Muth *et al. EURASIP Journal on Information Security* (2020) 2020:9
https://doi.org/10.1186/s13635-020-00108-9

EURASIP Journal on Information Security

RESEARCH Open Access

ELSA: efficient long-term secure storage of large datasets (full version)*

Philipp Muth[1*], Matthias Geihs[2], Tolga Arul[3], Johannes Buchmann[2] and Stefan Katzenbeisser[4]

Abstract

An increasing amount of information today is generated, exchanged, and stored digitally. This also includes long-lived and highly sensitive information (e.g., electronic health records, governmental documents) whose integrity and confidentiality must be protected over decades or even centuries. While there is a vast amount of cryptography-based data protection schemes, only few are designed for long-term protection. Recently, Braun et al. (AsiaCCS'17) proposed the first long-term protection scheme that provides renewable integrity protection and information-theoretic confidentiality protection. However, computation and storage costs of their scheme increase significantly with the number of stored data items. As a result, their scheme appears suitable only for protecting databases with a small number of relatively large data items, but unsuitable for databases that hold a large number of relatively small data items (e.g., medical record databases).

In this work, we present a solution for *efficient* long-term integrity and confidentiality protection of large datasets consisting of relatively small data items. First, we construct a renewable vector commitment scheme that is information-theoretically hiding under selective decommitment. We then combine this scheme with renewable timestamps and information-theoretically secure secret sharing. The resulting solution requires only a single timestamp for protecting a dataset while the state of the art requires a number of timestamps linear in the number of data items. Furthermore, we extend the scheme, that supports a single client, to a multi-client setting. Subsequently, we characterize the arising challenges with respect to integrity and confidentiality and discuss how our multi-client scheme tackles them. We implemented our solution and measured its performance in a scenario where 9600 data items are aggregated, stored, protected, and verified over a time span of 80 years. Our measurements show that our new solution completes this evaluation scenario an order of magnitude faster than the state of the art.

Keywords: Long-term security, Authenticity, Commitments, Timestamps, Proactive secret sharing

1 Introduction

1.1 Motivation and problem statement

Today, huge amounts of information are generated, exchanged, and stored digitally, and these amounts will further grow in the future. Much of this data contains sensitive information (e.g., electronic health records, governmental documents, enterprise documents) and requires protection of *integrity* and *confidentiality*. Integrity protection means that illegitimate and accidental changes of data can be discovered. Confidentiality protection means that only authorized parties can access the data. Depending on the use case, protection may be required for several decades or even centuries. Databases that require protection are often complex and consist of a large number data items of varying sizes that require continuous confidentiality protection and whose integrity must be verifiable independent from the other data items.

Today, integrity of digitally stored information is most commonly ensured using digital signatures (e.g., RSA [21]) and confidentiality is ensured using encryption (e.g., AES [19]). The commonly used schemes are secure under certain computational assumptions. For example, they

*Correspondence: muth@seceng.informatik.tu-darmstadt.de
This work has been co-funded by the DFG as part of project S6 within CRC 1119 CROSSING. A short version of this paper appears in the proceedings of ICISC'18.
[1]Seceng, TU Darmstadt, Hochschulstr 10, 64293 Darmstadt, Germany
Full list of author information is available at the end of the article

require that computing the prime factors of a large integer is infeasible. However, as computing technology and cryptanalysis advances over time, computational assumptions made today are likely to break at some point in the future (e.g., RSA will become insecure once quantum computers are available [24]). Consequently, *computationally secure* cryptographic schemes have a limited lifetime and are insufficient to provide *long-term security*.

Several approaches have been developed to mitigate long-term security risks. Bayer et al. [1] proposed a technique for prolonging the validity of a digital signature by using digital timestamps. Based on their idea, a variety of long-term integrity protection schemes have been developed. An overview of existing long-term integrity schemes is given by Vigil et al. in [25]. In contrast to integrity protection, confidentiality protection cannot be prolonged. There is no protection against an adversary that stores ciphertexts today, waits until the encryption is weakened, and then breaks the encryption and obtains the plaintexts. Thus, if long-term confidentiality protection is required, then strong confidentiality protection must be applied from the start. A very strong form of protection can be achieved by using *information theoretically secure* schemes, which are invulnerable to computational attacks. For example, key exchange can be realized using quantum key distribution [10], encryption can be realized using one-time pad encryption [23], and data storage can be realized using proactive secret sharing [14]. An overview of information theoretically secure solutions for long-term confidentiality protection is given by Braun et al. [4].

Recently, Braun et al. proposed LINCOS [3], which is the first long-term secure storage architecture that combines long-term integrity with long-term confidentiality protection. While their system achieves high protection guarantees, it is only designed for storing and protecting a single large data object, but not databases that consist of a large number of small data items. One approach to store and protect large databases with LINCOS is to run an instance of LINCOS for each data item in parallel. However, with this construction, the amount of work scales linearly with the number of stored data items. Especially, if the database consists of a large number of relatively small data items, this introduces a large communication and computation overhead.

1.2 Contribution

This paper is an extension of [7], which introduced an efficient solution to storing and protecting large and complex datasets over long periods of time. Concretely, we present the long-term secure storage architecture ELSA that uses renewable vector commitments in combination with renewable timestamps and proactive secret sharing to achieve this. In this full version of the paper, we also provide an extension to ELSA to support multiple clients simultaneously while achieving the same security guaranties. The main contribution of this extension is MCELSA, the multi-client version of ELSA, that is, Sections 5 and 6. We also improve several technical details with respect to [7].

In Section 3, we construct an extractable-binding and statistically hiding vector commitment scheme. Such a scheme allows for committing to a large number of data items by a single short commitment. The extractable binding property of the scheme enables renewable integrity protection [5], while the statistical hiding property ensures information theoretic confidentiality. Our construction is based on statistically hiding commitments and hash trees [18]. We prove that our construction is extractable binding given that the employed commitment scheme and hash function are extractable binding. Furthermore, we prove that our construction is statistically hiding under selective opening, which guarantees that by opening the commitments to some of the data items no information about unopened data items is leaked. The construction of extractable binding and statistically hiding vector commitments may be of independent interest, for example, in the context of zero knowledge protocols [9].

In Section 4, we introduce the secure long-term storage architecture ELSA, which uses our new vector commitment scheme construction to achieve efficient protection of large datasets. While protecting a dataset with LINCOS requires the generation of a commitment and a timestamp for each data item separately, ELSA requires only a single vector commitment and a single timestamp to protect the same dataset. Hence, the number of timestamps is decreased from linear in the number of data items to constant and this drastically reduces the communication and computation complexity of the solution. Moreover, as the vector commitment scheme is hiding under selective decommitment, the integrity of stored data items can still be verified individually without revealing information about unopened data items. ELSA uses a separate service for storing commitments and timestamps, which allows for renewing the timestamp protection without access to the stored confidential data. The decommitments are stored together with the data items at a set of shareholders using proactive secret sharing. We show that the long-term integrity security of ELSA can be reduced to the unforgeability security of the employed timestamp schemes and the binding security of the employed commitment schemes within their usage period. Long-term confidentiality security is based on the statistical hiding security of the employed commitment and secret sharing schemes.

In Section 5, we introduce an extension of ELSA called MCELSA that provides efficient secure multi-client long-term storage solution. For that, we transfer the single-

client scheme ELSA that we constructed in Section 4, to a multi-client setting. MCELSA provides the same integrity and confidentiality guaranties as ELSA and involves the same parties, yet supports multiple clients instead of a single one. The multi-client setting entails several challenges regarding the distribution of maintenance tasks and access management. With respect to maintenance, we modify the protocols and the tasks that the parties are entrusted with, to balance the work load between parties in the multi-client setting. In terms of access management, we aim to avoid a single point of failure that is an individual instance deciding document access. We achieve this by entrusting the shareholders, who store the documents, with determining when to hand over shares to a client. MCELSA is a true extension of ELSA, that is, if deployed in with a single client, its behavior is identical to ELSA.

Finally, we experimentally demonstrate (Section 6) the performance achieved by MCELSA in scenario where over a course of 80 years documents are aggregated, stored, protected, retrieved, and verified for each of several clients simultaneously with different layouts of access permissions. Our measurements show that MCELSA outperforms ELSA (the performance analysis of which can be found in [7]) and the former state of the secure long-term storage architecture LINCOS by combining maintenance tasks to minimize computational work load and storage demand. In MCELSA, storage, retrieval, and verification of a data item takes about a second on average. Overall, our evaluation shows that MCELSA provides practical performance and is suitable for storing and protecting large and complex databases over long periods of time.

1.3 Related work

Our notion of vector commitments is reminiscent of the one proposed by Catalano and Fiore [6]. However, they do not consider the hiding property and therefore do not analyze hiding under selective opening security. Also, they do not consider extractable binding security. Hofheinz [15] studied the notion of selective decommitment and showed that schemes can be constructed that are statistically hiding under selective decommitment. However, they do not consider constructions of vector commitments where a short commitment is given for a set of messages. In [2], Bitansky et al. propose the construction of a SNARK from extractable collision-resistant hash functions in combination with Merkle trees. While their construction is similar to the extractable-binding vector commitment scheme proposed in Section 3.2, our construction relies on a weaker property (i.e., extractable-binding hash functions) and our security analysis provides concrete security estimates.

