PROPYLA: **Privacy-Preserving Long-Term Secure Storage**

(Revised Version, Nov 2018)

Matthias Geihs, Nikolaos Karvelas, Stefan Katzenbeisser, and Johannes Buchmann
Technische Universität Darmstadt, Germany

ABSTRACT

An increasing amount of sensitive information today is stored electronically and a substantial part of this information (e.g., health records, tax data, legal documents) must be retained over long time periods (e.g., several decades or even centuries). When sensitive data is stored, then integrity and confidentiality must be protected to ensure reliability and privacy. Commonly used cryptographic schemes, however, are not designed for protecting data over such long time periods. Recently, the first storage architecture combining long-term integrity with long-term confidentiality protection was proposed (AsiaCCS'17). However, the architecture only deals with a simplified storage scenario where parts of the stored data cannot be accessed and verified individually. If this is allowed, however, not only the data content itself, but also the access pattern to the data (i.e., the information which data items are accessed at which times) may be sensitive information.

Here we present the first long-term secure storage architecture that provides long-term access pattern hiding security in addition to long-term integrity and long-term confidentiality protection. To achieve this, we combine information-theoretic secret sharing, renewable timestamps, and renewable commitments with an information-theoretic oblivious random access machine. Our performance analysis of the proposed architecture shows that achieving long-term integrity, confidentiality, and access pattern hiding security is feasible.

1 INTRODUCTION

1.1 Motivation and Problem Statement

Large amounts of sensitive data (e.g., health records, governmental documents, enterprise documents, land registries, tax declarations) are stored in cloud-based data centers and require integrity and confidentiality protection over long time periods (e.g., several decades or even centuries). Here, by *integrity* we mean that illegitimate changes to the data can be discovered and by *confidentiality* we mean that only authorized parties can access the data.

Typically, cryptographic signature and encryption schemes (e.g., RSA [24] and AES [23]) are used to ensure data integrity and confidentiality. However, the security of currently used cryptographic schemes relies on computational assumptions (e.g., that the prime factors of a large integer cannot be computed efficiently). Such schemes are called *computationally secure*. Due to technological progress, however, computers are becoming more powerful over time and computational assumptions made today are likely to break at some point in the future (e.g., factoring large integers will be possible using quantum computers). Thus, commonly used cryptographic schemes, which rely on a single computational assumption, are insufficient for protecting data over long periods of time (e.g., decades or even centuries).

To enable long-term integrity protection, Bayer et al. [1] proposed a method for renewing digital signatures by using timestamps. Following their approach, several other schemes had been developed of which [31] gives an overview. It should be noted that only recently the concrete security of these schemes has been investigated and proven [6, 7, 11]. While these results indicate that integrity protection can be renewed using timestamps, it appears entirely infeasible to prolong confidentiality. The reason is that once a ciphertext generated by a computationally secure encryption scheme is observed by an attacker, the attacker can always break the encryption using brute-force and a sufficiently large amount of computational resources over time. While honey encryption [18] can help against brute-force attacks in some scenarios, the only rigorous way to provide long-term confidentiality protection is to use *information-theoretically secure* schemes, as their security does not rely on computational assumptions. Information-theoretic solutions exist for many cryptographic tasks. For example, secure data storage can be realized using secret sharing [17, 25] and secure communication can be realized by combining Quantum Key Distribution [12] with One-Time-Pad Encryption [26]. We refer the reader to [4] for an overview of techniques relevant to long-term confidentiality protection. Recently, the first storage system that combines long-term integrity with long-term confidentiality protection has been proposed by Braun et al. [3]. They achieve this by combining secret sharing with renewable timestamps and unconditionally hiding commitments. However, their scheme does not allow for accessing and verifying parts of the stored data separately.

Here we consider a setting where the database consists of a list of blocks and each data block can be retrieved and verified separately. In this setting, however, not only the data content may be regarded as sensitive information, but also the access pattern to the data (i.e., which data blocks are accessed at which times). For example, if genome data is stored and accessed at a subset of locations S that are known to be associated with a certain property X, then it is likely that the person accessing the data is concerned with X. In such a scenario it is desirable not only to ensure the confidentiality of the data content, but also to ensure the confidentiality of the data access pattern.

1.2 Contribution

In this paper we solve the problem of long-term secure data storage with access pattern hiding security by presenting the storage architecture PROPYLA.

Long-term integrity and long-term confidentiality protection in PROPYLA are achieved similar as in [3]. Confidential data is

This is the revised version of a paper that appeared in the proceedings of the 6th International Workshop on Security in Cloud Computing (SCC'18) co-located with the 13th ACM ASIA Conference on Computer and Communications Security (ASIACCS'18). See the appendix for a version history.

stored at a set of shareholders using information-theoretic secure secret sharing and integrity protection is achieved by combining renewable commitments with renewable timestamps. An evidence service helps the user with maintaining integrity protection and renewing the protection when necessary.

We now explain how we additionally achieve long-term access pattern hiding security. The main idea is to integrate an information theoretically secure oblivious random access machine (ORAM) [13] with the shareholders and the evidence service so that none of these storage servers learns the access pattern of the client. The challenge here is to ensure that even under composition the individual security properties (i.e., renewable integrity protection, information-theoretic confidentiality, and information-theoretic access pattern hiding security) are preserved. To preserve information-theoretic access pattern hiding security, the data flows induced by any two different database accesses must be indistinguishable from the viewpoint of any of the storage servers. Typically, this is solved by using an ORAM in combination with shuffling and re-encrypting the accessed data blocks. This approach, however, cannot be directly applied in our case, as here commitments are stored at the evidence service which must be timestamped and therefore cannot be stored encrypted. We solve the problem of obfuscating the reshuffling by employing a recommitment technique that enables us to refresh commitments while maintaining their binding property. With regard to the secret shareholders it is sufficient to regenerate the secret shares on every data access in order to obfuscate the data block shuffling. This ensures that there is no correlation between the received and stored data blocks, and thus the shuffling of the data blocks cannot be traced. We also analyze the security of PROPYLA and show it indeed achieves long-term integrity, long-term confidentiality, and long-term access pattern hiding security. Finally, we evaluate the performance of PROPYLA and show that storage, computation, and communication costs appear manageable.

