal{E}$ is $\rho_{\mathcal{E}}$-bounded. Additionally assume that $\mathcal{A}_2$ makes at most $q(t)$ receiver oracle calls until time t. Define $\rho_{\mathcal{B}}(t) := \rho_1(t) + \rho_2(t) + q(t) * \rho_{\mathcal{E}}(t)$. The step count of $\mathcal{B}_{i,1}$ is bounded by $\rho_{\mathcal{B}}(t_i^b)$. It follows that for every t,

$$\begin{aligned}\mathbf{Adv}^{\mathrm{LtExtBind}}_{\mathsf{LtCom},\mathcal{C}}(\mathcal{A}_1, \mathcal{E}, \mathcal{A}_2, t) &\leq \sum_{i \in I_t} \mathbf{Adv}^{\mathrm{ExtBind}}_{\mathsf{CS}_i}(\mathcal{B}_{i,1}, \mathcal{E}_i, \mathcal{B}_{i,2}) \\ &\leq \sum_{i \in I_t} \epsilon_i \left(\rho_{\mathcal{B}}(t_i^b), \rho_{\mathcal{E}}(t_i^b), 1\right) \ .\end{aligned}$$

□

6.5. Evaluation

We evaluate the security loss over time for the long-term commitment scheme described in section 6.3. For our evaluation we consider a scenario where a commitment should last for a time period t. The security level of the long-term commitment scheme is evaluated in terms of the security level of the short-term commitment schemes that are used. For convenience, we assume that all used commitment schemes have the same security level before they become insecure. Here, by the security level we mean a bound on the success probability of the adversary. Concretely, consider a long-term commitment scheme that uses short-term commitment schemes $\mathcal{C} = \{\mathsf{CS}_i\}_i$. We assume that the extractable-binding security of a commitment scheme derives from the ratio of the adversary power p_A and the extractor power $p_{\mathcal{E}}$, multiplied by a base security level δ. Hence, we assume each CS_i is ϵ-secure extractable-binding before its breakage time t_i^b with $\epsilon(p_1, p_{\mathcal{E}}, p_2) = \frac{p_1+p_2}{p_{\mathcal{E}}}\delta$. By Theorem 6.5, we obtain that the long-term commitment scheme is ϵ'-secure extractable-binding with

$$\epsilon'(\rho_1, \rho_{\mathcal{E}}, \rho_2, q, t) = \delta * \sum_{i \in \{i : t_i^b \leq t\}} \frac{\rho_1(t_i^b) + \rho_2(t_i^b) + q(t_i^b) * \rho_{\mathcal{E}}(t_i^b) + \alpha}{\rho_{\mathcal{E}}(t_i^b)}$$

Listing 6.7: ExtBind adversary $\mathcal{B}_{\underline{i},1}(\underline{ck})$ constructed from long-term adversary $(\mathcal{A}_1, \mathcal{A}_2)$ and long-term extractor $\mathcal{E}$.

run
  $(s^*, c^*) \xleftarrow{\omega^*} \mathcal{A}_1^{\underline{\texttt{Clock}}}$;
  $\texttt{C} \leftarrow \texttt{C} \cup \{c^*\}$;
  $m^* \leftarrow \mathcal{E}(\texttt{CK}, \omega^*)$;
  $\texttt{M} \leftarrow \texttt{M} \cup \{m^*\}$;
  $(m^{**}, W^{**}) \leftarrow \mathcal{A}_2^{\underline{\texttt{Clock}},\underline{\texttt{Rec}}}(s^*)$;
  $\texttt{M} \leftarrow \texttt{M} \cup \{m^{**}\}$;
  $\texttt{W} \leftarrow \texttt{W} \cup W^{**}$;
until $(\texttt{time} \geq t_{\underline{i}}^b)$ ***or***
$(\exists m, m' \in \texttt{M}, w, w' \in \texttt{W}, (\underline{i}, c) \in \texttt{C}:$
$\mathsf{CS}_{\underline{i}}.\mathsf{Verify}(\texttt{CK}[\underline{i}], m, c, w) = \mathsf{CS}_{\underline{i}}.\mathsf{Verify}(\texttt{CK}[\underline{i}], m', c, w') = 1)$;
return $((m, w, m', w'), c)$;

simulator $\underline{\texttt{Clock}}(t)$:
$CK \leftarrow []$;
if $t > \texttt{time}$ **then**
  forall $i \in \{j : \texttt{time} < t_j^s \leq t\}$ **do**
    if $i = \underline{i}$ **then**
      $ck \leftarrow \underline{ck}$;
    else
      $ck \leftarrow \mathsf{CS}_i.\mathsf{Setup}$;
    end
    $\texttt{CK}[i] \leftarrow ck$;
    $CK \leftarrow CK \| (i, ck)$;
  end
  $\texttt{time} \leftarrow t$;
end
return CK;

simulator $\underline{\texttt{Rec}}(c)$:
$\texttt{C} \leftarrow \texttt{C} \cup \{c\}$;
$c = (i, c')$;
if $t_i^s \leq \texttt{time}$ **then**
  Let ω be the random coins consumed by $\mathcal{A}_1$ and $\mathcal{A}_2$ until this point;
  $m \leftarrow \mathcal{E}_i(\texttt{CK}[i], \omega, c')$;
  $m = (m', w')$;
  $\texttt{M} \leftarrow \texttt{M} \cup \{m'\}$;
  $\texttt{W} \leftarrow \texttt{W} \cup \{w'\}$;
end

Listing 6.8: ExtBind adversary $\mathcal{B}_{\underline{i},2}(s)$.

$s = (m_0, w_0, m_1, w_1)$;
$b \leftarrow_\$ \{0, 1\}$;
return (m_b, w_b);

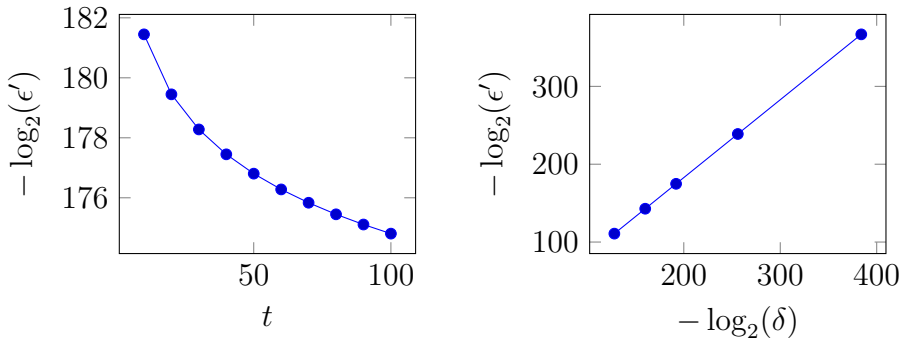

Figure 6.1.: Evaluation of the security level ϵ' of the long-term commitment scheme LtCom in terms of time t and short-term commitment security level δ. Here, we assume $R = 1$, $L = 5$, and we choose $\delta = 2^{-192}$ for evaluating over time t and we choose $t = 100$ for evaluating over δ.

for some constant α. Let the unit of time be years and assume that the number of commitment schemes that become available during each year is at most L and the number of renewals that are done per year is at most R, i.e., $|I_t| = |\{i : t_i^b \leq t\}| \leq t{*}L$ and $q(t) \leq t * R$. We suggest that it is reasonable that the computational power of the adversary is comparable to the computational power of the extractor. Hence, we assume that $\frac{\rho_1(t)}{\rho_\mathcal{E}} = \frac{\rho_2(t)}{\rho_\mathcal{E}} = 1$. We also observe that the step count α of the very simple second-stage adversary described in Listing 6.8 should be smaller than $\rho_\mathcal{E}$, hence, we assume $\frac{\alpha}{\rho_\mathcal{E}(t)} \leq 1$. We obtain the following bound on the long-term security level:

$$\epsilon'(\rho_1, \rho_\mathcal{E}, \rho_2, q, t) \leq 3 * t^2 * LR\delta \ .$$

In Figure 6.1, we show how the long-term security level develops over time and for different choices of the base security level δ. We observe that after 100 years the security level drops from 2^{-182} to 2^{-175} (for $R = 1$, $L = 5$, and $\delta = 2^{-192}$). We also observe that there is a constant difference of roughly 2^{18} between the base security level δ and the long-term security level ϵ' if the time period is kept fixed.

Part II.

Constructions

7. LINCOS: A Long-Term Secure Storage System providing Integrity and Confidentiality

Previous long-term protection schemes only supported protection of either integrity or confidentiality. However, much long-lived digitally stored information requires protection of both properties simultaneously (e.g., governmental documents, electronic health records).

Contribution. We present the long-term secure storage system LINCOS, which is the first system that provides simultaneous long-term protection of integrity and confidentiality. We first construct a long-term integrity scheme that does not leak any information about the protected data by using information-theoretically hiding commitments and computationally secure timestamps. We then combine the obtained long-term integrity scheme with proactive secret sharing in order to construct LINCOS and we provide a security analysis for this construction. We also present an implementation and experimental evaluation of LINCOS that uses quantum key distribution for establishing information-theoretically secure communication channels between the data owner and the shareholders. Our experiments are carried out within the Tokyo QKD Network, which is one of the most advanced QKD networks. Our experimental evaluation establishes the feasibility of LINCOS and shows that it is a promising solution for long-term secure data storage with integrity and confidentiality protection.

Publications. This chapter is based on publication [G2].

7.1. COPRIS: Confidentiality-preserving long-term integrity scheme

In this section we present the scheme COPRIS which ensures long-term integrity protection and is long-term confidentiality preserving, i.e., it does not leak any information about the protected data. Recall that by long-term protection we mean

protection for an indefinite time period. The scheme also provides long-term authenticity if the protected data is signed and the integrity of the signature is protected together with the signed data.

7.1.1. Scheme

COPRIS works as follows. A document owner stores a document d at some time t. He keeps d secret and uses COPRIS to construct a *proof of integrity* PI *for* d. *Later he may choose to reveal* d *to another party. This party then uses* PI *to verify that* d *existed at time* t. To preserve the confidentiality of d, the proof of integrity PI is constructed in such a way that no information about d is revealed to any third party. Outsourced storage of the confidential data is later dealt with in section 7.2.

We now explain the construction of the proof of integrity and its verification and we show that this construction has the desired security properties. The integrity proof is a pair (E, R), where E is an *evidence record* and R is a list of decommitment values. The evidence record is constructed interactively between the document owner and an *evidence service* which, in turn, interacts with a timestamp service. The list of decommitment values is constructed and kept secret by the document owner. The document owner may decide to reveal the decommitment values together with the document to a third party *verifier*.

In the following we describe COPRIS in detail. Figure 7.1 illustrates the functionality of COPRIS. Figure 7.2 lists the algorithms used in COPRIS.

Initial protection. The initial integrity proof is constructed as follows. The document owner runs algorithm Protect. Input is the document d and the (initially empty) list of decommitment values R. He selects a commitment scheme CS and computes a commitment $(c, r) \leftarrow \mathsf{CS.Commit}(d)$. He sets $R = (r)$ and sends the commitment value c to the evidence service. When the evidence service receives c, it runs algorithm AddEv. Input is c and the (initially empty) evidence record E. It requests a timestamp T on c from a timestamp service TS using protocol TS.Stamp at time t. The evidence record is initialized as $E = (c, T, t)$.

Timestamp renewal. Before the last timestamp becomes insecure it must be renewed. In this case, the evidence service executes algorithm RenewTs, where the input is the current evidence record E. It selects a new timestamp scheme TS and obtains a timestamp T on E at time t using protocol TS.Stamp. Then, it appends $(\perp, T, t)$ to E, where $\perp$ indicates that there is no commitment generated at timestamp renewal.

Commitment renewal. Before the last commitment created by the document owner becomes insecure, it must be renewed. The document owner runs the al-

gorithm RenewCom. Input is the document d and the decommitment value list R. The document owner selects a new commitment scheme CS and computes $(c, r) \leftarrow \mathsf{CS.Commit}(d, R)$. He add r to the list R at position $|E|$ and sends c to the evidence service. When the evidence service receives c, it runs algorithm AddEv. Input is c and the evidence record E.

Verification. When the document owner reveals d to the verifier, he also transmits the asserted existence time t and the integrity proof (E, R). Using this information, the verifier can validate the existence of d at time t as follows. Let $R = (r_0, \ldots, r_n)$ and $E = (c_0, T_0, t_0, \ldots, c_n, T_n, t_n)$.

We describe the verification procedure. We define t_{n+1} to be the time of verification and for $i \in \{0, \ldots, n\}$ we set $E_i = (c_0, T_0, t_0, \ldots, c_i, T_i, t_i)$ and $R_i = (r_0, \ldots, r_i)$. Furthermore, for $i \in \{0, \ldots, n\}$ let TS_i the timestamp scheme associated with T_i and if $c_i \neq \perp$, let CS_i denote the commitment scheme associated with c_i. Also, let $t_{\mathrm{NRC}(i)}$ denote the time of the next recommitment after c_i, i.e., the minimum $t_{i'}$ with $i' > i$ and $c_{i'} \neq \perp$ or t_{n+1} if $i = n$. The verifier uses his trust anchor TA, which must contain the necessary certificates for the public parameters of the signatures and commitments, and for $i \in \{0, \ldots, n\}$ verifies that

if $c_i \neq \perp$:

$$\mathsf{CS}_i.\mathsf{Verify}(TA, (d, R_{i-1}), c_i, r_i; t_{\mathrm{NRC}(i)}) = 1$$

$$\mathsf{TS}_i.\mathsf{Verify}(TA, (E_{i-1}, c_i), T_i, t_i; t_{i+1}) = 1$$

else:

$$\mathsf{TS}_i.\mathsf{Verify}(TA, E_{i-1}, T_i, t_i; t_{i+1}) = 1.$$

7.1.2. Security

In the following, we provide a security analysis of COPRIS. We show that COPRIS provides long-term integrity and authenticity protection and that no confidential data is leaked to the evidence and timestamp service.

We consider adversaries that may be active for an unbounded period of time while being computationally bounded per unit of time. We refer to chapter 3 for more details regarding this computational model. It reflects the indefinite lifetime of long-lived systems and of the data processed by them. The fact that adversaries have limited capabilities per unit of real time allows for the usage of computationally secure cryptographic primitives in long-lived systems.

Our security analysis is based on the results of Part I, where it is shown that (under certain computational assumptions) extractable commitments and timestamps can be used to argue about the knowledge at an earlier point in time based on the

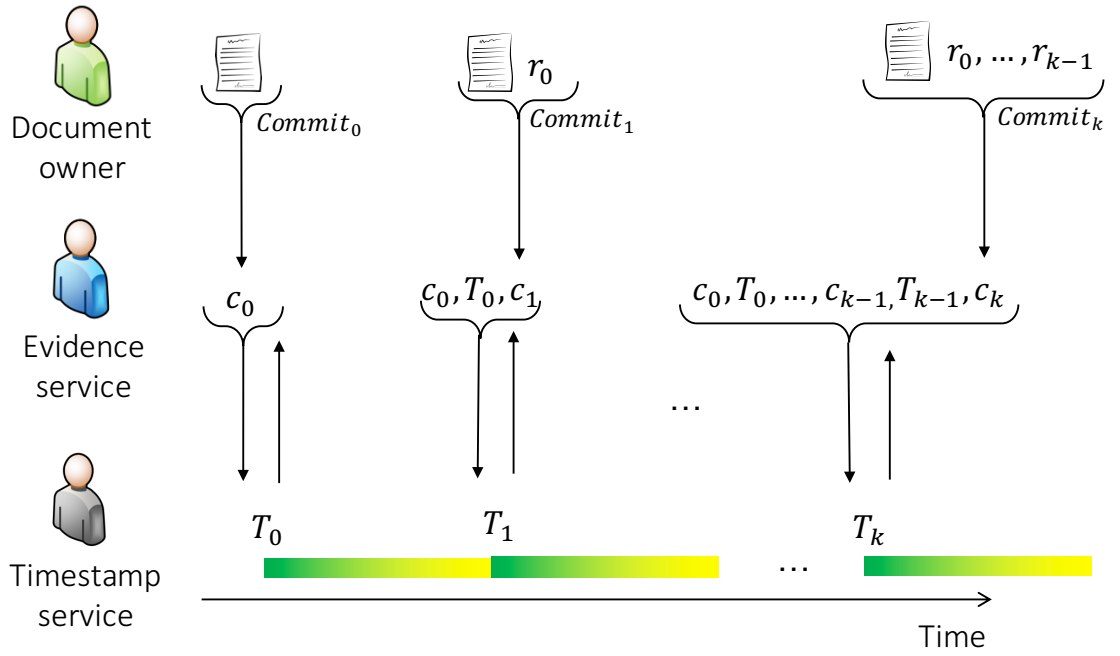

Figure 7.1.: Functionality of COPRIS.

current knowledge and the interaction with the considered cryptographic system. In our security analysis we will use the following notation. For a data object d and a time t, we write $d \in \mathcal{K}[t]$ to denote that d was known at time t. We remark that knowledge is not forgotten, that is, for any data object d and any two points in time t and t', it holds that if $d \in \mathcal{K}[t]$ and $t' > t$, then also $d \in \mathcal{K}[t']$. We also use the convention that for verification of timestamps and commitments a trust anchor TA is provided by a PKI that certifies the verification keys of the used timestamp and commitment scheme instances and specifies the corresponding instance validity periods.

We first state two arguments which are derived from the results of chapter 4, chapter 5, and chapter 6 about extractable timestamps and commitments. We will then use these arguments for showing that integrity proofs of COPRIS are long-term unforgeable. The first argument states that if somebody knows a timestamp (T, t) and a data object d at a time t', and (T, t) is valid for d at t', then has already been known d at time t (with high probability). The second argument states that if somebody knows a commitment value c at a time t, a message d and a decommitment r are known at time $t' > t$, and r is a valid decommitment from c to d at time t', thenthe message d has already been known at the commitment time t (with high probability).

Protect(d, R)**:**

1. Select commitment scheme CS
2. $(c, r) \leftarrow \mathsf{CS.Commit}(d)$
3. Send c to evidence service, set $R = (r)$

AddEv(c, E)**:**

1. Select timestamp scheme TS
2. $(T, t) \leftarrow \mathsf{TS.Stamp}(E, c)$
3. Append (c, T, t) to E

RenewTs(E)**:**

1. Select timestamp scheme TS
2. $(T, t) \leftarrow \mathsf{TS.Stamp}(E)$
3. Append $(\perp, T, t)$ to E

RenewCom(d, R)**:**

1. Select commitment scheme CS
2. $(c, r) \leftarrow \mathsf{CS.Commit}(d, R)$
3. Send c to evidence service, append r to R

Verify(TA, d, t, R, E)**:**

1. Let $R = (r_0, \ldots, r_n)$ and $E = (c_0, T_0, t_0, \ldots, c_n, T_n, t_n)$ such that r_i corresponds to c_i. For $i \in \{0, \ldots, n\}$, let $E_i = (c_0, T_0, t_0, \ldots, c_i, T_i, t_i)$ and $R_i = (r_0, \ldots, r_i)$, let TS_i be the timestamp scheme associated with T_i, and if $c_i \neq \perp$, let CS_i be the commitment scheme associated with c_i. Furthermore let $t_0 = t$ and t_{n+1} be the current time.
2. For $i \in \{0, \ldots, n\}$, if $c_i = \perp$, verify that $\mathsf{TS}_i.\mathsf{Verify}(TA, E_{i-1}, T_i, t_i; t_{i+1}) = 1$, and if $c_i \neq \perp$, verify that $\mathsf{CS}_i.\mathsf{Verify}(TA, (d, R_{i-1}), c_i, r_i; t_{\mathrm{NRC}(i)}) = 1$ and $\mathsf{TS}_i.\mathsf{Verify}(TA, (E_{i-1}, c_i), T_i, t_i; t_{i+1}) = 1$.