Weinert et al. [26] recently proposed a long-term integrity protection scheme that also uses hash trees to reduce the number of timestamps. However, their scheme does not support confidentiality protection, lacks a formal security analysis, and is less efficient than our construction. Only few work has been done with respect to combining long-term integrity with long-term confidentiality protection. The first storage architecture providing these two properties and most efficient to date is LINCOS [3]. Recently, another long-term secure storage architecture has been proposed by Geihs et al.[8] that provides access pattern hiding security in addition to integrity and confidentiality. On a high level, this is achieved by combining LINCOS with an information theoretically secure ORAM. While access pattern hiding security is an interesting property in certain scenarios where meta information about the stored data is known, it is achieved at the cost of additional computation and communication and it is out of the scope of this work.

2 Preliminaries

2.1 Notation

Let n be a non-negative integer. By $[n]$ we denote the set $\{0, 1, \ldots, n\}$. For a vector $V = (v_0, \ldots, v_n)$, and a set $I \subseteq [n]$, define the vector $V_I := (v_i)_{i \in I}$ as well as $V_i := v_i$ for an $i \in [n]$.

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), we define the statistical distance of A and B as $\Delta(A, B) := \sum_x |\Pr_A(x) - \Pr_B(x)|$.

2.2 Cryptographic primitives

We describe the cryptographic primitives that will be relevant for the understanding of this paper.

2.2.1 *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 ϵ-secure [11] if for any t-bounded algorithm $\mathcal{A}$ with access to a signing oracle O, we have

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(pk, m, s) = 1 \wedge m \notin Q : \\ \mathsf{Setup} \to (sk, pk), \mathcal{A}^O(pk) \to (m, s)\end{array}\right] \leq \epsilon(t),$$

where $O(m) = \{Q{+}{=}m; \mathsf{Sign}(sk, m) \to s; returns; \}$ and Q is the set of messages m that $\mathcal{A}$ has queried O upon.

2.2.2 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., based on hash functions or digital signature schemes [12]). Here, we only consider timestamp schemes based on digital signatures. This works as follows. On initialization, the 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 $\mathsf{SIG.Verify}(pk, [m, t], s) = 1$. Such a signature-based timestamp scheme is considered secure as long as the signature scheme is secure.

2.2.3 Commitment schemes

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.

A commitment scheme is considered secure if it is hiding and binding. There exist different flavors of defining binding security. Here, we are interested in extractable binding commitments as this enables renewable and long-term secure commitments [5]. A commitment scheme is ϵ-extractable-binding-secure if for any t_1-bounded algorithm $\mathcal{A}_1$, there exists an $t_{\mathcal{E}}$-bounded algorithm $\mathcal{E}$, such that for any t_2-bounded algorithm $\mathcal{A}_2$,

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(pk, m, c, d) = 1 \wedge m \neq m^* : \\ \mathsf{Setup} \rightarrow pk, \mathcal{A}_1(pk) \rightarrow_r c, \\ \mathcal{E}(pk, r) \rightarrow m^*, \mathcal{A}_2(k, r) \rightarrow (m, d)\end{array}\right] \leq \epsilon(t_1, t_{\mathcal{E}}, t_2).$$

For any public key k and message m, define $C_k(m)$ as the random variable that takes the value of c when sampling $\mathsf{Commit}(k, m) \rightarrow (c, d)$. A commitment scheme is ϵ-statistically hiding if for any $k \in \mathsf{Setup}$, any pair of messages (m_1, m_2), $\Delta(C_k(m_1), C_k(m_2)) \leq \epsilon$.

2.2.4 Keyed hash functions

A keyed hash function is a tuple of algorithms (K, H) with the following properties. K is a probabilistic algorithm that generates a key k. H is a deterministic algorithm that on input a key k and a message $x \in \{0,1\}^*$ outputs a short fixed length hash $y \in \{0,1\}^l$, for some $l \in \mathbb{N}$. We say a keyed hash function (K, F) is ϵ-extractable-binding if for any t_1-bounded algorithm $\mathcal{A}_1$, there exists a $t_{\mathcal{E}}$-bounded algorithm $\mathcal{E}$, such that for any t_2-bounded algorithm $\mathcal{A}_2$,

$$\Pr_{K \rightarrow k}\left[\begin{array}{c}H(k, x) = H(k, x^*) \wedge x \neq x^* : \\ \mathcal{A}_1(k) \rightarrow_r y, \mathcal{E}(k, r) \rightarrow x^*, \mathcal{A}_2(k, r) \rightarrow x\end{array}\right] \leq \epsilon(t_1, t_{\mathcal{E}}, t_2).$$

2.2.5 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 authorized subsets of the shareholders can reconstruct the secret, while all the other subsets of the shareholders have no information about the secret. In this work, we consider threshold secret sharing schemes [22], for which there exists a threshold parameter t (chosen by the data owner) such that any set of at least 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 [14]. 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.

For MCELSA, i.e., ELSA in a multi-client setting, we will adapt the protocols provided by the secret sharing scheme, so that they accommodate several clients simultaneously and the respective access permissions regarding the stored documents. Also, we introduce a new protocol UpdatePerm that allows clients to manage the access rights with respect to individual stored documents. We will go into further detail in Section 5.

2.2.6 Public key infrastructure

Some of the schemes, that we listed above, take a public key as a parameter for their protocols (see Sections 2.2.1, 2.2.2 and 2.2.4). In the single as well as the multiclient client case, this public key is supplied by the client who executes the protocol. While public key cryptography

itself does not provide long-term security due to increasing adversarial capabilities in terms of cryptoanalysis and computing power, it offers prolonged security. That is, by periodically performed maintenance and key updates, the parties' public keys can be considered secure. We therefore assume the existence of a public key infrastructure (PKI), which provides the client with the appropriate public key for the executed protocol. PKI will be treated as an implicit parameter for protocols in need of public keys. Thus we will omit public keys as parameters for ELSA's protocols.

3 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 4.

3.1 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 [6]. However, neither do they require their commitments to be extractable binding, nor do they consider their hiding property.

Definition 1 (Vector commitment scheme) *A vector commitment scheme is a sextuple* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$*, where* $L \in \mathbb{N}$ *is the maximum vector length,* $\mathcal{M}$ *is the message space, and* Setup, Commit, Open, *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 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.*
- $\mathsf{Open}(k, D, i) \to d$*: On input key k, vector decommitment D, and index i, this algorithm outputs a decommitment d for the ith 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 Open will always verify for the corresponding commitment and message.

Definition 2 (Correctness) *A vector commitment scheme* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *is correct, if for all* $n \in [L]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, $i \in [n]$,

$$\Pr\left[\begin{array}{c}\mathsf{Verify}(k, M_i, c, d) = 1 : \\ \mathsf{Commit}(k, M) \to (c, D), \mathsf{Open}(k, D, i) \to d\end{array}\right] = 1.$$

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 3 (Statistically hiding (under selective opening)) *Let* $S = (L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *be a vector commitment scheme. For* $n \in [L]$, $I \subseteq [n]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$*, we denote by* $\mathsf{CD}_k(M, I)$ *the random variable* $(c, \bar{D}_I)$*, where* $(c, D) \leftarrow \mathsf{Commit}(k, M)$ *and* $\bar{D} \leftarrow (\mathsf{Open}(D, i))_{i \in [n]}$*. Let* $\epsilon \in [0, 1]$*. 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* $(M_1)_I = (M_2)_I$, $k \in \mathsf{Setup}$,

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

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 4 (extractable binding) *Let* $\epsilon : \mathbb{N}^3 \to [0, 1]$*. We say a vector commitment scheme* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ *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* $\mathcal{A}_2$*,*

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

3.2 Construction: extractable binding

In the following, we show that the Merkle hash tree construction [18] can be casted into a vector commitment scheme and that this construction is extractable binding if the used hash function is extractable binding.

Construction 1 *Let* (K, H) *denote a keyed hash function and let* $L \in \mathbb{N}$*. The following is a description of the hash tree scheme by Merkle cast into the definition of vector commitments. We denote the vertices of the tree by* $h_{i,j}$*, where i is*

the level of $h_{i,j}$ within the Merkel tree and j is the index of $h_{i,j}$ in the ith level.

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

Commit$(k,(m_1,\ldots,m_n)) \to (c,D)$: *Set* $l \leftarrow \min\left\{i \in \mathbb{N} : n \le 2^i\right\}$. *For* $j \in \{0,\ldots,n-1\}$, *compute* $h_{l,j} \leftarrow H\left(k, m_{j+1}\right)$, *and for* $j \in \left\{n,\ldots,2^l-1\right\}$, *set* $h_{l,j} \leftarrow \perp$. *For* $i = l-1$ *to 0,* $j \in \left\{0,\ldots,2^i-1\right\}$, *set* $h_{i,j} \leftarrow H\left(k,\left(h_{i+1,2j}, h_{i+1,2j+1}\right)\right)$. *Finally, compute* $c \leftarrow H(k,\left(l,h_{0,0}\right))$ *and set* $D \leftarrow \left(h_{i,j}\right)_{i\in\{0,\ldots,l\},j\in\{0,\ldots,2^i-1\}}$, *and output* (c,D).