1.3 Organization

The paper is organized as follows. In Section 2, we state preliminaries. In Section 3, we describe our long-term secure storage architecture PROPYLA. In Section 4, we analyze its security and in Section 5, we evaluate its performance.

2 PRELIMINARIES

2.1 Time-Stamping

A timestamp scheme [15] consists of an algorithm Setup, a protocol Stamp executed between a client C and a timestamp service $\mathcal{TS}$, and an algorithm VerTs with the following properties.

- **Setup():** At initialization, the timestamp service $\mathcal{TS}$ runs this algorithm and outputs a public verification key pk.
- **Stamp:** On input a data object dat, the client runs this protocol with the timestamp server. When the protocol has finished, the client obtained a timestamp ts for dat, where ts.t denotes the time associated with the timestamp.
- **$\text{VerTs}_{pk}(\text{dat}, \text{ts})$:** Given the public key of the timestamp service, on input a data object dat and a timestamp ts, this algorithm returns 1, if the timestamp is valid for dat, and 0 otherwise.

A timestamp scheme is secure if it is infeasible for an adversary to generate a timestamp ts and a data object dat such that ts is valid for dat and dat did not exist at time ts.t. This security notion is formalized in [8–10].

2.2 Commitments

A commitment scheme is the digital analog of a sealed envelope and allows to commit to a message m without revealing it. It consists of the algorithms Setup, Commit, and VerCom, that have the following properties.

- **Setup():** This algorithm is run by a trusted party and generates the public commitment parameters cp.
- **$\text{Commit}_{cp}(\text{dat})$:** Given the commitment parameters cp, on input a data object dat, this algorithm outputs a commitment c, and a decommitment d.
- $\text{VerCom}_{cp}(\text{dat}, c, d)$ Given the commitment parameters cp, on input a data object dat, a commitment c, and a decommitment d, this algorithm outputs 1 if d is a valid decommitment from c to dat and 0 if it is invalid.

A commitment scheme is secure if it is hiding and binding. Hiding means that a commitment does not reveal anything about the message and binding means that a committer cannot decommit to a different message. Here, we are interested in information theoretically hiding and computationally binding commitments. For a more formal description of commitment schemes we refer to [14, 16].

2.3 Proactive Secret Sharing

A proactive secret sharing scheme [17] allows a client C to share a secret among a set of shareholders $\mathcal{SH}$ such that less than a threshold number of shareholder cannot learn the secret. It is defined by protocols Setup, Share, Reshare, and Reconstruct.

- **Setup:** In this protocol the client and the shareholders establish the system parameters.
- **Share:** This protocol is run between the client and the shareholders. The client takes as input a data object dat and after the protocol has ended each shareholder holds a share of the data.
- **Reshare:** This protocol is run periodically between the shareholders. The shareholders take as input their shares and after the protocol is finished, the shareholders have obtained new shares which are not correlated with the old shares. This protects against a mobile adversary who compromises one shareholder after another over time.
- **Reconstruct:** This protocol is run between the shareholders and the client. The shareholders take as input their shares and after the protocol is finished the client outputs the reconstructed data object dat. If the reconstruction fails, the client outputs $\perp$.

A secret sharing scheme is secure if no information about the secret is leaked given that at least a threshold number of shareholders is not corrupted. The threshold is usually determined in the setup phase and in some schemes can be changed during the resharing phase.

2.4 Oblivious RAM

An oblivious random access machine (ORAM) [13] allows a client to access a remotely stored database such that the storage server does not learn which data items are of current interest to the client. Here we assume that the client's data consists of N blocks of equal size and each block is associated with a unique identifier $\mathsf{id} \in \{1, \ldots, N\}$. The server holds a database of size $M > N$ blocks and each block in the server database is identified by a location $i \in \{1, \ldots, M\}$. A (stash-free) ORAM is defined by algorithms Setup, GenAP, and GetId with the following properties.

- $\mathsf{Setup}(N)$: This algorithm takes as input the client database size N and generates a client local state s and a server database size M.
- $\mathsf{GenAP}(s, \mathsf{id})$: This algorithm gets as input a client state s and a client block identifier $\mathsf{id} \in \{1, \ldots, N\}$, and outputs an access pattern $P \in \{1, \ldots, N\}^{2\times n}$, which is a sequence of server block location pairs, and a new client local state s'.
- $\mathsf{GetId}(s, i)$: This algorithm gets as input a client state s and a server block location $i \in \{1, \ldots, M\}$, and outputs the corresponding client block identifier $\mathsf{id} \in \{1, \ldots, N\}$.

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

An ORAM is secure if the access patterns generated by GenAP are indindistinguishable from each other. In the security experiment (Algorithm 2.1), an adversary can instruct the client to access blocks of its choice and then sees the induced access patterns. In order to break the security, the adversary has to distinguish the access patterns of two access instructions of its choice.

Definition 2.1 (ORAM Security). An oblivious RAM ORAM is information theoretically secure if for any integer N and probabilistic algorithm $\mathcal{A}$,

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

Algorithm 1: The ORAM access pattern hiding experiment $\mathbf{Exp}^{\mathrm{APH}}_{\mathsf{ORAM},N}(\mathcal{A})$.

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

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

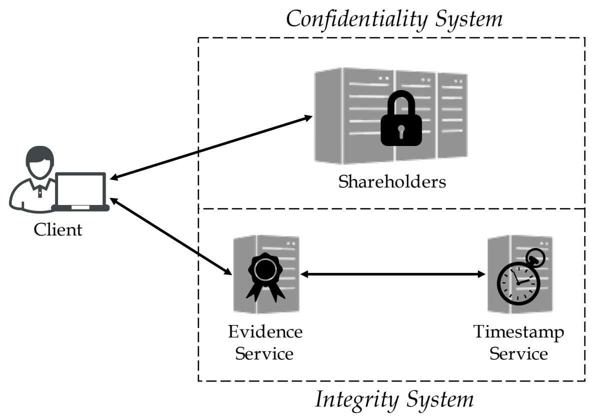

Figure 1: Overview of the storage architecture PROPYLA.