Figure 7.2.: Algorithms used in COPRIS for initial protection, adding evidence, timestamp renewal, commitment renewal, and verification.

Argument 7.1. *For any data object d, timestamp (T, t), and time t':*

$$(d, (T, t)) \in \mathcal{K}[t'] \wedge \mathsf{VerifyTs}_{TA}(d, T, t; t') = 1 \implies d \in \mathcal{K}[t] \ .$$

Argument 7.2. *For any commitment value c, time t, data object d, decommitment value r, and time $t' > t$:*

$$c \in \mathcal{K}[t] \wedge (d, r) \in \mathcal{K}[t'] \wedge \mathsf{VerCom}_{TA}(d, c, r; t') = 1 \implies d \in \mathcal{K}[t] \ .$$

By long-term unforgeability of COPRIS we mean that it is computationally infeasible to construct a valid integrity proof for a document d and a time t if d was unknown at time t.

It is essential for the security of COPRIS that the following assumptions hold.

I1. The commitment schemes used in the proof of integrity are computationally binding in their usage period.

I2. The timestamp schemes used in the proof of integrity are computationally unforgeable in their usage period.

I3. The verifier has a valid trust anchor.

Here, by *usage period* of the cryptographic schemes we mean the time interval that begins when the cryptographic scheme is chosen and ends when it is replaced by a new scheme. Also, by a valid trust anchor we mean a trust anchor that allows for the verification of all timestamps and commitments.

Argument 7.3. *Under assumptions I1, I2, and I3,* COPRIS *is long-term unforgeable.*

Proof sketch. Assume an adversary outputs (d, t, E, R) such that (E, R) is a valid integrity proof for d and time t. We show that in this case d was known at time t (with high probability).

We write $R = (r_0, \ldots, r_n)$ and $E = (c_0, T_0, t_0, \ldots, c_n, T_n, t_n)$. For $i \in \{0, \ldots, n\}$ let $E_i = (c_0, T_0, t_0, \ldots, c_i, T_i, t_i)$ and $R_i = (r_0, \ldots, r_i)$. Furthermore, we denote by CS_i the commitment scheme corresponding to c_i, for $c_i \neq \perp$, and by TS_i the timestamp scheme corresponding to T_i. Also, denote by t_{n+1} the time at which the adversary outputs (d, t, E, R). Finally, let $t_{\mathrm{NRC}(i)}$ denote the time of the next recommitment after c_i or t_{n+1} if $i = n$.

We show that for $i \in \{n, \ldots, 0\}$ the following holds:

- (c_i, T_i, t_i) was known at time t_{i+1}.
- If $c_i \neq \perp$, then (d, E_i, R_i) was known at time $t_{\mathrm{NRC}(i)}$.

We prove the first two statements recursively. We observe that for $i = n$ both statements are obviously true as (d, t, E, R) is presented at time $t_{n+1} = t_{\mathrm{NRC}(n)}$ and $E = E_n$ contains (c_n, T_n, t_n) and $R = R_n$. Next, we prove that if the statements are true for i, then they must also be true for $i - 1$. We observe that if $c_i = \perp$, then by the validity of the integrity proof we have $\mathsf{TS}_i.\mathsf{Verify}(TA, E_{i-1}, T_i, t_i; t_{i+1}) = 1$ and by assumptions I2 and I3 and Argument 7.1 we have that E_{i-1} was known at time t_i, which implies that $(c_{i-1}, T_{i-1}, t_{i-1})$ was also known at time t_i. Furthermore, we observe that if $c_i \neq \perp$, then we have $\mathsf{CS}_i.\mathsf{Verify}(TA, (d, R_{i-1}), c_i, r_i; t_{\mathrm{NRC}(i)}) = 1$ and $\mathsf{TS}_i.\mathsf{Verify}(TA, (E_{i-1}, c_i), T_i, t_i; t_{i+1}) = 1$. By assumptions I2 and I3 and Argument 7.1 it follows that (E_{i-1}, c_i) was known at time t_i, which implies that $(c_{i-1}, T_{i-1}, t_{i-1})$ was also known at time t_i, and by assumptions I1 and I3 and Argument 7.2 it follows that (d, R_{i-1}) was known at time $t_i = t_{\mathrm{NRC}(i-1)}$. We observe that, in particular for $i = 0$, this means that d was known at time $t_1 = t$ (with high probability). □

We remark that the concrete security level of the protection degrades slowly over time based on the chosen schemes and parameters. For more details, we refer to chapter 4, chapter 5, and chapter 6.

Next, we show that COPRIS is confidentiality preserving in the long-term, i.e., no information is leaked to the evidence and timestamp service (in an information-theoretic sense). This fact relies on the following assumption.

C1. The commitment schemes are information-theoretically hiding.

Argument 7.4. *Under assumption C1,* COPRIS *is information-theoretic confidentiality preserving.*

Proof sketch. The only data that is sent by the document owner to the evidence service and from there to the timestamp service are information-theoretically hiding commitments. By assumption C1, these commitments do not leak any information about the committed data. □

7.2. LINCOS: System for long-term integrity, authenticity, and confidentiality

In this section we describe our new long-term storage system LINCOS which allows for information-theoretic confidentiality and long-term integrity and authenticity protection. The situation is similar as in COPRIS. A document owner stores a document d at time t. He uses an *integrity system*, which is based on COPRIS, to construct an integrity proof PI. Additionally, he uses a *confidentiality system* for information-theoretic confidential storage of the secret document. The confidentiality system is also used for confidential storage of the secret decommitment values, which are generated during integrity proof construction.

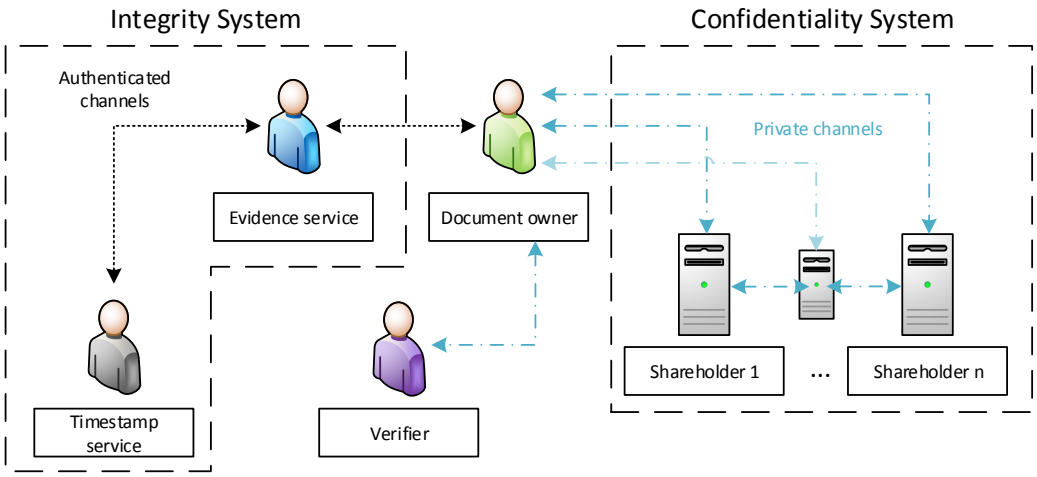

Figure 7.3.: Overview of the long-term secure storage system LINCOS.

7.2.1. System specification

An overview of LINCOS is shown in Figure 7.3. The involved parties are the *document owner*, the *evidence service*, a *timestamp service*, a set of *shareholders*, and a *verifier*. These parties are connected by private or authenticated channels as shown in Figure 7.3. While LINCOS is running, the respective channels are instantiated securely whenever a connection needs to be established. When referring to the evidence service, we use the same notation as in the description of COPRIS. That is, the document owner maintains a list of decommitment values R and the evidence service maintains an evidence record E. Both are initially empty.

Initial document protection. For initial protection of a document d, the document owner runs COPRIS.Protect. Input is a document d and the (initially empty) list of decommitment values R. The document owner chooses a confidentiality system involving several shareholders. The document owner uses protocol Share to distribute (d, R) among the shareholders.

Renewal of timestamps. In COPRIS, timestamps are renewed on a regular basis. For this, the evidence service uses COPRIS.RenewTs.

Renewal of commitments. In COPRIS also the commitments are renewed regularly. For this, the document owner does the following. First, he retrieves d and the sequence of decommitment values R from the confidentiality system by running protocol Retrieve. Then, he runs the algorithm COPRIS.RenewCom, thereby updating

the list of decommitment values R and the evidence record E. Finally, the document owner selects a potentially new confidentiality system and runs protocol Share to distribute the document d and the updated sequence of decommitment values R among the shareholders in the confidentiality system.

Renewal of secret shares. The shares stored by the shareholders are renewed on a regular basis. This prevents a mobile adversary to take advantage of shares he may have been able to obtain in the past. In this process, the current set of shareholders of the confidentiality system may also be replaced by a new set of shareholders operated by the same confidentiality system. This resharing is done by running protocol Reshare.

Verification. When the document owner decides to reveal the document d to a verifier and prove that it existed at time t, he executes the following steps. He requests the current evidence record E from the evidence service. He also retrieves the document d and the list of decommitment values R from the confidentiality system by running the protocol Retrieve. He sends the document d, time t, evidence record E, and the list of decommitment values R to the verifier through a private channel. The verifier uses the data that he received and his trust anchor TA and checks that $\mathsf{COPRIS.Verify}(TA, d, t, E, R) = 1$. This proves that d existed at time t and has not been changed.

7.2.2. Security

We show that under appropriate assumptions, LINCOS provides integrity protection for an indefinite period of time and information-theoretic confidentiality protection.

Adversaries are assumed to have the capabilities described in subsection 7.1.2. They run forever but are computationally bounded per unit of time. In addition, to analyze confidentiality, adversaries are assumed to be active and mobile. This means that adversaries may eavesdrop on channels or corrupt shareholders. A more detailed discussion of this model can be found in [OY91, CH94, HJKY95].

Integrity. Argument 7.3 states that in this adversary model, LINCOS provides long-term integrity and authenticity protection if assumptions I1, I2, and I3 from subsection 7.1.2 are satisfied.

Confidentiality. We say that LINCOS provides information-theoretic confidentiality protection if an adversary with capabilities as described above cannot recover any information about the stored document in an information-theoretic sense.

For information-theoretic confidentiality we require assumption C1 from subsection 7.1.2 and the following assumptions to hold.

C2. The private channels used in LINCOS provide information-theoretic confidentiality and computational authenticity at the time of data transmission.

C3. The proactive secret sharing schemes used in LINCOS provide information-theoretic confidentiality.

C4. During their usage periods, the secret sharing services used in LINCOS prevent mobile adversaries from learning k or more shares.

Argument 7.5. *Under assumptions C1, C2, C3, and C4 the system* LINCOS *provides information-theoretic confidentiality protection.*

Proof sketch. LINCOS is based on COPRIS, which is information-theoretic confidentiality preserving under assumption C1. Hence, an adversary cannot obtain any information about the confidential document by eavesdropping on the authenticated channel from the document owner to the evidence service or from the evidence service to the timestamp service.

Information-theoretic confidentiality of data sent through the private channels from the document owner to the shareholders or between the shareholders is guaranteed by assumption C2. Information-theoretic confidentiality of data stored at the shareholders is guaranteed by assumptions C3 and C4. □

7.3. Implementation

In this section we describe our implementation of the storage system LINCOS, which we presented in section 7.2.

LINCOS uses COPRIS for its integrity system and proactive secret sharing combined with appropriate private channels for its confidentiality system. We describe the implementation of these two systems. One important feature of our implementation is the possibility of replacing cryptographic components. This is required because of assumptions I1 and I2. Another feature is the realization of private channels using the Tokyo QKD Network [SFI+11].

7.3.1. Implementation of COPRIS

The parties involved in COPRIS are the document owner, the evidence service, and a timestamp service. As cryptographic componentes they use commitment and timestamp schemes. We implemented COPRIS in Java following the specification given in section 7.1. Here we describe the implementation of the components.

Commitment scheme. As the commitment scheme, which is used by the document owner to generate commitments to documents, we use the scheme proposed by Pedersen [Ped92]. It is computationally binding and information-theoretically hiding (assumptions I1 and C1) and is parametrized by two prime numbers p and q. The size of q determines the size of the data that can be committed to. The scheme is binding if it is infeasible to compute discrete logarithms in the unique q-order subgroup of $\mathbb{Z}_p$.

To allow committing to data of arbitrary length, data are first hashed, using a cryptographic hash function, and then committed to. Our implementation supports the SHA-2 hash function family which contains the hash functions SHA-224, SHA-256, SHA-384, and SHA-512. They have increasing security levels.

The hash function and the parameters of the commitment scheme are chosen such that binding security is achieved during the intended usage period as required by assumption I1. In practice, these choices can be made on the basis of trustworthy recommendations. For an overview of recommendations see [Gir18].

Timestamp scheme. The timestamp service used by the evidence service is implemented in accordance with standard RFC 3161 [ACPZ01]. Implementing it requires choosing a hash function and a digital signature scheme. We use the SHA-2 hash function family and the RSA digital signature scheme. The security of the used RSA instance depends on the bitlength of the RSA-modulus. Hash function and RSA-modulus are chosen such that they remain secure during their usage period as required by assumption I2. Again, see [Gir18] for an overview of recommendations on how to choose hash functions and security parameters in practice.

Authenticated channels. Authenticated channels are realized using TLS [DR08]. This protocol is state of the art and provides mutual authentication in a computational sense. Authenticated channels are important for the robustness of our system. They guarantee that the document owner connects with the intended evidence service. However, our security analysis does not require security properties of these channels. Therefore, we do not discuss the schemes and parameters chosen for TLS.

7.3.2. Secret sharing and private channels

In the following we describe the implementation of the confidentiality system in LINCOS which uses private channels and proactive secret sharing.

Private channels. LINCOS uses private channels to connect the document owner with the shareholders. By assumption C2, these channels are required to provide information-theoretic confidentiality and computational authenticity. For establishing such private channels we use the Tokyo QKD Network [SFI+11], which is shown

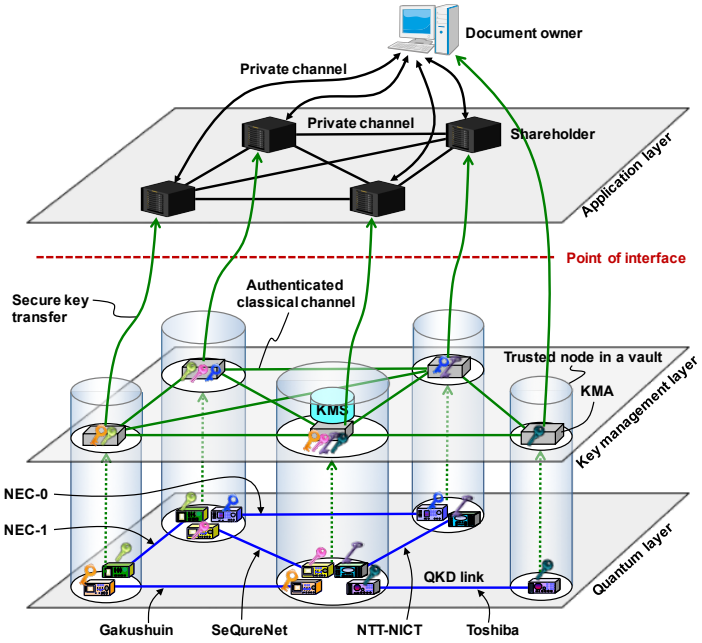

Figure 7.4.: The secret sharing scheme supported by the Tokyo QKD Network.

in Figure 7.4. A combination of Wegman-Carter authentication, QKD, and OTP encryption is used to achieve information-theoretic private and authenticated channels [LC99, May01, SBPC+09]. The network consists of three layers; the *quantum layer*, the *key management layer*, and the *application layer*. Secret sharing is run on the application layer. Parties on the application layer request and receive key material from the key management layer. The key management layer establishes an interface to the quantum layer where the raw key material is generated using QKD technology. To improve the capabilities of the network, keys are relayed on the key management layer by key management agents. In order to allow for assumption C2 to hold, further technical protection measures are in place as explained below. In the following, we explain the functionality of the network in more detail.

Wegman-Carter authentication [WC81] is used to guarantee authenticity of the channels. The initial authentication is done using a preshared key. Further key material for this authentication is generated using QKD.

We explain the key exchange mechanism. On the quantum layer, nodes that are directly connected via a QKD link can generate raw key material. QKD transmitters and receivers are assumed to be located in the trusted nodes. The exact configurations of the QKD links and protocols in use are shown in Table 7.1 with achieved key generation rates.

Once keys have been generated in the quantum layer, they are pushed up to the key management layer and then stored and managed by the key management

Name	**Protocol**	**Length** km	**Key rate** kb/s
NEC-0	BB84	50	200
NEC-1	BB84	22	200
Toshiba	BB84	45	300
NTT-NICT	DPS-QKD	90	10
Gakushuin	CV-QKD	2	100
SeQureNet	CV-QKD	2	10

Table 7.1.: Specifications of the QKD links.

agents (KMAs). All the KMAs are placed in the trusted nodes. The KMAs are connected by authenticated channels, and execute key relays by key encapsulation in a hop-by-hop fashion. Thus a key pair can be shared between two terminal nodes even if they are not directly connected by a QKD link. A reliable key management server (KMS) is also located at one of the trusted nodes, gathers link information (bit error rates, key rates, amounts of accumulated keys, etc.) from the KMAs, organizes a routing table, and provisions secure paths to the KMAs. Secure key transfer is made on request from the KMA to the document owner/shareholder via a protected classical channel, e.g., a tamper resistant cable of short distance. This guarantees non-interceptable key transfer from the QKD platform to the document owner/shareholder located outside the trusted nodes. Furthermore, the trusted nodes are protected by one-way firewalls to prevent attackers from sending malicious commands from the application layer to the QKD platform. Once supplied with the keys, the document owner and the shareholders are in charge of key management. Thus the boundary of responsibility (point-of-interface) is set between the QKD platform and the application layer. For further details we refer to [SFI+11].

Secret sharing. Our implementation of secret sharing is based on the secret sharing scheme proposed by Shamir [Sha79]. The scheme provides information-theoretic confidentiality as required by assumption C3. We use a (3,4)-threshold secret sharing, which suits the network structure of the Tokyo QKD Network. This means that the document owner distributes shares to 4 shareholders and 3 shareholders are needed for the reconstruction of the data. To allow for sharing data of arbitrary size, these data are decomposed into parts of appropriate size. Our implementation supports a simple resharing protocol where the data owner reconstructs the data and creates new shares. As required by assumption C4 we assume that resharing happens before the adversary corrupts more than 2 shareholders. In the future we plan to implement proactive secret sharing as suggested in [HJKY95]. It allows for taking the document owner out of the loop when resharing happens. Furthermore,

Security year	**SHA-2** instance	**RSA** $\log_2(n)$	**Pedersen** $\log_2(p)$, $\log_2(q)$
2040	SHA-224	2048	2048, 224
2065	SHA-224	3072	3072, 224
2085	SHA-256	4096	4096, 256
2103	SHA-384	5120	5120, 384
2116	SHA-384	6144	6144, 384

Table 7.2.: Parameter selection according to Lenstra [Len04].

it is desirable to have a system that does not involve the document owner in the commitment renewal process. However, this requires more research.