Open$(k,D,i^*) \to d$: *Let* $\left(h_{i,j}\right)_{i\in\{0,\ldots,l\},j\in\{0,\ldots,2^i-1\}} \leftarrow D$. *Set* $a_l \leftarrow i^*$. *For* $i' = l$ *to* 1, *set* $b_{i'} \leftarrow a_{i'} + 2(a_{i'} \bmod 2) - 1$, $g_{i'} \leftarrow h_{i',b_{i'}}$, *and* $a_{i'-1} \leftarrow \lfloor a_{i'}/2 \rfloor$. *Set* $d = \left(g_1,\ldots,g_l\right)$ *and output d.*

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

Theorem 1 *The vector commitment scheme described in Construction 1 is correct.*

Proof of Theorem 1. Let $(L,\mathcal{M},$ Setup, Commit, Open, Verify$)$ be the scheme described in Construction 1. Furthermore, let $n \in [L]$, $M \in \mathcal{M}^n$, $k \in$ Setup, $i \in [n]$, Commit$(k,M) \to (c,D)$, Open$(k,D,i) \to d$, and Verify$(k,M_i,c,d) \to b$. By the definition of algorithms Commit and Open, we observe that $d = \left(g_1,\ldots,g_l\right)$ are the siblings of the nodes in the hash tree $D = \left(h_{i,j}\right)_{i\in\{0,\ldots,l\},j\in\{0,\ldots,2^i-1\}}$ 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 2 *Let (K,H) be an ϵ-extractable-binding hash function. The vector commitment scheme described in Construction 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 of Theorem 2. Let (K,H) be an ϵ-extractable-binding keyed hash function and let the tuple $(L,\mathcal{M},$ Setup, Commit, Open, Verify$)$ be the vector commitment scheme described in Construction 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 $\mathcal{E}^H_{\mathcal{A}_1}(k,r) \to \left(l,h_{0,0}\right)$, 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 $\mathcal{E}^H_{\mathcal{A}_1}(k,r) \to_{r'} \left(l,h_{0,0}\right)$. Then, for $i \in \left\{0,\ldots,2^l-1\right\}$, sample r_i and compute $\mathcal{E}^H_{\mathcal{A}_{l,i}}\left(k,\left(r,r',r_i\right)\right) \to m_{i+1}$. Output $\left(m_1,\ldots,m_{2^l}\right)$.

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

3.3 Construction: extractable binding and statistically hiding

We now combine a statistically hiding and extractable binding commitment scheme with the vector commitment scheme from Construction 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 2 *Let* COM *be a commitment scheme and* VC *be a vector commitment scheme.*

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

Commit$(k,(m_1,\ldots,m_n)) \to (c,D)$: *Let* $k \to (k_1,k_2)$. *For* $i \in \{1,\ldots,n\}$, *compute* COM.Commit$(k_1,m_i) \to (c_i,d_i)$. *Then compute* VC.Commit$\left(k_2,(c_1,\ldots,c_n)\right) \to (c,D')$, *set* $D \leftarrow \left(\left((c_1,d_1),\ldots,(c_n,d_n)\right),D'\right)$, *and output* (c,D).

Open$(k,D,i) \to d$: *Let* $k \to (k_1,k_2)$ *and* $D \to \left(\left((c_1,d_1),\ldots,(c_n,d_n)\right),D'\right)$. *Compute* VC.Open$(k_2,D',i) \to d'$, *set* $d \leftarrow (c_i,d_i,d')$, *and output d.*

Verify$(k,m,c,d,i) \to b$: *Let* $k \to (k_1,k_2)$ *and* $d \to \left(c',d',d''\right)$. *Compute* COM.Verify$\left(k_1,m,c',d'\right) \to b_1$ *and then compute* VC.Verify$(k_2,c',c,d'',i) \to b_2$, *set* $b \leftarrow (b_1 \wedge b_2)$, *and output b.*

Theorem 3 *The vector commitment scheme described in Construction 2 is correct if* COM *and* VC *are correct.*

Proof of Theorem 3. Let $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$ be the scheme described in Construction 2. Let $n \in [L]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, $i \in [n]$, $\mathsf{Commit}(k, M) \to (c, D)$, $\mathsf{Open}(k, D, i) \to d$, and $\mathsf{Verify}(k, M_i, c, d) \to b$. We observe that $D = \left((c_i, d_i)_{i\in[n]}, D'\right)$ and furthermore $d \in \mathsf{VC.Open}(k_2, D', i)$. 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 4 *The vector commitment scheme described in Construction 2 is $L\epsilon$-statistically hiding (under selective opening) if the commitment scheme* COM *is ϵ-statistically hiding.*

We prove Theorem 4 in Appendix B.

Theorem 5 *If* COM *and* VC *of Construction 2 are ϵ-extractable-binding, Construction 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 of Theorem 5. 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 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 $\mathcal{E}_i\left(k, (r, r')\right) \to m_i$. 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 2 is ϵ'-extractable-binding with $\epsilon'(t_1, t_\mathcal{E}, t_2) = L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$. □

4 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 3) 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 (Fig. 1), 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.

4.1 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's security is weakened, RenewCom renews the protection if the commitment scheme's 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.

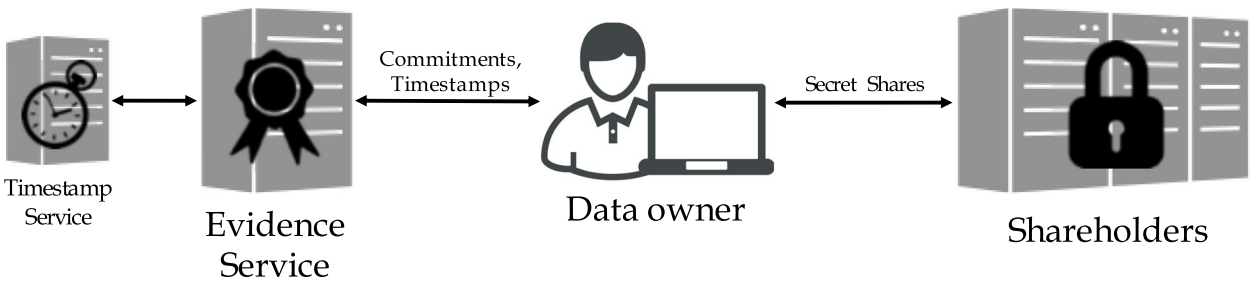


Fig. 1 Overview of the components of ELSA

We use the following notation. When we write SH.Store(name, dat) we mean that the data owner shares the data dat among the shareholders using protocol SHARE.Share associated with identifier name. If the shared data dat is larger than 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. When we write SH.Retrieve(name), we mean that the data owner retrieves the shares associated with identifier name from the shareholders and reconstructs the data using protocol SHARE.Reconstruct.

4.1.1 Initialization

The data owner uses algorithm ELSA.Init (Algorithm 1) to initialize the storage system. The algorithm gets as input a proactive secret sharing scheme SHARE, a tuple of shareholder addresses $(\text{shURL}_i)_{i\in[N]}$, a threshold t, and an evidence service address esURL. It then initializes the storage module SH by running protocol SHARE.Setup and the evidence service module ES by setting ES.`evidence` as an empty table and ES.`renewLists` as an empty list.

Algorithm 1: ELSA.Init(SHARE, $(shURL_i)_{i\in[N]}, t, esURL$)

SH.Init(SHARE, $(shURL_i)_{i\in[N]}, t$);
ES.Init($esURL$);

4.1.2 Data storage

The client uses algorithm ELSA.Store (Algorithm 2) to store data files $(\text{file}_i)_{i\in[n]}$, which works as follows. First a signature scheme SIG, a vector commitment scheme VC, and a timestamp scheme TS are chosen. 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 client first signs each of the data objects individually. The algorithm then stores the file data and the generated signature in the instance SH of the secret sharing storage system. Afterwards, a vector commitment (c, D) to the file data vector and the signatures are being generated. 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. We remark that signing each data object individually potentially allows for having a different signer for each data object. If the data objects are to be signed by the same user, an alternative is to sign the commitment instead of the data objects.

When the evidence service receives (`filenames`, VC, c, TS), it does the following in algorithm AddCom (Algorithm 3). It first timestamps the commitment (VC, c) and thereby obtains a timestamp ts. Then, it compiles the quintuple $l = (\text{VC}, c, \perp, \text{TS}, \text{ts})$ and assigns l as the first entry of the proof of integrity for each document in `filenames`. Also, it adds l to the list `renewLists`, which contains the lists that are updated on a timestamp renewal.

The third component of each entry of a proof of integrity is either a $\perp$ or it consists of a pair (d, j), where d is a decommitment value extracted from a vector decommitment and j is its index in said vector decommitment. For a given entry e of a proof of integrity, we shall denote this by either $(e.\text{decom.d}, e.\text{decom.index}) = (d, j)$ or $e.\text{decom} = \perp$. During the execution of ELSA.Store, each document's decommitment value is stored in SH, so we have $l.\text{decom} = \perp$.

Algorithm 2: ELSA.Store($(\text{file}_i)_{i\in[n]}$, SIG, VC, TS)

`filenames` $\leftarrow \{\}$;
for $i \in [n]$ **do**
  SIG.Sign(file_i.dat) $\rightarrow s_i$;
  SH.Store $(('data', \text{file}_i.\text{name}), (\text{file}_i.\text{dat}, \text{SIG}.Cert, s_i))$;
  `filenames`+=file_i.name ;
VC.Commit $\left((\text{file}_i.\text{dat}, \text{SIG}.Cert, s_i)_{i\in[n]}\right) \rightarrow (c, D)$;
for $i \in [n]$ **do**
  VC.Open$(D, i) \rightarrow d$;
  SH.Store $(('decom', \text{file}_i.\text{name}, 0), (d, i))$;
ES.AddCom(`filenames`, VC, c, TS);

Algorithm 3: ES.AddCom(`filenames`, VC, c, TS)

TS.Stamp$((\text{VC}, c)) \rightarrow ts$;
$l \leftarrow (\text{VC}, c, \perp, \text{TS}, ts)$;
for name $\in$ `filenames` **do**
  `evidence`[name] +=l;
  `renewLists`+=l;

4.1.3 Timestamp renewal

Algorithm ES.RenewTs (Algorithm 4) is performed by the evidence service regularly in order to protect against the weakening of the most recently used timestamp scheme. The algorithm gets a vector commitment scheme instance VC′ and a timestamp scheme instance TS as input. It first creates a vector commitment (c', D') for the list

of renewal items `renewLists`. Here, we only require the extractable-binding property of VC′, while the hiding property is not required as all of the data stored at the evidence service is independent of the secret data due to the use of unconditionally hiding commitments by the data owner. For each updated list item i, the freshly generated timestamp, commitment, and extracted decommitment are added to the corresponding evidence list `renewLists`$[i]$.