3 DESCRIPTION OF PROPYLA

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

3.1 Overview

PROPYLA comprises the following components: a client, an integrity system, which consists of an evidence service and timestamp service, and a confidentiality system, which consists of a set of shareholders (Figure 1).

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

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

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

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

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

To achieve access pattern hiding, the client makes accesses to the evidence service and the storage servers using an information theoretically secure ORAM. Therefore, the evidence service and the shareholders store a database consisting of $M > N$ blocks, where M is determined by the choice of the ORAM. The additional $M - N$ blocks provide the client with the necessary storage space so that it can reshuffle and access the data such that a uniform distribution of accesses over the server blocks is achieved. In order to preserve the access pattern hiding property, there must also be no correlation between the transmitted data of any two blocks. Typically, this is achived by re-encrypting the data on every access. In our solution, however, this technique cannot be applied because the evidence service must receive the commitments in plaintext so that it can timestamp them in order to renew the integrity protection. Instead, we use a recommitment technique to refresh the commitments: we let the client commit to the previous commitment and timestamp. Thereby, a new commitment is obtained that is indistinguishable from other fresh commitments while the connection to the originally timestamped commitment is maintained.

3.2 Protocols

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

Throughout the description of the protocols we use the following notation. For $i \in \{1, \dots, M\}$ and commitment c, we write $\mathcal{ES}.\mathsf{Write}(i, c)$ to denote that the client instructs the evidence service to store commitment c at block i. Furthermore, we write $E_{\mathcal{ES}} \leftarrow \mathcal{ES}.\mathsf{Read}(i)$ to denote that the client retrieves evidence $E_{\mathcal{ES}}$ of block i from the evidence service. Likewise, we denote by $\mathcal{SH}.\mathsf{Write}(i, (\mathsf{dat}, E))$ that the client stores data dat and evidence E to block i at the shareholders using protocol SHARE.Share and we write $(\mathsf{dat}, E) \leftarrow \mathcal{SH}.\mathsf{Read}(i)$ to denote that the client retrieves dat and E of block i from the shareholders using protocol SHARE.Reconstruct, where SHARE is the secret sharing scheme chosen by the client in the initialization phase. For a data object dat, we write $\mathsf{ts} \leftarrow \mathcal{TS}.\mathsf{Stamp}(\mathsf{dat})$ to denote that a timestamp ts for dat is obtained from timestamp service $\mathcal{TS}$. Likewise, we write $(c, d) \leftarrow \mathsf{CSI}.\mathsf{Commit}(\mathsf{dat})$ to denote that a commitment and decommitment are generated for dat using commitment scheme instance CSI. Throughout our description, we assume that the timestamp services are initialized appropriately using algorithm Setup of the corresponding timestamp scheme. Similarly, we assume that commitment scheme instances are initialized by a trusted third party using algorithm Setup of the corresponding commitment scheme. Also, we use the following notation for lists. We write [] to denote an empty list. For a list $E = [x_1, \dots, x_n]$ and $i \in \{1, \dots, n\}$, by $E[i:]$ we denote the sublist $[x_i, \dots, x_n]$, by $E[:-i]$ we denote the sublist $[x_1, \dots, x_{n-i+1}]$, and by $E[-i]$ we denote the element x_{n-i+1}.

3.2.1 Initialization. At initialization, the client chooses a secret sharing scheme SHARE, an ORAM scheme ORAM, and a database size N. It then initializes the ORAM via $(s, M) \leftarrow \mathsf{ORAM.Setup}(N)$ and allocates a database with M blocks at the evidence service. For each $i \in \{1, \dots, M\}$, the evidence service initializes block i with an empty evidence lists, $E_{\mathcal{ES}, i} = []$. Afterwards, the client picks a set of secret share holders and a reconstruction threshold, and then initializes the secret sharing database with M blocks using protocol SHARE.Setup. The shareholders initialize their databases such that for each $i \in \{1, \dots, M\}$ an empty data object $\mathsf{dat}_i = \perp$ and an empty evidence list $E_i = []$ is stored at block i. We remark that while we use a stash-free ORAM model for the benefit of a more comprehensible description, the construction can be adopted to work with stashed ORAMs in which case the client locally manages the stashed items as usual.

3.2.2 Read and Write. The client reads and writes data blocks using algorithm Access (see Algorithm 2). This algorithm gets as input an operation type $\mathsf{op} \in \{\text{'Write'}, \text{'Read'}\}$, a block identifier $\mathsf{id} \in \{1, \dots, N\}$, optionally data to be written dat', the ORAM state s, a commitment scheme instance CSI, and a reference to a timestamp service $\mathcal{TS}$. It then generates an access pattern, $(P, s) \leftarrow \mathsf{ORAM.GenAP}(s, \mathsf{id})$, and for each $(i_k, j_k) \in P$ does the following: first, it retrieves the new evidence $E_{\mathcal{ES}}$ for block i_k from the evidence service and it retrieves the stored data dat and the old evidence E from the shareholders. Then the new evidence $E_{\mathcal{ES}}$ is added to the shareholder evidence E. If this is a write operation ($\mathsf{op} = \text{'Write'}$) and $\mathsf{ORAM.GetId}(s, i_k) = \mathsf{id}$, then the block data dat is replaced with dat' and a commitment (c, d) to the new data is generated. Since this block is newly written, the existing evidence is discarded and the corresponding evidence is set to $E = [(\text{'Write'}, c, d, \perp)]$, where $\perp$ is a placeholder for the timestamp that will later be retrieved by the evidence service. If this is a read operation ($\mathsf{op} = \text{'Read'}$) and $\mathsf{ORAM.GetId}(s, i_k) = \mathsf{id}$, then the data dat is cached and returned when the access algorithm finishes. Finally, the algorithm refreshes the commitment by creating a new commitment to the block data and the previous commitment. This is necessary so that the evidence service cannot trace how the client rearranges the blocks. The refreshed commitment is stored at the evidence service at the new location j_k. The evidence service then timestamps the new commitment and stores the timestamp together with the commitment (Algorithm 3). Also, the client generates new secret shares of the data dat and the shareholder evidence E and stores them at the shareholders at the new location j_k. As the secret shares are newly generated, they do not correlate with the old shares and their relocation cannot be traced either.