7.4. Experimental evaluation

In the following, we present a performance analysis of LINCOS. We estimate the storage space required by the system and investigate data transmission limits imposed by QKD. We also measure the time required for integrity verification. To do so, we run the following experiment. A document is stored and protected using LINCOS over a period of 100 years, starting in 2016 and ending in 2116. Share and timestamp renewal happen every two years. The share renewal period is to be chosen such that mobile adversaries are unable to recover more shares than permissible. Also, the typical storage hardware maintenance service interval is two years. The timestamp renewal period is chosen in accordance with typical certificate renewal periods. Such certificates are required to verify the timestamps. Finally, commitment renewal happens every ten years. This is in accordance with the heuristic security assumptions for the commitment scheme parameters.

Parameter choice for the complexity-based cryptographic components is done according to the heuristics in [Len04]. The corresponding expected protection periods are presented in Table 7.2.

7.4.1. Storage space

We analyze the storage space required by the various parties of LINCOS. It is analyzed as a function of the bitlength $\mathsf{size_d}$ of the protected document d.

Shareholders. Each shareholder stores one share s per document. Its size is $\mathsf{size_s} = \mathsf{size_d} + \mathsf{size_R}$. Here R is the list of decommitment values accumulated over time. Its size is independent of the document size. The size of a single decommitment value equals the size of the parameter q of the commitment scheme. At present, a secure

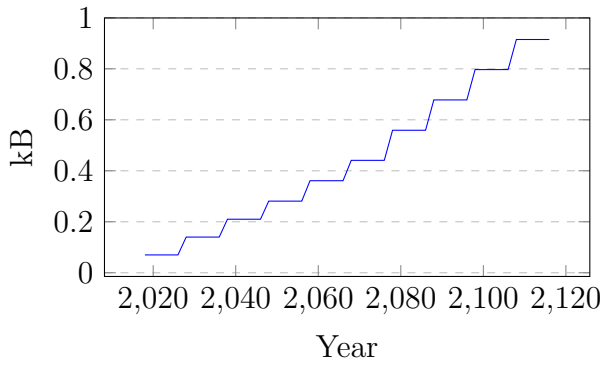

Figure 7.5.: Accumulated size of decommitment values.

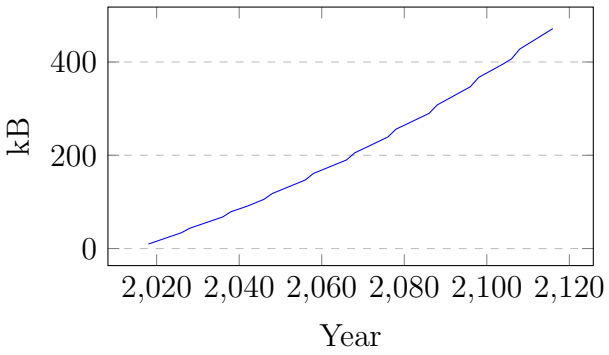

Figure 7.6.: Size of evidence record.

instantiation of the Pedersen commitment scheme requires a decommitment value size of 224 bit. The growth of size_R over 100 years is shown in Figure 7.5. The experiments show that it is at most 1 kB.

Evidence service. The evidence service stores one evidence record E per document. The size of the evidence record size_E is independent of the document size. It depends on the size and number of timestamps and commitments contained in the evidence record. It grows over time because a new timestamp and a new commitment are added with each renewal. The growth of size_E over time is shown in Figure 7.6. Our experiments show that the size of the evidence record accumulates over 100 years to $\mathsf{size}_\mathsf{E} \approx 500$ kB.

7.4.2. Data transmission

Our system uses authenticated and private channels. Authenticated channels easily allow for data rate of 1 Gb/s, while they are used for sending only a few hundred

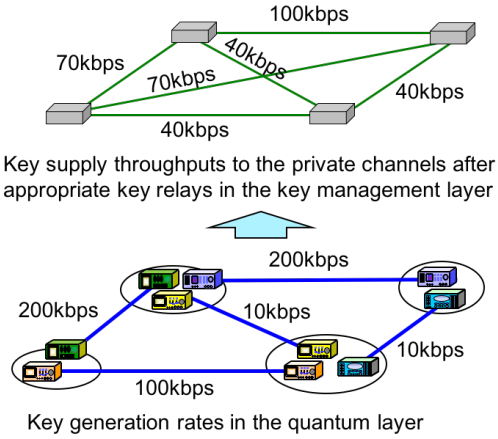

Figure 7.7.: Key generation rates of the QKD links, and key supply throughputs to the private channels after appropriate key relays in the key management layer.

kB of evidence data. So the cost for data transmission via authenticated channels is negligible.

Private channels are realized using OTP and QKD. The transmission rate of these channels is limited by the key generation rate of QKD. Therefore, in our analysis we focus on the QKD part.

Data rate of private channels. The QKD performance in the Tokyo QKD Network differs from link to link because fiber channel lengths as well as specifications of QKD devices are different from each other. Furthermore, some nodes are directly connected by a QKD link, others have to use key relay. The achieved secret key rates of the QKD links are summarized in Table 7.1. They range from 10 kb/s to 300 kb/s depending on the specification of the respective QKD link. To prevent being limited by the slowest QKD links (10 kb/s), keys are relayed between appropriate KMAs such that OTP keys can be supplied at a reasonable key supply throughput, which is denoted as $\mathsf{keyRate}_{\mathsf{QKD}}$. Such key relaying balances the key material across the network. The resulting throughput lies between the slowest and fastest key generation rates of the QKD links. In our current configuration of the QKD platform, this key relay allows to raise the minimum throughput of key supply for each pair of four shareholders to $\mathsf{keyRate}_{\mathsf{QKD}} = 40$ kb/s, as shown in Figure 7.7.

Storage and retrieval. When the document owner stores data in the confidentiality system, he sends one share to each shareholder. Likewise, when retrieving the data, the document owner receives one share per shareholder. Since $\mathsf{size}_{\mathsf{s}} = \mathsf{size}_{\mathsf{d}} + \mathsf{size}_{\mathsf{R}}$, the time required for generating the necessary OTP key material per

$\mathsf{size_s}$	**Sharing**	**Resharing**
1 kB	0.2 s	0.4 s
1 MB	3 min 20 s	6 min 40 s
1 GB	2.3 days	4.6 days

Table 7.3.: Key exchange time for sharing and resharing with $\mathsf{keyRate_{QKD}} = 40$ kb/s.

share transfer in a private channel is $t_s = \mathsf{size_s}/\mathsf{keyRate_{QKD}}$ seconds. Table 7.3 shows timings for shares of different sizes with $\mathsf{keyRate_{QKD}} = 40$ kb/s.

Share renewal. For share renewal, the document owner retrieves the current set of shares and distributes new shares to the shareholders. So the time for communicating the key material required for resharing is $2 * t_s$. Table 7.3 also lists timings for shares of different sizes.

The data that can be protected when resharing happens every two years as in our experiment has maximum size $\mathsf{size_s} = 2 \text{ years} * \mathsf{keyRate_{QKD}}/2 = 1 \text{ year} * \mathsf{keyRate_{QKD}}$. For the current key supply throughput of 40 kb/s we obtain $\mathsf{size_s} = 158$ GB. This data size approximately corresponds to human genomic data of 195 persons. In the near future (4 to 5 years), QKD technology with key rates of 1 Mb/s over 50 km is expected to be available. Then, data of size up to 3942 GB can be protected, which is roughly 4 TB or the size of the genomes of 4926 persons. If the key supply throughput can be increased to 1 Gb/s, data of size 4 PB can be handled, which corresponds to human genomic data of 4.9 million persons. Such a QKD performance can be expected to be realizable using dense wavelength division multiplexing of 1000 quantum channels as well as fast key distillation processing. This is a challenge, but will be feasible by employing integrated photonic technologies and dedicated key distillation engines on semiconductor chips.

7.4.3. Evidence verification

Figure 7.8 shows timings for verification of an integrity proof. The timings were measured on a machine with an 2.9GHz Intel Core i5 CPU and 8GB RAM running our Java implementation of the verification algorithm. As the evidence record and the list of decommitment values grow over time, the verification time increases. Verification of evidence accumulated over 100 years takes approximately 10 seconds. It can be expected that, because computers are getting faster, in a hundred years from now integrity proof verification will only take a fraction of this time.

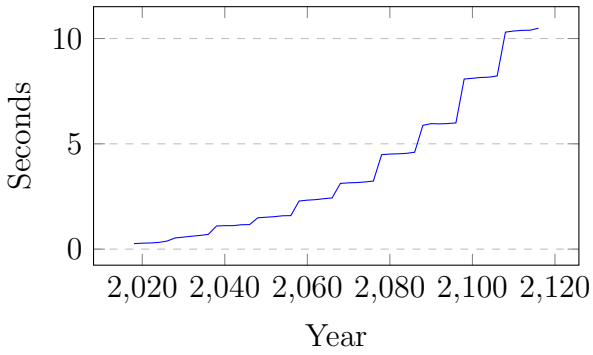

Figure 7.8.: Performance of evidence verification.

7.4.4. Summary

Our experimental evaluation shows the following situation. The long-term integrity system based on COPRIS has very good performance, in particular in view of the expected growth of computing power. So the time and space cost for time-stamping commitments instead of hash values and for renewing these commitments is negligible. As expected, information-theoretic confidentiality protection is expensive. One limiting factor is the additional space required by secret sharing. However, it does not exceed the additional storage space required by cloud storage solutions that use secret sharing for robustness reasons. The second limiting factor is QKD. It is technically complex and transmission rates are not yet fully satisfactory. But, as we have explained above, the development in this area is promising so that practical solutions can be expected in the future.

8. PROPYLA: Long-Term Secure Storage with Access Pattern Hiding

Usually, when a user accesses a database, the database service learns which data items are accessed at which times. In many scenarios, however, these access patterns are sensitive information and it is desirable to hide them from the database service. ORAM schemes [Gol87] allow for doing that, but existing ORAM-based storage systems do not support long-term protection of data integrity and confidentiality.

Contribution. In this chapter we present the storage architecture PROPYLA which is the first to provide long-term protection of data integrity, data confidentiality, and access pattern hiding security. To achieve this, we combine the cryptographic components of LINCOS (cf. chapter 7) with an information-theoretically secure ORAM. We prove our construction secure and show that compared to a multi-data-item version of LINCOS (which does not provide access pattern hiding security) there is only a small constant storage overhead and a poly-logarithmic (in the number of data items) computation and communication overhead.

Publications. This chapter is based on publication [G5].

8.1. Description of PROPYLA

In this section, we describe our new long-term secure storage architecture PROPYLA, which is the first storage architecture that provides long-term integrity, long-term confidentiality, and long-term access pattern hiding.

8.1.1. Overview

PROPYLA comprises the following components: a client, an integrity system, which consists of an evidence service and timestamp service, and a confidentiality system, which consists of a set of shareholders (Figure 8.1).

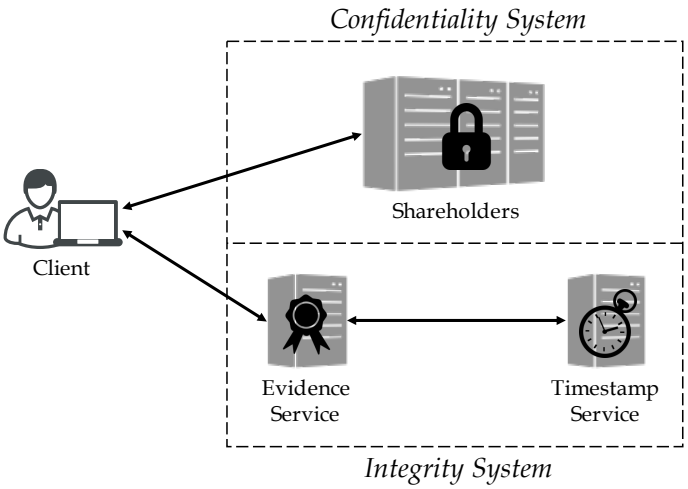

Figure 8.1.: Overview of the storage architecture PROPYLA.

We assume that the client stores a database D that consists of N data blocks that have equal length:

$$D = [\mathsf{dat}_1, \mathsf{dat}_2, \ldots, \mathsf{dat}_N] \ .$$

The data is stored at the shareholders using secret sharing, where each block is shared separately. Integrity protection of the data blocks is achieved by maintaining for each data block i an evidence block E_i. An evidence block E_i has the form

$$E_i = [(\mathsf{op}_{i,1}, c_{i,1}, d_{i,1}, \mathsf{ts}_{i,1}), (\mathsf{op}_{i,2}, c_{i,2}, d_{i,2}, \mathsf{ts}_{i,2}), \ldots]$$

and describes a history of operations that have been performed on the block. For an element $(\mathsf{op}_{i,j}, c_{i,j}, d_{i,j}, \mathsf{ts}_{i,j})$ of such an evidence block E_i, the first element $\mathsf{op}_{i,j} \in \{\text{'Write'}, \text{'Read'}, \text{'ReCom'}, \text{'ReTs'}\}$ describes the operation type that has been performed, and the elements $c_{i,j}$, $d_{i,j}$, and $\mathsf{ts}_{i,j}$ refer to the commitment, decommitment, and timestamp, which have been generated during that operation. Here, the commitments and timestamps form a chain, where later commitments and timestamps guarantee the validity of earlier commitments and timestamps. The evidence is partially stored at the evidence service and partially stored at the shareholders. The newest part of the evidence is stored at the evidence service so that the timestamps can be renewed by the evidence service without the help of the data owner.

To achieve access pattern hiding, the client makes accesses to the evidence service and the storage servers using an information theoretically secure ORAM. Therefore, the evidence service and the shareholders store a database consisting of $M > N$ blocks, where M is determined by the choice of the ORAM. The additional $M - N$ blocks provide the client with the necessary storage space so that it can reshuffle and

access the data such that a uniform distribution of accesses over the server blocks is achieved. In order to preserve the access pattern hiding property, there must also be no correlation between the transmitted data of any two blocks. Typically, this is achived by re-encrypting the data on every access. In our solution, however, this technique cannot be applied because the evidence service must receive the commitments in plaintext so that it can timestamp them in order to renew the integrity protection. Instead, we use a recommitment technique to refresh the commitments: we let the client commit to the previous commitment and timestamp. Thereby, a new commitment is obtained that is indistinguishable from other fresh commitments while the connection to the originally timestamped commitment is maintained.

8.1.2. Protocols

In the following we describe the protocols used in PROPYLA for initializing the system, accessing and protecting the data, renewing the protection, and integrity verification.

Throughout the description of the protocols we use the following notation. For $i \in \{1, \ldots, M\}$ and commitment c, we write $\mathcal{ES}.\mathsf{Write}(i, c)$ to denote that the client instructs the evidence service to store commitment c at block i. Furthermore, we write $E_{\mathcal{ES}} \leftarrow \mathcal{ES}.\mathsf{Read}(i)$ to denote that the client retrieves evidence $E_{\mathcal{ES}}$ of block i from the evidence service. Likewise, we denote by $\mathcal{SH}.\mathsf{Write}(i, (\mathsf{dat}, E))$ that the client stores data dat and evidence E to block i at the shareholders using protocol $\mathtt{SHARE}.\mathsf{Share}$ and we write $(\mathsf{dat}, E) \leftarrow \mathcal{SH}.\mathsf{Read}(i)$ to denote that the client retrieves dat and E of block i from the shareholders using protocol $\mathtt{SHARE}.\mathsf{Reconstruct}$, where $\mathtt{SHARE}$ is the secret sharing scheme chosen by the client in the initialization phase. For a data object dat, we write $\mathsf{ts} \leftarrow \mathcal{TS}.\mathsf{Stamp}(\mathsf{dat})$ to denote that a timestamp ts for dat is obtained from timestamp service $\mathcal{TS}$. Likewise, we write $(c, d) \leftarrow \mathsf{CS}.\mathsf{Commit}(\mathsf{dat})$ to denote that a commitment and decommitment are generated for dat using commitment scheme instance CS. Throughout our description, we assume that the timestamp services are initialized appropriately using algorithm Setup of the corresponding timestamp scheme. Similarly, we assume that commitment scheme instances are initialized by a trusted third party using algorithm Setup of the corresponding commitment scheme.

Initialization

At initialization, the client chooses a secret sharing scheme $\mathtt{SHARE}$, an ORAM scheme $\mathtt{ORAM}$, and a database size N. It then initializes the ORAM via $(s, M) \leftarrow \mathtt{ORAM}.\mathsf{Setup}(N)$ and allocates a database with M blocks at the evidence service. For each $i \in \{1, \ldots, M\}$, the evidence service initializes block i with an empty evidence lists, $E_{\mathcal{ES},i} = []$. Afterwards, the client picks a set of secret share holders and a reconstruction threshold, and then initializes the secret sharing database with M

blocks using protocol SHARE.Setup. The shareholders initialize their databases such that for each $i \in \{1, \ldots, M\}$ an empty data object $\mathsf{dat}_i = \perp$ and an empty evidence list $E_i = []$ is stored at block i. We remark that while we use a stash-free ORAM model for the benefit of a more comprehensible description, the construction can be adopted to work with stashed ORAMs in which case the client locally manages the stashed items as usual.

Read and write

The client reads and writes data blocks using algorithm Access (see Listing 8.1). This algorithm gets as input an operation type $\mathsf{op} \in \{$'Write', 'Read'$\}$, a block identifier $\mathsf{id} \in \{1, \ldots, N\}$, optionally data to be written dat', the ORAM state s, a commitment scheme instance CS, and a reference to a timestamp service $\mathcal{TS}$. It then generates an access pattern, $(P, s) \leftarrow \mathtt{ORAM}.\mathsf{GenAP}(s, \mathsf{id})$, and for each $(i_k, j_k) \in P$ does the following: first, it retrieves the new evidence $E_{\mathcal{ES}}$ for block i_k from the evidence service and it retrieves the stored data dat and the old evidence E from the shareholders. Then the new evidence $E_{\mathcal{ES}}$ is added to the shareholder evidence E. If this is a write operation (op = 'Write') and $\mathtt{ORAM}.\mathsf{GetId}(s, i_k) = \mathsf{id}$, then the block data dat is replaced with dat' and a commitment (c, d) to the new data is generated. Since this block is newly written, the existing evidence is discarded and the corresponding evidence is set to $E = [(\text{'Write'}, c, d, \perp)]$, where $\perp$ is a placeholder for the timestamp that will later be retrieved by the evidence service. If this is a read operation (op = 'Read') and $\mathtt{ORAM}.\mathsf{GetId}(s, i_k) = \mathsf{id}$, then the data dat is cached and returned when the access algorithm finishes. Finally, the algorithm refreshes the commitment by creating a new commitment to the block data and the previous commitment. This is necessary so that the evidence service cannot trace how the client rearranges the blocks. The refreshed commitment is stored at the evidence service at the new location j_k. The evidence service then timestamps the new commitment and stores the timestamp together with the commitment (Listing 8.2). Also, the client generates new secret shares of the data dat and the shareholder evidence E and stores them at the shareholders at the new location j_k. As the secret shares are newly generated, they do not correlate with the old shares and their relocation cannot be traced either.

Timestamp renewal

If the security of the currently used timestamp scheme is threatened, the evidence service renews the evidence as follows (Listing 8.3). It picks a new timestamp service $\mathcal{TS}$ that uses a more secure timestamp scheme and then for every $i \in [1, \ldots, M]$, it first creates a commitment (c', d') to the last commitment and timestamp stored in $E_{\mathcal{ES},i}$, then requests a timestamp ts for c', and finally adds ('ReTs', c', d', ts) to $E_{\mathcal{ES},i}$.