Algorithm 4: ES.RenewTs $(\mathsf{VC}', \mathsf{TS})$

$\mathsf{VC}'.\mathsf{Commit}(\texttt{renewLists}) \to (c', D')$;
$\mathsf{TS}.\mathsf{Stamp}\left((\mathsf{VC}', c')\right) \to ts$;
for $i \in [|\texttt{renewLists}|]$ **do**
  $\mathsf{VC}'.\mathsf{Open}(D', i) \to d'$;
  $\texttt{renewLists}[i] \mathrel{+}= (\mathsf{VC}', c', (d', i), \mathsf{TS}, ts)$;

4.1.4 Commitment renewal

The data owner runs algorithm ELSA.RenewCom (Algorithm 5) to protect against a weakening of the currently used commitment scheme. It chooses a new commitment scheme instance VC and a new timestamp scheme instance TS and proceeds as follows. First, the table of evidence lists ES.`evidence` is 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 reusing algorithm ES.AddCom.

4.1.5 Secret share renewal

There are two types of share renewal supported by ELSA. The first type (Algorithm 6) 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 (Algorithm 7) replaces the proactive sharing scheme entirely. This may be necessary if the scheme has additional security properties like verifiability (see proactive verifiable secret sharing [14]), whose security may be weakened. In this case, the data is retrieved, shared to the new shareholders, and finally, the old shareholders are shutdown.

Algorithm 5: ELSA.RenewCom(VC, TS)

`filenames` $\leftarrow \{\}$; `comIndices` $\leftarrow \{\}$;
`comCount` $\leftarrow \{\}$; $L \leftarrow [\,]$;
for name $\in$ ES.`evidence` **do**
  $(\mathsf{dat}, \mathsf{SIG}.Cert, s) \leftarrow \mathsf{SH.Retrieve}\left(('data', \mathsf{name})\right)$;
  ES.`evidence`[name] $\to e$;
  for $i \in [|e|]$ **do**
    if $e_i.\mathsf{decom} = \perp$ **then**
      $(e_i.\mathsf{decom.d}, e_i.\mathsf{decom.index}) \leftarrow$
      $\mathsf{SH.Retrieve}\left(('decom', \mathsf{name}, i)\right)$;
  $L \mathrel{+}= (\mathsf{dat}, \mathsf{SIG}.Cert, s, e)$;
  `comIndices`[name] $\leftarrow |L|$;
  `comCount`[name] $\leftarrow |e|$;
  `filenames`+=name;
VC.Commit$(L) \to (c, D)$;
for name $\in$ ES.`evidence` **do**
  VC.Open$(D,$ `comIndices`[name]$) \to d$;
  $\mathsf{SH.Share}\left(('decom', \mathsf{name}, \texttt{comCount}[\mathsf{name}]\right),$
  $(d,$ `comIndices`[name]$))$;
ES.AddCom(`filenames`, VC, c, TS);

Algorithm 6: ELSA.RenewShares()

SH.Reshare();

Algorithm 7: ELSA.RenewSharing$(\mathsf{SHARE}, (shURL_i)_{i\in[N]}, T)$

$\mathsf{SH}'.\mathsf{Init}(\mathsf{SHARE}, (shURL_i)_{i\in[N]}, T)$;
$I \leftarrow$ ES.`itemInfos`;
for $name \in I$ **do**
  $\mathsf{SH.Retrieve}\left('data', name\right) \to \mathsf{dat}$;
  $\mathsf{SH}'.\mathsf{Share}\left('data', name, \mathsf{dat}\right)$;
SH.Shutdown();
SH $\leftarrow$ SH′;

4.1.6 Data retrieval

The algorithm ELSA.Retrieve (Algorithm 8) 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.

4.1.7 Verification

Algorithm ELSA.Verify (Algorithm 9) describes how a verifier can check the integrity of a data item using the evidence produced by ELSA. Here, we denote by time(ts)

Algorithm 8: ELSA.Retrieve(name)

$e \leftarrow$ ES.evidence[name];
for $i \in [\,|e|\,]$ **do**
  if $e_i.decom = \perp$ **then**
    $(e_i.\text{decom}.d, e_i.\text{decom.index}) \leftarrow$
    SH.Retrieve $\left(('decom', \text{name}, i)\right)$;
SH.Retrieve $\left(('data', \text{name})\right) \rightarrow (\text{dat}, \text{SIG}.Cert, s)$;
$E \leftarrow (\text{SIG}.Cert, s, e)$;
return (dat, E);

Algorithm 9: ELSA.Verify (name, t_{store})

$e \leftarrow$ ES.evidence[name];
/* Get evidence record for name. */
$e' \leftarrow e$;
/* Make a copy of the proof of integrity, which will be completed below. */
for $i = 0, \ldots, |e'|$ **do**
  if $e'_i.decom = \perp$ **then**
    $e'_i.decom \leftarrow (d, j) =$
    SH.Retrieve (('decom', name, i));
$(dat, \text{SIG}.Cert, s) \leftarrow$ SH.Retrieve (('data', name));
$\left(\text{VC}, c, (d, j), \text{TS}, ts\right) \leftarrow e'_0$;
/* Check the first entry of the proof of integrity. */
$t \leftarrow t_{store}$;
$b \leftarrow$ SIG.Verify $(\text{SIG}.Cert, dat, s) \wedge$
VC.Verify $\left(\text{PKI}, dat, c, d, j\right) \wedge$
TS.Verify $(\text{PKI}, ((\text{VC}, c), t), ts)$;
for $i = 1$ *to* $|e|$ **do**
  $\left(\text{VC}, c, (d, j), \text{TS}, ts\right) \leftarrow e'_i$;
  $t = \text{time}(ts)$;
  if $e_i.\text{decom} = \perp$ **then**
    /* In this case, the i-th entry came from a commitment renewal. */
    $b \wedge =$
    VC.Verify $\left(\text{PKI}, \left(dat, \text{SIG}, s, e'_{[i-1]}\right), c, d, j\right) \wedge$
    TS.Verify $(\text{PKI}, ((\text{VC}, c), t), ts)$;
  else
    /* Otherwise, the i-th entry of e came from a timestamp renewal. */
    $b \wedge =$ VC.Verify $\left(\text{PKI}, e_{[i-1]}, c, d, j\right) \wedge$
    TS.Verify $(\text{PKI}, ((\text{VC}, c), t), ts)$;
return b;

the point in time, when a given timestamp ts was generated, i.e. TS.Verify $(\text{PKI}, (m, \text{time}(ts)), ts) = 1$ if $ts \leftarrow$ TS.Stamp(m). ELSA.Verify gets name, the identifier of the document in question and a point in time t_{store} as input. The algorithm returns 1, if dat, i.e. name's data, is authentic and has been stored at t_{store}.

In more detail, the verification algorithm works as follows. It first checks the initial element of name's proof of integrity, that is whether the signature s is valid for the data object dat under signature scheme instance SIG at the time of the first timestamp of the evidence list e. It also checks whether the corresponding commitment is valid for $(\text{dat}, \text{SIG}, s)$ at the time of the next commitment and the timestamp is valid at the next timestamp. Then, for each of the remaining $|e|$ entries of e, the algorithm checks whether the corresponding timestamp is valid at the time of the next timestamp and whether the corresponding commitments are valid at the time of the next commitments. The algorithm outputs 1 if all checks return valid, and it outputs 0 in any other case.

4.2 Security analysis

4.2.1 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 from [5].

A real-time bounded long-lived adversary $\mathcal{A}$ is defined as a sequence $\left(\mathcal{A}_{(0)}, \mathcal{A}_{(1)}, \mathcal{A}_{(2)}, \ldots\right)$ of machines with $\mathcal{A}_{(t)} \in \mathcal{M}_t$ and is associated with a global clock Clock. 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 $t < t'$. The adversary is given the power to advance the clock, but it cannot go backwards in time. When $\mathcal{A}^{\text{Clock}}$ is started, then actually the component $\mathcal{A}_{(0)}$ is run. Whenever a component $\mathcal{A}_{(t)}$ calls the clock oracle to set a new time t', then component $\mathcal{A}_{(t)}$ is stopped and the component $\mathcal{A}_{(t')}$ is run with input the internal state of $\mathcal{A}_{(t)}$. Real-time computational bounds are expressed as follows.

Let $\mathcal{A}^{\text{Clock}}$ be a computing machine associated with Clock, and let $\rho : \mathbb{N} \rightarrow \mathbb{N}$ be a function. 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 $\rho(t)$.

