Maliciously Secure Multi-Client ORAM*

Matteo Maffei[1] Giulio Malavolta[2] Manuel Reinert[3] Dominique Schröder[2]
[1] TU Wien, matteo.maffei@tuwien.ac.at
[2] Friedrich-Alexander Universität Erlangen-Nürnberg,
giulio.malavolta@fau.de, dominique.schroder@fau.de
[3] CISPA, Saarland University, reinert@cs.uni-saarland.de

April 13, 2017

Abstract

Oblivious RAM (ORAM) has emerged as an enabling technology to secure cloud-based storage services. The goal of this cryptographic primitive is to conceal not only the data but also the access patterns from the server. While the early constructions focused on a single client scenario, a few recent works have focused on a setting where multiple clients may access the same data, which is crucial to support data sharing applications. All these works, however, either do not consider malicious clients or they significantly constrain the definition of obliviousness and the system's practicality. It is thus an open question whether a natural definition of obliviousness can be enforced in a malicious multi-client setting and, if so, what the communication and computational lower bounds are.

In this work, we formalize the notion of maliciously secure multi-client ORAM, we prove that the server-side computational complexity of any secure realization has to be $\Omega(n)$, and we present a cryptographic instantiation of this primitive based on private information retrieval techniques, which achieves an $O(\sqrt{N})$ communication complexity. We further devise an efficient access control mechanism, built upon a novel and generally applicable realization of plaintext equivalence proofs for ciphertext vectors. Finally, we demonstrate how our lower bound can be bypassed by leveraging a trusted proxy, obtaining logarithmic communication and server-side computational complexity. We implemented our scheme and conducted an experimental evaluation, demonstrating the feasibility of our approach.

1 Introduction

Oblivious RAM. Cloud storage has rapidly become a central component in the digital society, providing a seamless technology to save large amounts of data, to synchronize them across multiple devices, and to *share* them with other parties. Popular data-sharing, cloud-based applications are e.g., personal health record management systems (PHRs), like those employed in Austria [25] and Estonia [23], collaborative platforms (e.g., Google Docs), and credit score systems (e.g., Experian, Equifax, and TransUnion in US) are just a few popular data-sharing, cloud-based applications taking advantage of such features. A stringent and well-understood security requirement is *access control*: read and write access should be granted only to authorized clients.

While access control protects user's data from other clients, encryption on the server's side is needed to obtain privacy guarantees against cloud administrators. Encryption is, however, not enough: as shown in the literature [34, 45], the capability to observe which data are accessed by which users allows the cloud administrator to learn sensitive information: for instance, it has been shown that the access patterns to a DNA sequence allow for determining the patient's disease. The property of hiding data accesses is called *obliviousness* and the corresponding cryptographic construction *Oblivious RAM* (O-RAM): while the first constructions were highly inefficient [26], recent groundbreaking research paved the way for a tremendous efficiency boost, exploiting ingenious tree based constructions [45, 51, 9, 2, 29, 16, 54, 38, 3, 52, 17], server side computations [41, 32], and trusted hardware [37, 55, 6, 49, 33].

*An extended abstract of this work will appear at ACNS'17 [40]

Work	MC	MD	PI	CS	Pr	AC	P	S comp.	C comp.	Comm.
Franz *et al.* [24]	✓	✗	✓	✗	✗	✓	✗	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n})$
GORAM [39]	✗	✗	✗	✓	✗	✓	✗	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
PIR-MCORAM (this work)	✓	✓	✗	✓	✗	✓	✗	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$
BCP-OPRAM [8]	✗	✗	✓	✓	✗	✗	✓	$\Omega(\log^3(n))$	$\Omega(\log^3(n))$	$\Omega(\log^3(n))$
CLT-OPRAM [11]	✗	✗	✓	✓	✗	✗	✓	$O(\log^2(n))$	$O(\log^2(n))$	$O(\log^2(n))$
PrivateFS [56]	✗	✗	✗	✓	✗	✗	✓	$O(\log^2(n))$	$O(1)$	$O(\log^2(n))$
Shroud [37]	✗	✗	✗	✗	✓	✗	✓	$O(\log^2(n))$	$O(1)$	$O(\log^2(n))$
TaoStore [49]	✗	✗	✗	✗	✓	✗	✓	$O(\log(n))$	$O(1)$	$O(\log(n))$
TAO-MCORAM (this work)	✓	✓	✗	✗	✓	✓	✓	$O(\log(n))$	$O(1)$	$O(\log(n))$

Table 1: Comparison of the related work supporting multiple clients to our constructions. The abbreviations mean: **MC**: oblivious against malicious clients, **MD**: supports multiple data owners sharing their data in one ORAM, **PI**: requires the periodic interaction with the data owner, **CS**: requires synchronization among clients, **AC**: access control, **Pr**: trusted proxy, **P**: parallel accesses, **S comp.**: server computation complexity, **C comp.**: client communication complexity, **Comm.**: communication complexity.