Listing 8.1: Access(op, id, dat′, s, CS, $\mathcal{TS}$), run by the client.

```
([(i_1, j_1), ..., (i_n, j_n)], s') ← ORAM.GenAP(id);
for k = 1, ..., n do
    // process k-th entry in access pattern
    E_ES ← ES.Read(i_k); (dat, E) ← SH.Read(i_k);      // read data from location i_k
    E[−1].ts ← E_ES[1].ts; E += E_ES[2 :];   // move evidence from ES to SH
    if op = 'Write' and T(i_k) = id then
        dat ← dat′; (c, d) ← CS.Commit(dat′); E = [('Write', c, d, ⊥)];
        // write and commit new data, discard old evidence
    else if op = 'Read' and T(i_k) = id then
        dat″ ← dat; E″ ← E;                    // save for output later
    end
    if op ≠ 'Write' or T(i_k) ≠ id then
        // refresh commitments
        if E[−1].op = 'Read' then
            E ← E[: −2];
        end
        (c, d) ← CS.Commit([E[−1].c, E[−1].ts]);
        E += [('Read', c, d, ⊥)];
    end
    ES.Write(j_k, E[−1].c, TS); SH.Write(j_k, (dat, E));   // write data to location j_k
end
return (s′, dat″, E″);
```

Listing 8.2: $\mathcal{ES}$.Write($i, c, \mathcal{TS}$), run by the evidence service.

```
ts ← TS.Stamp(c);
E_ES,i ← [(⊥, c, ⊥, ts)];
```

Listing 8.3: RenewTs(CS, $\mathcal{TS}$), run by the evidence service.

```
for i = 1 to M do
    (c_i, d_i) ← CS.Commit([E_ES,i[−1].c, E_ES,i[−1].ts]);
    ts_i ← TS.Stamp(c_i);
    E_ES,i += [('ReTs', c_i, d_i, ts_i)];
end
```

Commitment renewal

If the security of the currently used commitment scheme is threatened, the client renews the evidence using Listing 8.4. It starts by selecting a new commitment scheme instance CS. Then, for each block $i \in [1, \ldots, M]$, it does the following. It first retrieves the new evidence $E_{\mathcal{ES},i}$ from the evidence service and data block dat_i and old evidence block E_i from the shareholders. It then adds the new evidence $E_{\mathcal{ES},i}$ to the shareholder evidence E_i. Afterwards, it creates a new commitment (c_i, d_i) to the secret data dat_i and the evidence E_i. It then adds ('ReCom', $c_i, d_i, \perp$) to E_i. Finally, it sends c_i to the evidence service and distributes dat_i and E_i to the shareholders.

Listing 8.4: RenewCom($\mathsf{CS}, \mathcal{TS}$), run by the client.

```
for i = 1, ..., M do
    E_ES ← ES.Read(i); (dat, E) ← SH.Read(i);    // retrieve data from
     location i
    E[−1].ts ← E_ES[1].ts; E += E_ES[2 :];   // move evidence from ES to
     SH
    (c, d) ← CS.Commit([dat, E]);     // recommit to data and evidence
    E += [('ReCom', c, d, ⊥)];                        // add new evidence
    ES.Write(i, E[−1].c, TS);
    SH.Write(i, (dat, E)) ;   // store evidence and data at location i
end
```

Share renewal

In regular time intervals the shareholders renew the stored shares to protect against a mobile adversary by running protocol SHARE.Reshare.

Verification

The verification algorithm (Listing 8.5) uses a trust anchor TA that certifies the validity of public keys for timestamps and commitments. Here, $\mathsf{VerTs}_{\mathsf{TA}}(\mathsf{dat}, \mathsf{ts}; t_{\mathsf{ver}}) = 1$ denotes that timestamp ts is valid for dat at reference time t_{ver}, and we denote by $\mathsf{VerCom}_{\mathsf{TA}}(\mathsf{dat}, c, d; t_{\mathsf{ver}}) = 1$ that d is a valid decommitment from c to dat at time t_{ver}. On input a data object dat, a time t, an evidence block E, and the verification time t_{ver}, the verification algorithm of PROPYLA checks whether E is currently valid evidence for the existence of dat at time t, given that the current time is t_{ver}. In particular, the algorithm checks that the evidence is constructed correctly for dat, that the timestamps and commitments have been valid at their renewal time, and that the first timestamp refers to time t.

Listing 8.5: $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E; t_{\mathsf{ver}})$, run by any verifier.

```
/* verifies that dat existed at time t                                    */
Let E = [(op_1, c_1, d_1, ts_1), ..., (op_n, c_n, d_n, ts_n)];
Set t_{n+1} := t_ver;
For i ∈ [n], set E_i := [(op_1, c_1, d_1, ts_1), ..., (op_i, c_i, d_i, ts_i)];
For i ∈ [n], set t_i := ts_i.t;
For i ∈ [n], set t_{NRC(i)} := min({t_j | act_j = 'ReCom' ∧ j > i} ∪ {t_{n+1}});
for i = n to 1 do
    Assert VerTs_TA(c_i, ts_i; t_{i+1}) = 1;
    if op_i = 'Write' and i = 1 then
        Assert VerCom_TA(dat, c_1, d_1; t_{NRC(1)}) = 1;
    else if op_i = 'Read' or op_i = 'ReTs' then
        Assert VerCom_TA([c_{i-1}, ts_{i-1}], c_i, d_i; t_{NRC(i)}) = 1;
    else if op_i = 'ReCom' then
        Assert VerCom_TA([dat, E_{i-1}], c_i, d_i; t_{NRC(i)}) = 1;
    else
        Fail;
    end
end
Assert ts_1.t = t;
```

8.2. Security analysis

In this section we analyze the security of PROPYLA. Our security analysis requires a model of real time which is used for expressing the scheduling of protection renewal events in our security experiments and for expressing computational bounds on the adversary with respect to real time. We first describe our model of real time. Then, we show that PROPYLA provides long-term access pattern hiding, long-term confidentiality, and long-term integrity.

8.2.1. Computational model

For modeling the security of PROPYLA, we want to be able to express that certain events (e.g., renewal of timestamps) are performed according to a timed schedule. Concretely, in the security analysis of PROPYLA, we consider a *renewal schedule* $\mathcal{S}$ that describes at which times, and using which schemes the timestamp and commitment renewals are performed. Additionally, our computational model allows to capture that an adversary becomes computationally more powerful over time (e.g., it gets access to a quantum computer). See chapter 3 for a more detailed description of our computational model.

8.2.2. Long-term access pattern hiding

In the following we prove that PROPYLA achieves information theoretically secure access pattern hiding against the evidence service and the shareholders if the used ORAM, secret sharing scheme, and commitment schemes are information theoretically secure.

Formally, Access-Pattern-Hiding (APH) Security of PROPYLA is defined via game $\mathbf{Exp}^{\mathrm{APH}}_{\mathsf{PROPYLA}}$ (Listing 8.6), where an adversary, $\mathcal{A}$, instructs the client of PROPYLA to read and write database blocks at logical addresses chosen by the adversary. During these data accesses, $\mathcal{A}$ observes the data that is transferred between the client and the evidence service and a subset of less than the threshold number of shareholders. At some point in time, $\mathcal{A}$ gives two different access instructions to the client. The client picks one of them at random and executes it. The goal of the adversary $\mathcal{A}$ is to infer from the observed data stream which of the access instructions has been executed by the client.

For PROPYLA to be information theoretically secure access pattern hiding, it must hold that for any timestamp and commitment renewal schedule $\mathcal{S}$, a computationally unbounded adversary is not able to infer with probability other than $\frac{1}{2}$ which access instruction was made.

Definition 8.1 (APH-security of PROPYLA)**.** PROPYLA *is information theoretically APH-secure if for any renewal schedule* $\mathcal{S}$ *and any adversary* $\mathcal{A}$*:*

$$\Pr\left[\mathbf{Exp}^{\mathrm{APH}}_{\mathsf{PROPYLA}}(\mathcal{S},\mathcal{A})=1\right]=\frac{1}{2}\ .$$

Listing 8.6: The access pattern hiding experiment for PROPYLA, $\mathbf{Exp}^{\mathrm{APH}}_{\mathsf{PROPYLA}}(\mathcal{S},\mathcal{A})$.

$((\mathsf{op}_1,\mathsf{id}_1,\mathsf{dat}_1),(\mathsf{op}_2,\mathsf{id}_2,\mathsf{dat}_2)) \leftarrow \mathcal{A}^{\mathsf{Clock},\mathsf{Client}}();$
$b \xleftarrow{\$} \{1,2\};$
$\mathsf{VIEW} \leftarrow \mathsf{Client}(\mathsf{op}_b,\mathsf{id}_b,\mathsf{dat}_b);$
$b' \leftarrow \mathcal{A}^{\mathsf{Clock},\mathsf{Client}}(\mathsf{VIEW});$
if $b = b'$ **then**
  return 1;
else
  return 0;
end

oracle Clock(t):
if $t >$ time **then**
  Perform all renewals scheduled in $\mathcal{S}$ between time and t;
  time $\leftarrow t$;
end

oracle Client$(\mathsf{op},\mathsf{id},\mathsf{dat})$:
PROPYLA.Access$(\mathsf{op},\mathsf{id},\mathsf{dat})$;
Let VIEW denote the data received by the evidence service and a subset of less than the threshold number of shareholders during the execution of PROPYLA.Access;
return VIEW;

Theorem 8.1. *If* PROPYLA *is instantiated using an information theoretically secure ORAM, information theoretically hiding commitment schemes, and information theoretically secure secret sharing, then it provides information theoretic APH-security.*

Proof. We observe that the queries made by the client and observed by the adversary are either of the form (i,c), when the client instructs the evidence service to store commitment c at database location i, or of the form (i,s), when the client instructs a shareholder to store share s at location i. Furthermore, we observe that when using an information theoretically secure ORAM, there is no statistical correlation between the instructions $(\mathsf{op},\mathsf{id},\mathsf{dat})$ chosen by the adversary and the database locations i sent by the client. There is also no statistical correlation between the data and the commitments c or the secret shares s, as long as the adversary observes less than the threshold number of shareholders. This is (1) because we use information

theoretically hiding commitments and information theoretically secure secret sharing and (2) the shares and the commitments are renewed on every access.

As there is no statistical correlation between the accesses made by the client and the transmitted data, even a computationally unbounded adversary cannot infer which of the challenge instructions $(\mathsf{op}_1, \mathsf{id}_1, \mathsf{dat}_1)$ and $(\mathsf{op}_2, \mathsf{id}_2, \mathsf{dat}_2)$ was executed. □

8.2.3. Long-term confidentiality

Informally, long-term confidentiality of PROPYLA means that even an unbounded evidence service and a subset of colluding unbounded shareholders cannot learn anything about the content of the stored data. More formally, we require that there is no significant statistical correlation between the data stored by the client and the data observed by the evidence service and a subset of shareholders. We observe that this property immediately follows from the information-theoretic access pattern hiding security of PROPYLA.

8.2.4. Long-term integrity

Next, we provide an intuitive argument that PROPYLA provides long-term integrity protection based on the results of Part I. Here we consider an adversary that may be running for a very long time but who can only perform a limited amount of work per unit of time (see the adversary model description in subsection 8.2.1). Furthermore, we use the notation and statements introduced in subsection 7.1.2.

Argument 8.1. PROPYLA *provides long-term integrity protection, that is, it is infeasible for an adversary to produce evidence E valid for data* dat *and time t without having known* dat *at time t given that the used timestamp and commitment schemes are secure within their usage period.*

Proof sketch. Assume an adversary outputs (dat, t, E) at some point in time t_{n+1} and that TA is the trust anchor provided by the PKI at that time. We show that if E is valid evidence for data dat and time t (i.e., $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$), then dat was known at time t (with high probability).

Let $E = [(\mathsf{op}_1, c_1, d_1, \mathsf{ts}_1), \ldots, (\mathsf{op}_n, c_n, d_n, \mathsf{ts}_n)]$ without loss of generality. Then, for $i \in [1, \ldots, n]$, define $E_i = [(\mathsf{op}_1, c_1, d_1, \mathsf{ts}_1), \ldots, (\mathsf{op}_i, c_i, d_i, \mathsf{ts}_i)]$, $t_i = \mathsf{ts}_i.t$, and $t_{\mathrm{NRC}(i)}$ as the time of the next commitment renewal after commitment c_i. Additionally, we define $t_{\mathrm{NRC}(n)} = t_{n+1}$. In the following, we show recursively that for $i \in [n, \ldots, 1]$, statement

$$St(i) : (c_i, \mathsf{ts}_i) \in \mathcal{K}[t_{i+1}] \wedge (\mathsf{dat}, E_i) \in \mathcal{K}[t_{\mathrm{NRC}(i)}]$$

holds, that is, commitment value c_i and timestamp ts_i are known at the next timestamp time t_{i+1} and the data dat and partial evidence E_i are known at the next commitment renewal time $t_{\mathrm{NRC}(i)}$.

Statement $St(n)$ obviously holds because (dat, E) is output by the adversary at time t_{n+1} and includes c_n, ts_n, dat, and E_n. Next, we show for $i \in \{n, \ldots, 2\}$, that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ and $St(i)$ implies $St(i-1)$. By the definition of VerInt (Listing 8.5) we observe that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ implies $\mathsf{VerTs}_{\mathsf{TA}}(c_i, \mathsf{ts}_i; t_{i+1}) = 1$, that is, ts_i is valid for commitment c_i at time t_{i+1}. Furthermore, we observe that $St(i)$ implies $(c_i, \mathsf{ts}_i) \in \mathcal{K}[t_{i+1}]$, which means that c_i and ts_i were known at time t_{i+1}. We can now apply Argument 7.1 to obtain that commitment c_i was known at time t_i (i.e., $c_i \in \mathcal{K}[t_i]$). Next, we distinguish between the case $\mathsf{op}_i \in \{\text{'Read'}, \text{'ReTs'}\}$ and the case $\mathsf{op}_i \in \{\text{'ReCom'}\}$. We observe that if $\mathsf{op}_i \in \{\text{'Read'}, \text{'ReTs'}\}$, then $\mathsf{VerCom}_{\mathsf{TA}}(c_{i-1} \| \mathsf{ts}_{i-1}, c_i, d_i; t_{\mathrm{NRC}(i)}) = 1$ and by Argument 7.2 it follows that $(c_{i-1}, \mathsf{ts}_{i-1}) \in \mathcal{K}[t_i]$ (i.e., c_{i-1} and ts_{i-1} were known at time t_i). If $\mathsf{op}_i \in \{\text{'ReCom'}\}$, then $\mathsf{VerCom}_{\mathsf{TA}}(\mathsf{dat} \| E_{i-1}, c_i, d_i; t_{\mathrm{NRC}(i)}) = 1$ and it follows that $(\mathsf{dat}, E_{i-1}) \in \mathcal{K}[t_{\mathrm{NRC}(i-1)}]$. Together, we obtain that if $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ and $St(i)$, then $St(i-1)$ holds. By induction over i it follows that $St(1)$ holds.

Finally, we observe that $St(1)$ by Argument 7.1 and Argument 7.2 implies $\mathsf{dat} \in \mathcal{K}[t_1]$. Also, $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ implies $t_1 = t$, and thus we obtain $\mathsf{dat} \in \mathcal{K}[t]$. We conclude that if an adversary presents (dat, t, E) such that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$, then dat must have been known at time t (with high probability). □

8.3. Performance evaluation

In this section, we describe an instantiation of PROPYLA and analyze its performance.

8.3.1. Scheme instantiation

We instantiate PROPYLA using Path-ORAM [SvDS+13], Shamir Secret Sharing [Sha79], Halevi-Micali Commitments [HM96], RSA Signatures [RSA78], and XMSS Signatures [BDH11]. The implementation has been done in Java and we use the following parameters. For Path-ORAM we use a bucket size of 5. Our implementation of Halevi-Micali Commitments uses the hash functions SHA-224, SHA-256, and SHA-384 of the SHA-2 hash function family [Nat02]. For RSA, we use the standard JDK implementation and use SHA-224 with RSA-2048. For XMSS, we use the implementation from the Bouncy Castle Library [The18], which supports the hash functions SHA-256 and SHA-512, and we use a tree height of 10.

This instantiation has the required security properties. Path ORAM instantiated with an information theoretically secure random number generator (e.g., a quantum-based random number generator [SGG+00]) provides information theo-

Table 8.1.: Overview of the used commitment and signature scheme instances and their usage period.

Years	Signatures	Commitments
2018-2030	RSA-2048	HM-224
2031-2066	XMSS-256	HM-224
2067-2091	XMSS-256	HM-256
2091-2118	XMSS-512	HM-384

retic security [SvDS+13]. Shamir Secret Sharing and Halevi-Micali Commitments are information theoretic hiding. Therefore, by Theorem 8.1, the described instantiation of PROPYLA is information theoretic access pattern hiding. Information theoretic confidentiality also follows from Theorem 8.1 (see subsection 8.2.3). Finally, by Argument 8.1, long-term integrity is achieved as long as the used commitment and signature scheme instances are secure within their usage period. In Table 8.1 we list the commitment and signature scheme instances that we use together with their usage periods, which are based on the predictions by Lenstra and Verheul [LV01, Len04, Gir18]. Also, we assume that quantum computers will become a considerable threat by 2040 and therefore transition from RSA Signatures (which are known vulnerable to quantum computers) to XMSS Signatures (which are expected secure against quantum computer attacks) after 2030.

8.3.2. Evaluation

Scenario

We examine a use case where a client stores N data objects of size L for a time period of 100 years. To maintain long-term integrity protection, timestamps and commitments are renewed regularly. We renew timestamps every 2 years, which corresponds to the typical lifetime of a public key certificate for digital signature schemes, and we renew commitments every 10 years, which reflects their stronger security compared to signatures as they do not require any secret parameters (which may be compromised at some point in time). The schemes are instantiated as described in subsection 8.3.1 and shown in Table 8.1. The runtime measurements have been computed using our Java implementation running on a computer with a 2.9 GHz Intel Core i5 CPU and 8 GB RAM.

Results

In the following we present the results of our performance evaluation. Here, we focus on the overhead costs for computation, storage, and traffic induced by our long-term

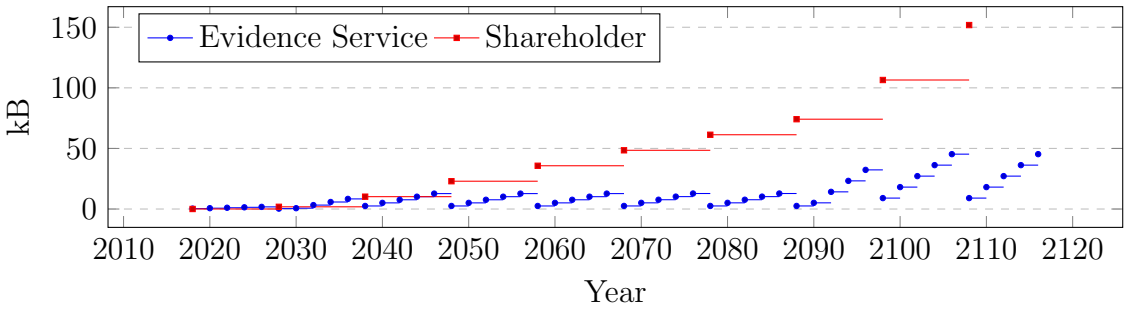

Figure 8.2.: Additional storage space required for commitments and timestamps per server block (independent of block size).

