ELSA: Efficient Long-Term Secure Storage of Large Datasets*

Matthias Geihs and Johannes Buchmann

TU Darmstadt, Germany

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. We implemented our solution and measured its performance in a scenario where 12 000 data items are aggregated, stored, protected, and verified over a time span of 100 years. Our measurements show that our new solution completes this evaluation scenario an order of magnitude faster than the state of the art.

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 docu-

* This work has been co-funded by the DFG as part of project S6 within CRC 1119 CROSSING. This is the proceedings version as published at ICISC'18. An extended version can be found at arXiv.org [8].

ments, 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 of relatively small data items 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 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 [11], 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

In this paper we propose 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.

Our first contribution (Section 3) is to 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 [6] 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 [10].

Our second contribution (Section 4) is the construction of the long-term secure 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.

Finally, we experimentally demonstrate (Section 5) the performance improvements achieved by ELSA in a scenario where 12 000 data items of size 10 kB are aggregated, stored, protected, retrieved, and verified during a timespan of 100 years. For this, we implemented ELSA and the state of the art long-term secure storage architecture LINCOS. Our measurements show that ELSA completes the evaluation scenario $17x$ faster than LINCOS and integrity protection

consumes $101x$ less memory. In particular, protection renewal is significantly faster with ELSA. Renewing the timestamps for approximately 12 000 data items takes 21.89 min with LINCOS and only 0.34 s with ELSA. Furthermore, storage of the timestamps and commitments consumes 1.75 GB of storage space with LINCOS and only 17.27 MB with ELSA at the end of the experiment. These improvements are achieved at slightly higher storage costs for the shareholders. Each shareholder consumes 559 MB with LINCOS and 748 MB with ELSA. The storage costs for integrity protection are independent of the size of the data items. Storage, retrieval, and verification of a data item takes less than a second. Overall, our evaluation shows that ELSA provides practical performance and is suitable for storing and protecting large and complex databases that consist of relatively small data items over long periods of time (e.g., health record or governmental document databases).

1.3 Related work

Our notion of vector commitments is reminiscent of the one proposed by Catalano and Fiore [7]. 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. [9] 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

For a probabilistic algorithm $\mathcal{A}$ and input x, we write $\mathcal{A}(x) \rightarrow_r y$ to denote that $\mathcal{A}$ on input x produces y using random coins r. For a vector $V = (v_1, \ldots, v_n)$, $n \in \mathbb{N}$, and set $I \subseteq [n]$, define $V_I := (v_i)_{i \in I}$, and for $i \in [n]$, define $V_i := v_i$. 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 briefly introduce the cryptographic primitives that are used in this paper. A more extensive description can be found in the full version [8].

Digital Signature Schemes A digital signature scheme SIG is defined by a message space $\mathcal{M}$ and algorithms Setup, Sign, and Verify. Algorithm $\mathsf{Setup} \rightarrow (sk, pk)$ for generateing a secret signing key sk and a public verification key pk. Algorithm $\mathsf{Sign}(sk, m) \rightarrow s$ gets as input a secret key sk and a message $m \in \mathcal{M}$ and outputs a signature s. Algorithm $\mathsf{Verify}(pk, m, s) \rightarrow b$ gets as input a public key pk, a message m, and a signature s, and outputs $b = 1$, if the signature is valid, and 0, if it is invalid. A signature scheme is ϵ-unforgeable-secure if the probability of a t-bounded adversary $\mathcal{A}$ forging a signature is bounded by $\epsilon(t)$.

Timestamp schemes A timestamp scheme [12] is a protocol between 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. Here, we consider signature-based timestamping, where the timestamp is a signature on the timestamped document and the current time. In this case, the timestamp service generating the signature must be trusted to use the correct time value.

Commitment schemes A (non-internactive) commitment scheme COM is defined by a message space $\mathcal{M}$ and algorithms Setup, Commit, and Verify. Algorithm $\mathsf{Setup} \rightarrow pk$ generates a public commitment key pk. Algorithm $\mathsf{Commit}(pk, m) \rightarrow (c, d)$ gets as input a public key pk and a message $m \in \mathcal{M}$ and outputs a commitment c and a decommitment d. Algorith $\mathsf{Verify}(pk, m, c, d) \rightarrow b$ gets as input a public key pk, a message m, a commitment c, and a decommitment d, and outputs $b = 1$, if the decommitment is valid, and 0, if it is invalid. A commitment scheme is considered secure if it is hiding (i.e., a commitment does not leak information) and binding (i.e., the committer cannot change his mind about the committed message). 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 [6].