3.2.3 Renew Timestamps. If the security of the currently used timestamp scheme is threatened, the evidence service renews the evidence as follows (Algorithm 4). It picks a new timestamp service $\mathcal{TS}$ that uses a more secure timestamp scheme and then for every $i \in [1, \dots, M]$, it first creates a commitment (c', d') to the last commitment and timestamp stored in $E_{\mathcal{ES}, i}$, then requests a timestamp ts for c', and finally adds $(\text{'ReTs'}, c', d', \mathsf{ts})$ to $E_{\mathcal{ES}, i}$.

3.2.4 Renew Commitments. If the security of the currently used commitment scheme is threatened, the client renews the evidence using Algorithm 5. It starts by selecting a new commitment scheme instance CSI. Then, for each block $i \in [1, \dots, M]$, it does the following. It first retrieves the new evidence $E_{\mathcal{ES}, i}$ from the evidence

Algorithm 2: Access(op, id, dat′, s, CSI, $\mathcal{TS}$), run by the client.

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

Algorithm 3: $\mathcal{ES}$.Write($i, c, \mathcal{TS}$), run by the evidence service.

```
ts ← $\mathcal{TS}$.Stamp($c$);
$E_{\mathcal{ES},i} \leftarrow [(\perp, c, \perp, \text{ts})]$;
```

Algorithm 4: RenewTs(CSI, $\mathcal{TS}$), run by the evidence service.

```
for $i = 1$ to $M$ do
    $(c_i, d_i) \leftarrow$ CSI.Commit($[E_{\mathcal{ES},i}[-1].c, E_{\mathcal{ES},i}[-1].\text{ts}]$);
    $ts_i \leftarrow \mathcal{TS}.\text{Stamp}(c_i)$;
    $E_{\mathcal{ES},i}$ += $[(\text{'ReTs'}, c_i, d_i, \text{ts}_i)]$;
```

service and data block dat_i and old evidence block E_i from the shareholders. It then adds the new evidence $E_{\mathcal{ES},i}$ to the shareholder evidence E_i. Afterwards, it creates a new commitment (c_i, d_i) to the secret data dat_i and the evidence E_i. It then adds ('ReCom', $c_i, d_i, \perp$) to E_i. Finally, it sends c_i to the evidence service and distributes dat_i and E_i to the shareholders.

3.2.5 Share Renewal. In regular time intervals the shareholders renew the stored shares to protect against a mobile adversary (see Section 2.3) by running protocol SHARE.Reshare.

3.2.6 Verification. The verification algorithm (Algorithm 6) uses a trust anchor TA that certifies the validity of public keys for timestamps and commitments. Here, by $\text{VerTs}_{\text{TA}}(\text{dat}, \text{ts}; t_{\text{ver}}) = 1$ we denote that timestamp ts is valid for dat at reference time t_{ver},

Algorithm 5: RenewCom(CSI, $\mathcal{TS}$), run by the client.

```
for $i = 1, \ldots, M$ do
    $E_{\mathcal{ES}} \leftarrow \mathcal{ES}.\text{Read}(i)$; $(\text{dat}, E) \leftarrow \mathcal{SH}.\text{Read}(i)$;
    // retrieve data from location $i$
    $E[-1].\text{ts} \leftarrow E_{\mathcal{ES}}[1].\text{ts}$; $E$ += $E_{\mathcal{ES}}[2:]$; // move evidence from ES to SH
    $(c, d) \leftarrow$ CSI.Commit($[\text{dat}, E]$); // recommit to data and evidence
    $E$ += $[(\text{'ReCom'}, c, d, \perp)]$; // add new evidence
    $\mathcal{ES}.\text{Write}(i, E[-1].c, \mathcal{TS})$;
    $\mathcal{SH}.\text{Write}(i, (\text{dat}, E))$; // store evidence and data at location $i$
```

and by $\text{VerCom}_{\text{TA}}(\text{dat}, c, d; t_{\text{ver}}) = 1$ we denote that d is a valid decommitment from c to dat at time t_{ver}. On input a data object dat, a time t, an evidence block E, and the verification time t_{ver}, the verification algorithm of PROPYLA checks whether E is currently valid evidence for the existence of dat at time t, given that the current time is t_{ver}. In particular, the algorithm checks that the evidence is constructed correctly for dat, that the timestamps and commitments have been valid at their renewal time, and that the first timestamp refers to time t.

Algorithm 6: $\text{VerInt}_{\text{TA}}(\text{dat}, t, E; t_{\text{ver}})$, run by any verifier.

```
/* verifies that dat existed at time $t$ */
Let $E = [(\text{op}_1, c_1, d_1, \text{ts}_1), \ldots, (\text{op}_n, c_n, d_n, \text{ts}_n)]$;
Set $t_{n+1} := t_{\text{ver}}$;
For $i \in [n]$, set $E_i := [(\text{op}_1, c_1, d_1, \text{ts}_1), \ldots, (\text{op}_i, c_i, d_i, \text{ts}_i)]$;
For $i \in [n]$, set $t_i := \text{ts}_i.t$;
For $i \in [n]$, set
  $t_{\text{NRC}(i)} := \min(\{t_j \mid act_j = \text{'ReCom'} \wedge j > i\} \cup \{t_{n+1}\})$;
for $i = n$ to $1$ do
    Assert $\text{VerTs}_{\text{TA}}(c_i, \text{ts}_i; t_{i+1}) = 1$;
    if $\text{op}_i$ = 'Write' and $i = 1$ then
        Assert $\text{VerCom}_{TA}(\text{dat}, c_1, d_1; t_{\text{NRC}(1)}) = 1$;
    else if $\text{op}_i$ = 'Read' or $\text{op}_i$ = 'ReTs' then
        Assert $\text{VerCom}_{\text{TA}}([c_{i-1}, \text{ts}_{i-1}], c_i, d_i; t_{\text{NRC}(i)}) = 1$;
    else if $\text{op}_i$ = 'ReCom' then
        Assert $\text{VerCom}_{\text{TA}}([\text{dat}, E_{i-1}], c_i, d_i; t_{\text{NRC}(i)}) = 1$;
    else
        Fail;
Assert $\text{ts}_1.t = t$;
```

4 SECURITY ANALYSIS

In this section we analyze the security of PROPYLA. Our security analysis requires a model of real time which is used for expressing the scheduling of protection renewal events in our security experiments and for expressing computational bounds on the adversary with respect to real time. We first describe our model of real time.