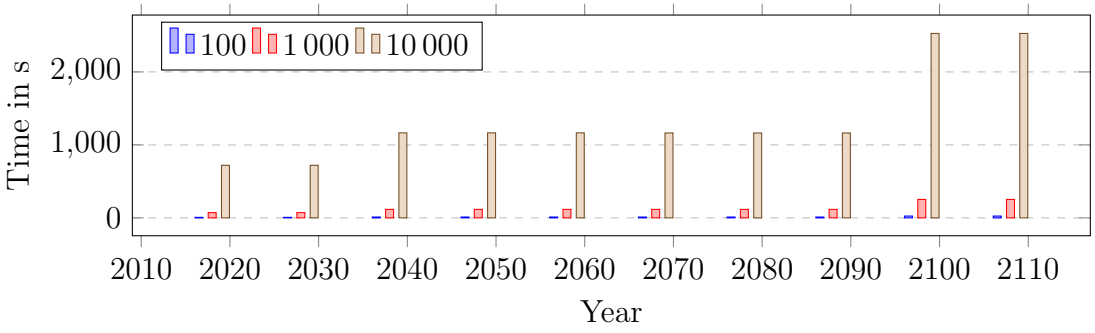

Figure 8.3.: Time required for renewing timestamps and commitments when storing $N = 100, 1000, 10000$ data objects of size $L = 1\,\text{MB}$.

protection scheme for generating and transmitting timestamps and commitments in comparison to standard Path-ORAM and Secret Sharing.

In Figure 8.2, we show the storage space consumption of the commitments and timestamps generated by our protection scheme, per shareholder and for the evidence service. For the shareholders, we observe that the storage space overhead increases over time with each commitment renewal. Obviously, it also depends on the chosen commitment and signatures schemes and their parameters. For the evidence service, the storage space overhead increases with each timestamp renewal and then is reset at commitment renewal, where the evidence is moved to the shareholders. As the commitment and signature sizes are independent of the size of the stored data objects, the same holds for the storage space overhead.

In Figure 8.3, we show the time required for renewing timestamps and commitments when storing $N = 100, 1000, 10000$ data objects of size $L = 1\,\text{MB}$. We observe that the renewal time scales linearly with the number of data objects and also depends on the chosen schemes and parameters.

In Figure 8.4, we show the computation time overhead for generating commitments and timestamps during one database access when storing $N = 100, 1000, 10000$

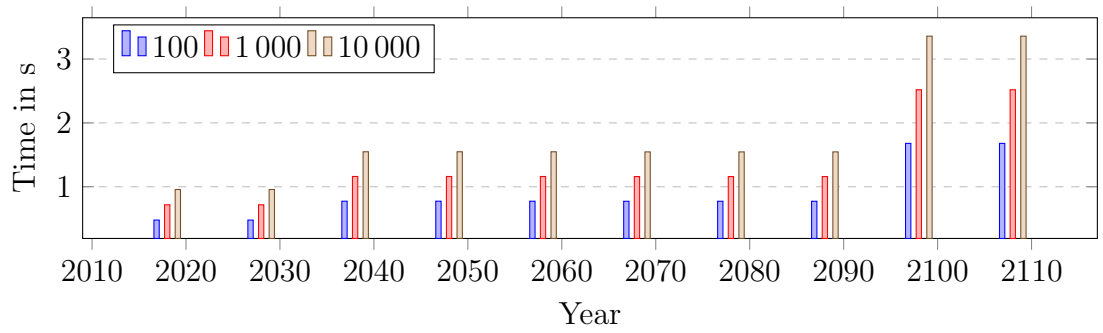

Figure 8.4.: Time overhead due to commitment and timestamp generation during one database access when storing $N = 100, 1000, 10000$ data objects of size $L = 1\,\text{MB}$.

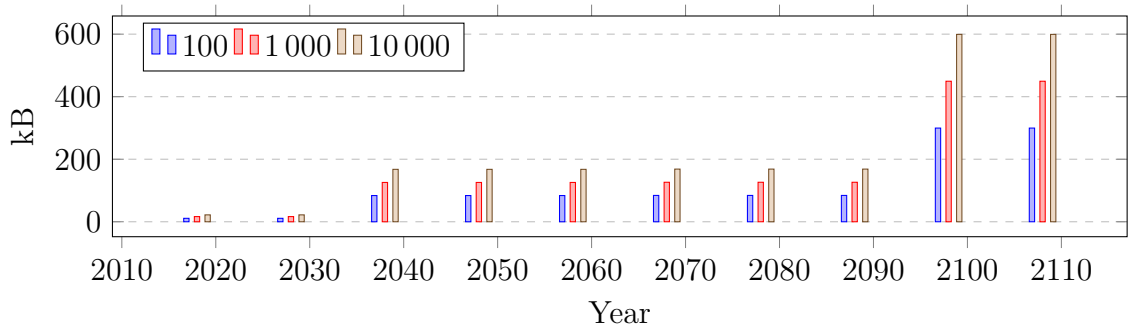

Figure 8.5.: Traffic overhead due to sending timestamps and commitments during one database access when storing $N = 100, 1000, 10000$ data objects (independent of data object size).

data objects of $L = 1\,\text{MB}$. As the number of blocks touched per database access using Path-ORAM is logarithmic in the number of data objects stored, we observe that the computation overhead also scales logarithmically with the number of data objects. Again, the computation time also depends on the chosen schemes and instances.

In Figure 8.5 we show the network traffic overhead due to sending timestamps and commitments during one database access when storing $N = 100, 1000, 10000$ data objects. We observe that, similar to the computation time overhead per database access, also the traffic overhead scales logarithmically with the number of data objects. As for the storage space overhead, also the traffic overhead is independent of the data object size.

Overall, our performance measurements show that the additional cost for adding long-term protection to a Path-ORAM database appears manageable. The storage overhead per block is independent of the block size. The computation time overhead and the traffic overhead are logarithmic in the database size and the traffic

overhead is independent of the data object size. The timestamp and commitment renewal time scales linearly with the database size. We remark that using Secret Sharing introduces a linear communication overhead in the number of share holders. However, this overhead seems inevitable when long-term confidentiality protection is required, as secret sharing is the only solution for information-theoretic confidentiality protection of data at rest.

We observe that for our instantiation, the most time consuming operation is the timestamp signature generation and the most storage consuming evidence data are the timestamp signatures. We expect that the performance of our solution could further be improved by using Merkle Hash Trees (MHT) [Mer90], as follows. During timestamp and commitment renewal, instead of time-stamping each data block separately, we would first create a MHT for the entire database, and then timestamp only the root of that tree. While asymptotically, this introduces an overhead of $O(N \cdot \log(N))$ (compared to just $O(N)$ for our solution), for our instantiation we expect better performance as hash function evaluation is much faster than signature generation.

9. ELSA: Efficient Long-Term Protection of Large Datasets

Often, real world databases are complex and consist of a large number of relatively small data items that must be retrievable individually (e.g., medical databases). LINCOS (see chapter 7) can be used to protect such databases, but the amount of work to be performed scales linearly with the number of stored data items and this causes performance issues.

Contribution. In this chapter we present the long-term secure storage architecture ELSA which efficiently stores and protects databases that consist of a large number of data items. The protection guarantees provided are long-term integrity and long-term confidentiality. Storing a set of data items or renewing protection with ELSA only requires a single timestamp, while this required with LINCOS a number of timestamps equal to the number of data items. The improvements are achieved by constructing and using an extractable-binding vector commitment scheme that is also information theoretically hiding under selective decommitment. We implemented ELSA and show performance figures for a scenario where a database consisting of 12 000 data items is aggregated, stored, protected, and verified over a time span of 100 years. Our results show that ELSA can be an order of magnitude faster than LINCOS when large databases consisting of many small data items are stored and protected.

Publications. This chapter is based on publication [G6].

9.1. Statistically hiding and extractable binding vector commitments

In this section, we define statistically hiding and extractable binding vector commitments, describe a construction, and prove the construction secure. This construction is the basis for our performance improvements that we achieve with our new storage architecture presented in section 9.2.

9.1.1. Definition

A vector commitment scheme allows to commit to a vector of messages $[m_1, \ldots, m_n]$. It is extractable-binding, if the message vector can be extracted from the commitment and the state of the committer and it is hiding under partial opening if an adversary cannot infer any valuable information about unopened messages, even if some of the committed messages have been opened. Our vector commitments are reminiscent of the vector commitments introduced by Catalano and Fiore [CF13]. However, neither do they require their commitments to be extractable binding nor do they consider their hiding property.

Definition 9.1 (Vector commitment scheme)**.** *A vector commitment scheme is a tuple* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$*, where* $L \in \mathbb{N}$ *is the maximum vector length,* $\mathcal{M}$ *is the message space, and* Setup*,* Commit*,* Extract*, and* Verify *are algorithms with the following properties.*

$\mathsf{Setup}() \to k$**:** *This algorithm generates a public key* k*.*

$\mathsf{Commit}(k, [m_1, \ldots, m_n]) \to (c, D)$**:** *On input public key* k *and message vector* $[m_1, \ldots, m_n] \in \mathcal{M}^n$*, where* $n \in [L]$*, this algorithm generates a commitment* c *and a vector decommitment* D*.*

$\mathsf{Extract}(k, D, i) \to d$**:** *On input key* k*, vector decommitment* D*, and index* i*, this algorithm outputs a decommitment* d *for the* i*-th message corresponding to* D*.*

$\mathsf{Verify}(k, m, c, d, i) \to b$**:** *On input key* k*, message* m*, commitment* c*, decommitment* d*, and an index* i*, this algorithm outputs* $b = 1$*, if* d *is a valid decommitment from position* i *of* c *to* m*, and otherwise outputs* $b = 0$*.*

A vector commitment scheme is correct, if a decommitment produced by Commit and Extract will always verify for the corresponding commitment and message.

Definition 9.2 (Correctness)**.** *A vector commitment scheme* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ *is correct if for all* $n \in [L]$*,* $M \in \mathcal{M}^n$*,* $k \in \mathsf{Setup}$*,* $i \in [n]$*,*

$$\Pr\begin{bmatrix}\mathsf{Commit}(k, M) \to (c, D), \\ \mathsf{Extract}(k, D, i) \to d, \\ \mathsf{Verify}(k, M_i, c, d) = 1\end{bmatrix} = 1 \ .$$

A vector commitment scheme is statistically hiding under selective opening, if the distribution of commitments and openings does not depend on the unopened messages.

Definition 9.3 (Statistically hiding (under selective opening))**.** *Let* $S = (L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ *be a vector commitment scheme. For* $n \in [L]$*,* $I \subseteq [n]$*,*

$M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, denote by $\mathsf{CD}_k(M, I)$ the random variable $(c, \bar{D}_I)$, where $(c, D) \leftarrow \mathsf{Commit}(k, M)$ and $\bar{D} \leftarrow (\mathsf{Extract}(D, i))_{i\in[n]}$. Let $\epsilon \in \mathbb{R}_{[0,1]}$. We say S is ϵ-statistically-hiding, if for all $n \in \mathbb{N}$, $I \subseteq [n]$, $M_1, M_2 \in \mathcal{M}^n$ with $(M_1)_I = (M_2)_I$, $k \in \mathsf{Setup}$,

$$\Delta(\mathsf{CD}_k(M_1, I), \mathsf{CD}_k(M_2, I)) \leq \epsilon \ .$$

A vector commitment scheme is extractable binding, if for every efficient committer, there exists an efficient extractor, such that for any efficient decommitter, if the committer gives a commitment that can be opened by a decommitter, then the extractor can already extract the corresponding messages from the committer at the time of the commitment.

Definition 9.4 (Extractable binding)**.** *Let $\epsilon : \mathbb{N}^3 \to \mathbb{R}_{[0,1]}$. We say a vector commitment scheme $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ is ϵ-extractable-binding, if for all t_1-bounded algorithms $\mathcal{A}_1$, $t_\mathcal{E}$-bounded algorithms $\mathcal{E}$, and t_2-bounded algorithms $\mathcal{A}_2$,*

$$\Pr\left[\begin{array}{c} \mathsf{Setup}() \to k, \mathcal{A}_1(k) \to_r c, \\ \mathcal{E}(k, r) \to [m_1, m_2, \ldots], \mathcal{A}_2(k, r) \to (m, c, d, i) : \\ \mathsf{Verify}(p, m, c, d, i) = 1 \wedge m_i \neq m \end{array}\right] \leq \epsilon(t_1, t_\mathcal{E}, t_2) \ .$$

9.1.2. Construction: Extractable binding

In the following, we show that the Merkle hash tree construction [Mer90] can be casted into a vector commitment scheme and that this construction is extractable binding if the used hash function is extractable binding.

Construction 9.1. *Let (K, H) denote a keyed hash function and let $L \in \mathbb{N}$. The following is a description of the hash tree construction proposed by Merkle cast into the definition of vector commitments.*

$\mathsf{Setup}() \to k$**:** *Run $K \to k$ and output k.*

$\mathsf{Commit}(k, [m_1, \ldots, m_n]) \to (c, D)$**:** *Set $l \leftarrow \min\{i \in \mathbb{N} : n \leq 2^i\}$. For $i \in \{0, \ldots, n-1\}$, compute $H(k, m_i) \to h_{i,l}$, and for $i \in \{n, \ldots, 2^l - 1\}$, set $h_{i,l} \leftarrow \perp$. For $i \in \{l-1, \ldots, 0\}$, $j \in \{0, \ldots, 2^i - 1\}$, compute $H(k, [h_{i-1,2j}, h_{i-1,2j+1}])$. Compute $H(k, [l, h_{0,0}]) \to c$. Set $D \leftarrow [h_{i,j}]_{i\in\{0,\ldots,l\}, j\in\{0,\ldots,2^i-1\}}$, and output (c, D).*

$\mathsf{Extract}(k, D, i^*) \to d$**:** *Let $D \to [h_{i,j}]_{i\in\{0,\ldots,l\}, j\in\{0,\ldots,2^i-1\}}$. Set $a_l \leftarrow i^*$. For $j \in \{l, \ldots, 1\}$, set $b_j \leftarrow a_j + 2(a_j \bmod 2) - 1$, $g_j \leftarrow h_{j,b_j}$, and $a_{j-1} \leftarrow \lfloor a_j/2 \rfloor$. Set $d = [g_1, \ldots, g_l]$ and output d.*

$\mathsf{Verify}(k, m, c, d, i^*) \to b^*$**:** *Let $d = [g_1, \ldots, g_l]$. Set $a_l \leftarrow i^*$ and compute $H(k, m) \to h_l$. For $i \in \{l, \ldots, 1\}$, if $a_i \bmod 2 = 0$, set $b_i \leftarrow [h_i, g_i]$, and if $a_i \bmod 2 = 1$, set $b_i \leftarrow [g_i, h_i]$, and then compute $H(k, b_i) \to h_{i-1}$ and set $a_{i-1} \leftarrow \lfloor a_i/2 \rfloor$. Compute $H(k, [l, h_0]) \to c'$. Set $b^* \leftarrow (c = c')$. Output b^*.*

Theorem 9.1. *The vector commitment scheme described in Construction 9.1 is correct.*

Proof. Let $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ be the scheme described in Construction 9.1. Furthermore, let $n \in [L]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, $i \in [n]$, $(c, D) \in \mathsf{Commit}(k, M)$, $d \in \mathsf{Extract}(k, D, i)$, and $b \in \mathsf{Verify}(k, M_i, c, d)$. By the definition of algorithms Commit and Extract, we observe that $d = [g_1, \ldots, g_l]$ are the siblings of the nodes in the hash tree $D = [h_{i,j}]_{i \in \{0,\ldots,l\}, j \in \{0,\ldots,2^i-1\}}$ on the path from leaf $H(k, M_i)$ to the root $h_{0,0}$. We observe that algorithm Verify recomputes this path starting with $H(k, M_i)$. Thus, $h_0 = h_{0,0} = c$. It follows that $b = 1$. □

Theorem 9.2. *Let (K, H) be an ϵ-extractable-binding hash function. The vector commitment scheme described in Construction 9.1 instantiated with (K, H) is ϵ'-extractable-binding with $\epsilon'(t_1, t_\mathcal{E}, t_2) = 2L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$.*

Proof. Let (K, H) be an ϵ-extractable-binding keyed hash function and let $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ be the vector commitment scheme described in Construction 9.1 instantiated with (K, H). To prove the theorem, we use the extractable-binding property of (K, H) to construct an extractor for the vector commitment scheme.

Fix $t_1, t_\mathcal{E}, t_2 \in \mathbb{N}$ and t_1-bounded algorithm $\mathcal{A}_1$. We observe that, because (K, H) is ϵ-extractable-binding, for any t-bounded algorithm $\mathcal{A}$, there exists a $t_\mathcal{E}$-bounded algorithm $\mathcal{E}^H_\mathcal{A}$ such that for any t_2-bounded algorithm $\mathcal{A}_2$, for experiment $\mathcal{A}(k) \to_r h$, $\mathcal{A}_2(k, r) \to m$, and $\mathcal{E}^H_\mathcal{A}(k, r) \to m'$, we have $H(k, m) = h$ and $m \neq m'$ with probability at most $\epsilon(t, t_\mathcal{E}, t_2)$.

Define $\mathcal{A}_{0,0}$ as an algorithm that on input k, samples r, computes $\mathcal{E}^H_{\mathcal{A}_1}(k, r) \to [l, h_{0,0}]$, and outputs $h_{0,0}$. Recursively, define $\mathcal{A}_{i,j}$ as an algorithm that on input k, samples r, computes $\mathcal{E}^H_{\mathcal{A}_{i-1,\lfloor j/2 \rfloor}}(k, r) \to [h_0, h_1]$, and outputs $h_{j \bmod 2}$.

Now define the vector extraction algorithm $\mathcal{E}$ as follows. On input (k, r), first compute $\mathcal{E}^H_{\mathcal{A}_1}(k, r) \to_{r'} [l, h_{0,0}]$. Then, for $i \in \{0, \ldots, 2^l - 1\}$, sample r_i and compute $\mathcal{E}^H_{\mathcal{A}_{l,i}}(k, [r, r', r_i]) \to m_{i+1}$. Output $[m_1, \ldots, m_{2^l}]$.

We observe that for any t_2-bounded algorithm $\mathcal{A}_2$, for experiment $\mathcal{A}_1(k) \to_r c$, $\mathcal{A}_2(k, r) \to (m, d, i)$, and $\mathcal{E}(k, r) \to M$, the probability of having $\mathsf{Verify}(k, m, c, d, i) = 1$ and $M_i \neq m$ is upper-bounded by the probability that at least one of the node extraction algorithms relied on by $\mathcal{E}$ fails. As there are at most $2L$ nodes in the tree, this probability is upper-bounded by $2L * \epsilon(\max\{t_1, t_\mathcal{E}\}, t_\mathcal{E}, t_2)$.

It follows that Construction 9.1 is ϵ'-extractable-binding with $\epsilon'(t_1, t_\mathcal{E}, t_2) = 2L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$. □

9.1.3. Construction: Extractable binding and statistically hiding

We now combine a statistically hiding and extractable binding commitment scheme with the vector commitment scheme from Construction 9.1 to obtain a statistically

hiding (under selective opening) and extractable binding vector commitment scheme. The idea is to first commit with the statistically hiding scheme to each message separately and then produce a vector commitment to these individually generated commitments.

Construction 9.2. *Let* COM *be a commitment scheme and* VC *be a vector commitment scheme.*

$\mathsf{Setup}() \to k$**:** *Run* $\mathsf{COM.Setup}() \to k_1$, $\mathsf{VC.Setup}() \to k_2$, *set* $k \leftarrow (k_1, k_2)$, *and output* k.