Keyed hash functions A keyed hash function is a tuple of algorithms (K, H) where K is a probabilistic algorithm that generates a key k and 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 $\mathcal{A}_2$,

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

Secret sharing schemes A proactive secret sharing scheme [14] is protocol between a data owner and a set of shareholders. It has a protocol Setup for generating system parameters, a protocol Share for sharing a data object, a protocol Reshare for refreshing the shares, and a protocol Reconstruct for reconstructing a data object from a given set of shares. In this work, we consider threshold secret sharing schemes, for which there exists a threshold parameter t (chosen by the data owner) such that any set of t shareholders can reconstruct the secret, but any set of less than t shareholders has no information about the secret.

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. The proofs of the presented theorems can be found in the full version [8].

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 [7]. 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 tuple* $(L, \mathcal{M}, \mathsf{Setup}, \mathsf{Commit}, \mathsf{Open}, \mathsf{Verify})$*, 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 i-th message corresponding to D.*

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

A vector commitment scheme is correct, if a decommitment produced by Commit and 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 $n \in [L]$, $M \in \mathcal{M}^n$, $k \in \mathsf{Setup}$, $i \in [n]$,*

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

A vector commitment scheme is statistically hiding under selective opening, if the distribution of commitments and openings does not depend on the unopened messages. For any public key k and message m, define $C_k(m)$ as the random variable that takes the value of c when sampling $\mathsf{Commit}(k, m) \to (c, d)$. A commitment scheme is ϵ-statistically-hiding if for any $k \in \mathsf{Setup}$, any pair of messages (m_1, m_2), $\Delta(C_k(m_1), C_k(m_2)) \leq \epsilon$.

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 $(M_1)_I = (M_2)_I$, $k \in \mathsf{Setup}$,*

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

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

Definition 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 $\mathcal{A}_2$,*

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

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.*

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

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

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

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

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

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 $\epsilon'(t_1, t_\mathcal{E}, t_2) = 2L * \epsilon(t_1 + t_\mathcal{E}/L, t_\mathcal{E}/L, t_2)$.*

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.*

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

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

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

$\mathsf{Verify}(k, m, c, d, i) \to b$**:** *Let* $k \to (k_1, k_2)$ *and* $d \to (c', d', d'')$*. Compute* $\mathsf{COM.Verify}(k_1, m, c', d') \to b_1$ *and then compute* $\mathsf{VC.Verify}(k_2, c', c, d'', i) \to b_2$*, 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.*

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.*

Theorem 5. *If* COM *and* VC *of Construction 2 are* ϵ*-extractable-binding, Construction 2 is an* ϵ'*-extractable-binding vector commitment scheme with* $\epsilon'(t_1, t_{\mathcal{E}}, t_2) = L * \epsilon(t_1 + t_{\mathcal{E}}/L, t_{\mathcal{E}}/L, t_2)$*.*

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 (Figure 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.

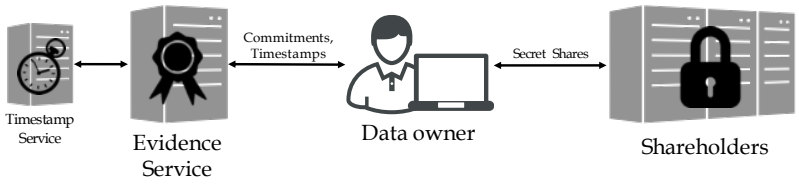

Fig. 1. Overview of the components of ELSA.

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 security is weakened, RenewCom renews the protection if the commitment scheme security is weakened, RenewShares renews the shares to protect against a mobile adversary who collects multiple shares over time, and Verify verifies the integrity of a retrieved file.

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 then the size of the message space of the secret sharing scheme, dat is first split into chunks that fit into the message space and then the chunks are shared individually. Each shareholder maintains a database that describes which shares belong to which data item name. When we write SH.Retrieve(name), we mean that the data owner retrieves the shares associated identifier name from the shareholders and reconstructs the data using protocol SHARE.Reconstruct.