Then, we show that PROPYLA provides long-term access pattern hiding, long-term confidentiality, and long-term integrity.

4.1 Model of Time

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

We model real time as proposed in [11], i.e., we use a global clock that advances whenever the adversary performs work. Formally, in a security experiment with an adversary $\mathcal{A}$, we assume a global clock Clock with a local state time determining the current time in the experiment. The adversary $\mathcal{A}$ is given the ability to advance time by calling $\mathsf{Clock}(t)$, i.e., at $\mathsf{time} = t$, it may call $\mathsf{Clock}(t')$ and set the time to $\mathsf{time} = t'$, for any t' with $t' > t$. When this happens, all actions scheduled between times t and t' are performed in order and afterwards the control is given back to the adversary. We remark that the adversary also uses up his computational power when it advances time. This is due to the fact that we restrict it to perform only a bounded number of operations per time interval. Our computational model also captures adversaries who increase their computational power over time. For more details on the adversary model we refer the reader to [6].

4.2 Long-Term Access Pattern Hiding

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

Formally, Access-Pattern-Hiding (APH) Security of PROPYLA is defined via game $\mathbf{Exp}^{\mathsf{APH}}_{\mathsf{PROPYLA}}$ (Algorithm 7), where an adversary, $\mathcal{A}$, instructs the client of PROPYLA to read and write database blocks at logical addresses chosen by the adversary. During these data accesses, $\mathcal{A}$ observes the data that is transferred between the client and the evidence service and a subset of less than the threshold number of shareholders. At some point in time, $\mathcal{A}$ gives two different access instructions to the client. The client picks one of them at random and executes it. The goal of the adversary $\mathcal{A}$ is to infer from the observed data stream which of the access instructions has been executed by the client.

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

Definition 4.1 (APH-Security of PROPYLA*).* PROPYLA is information theoretically APH-secure if for any renewal schedule $\mathcal{S}$ and any adversary $\mathcal{A}$:

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

Algorithm 7: $\mathbf{Exp}^{\mathsf{APH}}_{\mathsf{PROPYLA}}(\mathcal{S},\mathcal{A})$

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

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

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

Theorem 4.2. *If* PROPYLA *is instantiated using an information theoretically secure ORAM, information theoretically hiding commitment schemes, and information theoretically secure secret sharing, then it provides information theoretic APH-security.*

Proof. We observe that the queries made by the client and observed by the adversary are either of the form (i, c), when the client instructs the evidence service to store commitment c at database location i, or of the form (i, s), when the client instructs a shareholder to store share s at location i. Furthermore, we observe that when using an information theoretically secure ORAM, there is no statistical correlation between the instructions (op, id, dat) chosen by the adversary and the database locations i sent by the client. There is also no statistical correlation between the data and the commitments c or the secret shares s, as long as the adversary observes less than the threshold number of shareholders. This is (1) because we use information theoretically hiding commitments and information theoretically secure secret sharing and (2) the shares and the commitments are renewed on every access.

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

4.3 Long-Term Confidentiality

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

4.4 Long-Term Integrity

Next, we show that PROPYLA provides long-term integrity protection. Here we consider an adversary that may be running for a very long time but who can only perform a limited amount of work per unit of time (see the adversary model description in Section 4.1).

The goal of the adversary is to prove that a data object existed at a certain point in time when in reality it did not exist. To capture this notion more formally, we need to define what it means that a data object existed. Here, when we say that a data object dat existed at time t with respect to an adversary $\mathcal{A}$, we mean that $\mathcal{A}$ "knew" dat at time t (with high probability). This can be formalized using computational extractors (e.g., [6, 9, 11]). Our security analysis is based on [6, 11], where it is shown that (under certain computational assumptions) extractable commitments and timestamps can be used to argue about the knowledge of an adversary at a given point in time. Based on these results, we here give a slightly less technical security proof for the integrity of PROPYLA.

In our security analysis we use the following notation to describe the knowledge of an adversary $\mathcal{A}$. For a data object dat and a time t, we write dat $\in \mathcal{K}_{\mathcal{A}}[t]$ to denote that $\mathcal{A}$ knew dat at time t. We assume without loss of generality that adversaries do not forget knowledge, that is, for any data object dat and any two points in time t and t', if dat $\in \mathcal{K}_{\mathcal{A}}[t]$ and $t' > t$, then dat $\in \mathcal{K}_{\mathcal{A}}[t']$. We also use the convention that for verification of timestamps and commitments a trust anchor TA is provided by a PKI that certifies the verification keys of the used timestamp and commitment scheme instances and specifies the corresponding instance validity periods.

We state two lemmas that will be useful for proving long-term integrity protection of PROPYLA. These lemmas are derived from results of [6, 11] about extractable timestamps and commitments. The first lemma states that if an adversary $\mathcal{A}$ knows a timestamp ts and a message m at a time t, and ts is valid for m at t, then $\mathcal{A}$ has already known m at time ts.t (with high probability). The second lemma states that if an adversary $\mathcal{A}$ knows a commitment value at a time t, a message m and a decommitment d are known at time $t' > t$, and d is a valid decommitment at time t', then $\mathcal{A}$ has already known the message m at the commitment time t (with high probability).

Lemma 4.3. *For any message m, timestamp* ts, *and time t:*

$$(m, \mathsf{ts}) \in \mathcal{K}_{\mathcal{A}}[t] \wedge \mathsf{VerTs}_{\mathsf{TA}}(m, \mathsf{ts}; t) = 1 \implies m \in \mathcal{K}_{\mathcal{A}}[\mathsf{ts}.t]\ .$$

Lemma 4.4. *For any commitment value c, time t, message m, decommitment value d, and time $t' > t$:*

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

Next, we use Lemma 4.3 and Lemma 4.4 to show that PROPYLA provides long-term integrity protection.