$\mathsf{Commit}(k, [m_1, \ldots, m_n]) \to (c, D)$**:** *Let* $k \to (k_1, k_2)$. *For* $i \in \{1, \ldots, n\}$, *compute* $\mathsf{COM.Commit}(k_1, m_i) \to (c_i, d_i)$. *Next, compute* $\mathsf{VC.Commit}(k_2, [c_1, \ldots, c_n]) \to (c, D')$. *Set* $D \leftarrow ([(c_1, d_1), \ldots, (c_n, d_n)], D')$. *Output* (c, D).

$\mathsf{Extract}(k, D, i) \to d$**:** *Let* $k \to (k_1, k_2)$ *and* $D \to ([(c_1, d_1), \ldots, (c_n, d_n)], D')$. *Compute* $\mathsf{VC.Extract}(k_2, D', i) \to d'$. *Set* $d \leftarrow (c_i, d_i, d')$ *and output* d.

$\mathsf{Verify}(k, m, c, d, i) \to b$**:** *Let* $k \to (k_1, k_2)$ *and* $d \to (c', d', d'')$. *Verify* m *and* c *by computing* $\mathsf{COM.Verify}(k_1, m, c', d') \to b_1$ *and* $\mathsf{VC.Verify}(k_2, c', c, d'', i) \to b_2$. *Set* $b \leftarrow (b_1 \wedge b_2)$ *and output* b.

Theorem 9.3. *The vector commitment scheme described in Construction 9.2 is correct.*

Proof. Let $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ be the scheme described in Construction 9.2. Let $n \in [L]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, $i \in [n]$, $\mathsf{Commit}(k, M) \to (c, D)$, $\mathsf{Extract}(k, D, i) \to d$, and $\mathsf{Verify}(k, M_i, c, d) \to b$. We observe that we can write $D = ((c_i, d_i)_{i \in [n]}, D')$ and that $d \in \mathsf{VC.Extract}(k_2, D', i)$. By the correctness of COM, it follows that $\mathsf{COM.Verify}(k_1, M_i, c_i, d_i) = 1$, and by the correctness of VC, it follows that $\mathsf{HT.VC}(k_2, c_i, c, d, i) = 1$. Thus, $\mathsf{Verify}(k, M_i, c, d, i) = 1$. □

Theorem 9.4. *The vector commitment scheme described in Construction 9.2 is* $L\epsilon$*-statistically-hiding (under selective opening) if the commitment scheme* COM *is* ϵ*-statistically-hiding.*

Proof. Let $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Extract}, \mathsf{Verify})$ be the scheme described in Construction 9.2. For any $n \in [L]$, $I \subseteq [n]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, and event $A \subseteq \Omega(\mathsf{CD}_k(M, I))$, denote by $\Pr_M(A)$ the probability that A is observed when sampling from $\mathsf{CD}_k(M, I)$.

Let $n \in [L]$, $I \subseteq [n]$, $M_1, M_2 \in \mathcal{M}^n$ with $(M_1)_I = (M_2)_I$, $k \in \mathsf{Setup}$. By the definition of the statistical distance, we have

$$\Delta(\mathsf{CD}_k(M_1, I), \mathsf{CD}_k(M_2, I)) = \sum_{c, \bar{D}_I} \left| \Pr_{M_1}(c, \bar{D}_I) - \Pr_{M_2}(c, \bar{D}_I) \right| .$$

We observe that for any $M = (m_1, \ldots, m_n) \in \mathcal{M}^n$, algorithm $\mathsf{Commit}(k, M) \to (c, D)$ first computes $\mathsf{Com}(k_1, m_i) \to (c_i, d_i)$, for each $i \in [n]$, and then computes $\mathsf{HT.Tree}(k_2, [c_1, \ldots, c_n]) \to T$. Denote $\tilde{C} = (c_i)_{i\in[n]}$ and $\tilde{D} = (d_i)_{i\in[n]}$. By the law of total probability we have

$$\Pr_M(c, \bar{D}_I) = \sum_{\substack{\tilde{C}, \bar{D}_I: \\ c, \bar{D}_I}} \Pr_M(c, \bar{D}_I | \tilde{C}, \tilde{D}_I) * \Pr_M(\tilde{C}, \tilde{D}_I) \ .$$

Furthermore, we observe that c and T are determined by $(k, \tilde{C})$. We also observe that on input k, $D = (T, (c_i, d_i)_{i\in[n]})$, and $i \in [n]$, algorithm $\mathsf{Extract}$ computes $\mathsf{HT.Path}(k_2, T, i) \to P$ and outputs $d = (c_i, d_i, P)$. Hence, the output d is determined by (k, T, i, c_i, d_i). Hence, $\Pr_M(c, \bar{D}_I | \tilde{C}, \tilde{D}_I) > 0$ implies $\Pr_M(c, \bar{D}_I | \tilde{C}, \tilde{D}_I) = 1$. It follows that

$$\sum_{\substack{\tilde{C}, \bar{D}_I: \\ c, \bar{D}_I}} \Pr_M(c, \bar{D}_I | \tilde{C}, \tilde{D}_I) * \Pr_M(\tilde{C}, \tilde{D}_I) = \sum_{\substack{\tilde{C}, \bar{D}_I: \\ c, \bar{D}_I}} \Pr_M(\tilde{C}, \tilde{D}_I) \ .$$

Next, we observe from the description of algorithm Commit that each call $\mathsf{Com}(M_i) \to (c_i, d_i)$ is independent of the other calls $\mathsf{Com}(M_j) \to (c_j, d_j)$, $j \neq i$. Thus,

$$\Pr_M(\tilde{C}, \tilde{D}_I) = \Pr_M(\tilde{C}_I, \tilde{D}_I) * \prod_{i\in[n]\setminus I} \Pr_M(c_i) \ .$$

By the description of algorithms Commit and $\mathsf{Extract}$, we have that $(\tilde{C}_I, \tilde{D}_I)$ is determined by M_I. We observe that $(M_1)_I = (M_2)_I$. Thus,

$$\Pr_{M_1}(\tilde{C}_I, \tilde{D}_I) = \Pr_{M_2}(\tilde{C}_I, \tilde{D}_I) \ .$$

It follows that

$$\begin{aligned}
&\left| \Pr_{M_1}(\tilde{C}_I, \tilde{D}_I) * \left(\prod_{i\in[n]\setminus I} \Pr_{M_1}(c_i) \right) - \Pr_{M_2}(\tilde{C}_I, \tilde{D}_I) * \left(\prod_{i\in[n]\setminus I} \Pr_{M_2}(c_i) \right) \right| \\
&= \Pr_{M_1}(\tilde{C}_I, \tilde{D}_I) * \left| \left(\prod_{i\in[n]\setminus I} \Pr_{M_1}(c_i) \right) - \left(\prod_{i\in[n]\setminus I} \Pr_{M_2}(c_i) \right) \right| \\
&\leq \Pr_{M_1}(\tilde{C}_I, \tilde{D}_I) * \sum_{i\in[n]\setminus I} \left| \Pr_{M_1}(c_i) - \Pr_{M_2}(c_i) \right| \ .
\end{aligned}$$

We observe that $|\Pr_{M_1}(c_i) - \Pr_{M_2}(c_i)| \leq \epsilon$ because Com is ϵ-statistically-hiding. It follows that

$$\sum_{i\in[n]\setminus I} \left| \Pr_{M_1}(c_i) - \Pr_{M_2}(c_i) \right| \leq L\epsilon \ .$$

In summary, we obtain

$$\Delta(\mathsf{CD}_k(M_1, I), \mathsf{CD}_k(M_2, I)) \leq L\epsilon \ .$$

□

Theorem 9.5. *If* COM *and* VC *of Construction 9.2 are ϵ-extractable-binding, Construction 9.2 is an ϵ'-extractable-binding vector commitment scheme with $\epsilon'(t_1, t_\mathcal{E}, t_2) = L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$.*

Proof. Let $t_1, t_\mathcal{E}, t_2 \in \mathbb{N}$ and fix any t_1-bounded algorithm $\mathcal{A}_1$ that on input k outputs a commitment c.

We observe that because VC is ϵ-extractable-binding that there exists a $t_\mathcal{E}$-bounded extraction algorithm $\mathcal{E}_0$ such that for any t_2-bounded algorithm $\mathcal{A}_2$, for experiment $\mathsf{Setup} \to k$, $\mathcal{A}_1(k) \to_r c$, $\mathcal{A}_2(k, r) \to (c_i, d, i)$, and $\mathcal{E}_0(k, r) \to C$, we have $\mathsf{VC.Verify}(k_2, c_i, c, d, i) = 1$ and $C_i \neq c_i$ with probability at most $\epsilon(t_1, t_\mathcal{E}, t_2)$.

Define $\mathcal{A}_i$ as an algorithm that on input k, samples r, computes $\mathcal{E}_0(k, r) \to C$, and outputs C_i.

For $i > 0$, define $\mathcal{E}_i$ as a $t_\mathcal{E}$-bounded extraction algorithm such that for any t_2-bounded algorithm $\mathcal{A}_2$, for experiment $K \to k$, $\mathcal{A}_i(k) \to_r c_i$, $\mathcal{E}_i(k, r) \to m'_i$, $\mathcal{A}_2(k, r) \to (m_i, d_i)$, we have $\mathsf{COM.Verify}(k_1, m_i, c_i, d_i) = 1$ and $m_i \neq m'_i$ with probability at most $\epsilon(t_\mathcal{E}, t_\mathcal{E}, t_2)$.

Now we define a message vector extraction algorithm $\mathcal{E}$ as follows. On input (k, r), first compute $\mathcal{E}_1(k, r) \to_{r'} C$. Then, for $i \in [|C|]$, compute $\mathcal{E}_i(k, [r, r']) \to m_i$. Output $[m_1, \ldots, m_{|C|}]$.

We observe that for any t_2-bounded algorithm $\mathcal{A}_2$, for experiment $\mathcal{A}_1(k) \to_r c$, $\mathcal{A}_2(k, r) \to (m, d, i)$, and $\mathcal{E}(k, r) \to M$, we have $\mathsf{Verify}(k, m, c, d, i) = 1$ and $M_i \neq m$ with probability at most $L * \epsilon(t_1 + t_\mathcal{E}, t_\mathcal{E}, t_2)$.

It follows that the vector commitment scheme described in Construction 9.2 is ϵ'-extractable-binding with $\epsilon'(t_1, t_\mathcal{E}, t_2) = L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$. □

9.2. ELSA: Efficient long-term secure storage architecture

Now we present ELSA, a long-term secure storage architecture that efficiently protects large datasets. It provides long-term integrity and long-term confidentiality protection of the stored data.

ELSA uses statistically-hiding and extractable-binding vector commitments (as described in section 9.1) in combination with timestamps to achieve renewable and privacy preserving integrity protection. The confidential data is stored using proactive secret sharing to guarantee confidentiality protection secure against computational attacks. The data owner communicates with two subsystems (Figure 9.1),

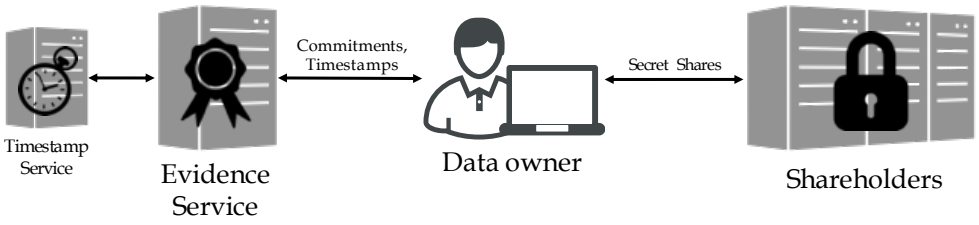

Figure 9.1.: Overview of the components of ELSA.

where one is responsible for data storage with confidentiality protection and the other one is responsible for integrity protection. The evidence service is responsible for integrity protection updates and the secret share holders are responsible for storing the data and maintaining confidentiality protection. The evidence service also communicates with a timestamp service that is used in the process of evidence generation.

9.2.1. Construction

We now describe the storage architecture ELSA in terms of the algorithms Init, Store, RenewTs, RenewCom, RenewShares, and Verify. Algorithm Init initializes the architecture, Store allows to store new files, RenewTs renews the protection if the timestamp scheme security is weakened, RenewCom renews the protection if the commitment scheme security is weakened, RenewShares renews the shares to protect against a mobile adversary who collects multiple shares over time, and Verify verifies the integrity of a retrieved file.

Initialization

The data owner uses algorithm ELSA.Init (Listing 9.1) to initialize the storage system. The algorithm gets as input a proactive secret sharing scheme SHARE, a set of shareholder addresses $(\text{shURL}_i)_{i\in[N]}$, a sharing threshold T, and an evidence service address esURL. It then initializes the storage module SH by means of the protocol SHARE.Setup and the evidence service module ES. During the initialization of the evidence service, the evidence list ES.`evidence` and the evidence renewal cache ES.`renewLists` are initialized as empty lists.

The interface to the storage system is as follows. We write SH.Store(name, dat) to denote that the data owner shares the data dat among the shareholders at location name by means of the protocol SHARE.Share. If the shared data dat is larger then the size of the message space of the secret sharing scheme, dat is first split into chunks that fit into the message space and then the chunks are shared individually. Each shareholder maintains a database that describes which shares belong to which data

item name. We write SH.Retrieve(name) to denote that the data owner retrieves the shares associated with the name name from the shareholders and reconstructs the data by means of the protocol SHARE.Reconstruct.

Listing 9.1: ELSA.Init(SHARE, $(\mathrm{shURL}_i)_{i\in[N]}, T, \mathrm{esURL}$)

```
SH.Init(SHARE, (shURL_i)_{i∈[N]}, T);
ES.Init(esURL);
```

Data storage

The client uses algorithm ELSA.Store (Listing 9.2) to store a set of data files $[\mathsf{file}_i]_{i\in[n]}$. For this, it chooses a signature instance SIG for signing the data, and a vector commitment scheme VC and a timestamp scheme TS that protect the data and the signature against ageing. Here, we assume that SIG is supplied with the secret key necessary for signature generation and VC is supplied with the public parameters necessary for commitment generation.

The algorithm first signs each of the data objects individually. It then stores the file data, the public key certificate of the signature scheme instance, and the generated signature at the secret sharing storage system. Afterwards, the algorithm generates a vector commitment (c, D) to the file data vector and the signatures. For each file, the corresponding decommitment is extracted and stored at the shareholders. The file names `filenames`, the commitment scheme instance VC, the commitment c, and the chosen timestamp scheme instance TS are sent to the evidence service.

When the evidence service receives $(\texttt{filenames}, \mathsf{VC}, c, \mathsf{TS})$, it does the following in algorithm AddCom (Listing 9.3). It first timestamps the commitment (VC, c) and thereby obtains a timestamp ts. Then, it starts a new evidence list $l = [(\mathsf{VC}, c, \mathsf{TS}, \mathrm{ts})]$ and assigns this list with all the file names in `filenames`. Also, it adds l to the list `renewLists`, which contains the lists that are updated on a timestamp renewal.

Timestamp renewal

Algorithm ES.RenewTs (Listing 9.4) is performed by the evidence service regularly in order to protect against the weakening of the currently used timestamp scheme and employ a new timestamp scheme. The algorithm gets as input a vector commitment scheme instance VC' and a timestamp scheme instance TS. It first creates a vector commitment (c', D') for the list of renewal items `renewLists`. Here, we are only interested in the extractable-binding property of VC', while the hiding property is not relevant as all of the data stored at the evidence service is already hidden

Listing 9.2: ELSA.Store($[\mathsf{file}_i]_{i\in[n]}$, SIG, VC, TS)

```
filenames ← {};
for i ∈ [n] do
    SIG.Sign(file_i.dat) → s_i;
    SH.Store(['data', file_i.name], [file_i.dat, SIG.Cert, s_i]);
    filenames += file_i.name;
end
VC.Commit([file_i.dat, SIG.Cert, s_i]_{i∈[n]}) → (c, D);
for i ∈ [n] do
    VC.Extract(D, i) → d;
    SH.Store(['decom', file_i.name, i], d);
end
ES.AddCom(filenames, VC, c, TS);
```

Listing 9.3: ES.AddCom(filenames, VC, c, TS)

```
TS.Stamp((VC, c)) → ts;
l ← [(VC, c, TS, ts)];
for name ∈ filenames do
    evidence[name] ← l;
    renewLists += l;
end
```

by the data owner. The freshly generated timestamp, commitment, and extracted decommitments are added to the corresponding evidence lists.

Listing 9.4: ES.RenewTs(VC', TS)

```
VC'.Commit(renewLists) → (c', D');
TS.Stamp((VC', c')) → ts;
for i ∈ [|renewLists|] do
    VC'.Extract(D', i) → d';
    renewLists[i] += (VC', c', d', TS, ts);
end
```

Commitment renewal

The data owner runs algorithm ELSA.RenewCom (Listing 9.5) to protect against a weakening of the security of the used commitment scheme. It chooses a new commitment scheme instance VC and a new timestamp scheme instance TS and proceeds as follows. First the current evidence lists ES.`evidence` are retrieved from the evidence service and complemented with the decommitment values stored at the shareholders. Next, a list with the data items, the signatures, and the current evidence for each data item is constructed. This list is then committed to using the vector commitment scheme VC. The decommitments are extracted and stored at the shareholders, and the commitment is added to the evidence at the evidence service using algorithm ES.AddComRenew.

Share renewal

There are two types of share renewal supported by ELSA. The first type (Listing 9.7) triggers the share renewal protocol of the secret sharing system (i.e., the protocol SHARE.Reshare). This interactive protocol refreshes the shares at the shareholders so that old shares, which may have leaked already, cannot be combined with the new shares, which are obtained after the protocol has finished, to reconstruct the stored data. The second type (Listing 9.8) replaces the sharing scheme entirely. This may be necessary if the configuration of the sharing scheme needs to be changed or it has additional security properties like verifiability (see proactive verifiable secret sharing [HJKY95]), which may require an update in order to remain secure. For the second type, the data is retrieved, stored at the new shareholders, and the old sharing system is shutdown.

Listing 9.5: ELSA.RenewCom(VC, TS)

```
comIndices ← {}; comCount ← {}; L ← [];
for name ∈ ES.evidence do
    SH.Retrieve(['data', name]) → (dat, SIG, s);
    ES.evidence[name] → e;
    for i ∈ |e| do
        if e_i.VC ≠ ⊥ then
            SH.Retrieve(['decom', name, i]) → e_i.d;
        end
    end
    L += (dat, SIG, s, e);
    comIndices[name] ← |L|;
    comCount[name] ← |e|;
end
VC.Commit(L) → (c, D);
for name ∈ ES.evidence do
    VC.Extract(D, comIndices[name]) → d;
    SH.Store(['decom', name, comCount[name]], d);
end
ES.AddComRenew(VC, c, TS);
```

Listing 9.6: ES.AddComRenew(VC, c, TS)

```
TS.Stamp((VC, c)) → ts;
l ← [(VC, c, TS, ts)];
renewLists ← [l];
for name ∈ evidence do
    evidence[name] += l;
end
```

Listing 9.7: ELSA.RenewShares()

```
SH.Reshare();
```

Listing 9.8: ELSA.RenewSharing(SHARE, $(\mathrm{shURL}_i)_{i\in[N]}, T$)

```
SH'.Init(SHARE, (shURL_i)_{i∈[N]}, T);
I ← ES.itemInfos;
for name ∈ I do
    SH.Retrieve('data/' + name) → dat;
    SH'.Store('data/' + name, dat);
end
SH.Shutdown();
SH ← SH';
```

Data Retrieval

The algorithm ELSA.Retrieve (Listing 9.9) describes the data retrieval procedure of ELSA. It gets as input the name of the data file that is to be retrieved. It then collects the evidence from the evidence service and the data from the shareholders. Next, the evidence is complemented with the decommitments and then the algorithm outputs the data with the corresponding evidence.