Initialization The data owner uses algorithm ELSA.Init to initialize the storage system. The algorithm gets as input a proactive secret sharing scheme SHARE, a set of shareholder addresses $(\text{shURL}_i)_{i\in[N]}$, a sharing threshold T, and an evidence service address esURL. It then initializes the storage module SH by 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.

Data storage The client uses algorithm ELSA.Store (Algorithm 1) to store a set of data files $[\mathsf{file}_i]_{i\in[n]}$, 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 algorithm first signs each of the data objects individually. It then stores the file data, the public key certificate of the signature scheme instance, and the generated signature at the secret sharing storage system. Afterwards, the algorithm generates a vector commitment (c, D) to the file data vector and the signatures. For each file, the corresponding decommitment is extracted and stored at the shareholders. The file names `filenames`, the commitment scheme instance VC, the commitment c, and the chosen timestamp scheme instance TS are sent to the evidence service.

When the evidence service receives $(\texttt{filenames}, \mathsf{VC}, c, \mathsf{TS})$, it does the following in algorithm AddCom (Algorithm 2). It first timestamps the commitment (VC, c) and thereby obtains a timestamp ts. Then, it starts a new evidence list $l = [(\mathsf{VC}, c, \mathsf{TS}, \text{ts})]$ and assigns this list with all the file names in `filenames`. Also, it adds l to the list `renewLists`, which contains the lists that are updated on a timestamp renewal.

Algorithm 1: $\mathsf{ELSA.Store}([\mathsf{file}_i]_{i\in[n]}, \mathsf{SIG}, \mathsf{VC}, \mathsf{TS})$

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

Algorithm 2: $\mathsf{ES.AddCom}(\mathtt{filenames}, \mathsf{VC}, c, \mathsf{TS})$

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

Timestamp renewal Algorithm $\mathsf{ES.RenewTs}$ (Algorithm 3) is performed by the evidence service regularly in order to protect against the weakening of the currently used timestamp scheme. The algorithm gets as input a vector commitment scheme instance VC' and a timestamp scheme instance TS. It first creates a vector commitment (c', D') for the list of renewal items `renewLists`. Here, we 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 $\mathtt{renewLists}[i]$.

Algorithm 3: $\mathsf{ES.RenewTs}(\mathsf{VC}', \mathsf{TS})$

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

Commitment renewal The data owner runs algorithm ELSA.RenewCom (Algorithm 4) 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 are retrieved from the evidence service and complemented with the decommitment values stored at the shareholders. Next, a list with the data items, the signatures, and the current evidence for each data item is constructed. This list is then committed using the vector commitment scheme VC. The decommitments are extracted and stored at the shareholders, and the commitment is added to the evidence at the evidence service using algorithm ES.AddComRenew.

Algorithm 4: ELSA.RenewCom(VC, TS)

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

Algorithm 5: ES.AddComRenew(VC, c, TS)

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

Secret share renewal There are two types of share renewal supported by ELSA. The first type (ELSA.RenewShares()) 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 6) 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 6: ELSA.RenewSharing(SHARE, $(\mathrm{shURL}_i)_{i\in[N]}, T$)

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

Data retrieval The algorithm ELSA.Retrieve (Algorithm 7) 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.

Algorithm 7: ELSA.Retrieve(name)

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

Verification Algorithm ELSA.Verify (Algorithm 8) describes how a verifier can check the integrity of a data item using the evidence produced by ELSA. Here

we denote by $\mathrm{NTT}(i, e, t_{\mathsf{verify}})$ the time of the next timestamp after entry i of e and by $\mathrm{NCT}(i, e, t_{\mathsf{verify}})$ the time of the timestamp corresponding to the next commitment after entry i, and we set $\mathrm{NTT}(i, e, t_{\mathsf{verify}}) = t_{\mathsf{verify}}$ if i is the last timestamp and $\mathrm{NCT}(i, e, t_{\mathsf{verify}}) = t_{\mathsf{verify}}$ if i is the last commitment in e. The algorithm gets as input a reference to the considered PKI (e.g., a trust anchor), the current verification time t_{verify}, the data to be checked dat, the storage time t_{store}, and the corresponding evidence $E = (\mathsf{SIG}, s, e)$. The algorithm returns true, if dat is authentic and has been stored at time t_{store}.