4.2.2 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 (Algorithm 10) 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 5 (Integrity) *We say* ELSA *is* $(\mathcal{M}, \epsilon)$*-unforgeable for schemes* $\mathcal{S}$*, if for any p-bounded machine* $\mathcal{A} \in \mathcal{M}$*,*

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

The experiment $\mathsf{Exp}^{\mathsf{Forge}}$ *is described in Algorithm 10.*

Algorithm 10: $\mathsf{Exp}_{\mathcal{S}}^{\mathsf{Forge}}(\mathcal{A})$

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

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

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

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

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

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

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

Next, we prove that the 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 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* $\epsilon(p) = \left(\sum_{i \in \mathsf{SIG}} \epsilon_i\left(p\left(t_i^b\right) p_E\left(t_i^b\right) L^2\right)\right) + \left(\sum_{i \in \mathsf{COM}} \epsilon_i\left(p\left(t_i^b\right), p_E\left(t_i^b\right), p\left(t_i^b\right)\right)\right)$.

The proof of Theorem 6 can be found in Appendix B.

4.2.3 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 11), where we use the following notation. For a secret sharing scheme SHARE, we denote by $\mathsf{SHARE}.AS$ 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 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* $L \in \mathbb{N}$*,*

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

The experiment $\mathsf{Exp}^{\mathsf{DIST}}$ *is described in Algorithm 11.*

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

Algorithm 11: $\mathsf{Exp}^{\mathsf{Dist}}_{\mathcal{S},L}(\mathcal{A}, Files, I)$

SetupExperiment();
$\mathcal{A}^{\mathsf{Clock},\mathsf{ELSA}'} \rightarrow b$;
return b;

SetupExperiment():
time $\leftarrow 0$; $N \leftarrow 0$;
Generate parameters for $\mathcal{S}$;
ELSA.Init($\mathcal{S}$.SHARE, SH, ES);

Clock(t):
if $t >$ time **then**
  time $\leftarrow t$;

ELSA$'(op, param)$:
if $N < L$ **then**
  N+=1; $\mathcal{T} \leftarrow \mathcal{T}'$;
  if $op =$ Store $\wedge$ $param \notin \mathcal{S}$.SHARE.AS **then**
    $\langle$ELSA.Store($Files, param$)$\rangle_{\mathsf{View}}$; $\mathcal{T} \leftarrow param$;
  else if $op =$ Retrieve $\wedge$ $param \in I$ **then**
    $\langle$ELSA.Retrieve($param$)$\rangle_{\mathsf{View}}$;
  else if $op = ReTs$ **then**
    $\langle$ELSA.RenewTs($param$)$\rangle_{\mathsf{View}}$;
  else if $op = ReCom$ **then**
    $\langle$ELSA.RenewCom($param$)$\rangle_{\mathsf{View}}$;
  else if $op = ReShare \wedge param \notin \mathcal{S}$.SHARE.$AS$ **then**
    $\langle$ELSA.RenewShares()$\rangle_{\mathsf{View}}$; $\mathcal{T}' \leftarrow param$;
  return View(ES, $\mathsf{SH}_{\mathcal{T}}$, *Receiver*);

Theorem 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 of Theorem 7. 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 regard 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 [10]), so that a network provider, who could potentially intercept all of the secret share packages, cannot learn any information about the communicated data. Implementations of QKD have, however, not yet reached a state in which they may be deemed a viable method to connect a larger number of parties. Thus, manual key exchange may be considered as a temporary solution. If information theoretically secure channels are not available, we recommend to use very strong symmetric encryption (e.g., AES-256 [19]).

5 MCELSA: ELSA for multiple clients

In Section 4, we proposed ELSA, a long-term storage solution, that provides information theoretical confidentiality and continuously prolonged computational integrity for the documents stored within it. In this section, we will adapt ELSA so that several clients can interact with the same instance simultaneously. We denote this multi-client version by MCELSA. MCELSA delivers the same guarantees as ELSA with respect to the stored documents' integrity and confidentiality.

Compared to the single-client setting, the multi-client setting introduces new challenges concerning access management and the distribution of maintenance tasks. In this section, we go into further detail about the nature of these challenges and propose solutions for them:

- In any multi-client storage system, access management breaks down to hindering unauthorized parties (clients, shareholders, or the evidence service) from obtaining a document while enabling access for authorized parties. An intuitive solution is to employ an external party that decides for each client's queries whether to grant access or deny it. That party itself may not have access to stored documents. This solution contradicts MCELSA's aim to provide robust confidentiality by storing documents in a secret sharing scheme and thereby eliminating central points of failure. More precisely, an attacker who corrupted the access deciding party would gain unlimited document access. Hence, we dismiss this approach and instead shift the access management to the shareholders. We go into more detail in Section 5.1.
- ELSA's maintenance in essence consists of timestamp, commitment, and share renewal (see Algorithm 4, 5, and 6). We will amend ELSA's protocols in Section 5.2 to the multi-client setting, so that they distribute these tasks among the clients, shareholders and evidence service in order to minimize the combined workload while respecting access privileges.

5.1 Modifications to the secret sharing schemes

Before going into detail regarding the modifications to ELSA's protocols, we first discuss the amendments to the

underlying schemes. While we leave the signature, vector commitment and timestamp schemes unchanged in comparison to the single-client setting, we do need to apply some modifications to the employed secret sharing scheme in order for it to support the aforementioned access management.

We elaborate the modifications in terms of the protocols, which the secret sharing scheme provides. Access management will be realized by having each individual shareholder decide upon receiving a query from a client c, whether c has the necessary permissions for the query he issued. Each shareholder then acts accordingly independent of the other shareholders. Below, we list the modified protocols with an appended asterisk and compare them to SH's original protocols. The updated secret sharing scheme will be denoted by SH*.

- Setup* establishes the secret sharing parameters as SH.Setup does, too. Yet, it additionally fixes the layout, in which the permissions for each document are to be denoted. Subsequent data access queries must adhere to these parameters.
- Store* takes a tuple of documents $(\mathsf{file}_i)_{i\in[n]}$ and a list of permissions $(p_i)_{i\in[n]}$ as an additional argument. It generates shares of the documents and sends the shares to the shareholders for storage, yet it attaches the permissions to each share.
- Retrieve* takes a document identifier name as input. Compared to the original Retrieve, it introduces a further step. Assume a client c issued Retrieve*(name). Upon receiving this request, each shareholder consults the permissions attached to his share of name. He then proceeds to transmit his share to c, if c is authorized to retrieve name. Otherwise, c's query is ignored. Also, if a shareholder hands over his shares, he sends the permissions alongside it. Thus, a client has to convince an authorized set of shareholders to send their shares in order to obtain it and if he does, he also learns the respective permissions.
- Reshare* has the shareholders derive new, uncorrelated shares of the stored documents, that is, it behaves as SH.Reshare does, yet we point out that the attached permissions for each share must remain unchanged.
- UpdatePerm*: We extend the secret sharing scheme by this protocol to enable the clients to change the permissions with respect to a document without having to retrieve and share it anew. Thus, if a client issues UpdatePerm* with a document list identifier and a permissions list as arguments, the shareholder check whether that client is authorized for his query and, if so, they swap the old permissions attached to their shares for the ones provided by the client. Thereby, the permissions are updated without reassembling the documents.

With the modifications above, we shift the task of access management to the shareholders, which avoids a central point of failure and leaves ELSA's general setup unchanged apart from introducing further clients. Furthermore, we hereby achieve the same confidentiality guarantees as with ELSA, that is, if a party wants to obtain a document, to which it is not permitted access, it has to corrupt an authorized set of shareholders, i.e., break the secret sharing scheme SH.

We also let the shareholders maintain an initially empty list of document identifiers for each client. This list will be denoted by l_u, where u goes over all clients. The purpose of the list will be explained in Section 5.2.2.

5.2 Modifications to ELSA's protocols

In Section 5.1, we amended the secret sharing scheme in which the documents and their respective decommitment values are stored so that it supports multiple clients and access management. This of course leads to changed interfaces for SH's protocols, which implies some amendments to ELSA's protocols, too. We split the protocols into three categories: data access, maintenance, and integrity verification.

5.2.1 Data access

The protocols for data access, i.e., storing and retrieving documents, in ELSA are Store (Algorithm 2) and Retrieve (Algorithm 8). We briefly describe their updated workflow and point out the applied changes.

Data storage. A client in MCELSA stores documents $(\mathsf{file}_i)_{i\in[n]}$ by executing MCELSA.Store (Algorithm 14) with parameters $(\mathsf{file}_i)_{i\in[n]}, \mathsf{SIG}, \mathsf{VC}, \mathsf{TS}$, and $(p_i)_{i\in[n]}$, where (file_i), SIG, VC, and TS agree with those of ELSA.Store and p_i denotes the permissions with respect to the document file_i, $i \in [n]$. He executes MCELSA.Store as he would execute ELSA.Store, yet when storing the documents and decommitment values via SH*.Share*, he passes $(p_i)_{i\in[n]}$ as the additional argument for the access permissions. Thereby, the client establishes each parties' access rights to the documents $(\mathsf{file}_i)_{i\in[n]}$ (and the respective decommitment values).

Data retrieval. To obtain a document identified by name, a client issues MCELSA.Retrieve(name). This implies executing SH.Retrieve(name) with respect to the document and its decommitment values from the shareholders and retrieving the remaining proof of integrity from the evidence service. The modification compared to ELSA.Retrieve is that the client only obtains the document if he is authorized and, if so, he also obtains the permissions concerning name due to the changes in the secret sharing scheme SH*.