Listing 9.9: ELSA.Retrieve(name)

```
e ← ES.evidence[name];
for i ∈ [|e|] do
    if e_i.VC ≠ ⊥ then
        SH.Retrieve(['decom', name, i]) → e_i.d;
    end
end
SH.Retrieve(['data', name]) → (dat, SIG, s);
E ← (SIG, s, e);
return (dat, E);
```

Verification

Algorithm ELSA.Verify (Listing 9.10) describes how a verifier can check the integrity of a data item retrieved from ELSA. This algorithm gets as input a reference to the PKI (e.g., a trust anchor), the current verification time t_{verify}, the data to be checked dat, the storage time t_{store}, and the corresponding evidence E. The algorithm returns true, if dat is authentic and has been stored at time t_{store}.

In more detail, the verification algorithm proceeds as follows. It first checks whether the signature s contained in E is valid for the data object dat under signature scheme instance SIG. It also checks whether E contains a valid decommitment

from the first commitment to the data and the signature $(\mathsf{dat}, \mathsf{SIG}, s)$ and the decommitment has existed when the commitment was still valid. Here, $\mathrm{NTT}(i, E', t_{\mathsf{verify}})$ denotes the time of the next timestamp after position i in E' and $\mathrm{NCT}(i, E', t_{\mathsf{verify}})$ denotes the time of the timestamp corresponding to the next commitment after position i and we set $\mathrm{NTT}(i, E', t_{\mathsf{verify}}) = t_{\mathsf{verify}}$ if i is the last timestamp and $\mathrm{NCT}(i, E', t_{\mathsf{verify}}) = t_{\mathsf{verify}}$ if i is the last commitment in E'.

Next, the algorithm iterates over the remaining $|E'| - 1$ entries of E'. For each entry it checks whether it contains a recommitment. In this case, both, the commitment c and the timestamp ts, are checked. If a timestamp renewal has been performed, the commitment c' and the timestamp ts are checked. If all checks succeed, the algorithm returns 1, otherwise it returns 0.

Listing 9.10: $\mathsf{ELSA.Verify}(\mathsf{PKI}, t_{\mathsf{verify}} : \mathsf{dat}, t_{\mathsf{store}}, E) \to b$

```
(SIG, s, e) ← E;
((VC, c, d), (VC′, c′, d′), (TS, ts)) ← e_i;
t_nt ← NTT(1, e, t_verify); t_nc ← NCT(1, e, t_verify);
b ← SIG.Verify(PKI, ts.t : dat, s);
b ∧= VC.Verify(PKI, t_nc : (dat, SIG, s), c, d);
b ∧= TS.Verify(PKI, t_nt : c, ts, t_store);
L ← (VC, c, TS, ts);
for i ∈ [2, ..., |e|] do
    ((VC, c, d), (VC′, c′, d′), (TS, ts)) ← e_i;
    t_nt ← NTT(i, e, t_verify); t_nc ← NCT(i, e, t_verify);
    if VC = ⊥ then
        b ∧= VC′.Verify(PKI, t_nt : L, c′, d′);
        b ∧= TS.Verify(PKI, t_nt : c′, ts, ts.t);
        L += (VC′, c′, d′, TS, ts);
    else
        dat′ ← (dat, Cert, s, e[1, i − 1]);
        b ∧= VC.Verify(PKI, t_nc : dat′, c, d);
        b ∧= TS.Verify(PKI, t_nt : c, ts, ts.t);
        L ← (VC, c, TS, ts);
    end
end
return b;
```

9.2.2. Security

We now analyze the security of ELSA.

Computational Model

In a long-running system we have to consider adversaries that increase their computational power over time. For example, they may increase their computation speed or acquire new computational devices, such as quantum computers. We capture this by using the computational model described in chapter 3.

Integrity

First, we analyze integrity protection, by which we mean that it should be infeasible for an adversary to forge a valid evidence list for a data object, but the data object has not been authentically signed at the claimed time. This is captured in the following definition, where we define security with respect to a set of schemes $\mathcal{S}$ (e.g., commitment schemes, signature schemes, timestamp schemes) that are available within the context of the adversary. Also, we assume that each scheme instance $\mathcal{S}_i$ is associated with a breakage time t_i^b after which it is considered insecure. The experiment (Listing 9.11) has a setup phase, where all the scheme instances are initialized by means of parameter generation. We write $\mathcal{S}_i.\mathsf{Setup}() \to (sk, pk)$ to denote that a scheme potentially generates a secret parameter sk (e.g., a private signing key) and a public parameter pk (e.g., a public verification key or the parameters of a commitment scheme instance). We allow the adversary to access the secret parameters of an instance once it is considered insecure (via oracle Break).

Definition 9.5 (Integrity)**.** *We say* ELSA *is* $(\mathcal{M}, \epsilon)$*-unforgeable for schemes* $\mathcal{S}$*, if for any* p*-bounded machine* $\mathcal{A} \in \mathcal{M}$*,*

$$\Pr\left[\mathsf{Exp}^{\mathsf{Forge}}_{\mathcal{S}}(\mathcal{A}) = 1\right] \leq \epsilon(p) \ .$$

The experiment $\mathsf{Exp}^{\mathsf{Forge}}$ *is described in Listing 9.11.*

Next, we prove that the unforgeability security of ELSA can be reduced to the extractable binding security of the used commitment schemes and the unforgeability of the used signature schemes within their validity period.

Theorem 9.6. *Let* $\mathcal{M} = (\mathcal{M}_t)_t$ *specify which computational technology is available at which point in time and let* $\mathcal{S}$ *be a set of cryptographic schemes, where each scheme* $\mathcal{S}_i \in \mathcal{S}$ *is associated with a breakage time* t_i^b *and is* ϵ_i*-secure against adversaries using computational technology* $\mathcal{M}_{t_i^b}$*. In particular, we require unforgeability-security for signature schemes and extractable-binding-security for commitment schemes. Let* p_E *be any computational bound and* L *be an upper bound on the maximum vector length of the commitment schemes in* $\mathcal{S}$*. Then,* ELSA *is* $(\mathcal{M}, \epsilon)$*-unforgeable for* $\mathcal{S}$ *with* $\epsilon(p) = (\sum_{i \in \mathsf{SIG}} \epsilon_i(p(t_i^b) p_E(t_i^b) L^2)) + (\sum_{i \in \mathsf{COM}} \epsilon_i(p(t_i^b), p_E(t_i^b), p(t_i^b)))$.

Listing 9.11: $\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$

SetupExperiment();
$\mathcal{A}^{\mathsf{Clock},\mathsf{PKI},\mathsf{SIG},\mathsf{TS},\mathsf{Break}} \rightarrow (\mathsf{dat}, t_{\mathsf{store}}, E)$;
$t_{\mathsf{verify}} \leftarrow \mathtt{time}$;
$b \leftarrow \mathsf{Verify}(\mathsf{PKI}, t_{\mathsf{verify}}; \mathsf{dat}, t_{\mathsf{store}}, E) \wedge \mathsf{dat} \notin Q_{t_{\mathsf{store}}}$;
return b;

SetupExperiment():
$\mathtt{time} \leftarrow 0$;
for $i \in [|\mathcal{S}|]$ **do**
  $\mathcal{S}_i.\mathsf{Setup} \rightarrow (sp, pp)$;
  $SP[i] \leftarrow sp$; $PP[i] \leftarrow pp$;
end

Clock(t):
if $t > \mathtt{time}$ **then**
  $\mathtt{time} \leftarrow t$;
end

PKI(i):
return $PP[i]$;

Break(i):
if $\mathtt{time} \geq \mathcal{S}_i.tb$ **then**
  return $SP[i]$;
end

SIG(i, m):
Assert $\mathcal{S}_i.type = sig$;
$Q[\mathtt{time}] \mathrel{+}= m$;
$\mathcal{S}_i.\mathsf{Sign}(SP[i]; m) \rightarrow s$;
return s;

TS(i, m):
Assert $\mathcal{S}_i.type = ts$;
$\mathsf{Sign}(i; [\mathtt{time}, m]) \rightarrow s$;
return $(\mathtt{time}, s)$;

Proof. Suppose any p-bounded machine $\mathcal{A}$ that interacts with interfaces Clock, PKI, SIG, and TS, and outputs $(\mathsf{dat}, t_{\mathsf{store}}, E)$. For each signature scheme $i \in \mathsf{SIG}$, construct a machine $\mathcal{B}_i$ with the goal to break the unforgeability of scheme i until time t_i^b. $\mathcal{B}_i$ in the signature unforgeability experiment gets as input a public key pk and access to a signing oracle Sign. Its goal is to output (m, s) such that $\mathcal{S}_i.\mathsf{Verify}(pk, m, s) = 1$ and the oracle Sign was not queried with m. $\mathcal{B}_i$, on input pk, does the following. It runs $\mathcal{A}$ until time t_i^b and simulates the environment of $\mathcal{A}$ with the following difference. $\mathcal{B}_i$ sets the public key of signature scheme i to pk and whenever the simulation of the experiment for $\mathcal{A}$ requires the generation of a signature for scheme i, $\mathcal{B}_i$ requests the signature from the oracle Sign. While $\mathcal{A}$ is running, $\mathcal{B}_i$ searches the outputs of $\mathcal{A}$ for a valid message-signature-pair, where the message has not been queried to the signing oracle thus far. $\mathcal{B}_i$ also uses the extractable-binding property of the commitment schemes, as follows. Whenever, $\mathcal{A}$ queries the timestamp service TS with a commitment, then $\mathcal{B}_i$ uses a p_E-bounded commitment message extractor to extract the corresponding messages out of $\mathcal{A}$. Let L be an upper bound on the maximum length supported by all the used vector commitment schemes COM. Then, $\mathcal{B}_i$ runs in at most $p_{\mathcal{B}_i} = p(t_i^b) * p_E(t_i^b) * L^2$ operations and adheres to the computational model $\mathcal{M}_{t_i^b}$.

We observe that an evidence object E is a sequence $(\mathsf{SIG}, s, CDT_1, \ldots, CDT_n)$, where $CDT_i = (\mathsf{VC}_i, c_i, d_i, \mathsf{VC}'_i, c'_i, d'_i, \mathsf{TS}_i, \mathrm{ts}_i)$. Define $CT_i = (\mathsf{VC}'_i, c'_i, d'_i, \mathsf{TS}_i, \mathrm{ts}_i)$. The verification algorithm of ELSA ensures that ts_i is a valid timestamp for $(\mathsf{VC}_j, c_j, CT_j, \ldots, CT_{i-1})$, where j is the largest index such that $\mathsf{VC}_j \neq \perp$ and $j \leq i$. If $\mathsf{VC}_j \neq \perp$, then it also ensures that d_i is a valid opening of c_i to $(\mathsf{dat}, \mathsf{SIG}, s, CDT_1, \ldots, CDT_{i-1})$. It follows that in every run in which $\mathcal{A}$ outputs $(\mathsf{dat}, t_{\mathsf{store}}, E)$ with $\mathsf{Verify}(\mathsf{PKI}, t_{\mathsf{verify}}; \mathsf{dat}, t_{\mathsf{store}}, E) = 1$ and $\mathsf{dat} \notin Q_{t_{\mathsf{store}}}$, there is at least one $\mathcal{B}_i$ that wins its unforgeability experiment before time t_i^b or at least one of the extractors used by $\mathcal{B}_i$ fails. Hence, the probability that $\mathcal{A}$ breaks ELSA is upper bounded by $(\sum_{i \in \mathsf{SIG}} \epsilon_i(p_{\mathcal{B}_i})) + (\sum_{i \in \mathsf{COM}} \epsilon_i(p(t_i^b), p_E(t_i^b), p(t_i^b)))$, where COM denotes the set of commitment schemes and SIG denotes the set of signature schemes of $\mathcal{S}$. □

Confidentiality

Next, we analyze confidentiality protection of ELSA. Intuitively, we require that an adversary with unbounded computational power who observes the data that is received by the evidence service and a subset of the shareholders does not learn any substantial information about the stored data. In particular, it should be guaranteed that an adversary does not learn anything about unopened files even if it retrieves some of the other files and the corresponding signatures, commitments, and timestamps.

We model this intuition by requiring that any (unbounded) adversary $\mathcal{A}$ should not be able to distinguish whether it interacts with a system that stores a file vector F_1 or a system that stores another file vector F_2, if $\mathcal{A}$ only opens a subset I of

files and F_1 and F_2 are identical on I. This is modeled in experiment $\mathsf{Exp}^{\mathsf{DIST}}$ (Algorithm 9.12), where we use the following notation. For a secret sharing scheme SHARE we denote by SHARE.AS the set of authorized shareholder subsets that can reconstruct the secret. For a protocol P, we write $\langle \mathsf{P} \rangle_{\mathsf{View}}$ to denote an execution of P where View contains all the data sent and received by the involved parties. For an involved party $\mathcal{P}$, we write $\mathsf{View}(\mathcal{P})$ to denote the data sent and received by party $\mathcal{P}$.

Definition 9.6 (Confidentiality)**.** *We say* ELSA *is ϵ-statistically-hiding for $\mathcal{S}$, if for all machines $\mathcal{A}$, subsets I, sets of files F_1, F_2 with $(F_1)_I = (F_2)_I$, for all $L \in \mathbb{N}$,*

$$\left|\Pr\left[\mathsf{Exp}^{\mathsf{Dist}}_{\mathcal{S},L}(\mathcal{A}, F_1, I) = 1\right] - \Pr\left[\mathsf{Exp}^{\mathsf{Dist}}_{\mathcal{S},L}(\mathcal{A}, F_2, I) = 1\right]\right| \leq \epsilon(L) \ .$$

The experiment $\mathsf{Exp}^{\mathsf{DIST}}$ is described in Algorithm 9.12.

Next, we show that ELSA indeed provides confidentiality protection as defined above if statistically hiding secret sharing and commitment schemes are used.

Theorem 9.7. *Let $\mathcal{S}$ be a set of schemes, where $\mathcal{S}$.SHARE is an ϵ-statistically-hiding secret sharing scheme and every commitment scheme in $\mathcal{S}$ is ϵ-statistically-hiding. Then,* ELSA *is ϵ'-statistically-hiding for $\mathcal{S}$ with $\epsilon'(L) = 2L\epsilon$.*

Proof. We observe that the statistical distance between the view of the evidence service and the receiver for F_1 and the view for F_2 is at most ϵ per call to ELSA′ because the vector commitment schemes in $\mathcal{S}$ are ϵ-statistically-hiding. The statistical distance between the view of an unauthorized subset of shareholders for F_1 and the view of the same subset of shareholders for F_2 is also at most ϵ because the secret sharing scheme is ϵ-statistically-hiding. It follows that the statistical distance for each query of the adversary diverges by at most 2ϵ. Hence, overall the statistical distance is bounded by $2L\epsilon$. □

With regards to protecting the network communication between the data owner and the shareholders we ideally require that information theoretically channels are used (e.g., based on Quantum Key Distribution and One-Time-Pad Encryption [GRTZ02]), so that a network provider, who could potentially intercept all of the secret share packages, cannot learn any information about the communicated data. If information theoretically secure channels are not available, we recommend to use very strong symmetric encryption (e.g., AES-256 [Nat01]).

9.3. Performance evaluation

We compare the performance of our new architecture ELSA with the performance of the storage architecture LINCOS (cf. chapter 7), which is the fastest existing storage architecture that provides long-term integrity and long-term confidentiality.

Listing 9.12: $\mathsf{Exp}^{\mathsf{Dist}}_{\mathcal{S},L}(\mathcal{A}, Files, I)$

```
SetupExperiment();
A^{Clock,ELSA'} → b;
return b;
```

```
SetupExperiment():
time ← 0; N ← 0;
Generate parameters for S;
ELSA.Init(S.SHARE, SH, ES);
```

```
Clock(t):
if t > time then
    time ← t;
end
```

```
ELSA'(op, param):
if N < L then
    N += 1; T ← T';
    if op = Store ∧ param ∉ S.SHARE.AS then
        ⟨ELSA.Store(Files, param)⟩_View;
        T ← param;
    else if op = Retrieve ∧ param ∈ I then
        ⟨ELSA.Retrieve(param)⟩_View;
    else if op = ReTs then
        ⟨ELSA.RenewTs(param)⟩_View;
    else if op = ReCom then
        ⟨ELSA.RenewCom(param)⟩_View;
    else if op = ReShare ∧ param ∉ S.SHARE.AS then
        ⟨ELSA.RenewShares()⟩_View;
        T' ← param;
    end
    return View(ES, SH_T, Receiver);
end
```

9.3.1. Protection Scenario

For our evaluation we consider a scenario inspired by the task of securely storing electronic health records in a medium sized doctor's office. The storage time frame is 100 years. Every month, 10 new data items of size 10 kB (e.g., prescription data of patients) are added. Every year, one document from each of the previous years is retrieved and verified (e.g., historic prescription data is read from the archives).

We assume the following renewal schedule for protecting the evidence against the weakening of cryptographic primitives. The signatures are renewed every 2 years, as this is a typical lifetime of a public key certificate, which is needed to verify the signatures. While signature scheme instances can only be secure as long as the

Table 9.1.: Overview of the used commitment and signature scheme instances and their usage period.

Years	Signatures	Commitments
2018-2030	RSA-2048	HM-256
2031-2066	XMSS-256	HM-256
2067-2091	XMSS-256	HM-256
2091-2118	XMSS-512	HM-512

corresponding private signing key is not leaked to an adversary, commitment scheme instances do not involve the usage of any secret parameters. Therefore, their security is not threatened by key leakage and we assume that they only need to be renewed every 10 years in order to adjust the cryptographic parameter sizes or to choose a new and more secure scheme.

In our architecture we instantiate signature and commitment schemes as follows. As signature scheme, we first use the RSA Signature Scheme [RSA78] and then switch to the post-quantum secure XMSS signature scheme [BDH11] by 2030, as we anticipate the development of large-scale quantum computers. Both of these schemes satisfy unforgeability security as required by Theorem 9.6. As the vector commitment scheme we use Construction 9.2 with the statistically hiding commitment scheme by Halevi and Micali [HM96] whose security is based on the security of the used hash function which we instantiate with members of the SHA-2 hash function family [Nat02]. If we model hash functions as random oracles, the extractable-binding property required by Theorem 9.6 is provided. This vector commitment scheme construction also provides statistical hiding security as required by Theorem 9.7. We adjust the signature and commitment scheme parameters over time as proposed by Lenstra and Verheul [LV01, Len04]. The resulting parameter sets are shown in Table 9.1.

For the storage system, we use the secret sharing scheme by Shamir [Sha79]. We run this scheme with 4 shareholders and a threshold of 3 shareholders are required for reconstruction. Secret shares are renewed every 5 years, where the resharing is carried out centrally by the data owner.

9.3.2. Results

We now present the results of our performance analysis. Figure 9.2 compares the computation time of the two systems, ELSA and LINCOS, and Figure 9.3 and Figure 9.4 compare the storage costs. The implementation was done using the programming language Java. The experiments were performed on a computer with a quad-core AMD Opteron CPU running at 2.3 GHz and the Java virtual machine was assigned 32 GB of RAM.