In more detail, the verification algorithm works as follows. It first checks 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 $(\mathsf{dat}, \mathsf{SIG}, s)$ at the time of the next commitment and the timestamp is valid at the next timestamp. Then, for each of the remaining $|e| - 1$ 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.

Algorithm 8: $\mathsf{ELSA.Verify}(\mathsf{PKI}, t_{\mathsf{verify}} : \mathsf{dat}, t_{\mathsf{store}}, E) \to b$

$(\mathsf{SIG}, s, e) \leftarrow E$;
$((\mathsf{VC}, c, d), (\mathsf{VC}', c', d'), (\mathsf{TS}, \mathrm{ts})) \leftarrow e_1$;
$t_{\mathrm{nt}} \leftarrow \mathrm{NTT}(1, e, t_{\mathsf{verify}})$; $t_{\mathrm{nc}} \leftarrow \mathrm{NCT}(1, e, t_{\mathsf{verify}})$;
$b \leftarrow \mathsf{SIG.Verify}(\mathsf{PKI}, \mathrm{ts}.t : \mathsf{dat}, s)$;
$b \wedge= \mathsf{VC.Verify}(\mathsf{PKI}, t_{\mathrm{nc}} : (\mathsf{dat}, \mathsf{SIG}, s), c, d)$;
$b \wedge= \mathsf{TS.Verify}(\mathsf{PKI}, t_{\mathrm{nt}} : c, \mathrm{ts}, t_{\mathsf{store}})$;
$L \leftarrow (\mathsf{VC}, c, \mathsf{TS}, \mathrm{ts})$;
for $i \in [2, \ldots, |e|]$ **do**
  $((\mathsf{VC}, c, d), (\mathsf{VC}', c', d'), (\mathsf{TS}, \mathrm{ts})) \leftarrow e_i$;
  $t_{\mathrm{nt}} \leftarrow \mathrm{NTT}(i, e, t_{\mathsf{verify}})$; $t_{\mathrm{nc}} \leftarrow \mathrm{NCT}(i, e, t_{\mathsf{verify}})$;
  if $\mathsf{VC} = \perp$ **then**
    $b \wedge= \mathsf{VC}'.\mathsf{Verify}(\mathsf{PKI}, t_{\mathrm{nt}} : L, c', d')$;
    $b \wedge= \mathsf{TS.Verify}(\mathsf{PKI}, t_{\mathrm{nt}} : c', \mathrm{ts}, \mathrm{ts}.t)$;
    $L \mathrel{+}= (\mathsf{VC}', c', d', \mathsf{TS}, \mathrm{ts})$;
  else
    $\mathsf{dat}' \leftarrow (\mathsf{dat}, Cert, s, e[1, i-1])$;
    $b \wedge= \mathsf{VC.Verify}(\mathsf{PKI}, t_{\mathrm{nc}} : \mathsf{dat}', c, d)$;
    $b \wedge= \mathsf{TS.Verify}(\mathsf{PKI}, t_{\mathrm{nt}} : c, \mathrm{ts}, \mathrm{ts}.t)$;
    $L \leftarrow (\mathsf{VC}, c, \mathsf{TS}, \mathrm{ts})$;
return b;

4.2 Security analysis

The security analysis can be found in the full version [8].

5 Performance evaluation

We compare the performance of our new architecture ELSA with the performance of the storage architecture LINCOS [3], which is the fastest existing storage architecture that provides long-term integrity and long-term confidentiality.

5.1 Evaluation scenario

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

We assume the following renewal schedule for protecting the evidence against the weakening of cryptographic primitives. The signatures are renewed every 2 years, as this is a typical lifetime of a public key certificate, which is needed to verify the signatures. While signature scheme instances can only be secure as long as the 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. Secret shares are renewed every 5 years, which we believe could be a typical shareholder life cycle.

In our architecture we instantiate the cryptographic schemes as follows. As signature scheme, we first use the RSA Signature Scheme [21] and then switch to the post-quantum secure XMSS signature scheme [5] by 2030, as we anticipate the development of large-scale quantum computers. 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 the hash functions as random oracles, they provide the necessary extractable-binding property. We adjust the signature and commitment scheme parameters over time as proposed by Lenstra and Verheul [17, 16]. The resulting parameter sets are shown in Table 1. For the storage system, we use the secret sharing scheme by Shamir [22]. We run this scheme with 4 shareholders and a threshold of 3 shareholders are required for reconstruction. Resharing is carried out centrally by the data owner.