Permission modification. Changing the permissions on one or more documents identified by $(\mathsf{name}_i)_{i\in[n]}$ can be achieved by retrieving said documents via MCELSA.Retrieve $\left((\mathsf{name}_i)_{i\in[n]}\right)$ and subsequently storing them via MCELSA.Store $\left((\mathsf{name}_i)_{i\in[n]}, (p'_i)_{i\in[n]}\right)$ with $[p'_i]_{i\in[n]}$ being the updated permissions. The drawback of this approach is that the documents are reconstructed at the client's site. Thus, we introduce the protocol MCELSA.UpdatePerm. The parameters for this protocol are $(\mathsf{name}_i)_{i\in[n]}$ and $(p'_i)_{i\in[n]}$. The protocol executes SH*.UpdatePerm* (name_i, p'_i) for each $i \in [n]$; if the client is authorized, the shareholders then swap name_i's permissions for p'_i.

5.2.2 Maintenance

Adapting the protocols for data access to the multi-client scenario was fairly straightforward, yet transferring the maintenance tasks is somewhat more involved. As we mentioned before, there are three procedures to be discussed: timestamp, share, and commitment renewal.

Timestamp renewal. The amendments to this protocol are purely for optimization purposes: whereas in ELSA, the evidence service ES renewed the timestamps on all proofs of integrity when executing ES.RenewTs (Algorithm 4), we take a more refined approach in MCELSA to reduce memory consumption. MCELSA.RenewTs (Algorithm 15) works as follows: ES updates all expiring timestamps by generating a vector commitment (c, D) on `renewLists` and obtains a timestamp ts' on (c, D) from a given timestamp scheme TS. As a last step, ES appends the quintuple $(\mathsf{VC}', c', (d', i), \mathsf{TS}, ts)$ to each proof of integrity `evidence`[name] whose last used timestamp scheme coincides with that of $\texttt{renewLists}_i$, that is, it satisfies $\texttt{evidence}[\mathsf{name}]_{|\texttt{evidence}[\mathsf{name}]|}.\mathsf{TS} = \texttt{renewLists}_i.\mathsf{TS}$.

Share renewal. ELSA's protocol for renewing the shares of stored documents stays unchanged apart from executing SH*.Reshare* instead of SH.Reshare. Thereby new, uncorrelated shares of the documents replace the old shares while keeping the access permissions.

Commitment renewal. Renewing the commitments is the most involved task to transfer to the multi-client setting. In ELSA, the client first collects all stored documents as well as their complete proofs of integrity in a list L. He then generates a vector commitment (c, D) on said list and has the evidence service append the commitment along with a freshly generated timestamp to the existing proofs of integrity via ES.AddCom. The newly generated decommitment values are then stored within the secret sharing scheme.

The obvious complication with ELSA's approach in the multi-client setting is that a client, who is authorized to retrieve all stored documents, does not even have to exist or have to be online! Two solutions would seem to offer themselves if we wished to uphold ELSA's solution of having a single party recommit to the stored documents: either an external trusted party gains access to all documents and recommits to them or an involved party (for example, the evidence service or a client) gains access to all documents and recommits to them. Both approaches strongly contradict MCELSA's confidentiality aim, i.e., only authorized parties gain access to a document.

We solve this conflict by employing—if necessary—more than one client to assist in commitment renewal. Also, in order to keep storage demand of MCELSA as low as possible, we have clients recommit only to documents that are affected by an expiring commitment instead of all documents. Thus, commitment renewal is split into three steps:

1. Assume that a vector commitment (c, D) is expiring. The evidence service ES, having knowledge of all commitment values in ELSA, executes MCELSA.*detRecom* (Algorithm 12) and compiles a list L of documents that are affected by (c, D)'s expiry. ES then hands L to the shareholders.
2. The shareholders have knowledge of the access permissions with respect to the stored documents. Thus, they will assign the documents in L to clients in such a manner that as few as possible clients become involved in the commitment renewal procedure while respecting the access permissions. This minimizes the number of commitments to maintain. The lists l_u, that the shareholders maintain, now come into action. For each client u, l_u contains the identifiers of the documents that had been assigned to u for commitment renewal. Once u has generated a new vector commitment for his list l_u, it is emptied. Yet, in a long-term storage solution such as MCELSA, a further commitment may expire while a list l_u still contains a document. Thus, upon receiving L from the evidence service, the shareholders merge L with the lists l_u and distribute the resulting list among the clients. Finally, each client u's newly assigned documents are denoted in l_u. For an exact description, how to assign the extended list L to the clients, see MCELSA.*distrRecom* (Algorithm 13). For MCELSA.*detRecom*, we have to introduce the following notation: by $\texttt{perm}_u$, we denote the set of documents that a client u is permitted to retrieve; also, the set of all clients is denoted by $\mathcal{U}$. The cardinality of a set X will be denoted by $\#X$.
3. Whenever a client u connects to MCELSA, he is handed l_u and executes MCELSA.*userRenewCom* (Algorithm 16). For that, he retrieves all documents in l_u (by design he is permitted access to each $\mathsf{name} \in l_u$) and their respective complete proof of

integrity (this includes the data stored at ES and the decommitments that were deposited in the secret sharing scheme). Afterwards, u commits to the gathered documents and proofs of integrity. The resulting decommitment values are then stored in SH appropriately, whereas the commitment value is handed to ES via ES.AddCom $(l_u, \mathsf{VC}, c, \mathsf{TS})$ to be appended to the proofs of integrity associated with the documents in l_u. In comparison to ELSA.RenewCom, the client that executes MCELSA.*userRenewCom* recommits to all documents in l_u instead of all stored documents.

Algorithm 12: MCELSA.*detRecom* (c)

```
recomList ← {};
for name ∈ ES.evidence do
    e ← evidence[name];
    i = |e|;
    repeat
        if e_i.c = c ∧ e_i.d = ⊥ then
            recomList+=name;
    until i = 0 ∨ e_i.d = ⊥;
return recomList;
```

Algorithm 13: MCELSA.*detRecom* (recomList)

```
recomList ← recomList ∪ ⋃_{u∈U} l_u;
for u ∈ U do
    l_u ← {};
The shareholders determine u_0 ∈ U with
#(perm_{u_0} ∩ recomList) = max_{u∈U} #(perm_u ∩ recomList)
l_{u_0} ← perm_{u_0} ∩ recomList;
n ← 1;
while
recomList \ ⋃_{i=0}^{n-1} l_{u_i} ≠ ∅ ∧ {u_0, ... u_{n-1}} ≠ U do
    The shareholders determine u_n ∈ U \ {u_i}_{i=0}^{n-1} with
    # ( perm_{u_n} ∩ recomList \ ⋃_{i=0}^{n-1} l_{u_i} ) = max_{u∈U\{u_i}_{i=0}^{n-1}}
    # ( perm_u ∩ recomList \ ⋃_{i=0}^{n-1} l_{u_i} )
    l_{u_n} ← perm_{u_n} ∩ recomList \ ⋃_{i=0}^{n-1} l_{u_i};
    n ← n + 1;
```

For Algorithms 13 and 16 to work correctly, we make a few basic assumptions:

- The access permissions, that are attached to the files in L at the shareholders site, coincide. If this were not the case, then constellations can be constructed quite easily in which two shareholders arrive at two distinct document distributions after MCELSA.*detRecom*'s execution.
- The clients connect to MCELSA within sufficiently short timespans to recommit to the documents in l_u in a timely manner.
- For each document stored in MCELSA, there exists a client, who has permission to retrieve it.

We now prove, that by executing MCELSA.*detRecom* the shareholders correctly distribute the list of documents, they receive from the evidence service, among the clients. That is, each document is assigned to a client.

Proposition 1 *Let* recomList *be a set of documents and U be a set of clients, so that, for each document* name $\in$ recomList $\cup \bigcup_{u\in U} l_u$*, there is client $u \in U$ who has permission to reconstruct* name*. After Algorithm 13's execution, we have*

$$\texttt{recomList} \cup \bigcup_{u\in U} l'_u \subseteq \bigcup_{u\in U} l_u,$$

where l'_u denotes l_u before Algorithm 13's execution.

Proof Assume the contrary, that is

$$\texttt{recomList} \cup \bigcup_{u\in U} l'_u \supsetneq \bigcup_{u\in U} l_u$$

after Algorithm 13 has terminated. Let U' be the set of all clients, that have been chosen during the algorithm's execution. Then, there is a document name $\in$ recomList $\cup \bigcup_{u\in U} l'_u \setminus \bigcup_{u\in U'} l_u$ with name $\notin l_u$ for all $u \in U'$, in particular, we have name $\notin l_{u_0}$. Let U_{name} be the set of clients $u \in U$ that have permission to reconstruct name. Now, we have to distinguish two cases:

- $U_{\mathsf{name}} \subseteq U'$: Let u^* be the first client in U_{name} selected by Algorithm 13; w.l.o.g. $u^* = u_0$. We have name $\in l_{u^*} \cap$ recomList, hence name $\in l_{u_0} = l_{u^*}$, a contradiction.
- $U_{\mathsf{name}} \not\subseteq U'$: Now, let $u^* \in U_{\mathsf{name}} \setminus U'$ and $n^* :=\ = |U'|$. Thereby, we have name $\in$ recomList $\setminus \bigcup_{i=0}^{n^*} l_{u_i} \neq \emptyset$ and $u^* \in U \setminus U' \neq \emptyset$; hence, the while-loop has not yet terminated and therefore neither has Algorithm 13.