Theorem 4.5. PROPYLA *provides long-term integrity protection, that is, it is infeasible for an adversary to produce evidence E valid for data* dat *and time t without having known* dat *at time t given that the used timestamp and commitment schemes are secure within their usage period.*

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

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

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

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

Statement $St(n)$ obviously holds because (dat, E) is output by the adversary at time t_{n+1} and includes c_n, ts_n, dat, and E_n. Next, we show for $i \in \{n, \ldots, 2\}$, that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ and $St(i)$ implies $St(i-1)$. By the definition of VerInt (Algorithm 6) we observe that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ implies $\mathsf{VerTs}_{\mathsf{TA}}(c_i, \mathsf{ts}_i; t_{i+1}) = 1$, that is, ts_i is valid for commitment c_i at time t_{i+1}. Furthermore, we observe that $St(i)$ implies $(c_i, \mathsf{ts}_i) \in \mathcal{K}_{\mathcal{A}}[t_{i+1}]$, which means that c_i and ts_i were known at time t_{i+1}. We can now apply Lemma 4.3 to obtain that commitment c_i was known at time t_i (i.e., $c_i \in \mathcal{K}_{\mathcal{A}}[t_i]$). Next, we distinguish between the case $\mathsf{op}_i \in \{\text{'Read'}, \text{'ReTs'}\}$ and the case $\mathsf{op}_i \in \{\text{'ReCom'}\}$. We observe that if $\mathsf{op}_i \in \{\text{'Read'}, \text{'ReTs'}\}$, then $\mathsf{VerCom}_{\mathsf{TA}}(c_{i-1} \| \mathsf{ts}_{i-1}, c_i, d_i; t_{\mathrm{NRC}(i)}) = 1$ and by Lemma 4.4 it follows that $(c_{i-1}, \mathsf{ts}_{i-1}) \in \mathcal{K}_{\mathcal{A}}[t_i]$ (i.e., c_{i-1} and ts_{i-1} were known at time t_i). If $\mathsf{op}_i \in \{\text{'ReCom'}\}$, then $\mathsf{VerCom}_{\mathsf{TA}}(\mathsf{dat} \| E_{i-1}, c_i, d_i; t_{\mathrm{NRC}(i)}) = 1$ and it follows that $(\mathsf{dat}, E_{i-1}) \in \mathcal{K}_{\mathcal{A}}[t_{\mathrm{NRC}(i-1)}]$. Together, we obtain that if $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ and $St(i)$, then $St(i-1)$ holds. By induction over i it follows that $St(1)$ holds.

Finally, we observe that $St(1)$ by Lemma 4.3 and Lemma 4.4 implies dat $\in \mathcal{K}_{\mathcal{A}}[t_1]$. Also, $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$ implies $t_1 = t$, and thus we obtain dat $\in \mathcal{K}_{\mathcal{A}}[t]$. We conclude that if the adversary presents (dat, t, E) such that $\mathsf{VerInt}_{\mathsf{TA}}(\mathsf{dat}, t, E) = 1$, then it must have known dat at time t. □

5 INSTANTIATION AND EVALUATION

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

5.1 Scheme Instantiation

We instantiate PROPYLA using Path-ORAM [29], Shamir Secret Sharing [25], Halevi-Micali Commitments [16], RSA Signatures [24], and XMSS Signatures [5]. The implementation has been done in Java and we use the following parameters. For Path-ORAM we use a bucket size of 5. Our implementation of Halevi-Micali Commitments uses the hash functions SHA-224, SHA-256, and SHA-384 [22]. For RSA, we use the standard JDK implementation and use SHA-224 with RSA-2048. For XMSS, we use the implementation from the Bouncy Castle Library [30], which supports the hash functions SHA-256 and SHA-512, and we use a tree height of 10.

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

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

This instantiation has the required security properties. Path ORAM instantiated with an information theoretically secure random number generator (e.g., a quantum-based random number generator [27]) provides information theoretic security [28]. Shamir Secret Sharing and Halevi-Micali Commitments are information theoretic hiding. Therefore, by Theorem 4.2, the described instantiation of PROPYLA is information theoretic access pattern hiding. Information theoretic confidentiality also follows from Theorem 4.2 (see Section 4.3). Finally, by Theorem 4.5, long-term integrity is achieved as long as the used commitment and signature scheme instances are secure within their usage period. In Table 1 we list the commitment and signature scheme instances that we use together with their usage periods, which are based on the predictions by Lenstra and Verheul [2, 19, 20]. Also, we assume that quantum computers will become a considerable threat by 2040 and therefore transition from RSA Signatures (which are known vulnerable to quantum computers) to XMSS Signatures (which are expected secure against quantum computer attacks) after 2030.

5.2 Evaluation

5.2.1 Scenario. In the following we examine a use case where a client stores N data objects of size L for a time period of 100 years using PROPYLA. To maintain long-term integrity protection, timestamps are renewed every 2 years, which corresponds to the typical lifetime of a public key certificate, and commitments are renewed every 10 years, which corresponds to the longer lifetime of commitments because they do not involve any secret parameters that could compromise security. Cryptographic schemes in PROPYLA are instantiated as described in Section 5.1. Computation times of PROPYLA are estimated by counting the number of the involved primitive operations (e.g., commitments, timestamps) and multiplying these numbers by the running time of the respective operations, which were measured on a computer with a 2.9 GHz Intel Core i5 CPU. We remark that our performance analysis does not consider network latency.

5.2.2 Results. We now present the results of our performance evaluation focusing on the costs of enabling long-term integrity protection. In particular, we measure the costs for generating, communicating, and storing timestamps and commitments.