5.2 Results

We now present the results of our performance analysis. Figure 2 compares the computation time and storage costs of the two systems, ELSA and LINCOS. Our

Table 1. Overview of the used commitment and signature scheme instances and their usage period.

Years	Signatures	Commitments
2018-2030	RSA-2048	HM-256
2031-2090	XMSS-256	HM-256
2091-2118	XMSS-512	HM-512

implementation was done using the programming language Java. The experiments were performed on a computer with a quad-core AMD Opteron CPU running at 2.3 GHz and the Java virtual machine was assigned 32 GB of RAM.

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

Next, we observe that ELSA is also more efficient compared to LINCOS when it comes to the consumed storage space at the evidence service. This is, again, because ELSA requires fewer timestamps to be generated and stored than LINCOS. After running for 100 years, the evidence service of ELSA consumes only 17.27 MB while the evidence service of LINCOS consumes 1.75 GB of storage space. We observe by Figure 2 that ELSA consumes slightly more storage space at the shareholders than LINCOS. This is because additional decommitment information for the vector commitments must be stored. After running for 100 years, a shareholder of ELSA consumes about 748 MB while a shareholder of LINCOS consumes about 559 MB of storage space.

References

1. Bayer, D., Haber, S., Stornetta, W.S.: Improving the efficiency and reliability of digital time-stamping. In: Capocelli, R., De Santis, A., Vaccaro, U. (eds.) Sequences II: Methods in Communication, Security, and Computer Science. pp. 329–334. Springer New York, New York, NY (1993)
2. Bitansky, N., Canetti, R., Chiesa, A., Goldwasser, S., Lin, H., Rubinstein, A., Tromer, E.: The hunting of the snark. Journal of Cryptology **30**(4), 989–1066 (Oct 2017). https://doi.org/10.1007/s00145-016-9241-9, https://doi.org/10.1007/s00145-016-9241-9
3. Braun, J., Buchmann, J., Demirel, D., Geihs, M., Fujiwara, M., Moriai, S., Sasaki, M., Waseda, A.: Lincos: A storage system providing long-term integrity, authenticity, and confidentiality. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 461–468. ASIA CCS '17, ACM, New York, NY, USA (2017)

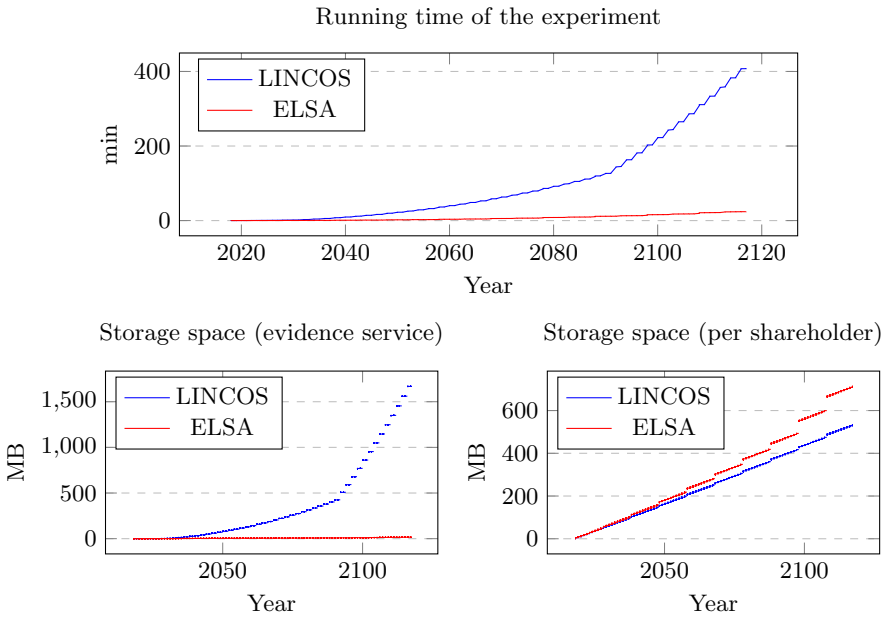

Fig. 2. Running time of the experiment and storage space consumption of the evidence service and per shareholder.