In conclusion, we obtain name $\in \bigcup_{u\in U'} l_u$, a contradiction. □

We do not list all of MCELSA's algorithms here; an overview of those we skipped can be found in A Algorithms.

5.2.3 *Integrity verification*

In Section 5.2, we modified ELSA's maintenance protocols in order to support multiple clients. Ye,t this has purposefully not changed the make of the proofs of integrity for the stored documents. Thus, we leave the protocol ELSA.Verify (Algorithm 9) unmodified. We should however point out, that—due to access permission limitations—only the integrity of documents that a client u is authorized to retrieve can be verified by u, since he has to reconstruct the document as well as the according decommitment values during ELSA.Verify's execution.

6 Performance evaluation

In our performance evaluation, we omit ELSA, since it is just a special case of MCELSA with a single client. We evaluate the performance of our new architecture MCELSA in terms of resource demand for the evidence service and the secret sharing instance. We will also measure the time elapsed for integrity verification of individually stored documents.

6.1 Testing parameters

We simulate a continued runtime of 80 years for MCELSA, i.e., approximately one human lifetime. During that period, we let each client store one document each month for the whole duration of the experiment.

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 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.

As the vector commitment scheme, we use Construction 2 with the statistically hiding commitment scheme by Halevi and Micali [13] whose security is based on the security of the used hash function which we instantiate with members of the SHA-2 hash function family [20]. If we model hash functions as random oracles, the extractable-binding property required by Theorem 6 is provided. This vector commitment scheme construction also provides statistical hiding security as required by Theorem 7. We adjust the signature and commitment scheme parameters over time as proposed by Lenstra and Verheul [16, 17].

Concerning the document storage, i.e., the secret sharing scheme, we instantiate it as a standard Shamir secret sharing [22] with four shareholders and a reconstruction threshold of three, that is, any set of three or more clients can reconstruct a stored document.

Now, since the layout of the access permissions strongly influences the resulting maintenance task distribution, we distinguish the three following cases.

> Single client: We form a baseline for our performance evaluation by testing MCELSA with a single client. We compare these results with those of LINCOS, up until now the fastest storage architecture that provides long-term integrity and confidentiality.
> Isolated clients: Ten clients are being simulated for this test, where each client is permitted to access only his own documents. The provided functionality is that of running ten parallel instances of ELSA, yet in using MCELSA, only one instance of SH and one instance of ES is needed.
> Privileged clients: In this case, we simulate a small doctor's office. Again, we consider ten clients, eight of which represent patients and two of which doctors. Each patient may of course only access his own health record, whereas the doctors can access those of all their patients.

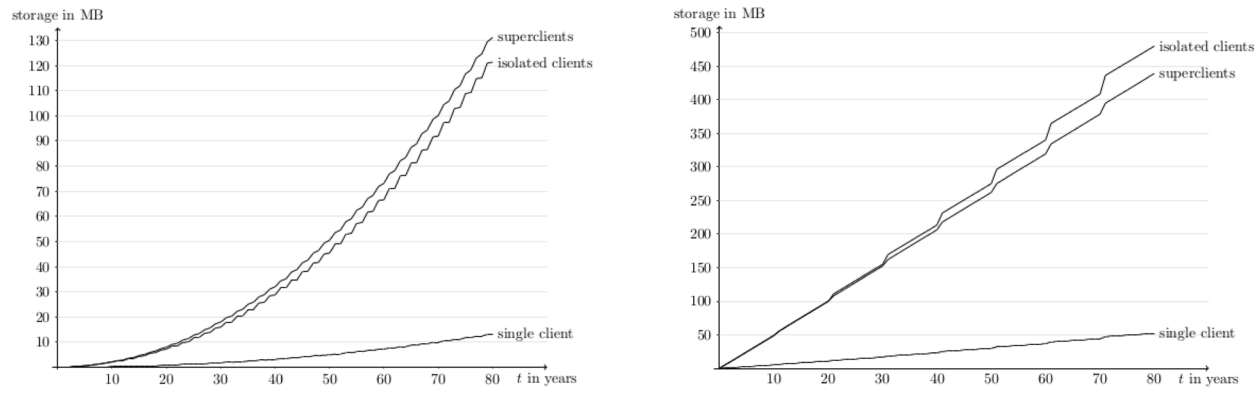
After the 80th year has passed, the client(s) verify the integrity of the stored documents and we measure the time that is needed for the integrity verification.

Our implementation was done using the programming language Java. In order to have consistent environment parameters, the experiments were performed in a virtual Linux machine with a two-core processor running at 2.8 GHz, 16GB of storage space, and 8GB of RAM.

6.2 Results

We now present the results of our performance analysis. Figure 2a shows the storage demand at the evidence service's site over the course of eighty years. The storage need at the shareholders is plotted in Fig.2b. In Fig.3, we summarize our evaluation results. We should remark that the longer a document has been stored in MCELSA, the more entries its proof of integrity has, thus the longer the integrity verification takes. More precisely, assuming that verifying a single entry of a proof of integrity takes constant time, the time needed for integrity verification goes linearly with the time, for which the document had been stored.

We observe that increasing the number of client scales reasonably well in MCELSA, more than that, MCELSA performs about 5% better than ELSA; this can be seen by comparing with ELSA's performance results in [7]. We attribute this increased performance to the optimizations,



(a) Storage need at the evidence service

(b) Storage need at the secret sharing service

Fig. 2 Plots of MCELSA's evaluation results concerning storage demand

that we applied to the maintenance protocols, and to some improvements in the simulation implementation.

In conclusion, we see that the storage demand in the scenario of several isolated clients is approximately the same as that of a single client multiplied with the number of engaged isolated clients.

We furthermore observe that we drastically decreased the used resources by employing two clients, which were able to retrieve all documents in the super client scenario.

7 Conclusions

We have proposed MCELSA as an extension of ELSA. MCELSA is a long-term secure storage architecture that provides efficient protection for large datasets and supports multiple clients simultaneously while upholding the same security guaranties that ELSA provides with respect to integrity and confidentiality of the stored data.

The multi-client setting raised the issues of access management and distribution of maintenance tasks. We achieved robust distributed access management by entrusting the shareholders with determining individual access privileges. We also proposed a set of algorithms to distribute the entailed maintenance tasks among the engaged parties with the aim of minimizing MCELSA's resource demand.

ELSA improved the performance of existing state of the art long-term storage solutions. MCELSA in turn achieves further performance improvements by optimizing ELSA's protocols and implementation. This became evident, when testing MCELSA's performance in terms of time elapsed for document verification, commitment renewal and storage demand. We observed that MCELSA outperforms ELSA regarding resources demand per client.

We achieve nearly constant computation and storage costs for timestamp renewal by using a new type of vector commitment, while for existing solutions, these costs scale linearly with the number of stored items. Our performance measurements show that MCELSA provides practical performance for application scenarios, where the stored data has to be maintained of extended periods of time.

We envision the following research directions for future work. It would be interesting to investigate how to combine our vector commitments with ORAM technology in order to obtain a storage architecture that additionally provides long-term access pattern hiding security. Another interesting direction is to combine secure multiparty computation protocols with our storage architecture in order to allow for computation on the stored and protected data. Also, it would be interesting to see whether commitment renewal without the help of the data owner is possible.

Scenario	Storage SH	Storage ES	Verificaton time per file
Single Client	53.26 MiB	13.115 MiB	1.41s
Isolated Clients	670.44 MiB	121.305 MiB	1.18s
Super Clients	439.30 MiB	131.198 MiB	0.94s

Fig. 3 Evaluation of the three tested scenarios

Appendix

A Algorithms

In this section we list the algorithms that we omitted in Section 5.

Algorithm 14: MCELSA.Store $\left(\left(\text{file}_i\right)_{i\in[n]}, \left(p_i\right)_{i\in[n]}, \text{SIG}, \text{VC}, \text{TS}\right)$

```
filenames ← {};
for i ∈[n] do
    s_i ← SIG.Sign (file_i.dat) s_i;
    SH.Store* (('data', file_i.name), (file_i.dat, SIG, s_i), p_i);
    filenames+=file_i.name;
(c, D) ← VC.Commit ((file_i.dat, SIG, s_i)_{i∈[n]});
for i ∈[n] do
    d_i ← VC.Open (D, i);
    SH.Store* (('decom', file_i.name), (d_i, i));
ES.AddCom (filenames, VC, c, ts);
```

Algorithm 15: MCELSA.RenewTs(VC′, TS)

```
(c′, D′) ← VC′.Commit (renewLists);
ts ← TS.Stamp ((VC′, c′));
renewLists ←[];
for i ∈ [|renewLists|] do
    d′ ← VC.Open (D′, i);
    for name ∈ evidence do
        if evidence [name]_{|evidence[name]|}.TS =
        renewLists_i.TS then
            evidence[name] += (VC′, c′, (d′, i), TS, ts);
renewLists+= (VC′, c′, ⊥, TS, ts);
```

Algorithm 16: MCELSA.*userRenewCom* $(l_u, \text{VC}, \text{TS})$

```
DocIndices ← {};
comCount ← {};
L ←[];
for name ∈ l_u do
    (dat, SIG.Cert, s, p) ←
    SH.Retrieve* ('data', name);
    e ← ES.evidence[name];
    for i ∈ [|e|] do
        if e_i.d = ⊥ then
            e_i.d ← SH.Retrieve* ('decom', name, i);
    L+ = (dat, SIG.Cert, s, e);
    DocIndices[name] ← |L|;
    comCount[name] ← |e|;
VC.Commit(L) → (c, D);/* L contains all
    documents and PoI 's.                    */
for name ∈ l_u do
    d ← VC.Open (D, DocIndices[name]);
    SH.Store* (('decom', name, comIndices[name]),
    (d, DocIndex[name]));
ES.AddCom (l_u, VC, c, TS);
```

B Proofs

Proof of Theorem 4. Let $(L, \mathcal{M}, \text{Setup}, \text{Commit}, \text{Open}, \text{Verify})$ be the scheme described in Construction 2. For any $n \in[L]$, $I \subseteq[n]$, $M \in \mathcal{M}^n$, $k \in$ Setup, and event $A \subseteq \Omega(\text{CD}_k(M, I))$, denote by $\Pr_M(A)$ the probability that A is observed when sampling from $\text{CD}_k(M, I)$.