In Figure 2, we show the space required for storing commitments, decommitments, and timestamps. We observe that the storage space required per shareholder increases with each commitment renewal and that the magnitude of the increase depends on the commitment and signature scheme parameters. With respect to the evidence service we observe that the required storage space increases with each

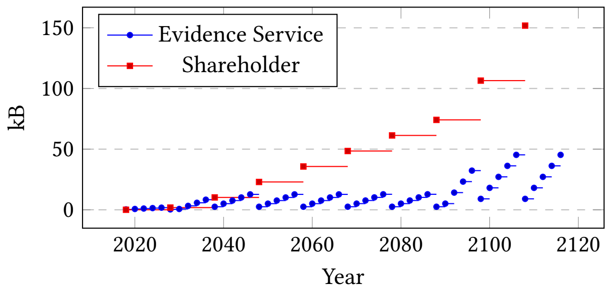

Figure 2: Storage costs for timestamps and commitments per server block (independent of block size L).

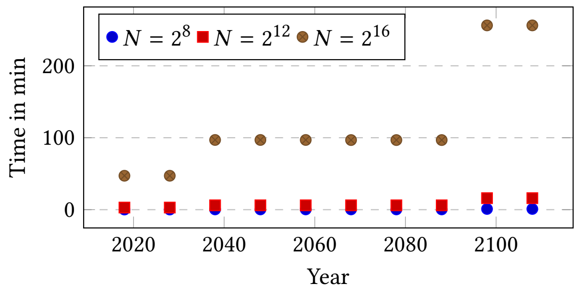

Figure 3: Computation time for renewing timestamps and commitments when storing $N \in \{2^8, 2^{12}, 2^{16}\}$ data objects of size $L = 100$kB.

timestamp renewal and is reset with each commitment renewal, when commitments and timestamps are moved to the shareholders. The storage space required for integrity protection is independent of the data block size because commitment and signature sizes are independent of the data size. After 100 years, about 50 kB of storage per block is required at the evidence service and about 150 kB at a shareholder.

In Figure 3, we show the computation time required for renewing timestamps and commitments for database size $N \in \{2^8, 2^{12}, 2^{16}\}$ and data block size $L = 100$kB. We observe that the renewal time scales linearly with the number of data objects and depends on the commitment and signature scheme parameters. Renewing the protection after 100 years takes about 1 min for $N = 2^8$ data objects and 256 min for $N = 2^{16}$ data objects.

In Figure 4, we show the computation time required for generating commitments and timestamps during one database access made by the client for database size $N \in \{2^8, 2^{12}, 2^{16}\}$ and block size $L = 100$kB. We observe that the computation cost scales logarithmically with the database size N, which is because Path-ORAM requires $\log(N)$ server accesses per client access. Furthermore, we observe that the computation cost also significantly depends on the commitment and timestamp scheme parameters. Concretely, the computation cost per access ranges from about 0.3 s in 2018 for $N = 2^8$ and about 3.7 s in 2118 for $N = 2^{16}$.

In Figure 5 we show the communication cost per database access for ORAM-SS, which is a combination of ORAM and Secret Sharing without long-term integrity protection, and PROPYLA. For $N = 2^{12}$

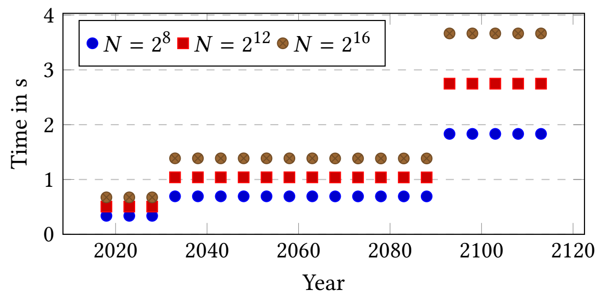

Figure 4: Computation time for generating timestamps and commitments per database access for database size $N \in \{2^8, 2^{12}, 2^{16}\}$ and block size $L = 100$kB.

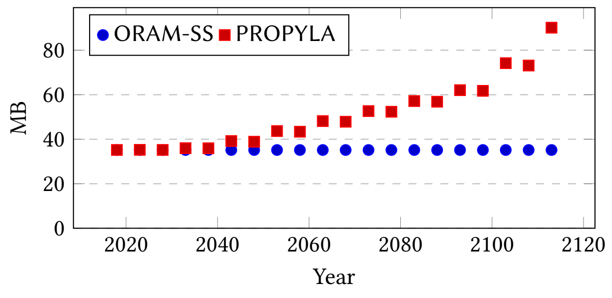

Figure 5: Communication cost per database access for database size $N = 2^{12}$ and block size $L = 100$kB.

and $L = 100$kB we observe that the communication overhead for adding long-term integrity protection ranges between 0.06% in year 2018 and 259% in 2118. The reason is that over time evidence is accumulated which needs to be communicated on every access. The communication cost per database access scales logarithmically with the database size N and linearly with the number of protection renewals. It also depends on the commitment and timestamp scheme parameters.

6 CONCLUSIONS

We presented the first long-term secure storage architecture that combines long-term integrity, long-term confidentiality, and long-term access pattern hiding protection. Overall, our performance measurements show that these protection goals can be achieved with manageable communication, computation, and storage costs. The storage and communication cost per block and per storage server is independent of the block size L. The computation and communication costs are logarithmic in the database size N and also depend on the amount of accumulated evidence. The protection renewal time is linear in N.

We envision that the performance of PROPYLA can further be improved by using Merkle Hash Trees (MHT) [21] in order to reduce the number of timestamps as follows. During timestamp and commitment renewal, instead of timestamping each data block separately, one could first create a MHT for the entire database and then timestamp only the root of that tree. While asymptotically this increases the storage and computation complexity from $O(N)$ to $O(N \log(N))$, we expect that concrete computation costs decrease because typical hash functions are much faster to evaluate than the signing algorithms used for timestamping. Furhtermore, hash values are also much smaller in size compared to signatures. We leave the exact description, implementation, and evaluation of this approach for future work.

ACKNOWLEDGMENTS

This work has been co-funded by the DFG as part of projects S5 and S6 within the CRC 1119 CROSSING.

VERSION HISTORY

Major changes compared to proceedings version:

- An error in the description of the commitment renewal procedure in Algorithm 2 has been fixed.
- The analysis of the communication cost in Section 5 has been revised.