4. Braun, J., Buchmann, J., Mullan, C., Wiesmaier, A.: Long term confidentiality: a survey. Designs, Codes and Cryptography **71**(3), 459–478 (2014)
5. Buchmann, J., Dahmen, E., Hülsing, A.: Xmss - a practical forward secure signature scheme based on minimal security assumptions. In: Yang, B.Y. (ed.) Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 – December 2, 2011. Proceedings. pp. 117–129. Springer Berlin Heidelberg, Berlin, Heidelberg (2011)
6. Buldas, A., Geihs, M., Buchmann, J.: Long-term secure commitments via extractable-binding commitments. In: Pieprzyk, J., Suriadi, S. (eds.) Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part I. pp. 65–81. Springer International Publishing, Cham (2017)
7. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Kurosawa, K., Hanaoka, G. (eds.) Public-Key Cryptography – PKC 2013. pp. 55–72. Springer Berlin Heidelberg, Berlin, Heidelberg (2013)
8. Geihs, M., Buchmann, J.: Elsa: Efficient long-term secure storage of large datasets (full version). arXiv:1810.11888 (2018)
9. Geihs, M., Karvelas, N., Katzenbeisser, S., Buchmann, J.: Propyla: Privacy preserving long-term secure storage. In: Proceedings of the 6th International Workshop on Security in Cloud Computing. pp. 39–48. SCC '18, ACM, New York, NY, USA (2018). https://doi.org/10.1145/3201595.3201599, http://doi.acm.org/10.1145/3201595.3201599
10. Gennaro, R., Micali, S.: Independent zero-knowledge sets. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) Automata, Languages and Programming. pp.

34–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2006)
11. Gisin, N., Ribordy, G., Tittel, W., Zbinden, H.: Quantum cryptography. Rev. Mod. Phys. **74**, 145–195 (Mar 2002)
12. Haber, S., Stornetta, W.S.: How to time-stamp a digital document. Journal of Cryptology **3**(2), 99–111 (Jan 1991). https://doi.org/10.1007/BF00196791, https://doi.org/10.1007/BF00196791
13. Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings. pp. 201–215. Springer Berlin Heidelberg, Berlin, Heidelberg (1996)
14. Herzberg, A., Jarecki, S., Krawczyk, H., Yung, M.: Proactive secret sharing or: How to cope with perpetual leakage. In: Coppersmith, D. (ed.) Advances in Cryptology — CRYPTO' 95. pp. 339–352. Springer Berlin Heidelberg, Berlin, Heidelberg (1995)
15. Hofheinz, D.: Possibility and impossibility results for selective decommitments. Journal of Cryptology **24**(3), 470–516 (Jul 2011). https://doi.org/10.1007/s00145-010-9066-x, https://doi.org/10.1007/s00145-010-9066-x
16. Lenstra, A.K.: The Handbook of Information Security, chap. Key lengths. Wiley (2004)
17. Lenstra, A.K., Verheul, E.R.: Selecting cryptographic key sizes. Journal of Cryptology **14**(4), 255–293 (2001)
18. Merkle, R.C.: A certified digital signature. In: Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings. pp. 218–238. Springer New York, New York, NY (1989)
19. National Institute of Standards and Technology: FIPS 197: Announcing the advanced encryption standard (AES) (2001)
20. National Institute of Standards and Technology: FIPS PUB 180-4: Secure hash standard (SHS) (2015)
21. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21**(2), 120–126 (Feb 1978)
22. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (Nov 1979)
23. Shannon, C.E.: Communication theory of secrecy systems. Bell System Technical Journal **28**(4), 656–715 (1949)
24. Shor, P.W.: Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. SIAM J. Comput. **26**(5), 1484–1509 (Oct 1997). https://doi.org/10.1137/S0097539795293172, http://dx.doi.org/10.1137/S0097539795293172
25. Vigil, M.A.G., Buchmann, J.A., Cabarcas, D., Weinert, C., Wiesmaier, A.: Integrity, authenticity, non-repudiation, and proof of existence for long-term archiving: A survey. Computers & Security **50**, 16–32 (2015)
26. Weinert, C., Demirel, D., Vigil, M., Geihs, M., Buchmann, J.: Mops: A modular protection scheme for long-term storage. In: Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security. pp. 436–448. ASIA CCS '17, ACM, New York, NY, USA (2017)