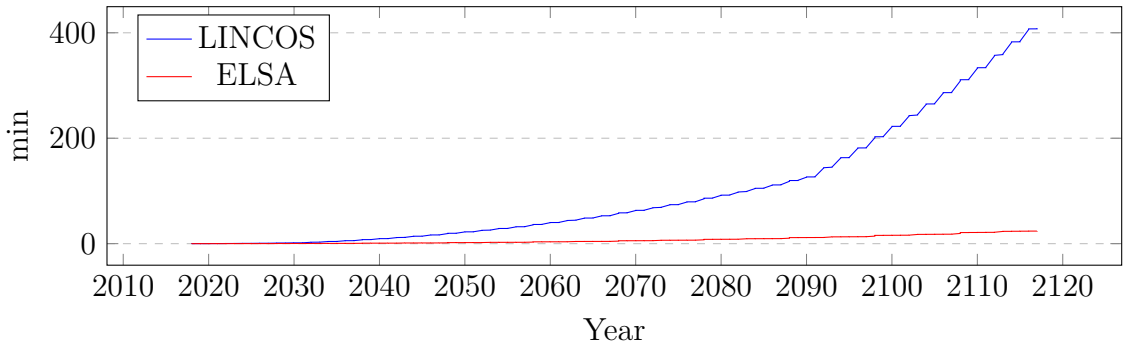

Figure 9.2.: Running time of the experiment.

Figure 9.2 shows that ELSA is much more computationally efficient compared to LINCOS. Completing the experiment using LINCOS took approximately 6.81 h, while it took only 24 min using ELSA. The biggest difference in the timings is observed when renewing timestamps. Timestamp renewal with LINCOS for year 2116 takes 21.89 min, while it takes only 0.34 s with ELSA. Data storage performance is also considerably faster with ELSA than with LINCOS. The same holds for the commitment renewal procedure. Data retrieval and verification performance is similar for the two systems.

Next, we observe by Figure 9.3 that ELSA is also more efficient compared to LINCOS when it comes to the consumed storage space at the evidence service. This is, again, because ELSA requires fewer timestamps to be generated and stored than LINCOS. After running for 100 years, the evidence service of ELSA consumes only 17.27 MB while the evidence service of LINCOS consumes 1.75 GB of storage space. We observe by Figure 9.4 that ELSA consumes slightly more storage space at the shareholders than LINCOS. This is because additional decommitment information for the vector commitments must be stored. After running for 100 years, a shareholder of ELSA consumes about 748 MB while a shareholder of LINCOS consumes about 559 MB of storage space.

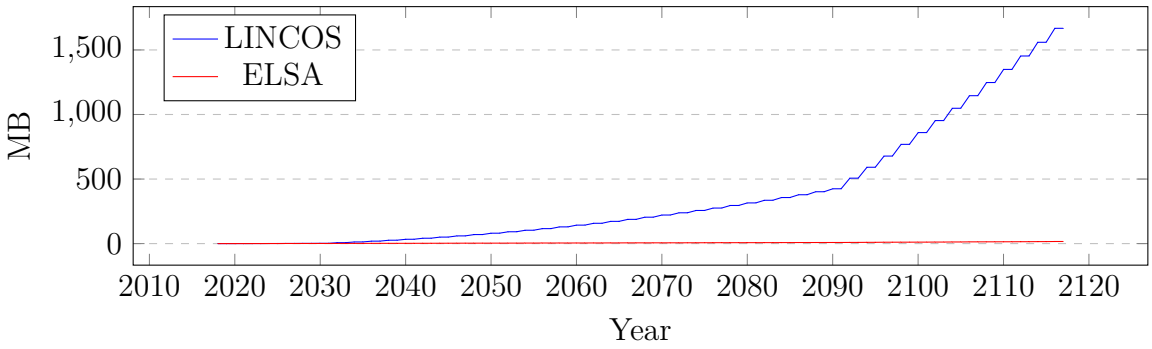

Figure 9.3.: Storage space consumed by the evidence service.

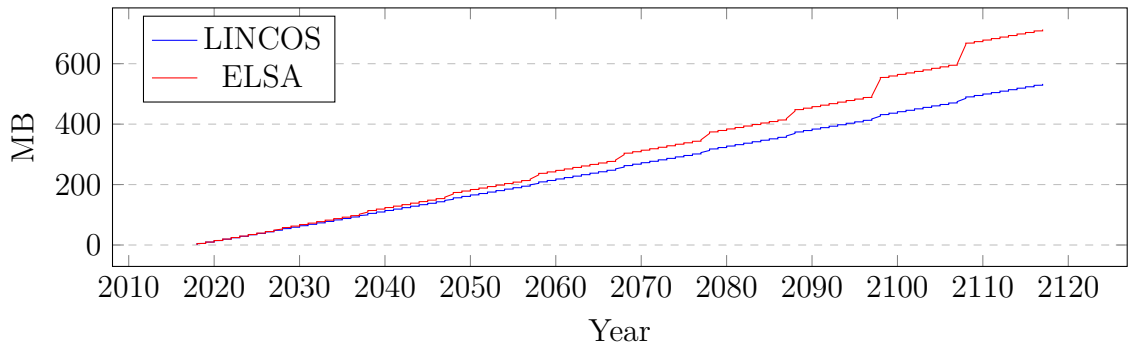

Figure 9.4.: Storage space consumed by a shareholder.

Conclusions

10. Conclusions

Summary of contributions. The contributions of this thesis are two-fold. Part I improves our understanding of the computational security of cryptographic schemes that are intended to provide protection over long periods of time (i.e., decades or centuries). Part II solves the problem of combining renewable integrity protection with information-theoretic confidentiality protection.

In Part I we first developed a new computational model that captures long-lived computationally-bounded entities whose computational power per unit of time increases over time. Based on this model, we then provided security proofs for long-term integrity protection schemes based on signature-based timestamps and hash-based timestamps. We also proposed a new cryptographic primitive called long-term commitment which is a cryptographic commitment that is information-theoretically hiding and long-term binding.

In Part II, we then gave the first construction of a long-term secure storage system that provides renewable integrity and authenticity protection and information-theoretically secure confidentiality protection. We also proposed a variation of this construction which additionally provides long-term access pattern hiding security and a variation which achieves improved performance for databases that contain a large number of small data items.

Directions for future work. We see several directions for future work. One important open problem is to prove the security of efficient and privacy-preserving long-term integrity schemes based on standard cryptographic assumptions. So far, our analysis of hash-based long-term time-stamping and long-term commitments either requires an idealized setting (e.g., the ideal primitive model), or non-standard assumptions (e.g., the existence of extractable collision resistant hash functions).

With respect to our long-term secure storage architecture, it would be worthwhile to extend it to support multiple data owners that store their data collectively and can selectively share parts of it with each other. Foreseeably, this requires the integration of access control mechanisms into the existing architecture and the development of new ORAM schemes that support multiple data owners. Another important research challenge is to enable secure computation on the stored and protected data. This could be achieved, for example, by using secure multiparty computation protocols. Such protocols could also be used for performing commitment renewals, which are currently performed by the data owner. Then, long-term integrity and confidentiality protection without the help of the data owner would be possible.

With respect to the components of our storage architecture, it would be desirable to improve the efficiency of proactive secret sharing for storing large amounts of data. The existing schemes are designed for small datasizes on the order of a few kilobytes, but storage architectures must often handle data of size several megabytes up to several gigabytes. Furthermore, a wide class of proactive secret sharing schemes relies on computationally binding and statistically hiding homomorphic commitment schemes, but the currently used schemes are insecure against quantum computers. Thus, post-quantum secure commitment scheme candidates for proactive secret sharing must be explored. Besides proactive secret sharing, our architecture ideally requires information-theoretically secure channels to provide long-term confidentiality protection of data in transit. The most promising technology for this currently is Quantum Key Distribution, but its performance and availability is still limited. Current QKD technology only allows for channel transmission rates of a few hundred kilobits per second over a distance of at most a few hundred kilometers. The performance of QKD technology must be improved and the practicality of quantum repeaters, which are required for establishing end-to-end secure connections over longer distances, must be demonstrated. Moreover, new connection establishment and channel protocols with support for Quantum Key Distribution and information-theoretically encryption must be developed and their security must be analyzed.

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[ACPZ01] C. Adams, P. Cain, D. Pinkas, and R. Zuccherato. Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP). RFC 3161 (Proposed Standard), August 2001. Updated by RFC 5816.

[BBMW14] Johannes Braun, Johannes Buchmann, Ciaran Mullan, and Alex Wiesmaier. Long term confidentiality: a survey. *Designs, Codes and Cryptography*, 71(3):459–478, Jun 2014.

[BCC+17] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The hunting of the snark. *Journal of Cryptology*, 30(4):989–1066, Oct 2017.

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 1–15, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[BCMS97] Gilles Brassard, Claude Crepeau, Dominic Mayers, and Louis Salvail. A brief review on the impossibility of quantum bit commitment. arXiv quant-ph/9712023, 1997.

[BDH11] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. Xmss - a practical forward secure signature scheme based on minimal security assumptions. In Bo-Yin Yang, editor, *Post-Quantum Cryptography*, pages 117–129, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[Ben80] Paul Benioff. The computer as a physical system: A microscopic quantum mechanical hamiltonian model of computers as represented by turing machines. *Journal of Statistical Physics*, 22(5):563–591, May 1980.

[BHS93] Dave Bayer, Stuart Haber, and W. Scott Stornetta. Improving the efficiency and reliability of digital time-stamping. In *Sequences II: Methods in Communication, Security and Computer Science*, pages 329–334. Springer-Verlag, 1993.

[BJLS16] Christoph Bader, Tibor Jager, Yong Li, and Sven Schäge. On the impossibility of tight cryptographic reductions. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 273–304, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[BKW13] Johannes A. Buchmann, Evangelos G. Karatsiolis, and Alexander Wiesmaier. *Introduction to Public Key Infrastructures*. Springer, 2013.

[BL06] Ahto Buldas and Sven Laur. Do broken hash functions affect the security of time-stamping schemes? In Jianying Zhou, Moti Yung, and Feng Bao, editors, *Applied Cryptography and Network Security*, pages 50–65, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[BL07] Ahto Buldas and Sven Laur. Knowledge-binding commitments with applications in time-stamping. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography – PKC 2007*, pages 150–165, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.

[BL13] Ahto Buldas and Risto Laanoja. Security proofs for hash tree time-stamping using hash functions with small output size. In Colin Boyd and Leonie Simpson, editors, *Information Security and Privacy*, pages 235–250, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BLSW05] Ahto Buldas, Peeter Laud, Märt Saarepera, and Jan Willemson. Universally composable time-stamping schemes with audit. In Jianying Zhou, Javier Lopez, Robert H. Deng, and Feng Bao, editors, *Information Security*, pages 359–373, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[BN08] Ahto Buldas and Margus Niitsoo. Can we construct unbounded time-stamping schemes from collision-free hash functions? In Joonsang Baek, Feng Bao, Kefei Chen, and Xuejia Lai, editors, *Provable Security*, pages 254–267, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[BN10] Ahto Buldas and Margus Niitsoo. Optimally tight security proofs for hash-then-publish time-stamping. In Ron Steinfeld and Philip Hawkes, editors, *Information Security and Privacy*, pages 318–335, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[BS04] Ahto Buldas and Märt Saarepera. On provably secure time-stamping schemes. In Pil Joong Lee, editor, *Advances in Cryptology - ASIACRYPT 2004*, pages 500–514, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[BSG11] A. Jerman Blazic, S. Saljic, and T. Gondrom. Extensible Markup Language Evidence Record Syntax (XMLERS). RFC 6283 (Proposed Standard), July 2011.

[Buc16] Johannes Buchmann. *Einführung in die Kryptographie*. Springer Spektrum, Berlin, Heidelberg, 2016.

[CCK+08] Ran Canetti, Ling Cheung, Dilsun Kaynar, Nancy Lynch, and Olivier Pereira. Modeling computational security in long-lived systems. In Franck van Breugel and Marsha Chechik, editors, *CONCUR 2008 - Concurrency Theory*, pages 114–130, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In Joe Kilian, editor, *Advances in Cryptology — CRYPTO 2001*, pages 19–40, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[CF13] Dario Catalano and Dario Fiore. Vector commitments and their applications. In Kaoru Kurosawa and Goichiro Hanaoka, editors, *Public-Key Cryptography – PKC 2013*, pages 55–72, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[CH94] Ran Canetti and Amir Herzberg. Maintaining security in the presence of transient faults. In Yvo G. Desmedt, editor, *Advances in Cryptology — CRYPTO '94*, pages 425–438, Berlin, Heidelberg, 1994. Springer Berlin Heidelberg.

[CLP14] Kai-Min Chung, Zhenming Liu, and Rafael Pass. Statistically-secure oram with $\tilde{O}(\log^2 n)$ overhead. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 62–81, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, pages 151–158, New York, NY, USA, 1971. ACM.

[CR72] Stephen A. Cook and Robert A. Reckhow. Time-bounded random access machines. In *Proceedings of the Fourth Annual ACM Symposium on Theory of Computing*, STOC '72, pages 73–80, New York, NY, USA, 1972. ACM.

[Deu85] David Deutsch. Quantum theory, the church-turing principle and the universal quantum computer. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 400(1818):97–117, 1985.

[DH76] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, Nov 1976.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685.

[DRS09] Yevgeniy Dodis, Thomas Ristenpart, and Thomas Shrimpton. Salvaging merkle-damgård for practical applications. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 371–388, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[ETS10a] ETSI. CMS advanced electronic signatures (CAdES), Jul. 2010.

[ETS10b] ETSI. XML advanced electronic signatures (XAdES), jan 2010.

[Fey82] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6-7):467–488, 1982.

[GBP07] T Gondrom, R Brandner, and U Pordesch. Evidence Record Syntax (ERS), 2007.

[Gir18] Damien Giry. https://www.keylength.com, 2018.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, 1988.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, May 1996.

[Gol87] O. Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing*, STOC '87, pages 182–194, New York, NY, USA, 1987. ACM.

[GRTZ02] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. Quantum cryptography. *Rev. Mod. Phys.*, 74:145–195, Mar 2002.

[Hab06] Stuart Haber. Content integrity service for long-term digital archives. In *Archiving 2006*, pages 159–164, Ottawa, Canada, 2006. IS&T.

[HJKY95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In Don Coppersmith, editor, *Advances in Cryptology — CRYPTO' 95*, pages 339–352, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[HM96] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 201–215, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[HS91] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology-CRYPT0' 90*, pages 437–455, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[Kni96] E. Knill. Conventions for quantum pseudocode. Technical report, Los Alamos National Laboratory, 6 1996.

[LC99] Hoi-Kwong Lo and H. F. Chau. Unconditional security of quantum key distribution over arbitrarily long distances. *Science*, 283(5410):2050–2056, 1999.

[Len04] Arjen K Lenstra. *The Handbook of Information Security*, chapter Key lengths. Wiley, 2004.

[LG04] Dimitrios Lekkas and Dimitris Gritzalis. Cumulative notarization for long-term preservation of digital signatures. *Computers & Security*, 23(5):413–424, 2004.

[LV01] Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology*, 14(4):255–293, Sep 2001.

[May97] Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Physical review letters*, 78(17):3414, 1997.

[May01] Dominic Mayers. Unconditional security in quantum cryptography. *Journal of the ACM (JACM)*, 48(3):351–406, 2001.

[Mer90] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *Advances in Cryptology — CRYPTO' 89 Proceedings*, pages 218–238, New York, NY, 1990. Springer New York.

[Nat77] National Institute of Standards and Technology (NIST). Data encryption standard (DES) (FIPS PUB 46). Federal Information Processing Standards Publications (FIPS PUBS), January 1977.

[Nat95] National Institute of Standards and Technology (NIST). Secure hash standard (SHS) (FIPS PUB 180-1). Federal Information Processing Standards Publications (FIPS PUBS), April 1995.

[Nat01] National Institute of Standards and Technology (NIST). Advanced encryption standard (AES) (FIPS PUB 197). Federal Information Processing Standards Publications (FIPS PUBS), November 2001.

[Nat02] National Institute of Standards and Technology (NIST). Secure hash standard (SHS) (FIPS PUB 180-2). Federal Information Processing Standards Publications (FIPS PUBS), August 2002.

[Nat17] National Institute of Standards and Technology (NIST). https://csrc.nist.gov/projects/post-quantum-cryptography/round-1-submissions. Website, 2017.

[OY91] Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '91, pages 51–59, New York, NY, USA, 1991. ACM.

[Ped92] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 129–140, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[RS04] Phillip Rogaway and Thomas Shrimpton. Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance. In Bimal Roy and Willi Meier, editors, *Fast Software Encryption*, pages 371–388, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[RSA78] Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communincations of the ACM*, 21(2):120–126, 1978.

[SBPC+09] Valerio Scarani, Helle Bechmann-Pasquinucci, Nicolas J. Cerf, Miloslav Dušek, Norbert Lütkenhaus, and Momtchil Peev. The security of practical quantum key distribution. *Rev. Mod. Phys.*, 81:1301–1350, Sep 2009.

[Sch14] Jörg Schwenk. Modelling time for authenticated key exchange protocols. In Mirosław Kutyłowski and Jaideep Vaidya, editors, *Computer Security - ESORICS 2014*, pages 277–294, Cham, 2014. Springer International Publishing.

[SFI+11] M. Sasaki, M. Fujiwara, H. Ishizuka, W. Klaus, K. Wakui, M. Takeoka, S. Miki, T. Yamashita, Z. Wang, A. Tanaka, K. Yoshino, Y. Nambu, S. Takahashi, A. Tajima, A. Tomita, T. Domeki, T. Hasegawa, Y. Sakai, H. Kobayashi, T. Asai, K. Shimizu, T. Tokura, T. Tsurumaru, M. Matsui,

T. Honjo, K. Tamaki, H. Takesue, Y. Tokura, J. F. Dynes, A. R. Dixon, A. W. Sharpe, Z. L. Yuan, A. J. Shields, S. Uchikoga, M. Legré, S. Robyr, P. Trinkler, L. Monat, J.-B. Page, G. Ribordy, A. Poppe, A. Allacher, O. Maurhart, T. Länger, M. Peev, and A. Zeilinger. Field test of quantum key distribution in the tokyo qkd network. *Opt. Express*, 19(11):10387–10409, May 2011.

[SGG+00] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. Optical quantum random number generator. *Journal of Modern Optics*, 47(4):595–598, 2000.

[Sha79] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, November 1979.

[Sho99] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Review*, 41(2):303–332, 1999.

[SvDS+13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path oram: An extremely simple oblivious ram protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, pages 299–310, New York, NY, USA, 2013. ACM.

[The18] The Legion of the Bouncy Castle. https://www.bouncycastle.org, 2018.

[Tur37] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[Unr16] Dominique Unruh. Computationally binding quantum commitments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, pages 497–527, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[VBC+15] 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*, 50:16 – 32, 2015.

[VCBH13] Martín A. Gagliotti Vigil, Daniel Cabarcas, Johannes Buchmann, and Jingwei Huang. Assessing trust in the long-term protection of documents. In *ISCC*, pages 185–191, 2013.

[WC81] Mark N. Wegman and J.Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265 – 279, 1981.

Erklärung zur Dissertation

Gemäß § 9 Abs. 1 der 8. Novelle der Promotionsordnung der Technischen Universität Darmstadt erkläre ich hiermit, dass ich die vorliegende Arbeit – abgesehen von den in ihr ausdrücklich genannten Hilfen – selbständig verfasst habe.

Ort, Datum	Unterzeichner