REFERENCES

[1] Dave Bayer, Stuart Haber, and W. Scott Stornetta. 1993. Improving the Efficiency and Reliability of Digital Time-Stamping. In *Sequences II: Methods in Communication, Security, and Computer Science*, Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro (Eds.). Springer New York, New York, NY, 329–334.

[2] BlueKrypt. 2018. https://www.keylength.com. Website. (2018).

[3] Johannes Braun, Johannes Buchmann, Denise Demirel, Matthias Geihs, Mikio Fujiwara, Shiho Moriai, Masahide Sasaki, and Atsushi Waseda. 2017. 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 (ASIA CCS '17)*. ACM, New York, NY, USA, 461–468.

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

[5] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. 2011. XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29 – December 2, 2011. Proceedings*, Bo-Yin Yang (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 117–129.

[6] Ahto Buldas, Matthias Geihs, and Johannes Buchmann. 2017. Long-Term Secure Commitments via Extractable-Binding Commitments. In *Information Security and Privacy: 22nd Australasian Conference, ACISP 2017, Auckland, New Zealand, July 3–5, 2017, Proceedings, Part I*, Josef Pieprzyk and Suriadi Suriadi (Eds.). Springer International Publishing, Cham, 65–81.

[7] Ahto Buldas, Matthias Geihs, and Johannes Buchmann. 2017. Long-Term Secure Time-Stamping Using Preimage-Aware Hash Functions. In *Provable Security*, Tatsuaki Okamoto, Yong Yu, Man Ho Au, and Yannan Li (Eds.). Springer International Publishing, Cham, 251–260.

[8] Ahto Buldas and Risto Laanoja. 2013. Security Proofs for Hash Tree Time-Stamping Using Hash Functions with Small Output Size. In *Information Security and Privacy: 18th Australasian Conference, ACISP 2013, Brisbane, Australia, July 1-3, 2013. Proceedings*, Colin Boyd and Leonie Simpson (Eds.). Springer, Berlin, Heidelberg, 235–250.

[9] Ahto Buldas and Sven Laur. 2007. Knowledge-Binding Commitments with Applications in Time-Stamping. In *Public Key Cryptography - PKC 2007, 10th International Conference on Practice and Theory in Public-Key Cryptography, Beijing, China, April 16-20, 2007, Proceedings*. Springer, Berlin, Heidelberg, 150–165.

[10] Ahto Buldas and Märt Saarepera. 2004. On Provably Secure Time-Stamping Schemes. In *Advances in Cryptology - ASIACRYPT 2004, 10th International Conference on the Theory and Application of Cryptology and Information Security, Jeju Island, Korea, December 5-9, 2004, Proceedings*. Springer, Berlin, Heidelberg, 500–514.

[11] Matthias Geihs, Denise Demirel, and Johannes Buchmann. 2016. A Security Analysis of Techniques for Long-Term Integrity Protection. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*. IEEE, Piscataway, NJ, USA, 449–456.

[12] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. 2002. Quantum cryptography. *Rev. Mod. Phys.* 74 (Mar 2002), 145–195. Issue 1.

[13] O. Goldreich. 1987. Towards a Theory of Software Protection and Simulation by Oblivious RAMs. In *Proceedings of the Nineteenth Annual ACM Symposium on Theory of Computing (STOC '87)*. ACM, New York, NY, USA, 182–194.

[14] Oded Goldreich. 2001. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, Cambridge, UK.

[15] Stuart Haber and W. Scott Stornetta. 1990. How to Time-Stamp a Digital Document. In *Advances in Cryptology - CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990, Proceedings*, Alfred J. Menezes and Scott A. Vanstone (Eds.). Springer, Berlin, Heidelberg, 437–455.

[16] Shai Halevi and Silvio Micali. 1996. Practical and Provably-Secure Commitment Schemes from Collision-Free Hashing. In *Advances in Cryptology — CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, Neal Koblitz (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 201–215.

[17] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. 1995. Proactive Secret Sharing Or: How to Cope With Perpetual Leakage. In *Advances in Cryptology — CRYPTO' 95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings*, Don Coppersmith (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 339–352.

[18] Ari Juels and Thomas Ristenpart. 2014. Honey Encryption: Security Beyond the Brute-Force Bound. In *Advances in Cryptology – EUROCRYPT 2014*, Phong Q. Nguyen and Elisabeth Oswald (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 293–310.

[19] Arjen K Lenstra. 2004. Key Lengths. In *Handbook of Information Security, Volume 2, Information Warfare, Social, Legal, and International Issues and Security Foundations*. Wiley, Indianapolis, IN, USA.

[20] Arjen K. Lenstra and Eric R. Verheul. 2001. Selecting Cryptographic Key Sizes. *J. Cryptology* 14, 4 (2001), 255–293.

[21] Ralph C. Merkle. 1990. A Certified Digital Signature. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, Gilles Brassard (Ed.). Springer New York, New York, NY, 218–238.

[22] National Institute of Standards and Technology. 2015. FIPS PUB 180-4: Secure Hash Standard (SHS). Federal information processing standards publication. (August 2015).

[23] NIST. 2001. FIPS 197: Announcing the advanced encryption standard (AES). (2001).

[24] R. L. Rivest, A. Shamir, and L. Adleman. 1978. A Method for Obtaining Digital Signatures and Public-key Cryptosystems. *Commun. ACM* 21, 2 (Feb. 1978), 120–126.

[25] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613.

[26] C. E. Shannon. 1949. Communication Theory of Secrecy Systems*. *Bell System Technical Journal* 28, 4 (1949), 656–715.

[27] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. 2000. Optical quantum random number generator. *Journal of Modern Optics* 47, 4 (2000), 595–598. arXiv:http://dx.doi.org/10.1080/09500340008233380

[28] Emil Stefanov and Elaine Shi. 2012. Path O-RAM: An Extremely Simple Oblivious RAM Protocol. (2012).

[29] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: an extremely simple oblivious RAM protocol. In *2013 ACM SIGSAC Conference on Computer and Communications Security, CCS'13, Berlin, Germany, November 4-8, 2013*, Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung (Eds.). ACM, New York, NY, USA, 299–310.

[30] The Legion of the Bouncy Castle. 2018. https://www.bouncycastle.org/. (2018).

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