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

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

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

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

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

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

Next, we observe from the description of algorithm Commit that each call $\text{Com}(M_i) \to (c_i, d_i)$ is independent

of the other calls $\mathsf{Com}(M_j) \to (c_j, d_j), j \neq i$. Thus,

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

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

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

It follows that

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

We observe that because Com is ϵ-statistically hiding, we have that $\left|\Pr_{M_1}(c_i) - \Pr_{M_2}(c_i)\right| \leq \epsilon$. 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

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

□

Proof of Theorem 6. 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 $p_{\mathcal{B}_i} = p\left(t_i^b\right) * p_E\left(t_i^b\right) * L^2$ operations and adheres to the computational model $\mathcal{M}_{t_i^b}$.

We observe that an evidence object E is a tuple $(\mathsf{SIG}, s, CDT_1, \ldots, CDT_n)$, where $CDT_i = \left(\mathsf{VC}_i, c_i, d_i, \mathsf{VC}_i', c_i', d_i', \mathsf{TS}_i, ts_i\right)$. Define $CT_i = \left(\mathsf{VC}_i', c_i', d_i', \mathsf{TS}_i, ts_i\right)$. The verification algorithm of ELSA ensures that ts_i is a valid timestamp for $\left(\mathsf{VC}_j, c_j, CT_j, \ldots, CT_{i-1}\right)$, 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 $\mathsf{Verify}\left(\mathsf{PKI}, t_{\mathsf{verify}}; \mathsf{dat}, t_{\mathsf{store}}, E\right) = 1$ 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 $\left(\sum_{i\in\mathsf{SIG}} \epsilon_i\left(p_{\mathcal{B}_i}\right)\right) + \left(\sum_{i\in\mathsf{COM}} \epsilon_i\left(p\left(t_i^b\right), p_E\left(t_i^b\right), p\left(t_i^b\right)\right)\right)$, where COM denotes the set of commitment schemes and SIG denotes the set of signature schemes of $\mathcal{S}$. □

Abbreviations

PKI: public key infrastructure; PoI: proof of integrity. This refers to the recursive list of vector commitments and timestamps that is maintained by ELSA or MCELSA, respectively, for each stored document or date

Acknowledgments

We thank Lucas Buschlinger, Timm Lippert, Christoph Beckmann, and Thomas Glaser for supporting us in implementing our testing scenarios.

Authors' contributions

JB, SK, and TA sparked the ideas for this research and supervised it. MG designed the single-client scheme and analyzed its security. PM transferred it to the multi-client setting and solved the challenges arising therefrom. PM furthermore evaluated MCELSA's performance. MG and PM are the major contributors in writing the manuscript. JB and SK supported its revision. All authors read and approved the final manuscript.

Funding

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – SFB 1119 – 236615297.

Availability of data and materials

The measurements presented in Section 6 were obtained from running a Java implementation of MCELSA in a virtual machine with the parameters described above. The source code for that implementation is available from the corresponding author on reasonable request.

Competing interests

The authors declare that they have no competing interests.

Author details

[1]Seceng, TU Darmstadt, Hochschulstr 10, 64293 Darmstadt, Germany. [2]CDC, TU Darmstadt, Darmstadt, Germany. [3]Seceng, TU Darmstadt, Darmstadt Germany. [4]Computer Engineering, Universität Passau, Passau, Germany.

Received: 16 January 2020 Accepted: 30 April 2020
Published online: 27 May 2020

References

1. D. Bayer, S. Haber, W. S. Stornetta, in *Sequences II: Methods in Communication, Security, and Computer Science*, ed. by R. Capocelli, A. De Santis, and U. Vaccaro. Improving the efficiency and reliability of digital time-stamping (Springer New York, New York, NY, 1993), pp. 329–334
2. N. Bitansky, R. Canetti, A. Chiesa, S. Goldwasser, H. Lin, A. Rubinstein, E. Tromer, The hunting of the snark. J. Cryptology. **30**(4), 989–1066 (2017). https://doi.org/10.1007/s00145-016-9241-9

3. J. Braun, J. Buchmann, D. Demirel, M. Geihs, M. Fujiwara, S. Moriai, M. Sasaki, A. Waseda, in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. Lincos: a storage system providing long-term integrity, authenticity, and confidentiality (ACM, New York, NY, USA, 2017), pp. 461–468. ASIA CCS '17
4. J. Braun, J. Buchmann, C. Mullan, A. Wiesmaier, Long term confidentiality: a survey. Des. Codes Crypt. **71**(3), 459–478 (2014)
5. A. Buldas, M. Geihs, J. Buchmann, in *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017 Proceedings, Part I, July 3–5, 2007*, ed. by J. Pieprzyk, S. Suriadi. Long-term secure commitments via extractable-binding commitments (Springer International Publishing, Cham, Auckland, New Zealand, 2017), pp. 65–81
6. D. Catalano, D. Fiore, in *Public-key cryptography – PKC 2013*, ed. by K. Kurosawa, G. Hanaoka. Vector commitments and their applications (Springer Berlin Heidelberg, Berlin, Heidelberg, 2013), pp. 55–72
7. M. Geihs, J. Buchmann, ELSA: efficient long-term secure storage of large datasets. Springer Lect. Notes Comput. Sci. **11396**, 269–286 (2018). CoRR abs/1810.11888. http://arxiv.org/abs/1810.11888
8. M. Geihs, N. Karvelas, S. Katzenbeisser, J. Buchmann, in *Proceedings of the 6th International Workshop on Security in Cloud Computing. SCC '18*. Propyla: privacy preserving long-term secure storage (ACM, New York, NY, USA, 2018), pp. 39–48. http://doi.acm.org/10.1145/3201595.3201599
9. R. Gennaro, S. Micali, in *Automata, languages and programming*, ed. by M. Bugliesi, B. Preneel, V. Sassone, and I. Wegener. Independent zero-knowledge sets (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006), pp. 34–45
10. N. Gisin, G. Ribordy, W. Tittel, H. Zbinden, Quantum cryptography. Rev Mod. Phys. **74**, 145–195 (2002)
11. S. Goldwasser, S. Micali, R. L. Rivest, A digital signature scheme secure against adaptive chosen-message attacks. SIAM J. Comput. **17**(2), 281–308 (1988). http://dx.doi.org/10.1137/0217017
12. S. Haber, W. S. Stornetta, How to time-stamp a digital document. J. Cryptol. **3**(2), 99–111 (1991). https://doi.org/10.1007/BF00196791
13. S. Halevi, S. Micali, in *Proceedings in Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996*, ed. by N. Koblitz. Practical and provably-secure commitment schemes from collision-free hashing (Springer Berlin Heidelberg, Berlin, Heidelberg, 1996), pp. 201–215
14. A. Herzberg, S. Jarecki, H. Krawczyk, M. Yung, in *Advances in Cryptology — CRYPTO' 95*, ed. by D. Coppersmith. Proactive secret sharing or How to cope with perpetual leakage (Springer Berlin Heidelberg, Berlin, Heidelberg, 1995), pp. 339–352
15. D. Hofheinz, Possibility and impossibility results for selective decommitments. J. Cryptol. **24**(3), 470–516 (2011). https://doi.org/10.1007/s00145-010-9066-x
16. A. K. Lenstra, *The Handbook of Information Security, chap. Key lengths*. (Wiley, Hoboken, 2004)
17. A. K. Lenstra, E. R. Verheul, Selecting cryptographic key sizes. J. Cryptol. **14**(4), 255–293 (2001)
18. R. C. Merkle, in *Proceedings in Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989*. A certified digital signature (Springer New York, New York, NY, 1989), pp. 218–238
19. National Institute of Standards and Technology: FIPS 197: Announcing the advanced encryption standard (AES) (2001). https://doi.org/10.6028/NIST.FIPS.180-4
20. National Institute of Standards and Technology: FIPS PUB 180-4: Secure hash standard (SHS) (2015)
21. R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM. **21**(2), 120–126 (1978)
22. A. Shamir, How to share a secret. Commun. ACM. **22**(11), 612–613 (1979)
23. C. E. Shannon, Communication theory of secrecy systems. Bell Syst. Tech. J. **28**(4), 656–715 (1949)
24. P. W. Shor, Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (1997). http://dx.doi.org/10.1137/S0097539795293172
25. M. A. G. Vigil, J. A. Buchmann, D. Cabarcas, C. Weinert, A. Wiesmaier, Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: a survey. Comput. Secur. **50**, 16–32 (2015)
26. C. Weinert, D. Demirel, M. Vigil, M. Geihs, J. Buchmann, in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. ASIA CCS '17*. Mops: a modular protection scheme for long-term storage (ACM, New York, NY, USA, 2017), pp. 436–448

Publisher's Note