Except for a few recent noticeable exceptions, discussed below, a fundamental limitation of all these constructions is that they target a single-client architecture, where the data owner is the only party allowed to read outsourced data, which does not make them suitable for data sharing services. The fundamental challenge to solve is to enforce access control and obliviousness *simultaneously*. These properties are seemingly contradictory: can the server check the correctness of data accesses with respect to the access control policy at all, if it is not allowed to learn anything about them?

Multi-client ORAM. A few recent constructions gave positive answers to this question, devising ORAM constructions in the multi-client setting, which specifically allow the data owner to share data with other clients while imposing fine-grained access control policies. Although, at a first glance, these constructions share the same high-level goal, they actually differ in a number of important aspects. Therefore we find it interesting to draw a systematic comparison among these approaches (cf. Table 1). First of all, obliviousness is normally defined against the server, but in a multi-client setting it is important to consider it against the clients too (**MC**), since they might be curious or, even worse, collude with the server. This latter aspect is important, since depending on the application, the cloud administrator might create fake clients or just have common interests with one of the legitimate clients. Some constructions allow multiple data owners to operate on the same ORAM (**MD**), while others require them to use disjoint ORAMs: the latter are much less efficient, since if the client does not want to reveal the owner of the accessed entry (e.g., to protect her anonymity, think for instance of the doctor accessing the patient's record), then the client has to perform a fake access to each other ORAM, thereby introducing a multiplicative factor of $O(m)$, where m is the number of data owners. Some constructions require the data owner to periodically access the dataset in order to validate previous accesses (**PI**), some others rely on server-side client synchronization, which can be achieved for instance by a shared log on the server, a gossiping protocol among clients, etc. (**CS**), while others assume a trusted proxy (**Pr**). Among these, gossiping is the mildest assumption since it can be realized directly on the server side as described by [36]. Another aspect to consider is the possibility for the data owner to specify fine-grained access control mechanisms (**AC**). Finally, some constructions enable concurrent accesses to the ORAM (**P**). The final three columns compare the asymptotic complexity of server-side and client-side computations as well as communication.

Franz *et al.* pioneered the line of work on multi-client ORAM, introducing the concept of delegated ORAM [24]. The idea of this construction, based on simple symmetric cryptography, is to let clients commit their changes to the server and to let the data owner periodically validate them according to the access control policy, finally transferring the valid entries into the actual database. Assuming periodic accesses from the data owner, however, constrains the applicability of this technique. Furthermore, this construction does not support multiple data owners. Finally, it guarantees the obliviousness of access patterns with respect to the server as well as malicious clients, excluding however the accesses on data readable by the adversary. While excluding write operations is necessary (an adversary can clearly notice that the data has changed), excluding read operations is in principle not necessary and limits the

applicability of the obliviousness definition: for instance, we would like to hide the fact that an oncologist accessed the PHR of a certain patient even from parties with read access to the PHR (e.g., the pharmacy, which can read the prescription but not the diagnosis).

More recently, Maffei *et al.* [39] proposed the notion of group ORAM, in which the server performs access control by verifying client-provided zero-knowledge proofs: this approach enables direct client accesses without any interaction with the data owner and more generic access control policies. The scheme relies on a gossiping protocol, but malicious clients are considered only in the context of access control and, indeed, obliviousness does not hold against them.

Another line of work, summarized in the lower part of Table 1, focuses on the parallelization of client accesses, which is crucial to scale to a large number of clients, while retaining obliviousness guarantees. Most of them [37, 55, 6, 49] assume a trusted proxy performing accesses on behalf of users, with TaoStore [49] being the most efficient and secure among them. These constructions do not formally consider obliviousness against malicious clients nor access control, although a contribution of this work is to prove that a simple variant of TaoStore [49] guarantees both. Finally, instead of a trusted proxy, BCP-OPRAM [8] and CLT-OPRAM [11] rely on a gossiping protocol while PrivateFS [56] assumes a client-maintained log on the server-side, but they do not achieve obliviousness against malicious clients nor access control. Moreover, PrivateFS guarantees concurrent client accesses only if the underlying ORAM already does so.

To summarize, the progress in the field does not answer a few foundational questions, which touch the core of the application of ORAM technologies in cloud-based data-sharing applications. First, is it possible at all to enforce the obliviousness of data accesses without constraining the security definition or placing severe system assumptions? If the answer is positive, it would be interesting to know at what computational cost.

Our contributions. This work answers the questions above, providing a foundational framework for multi-client ORAM. In particular,

- We give for the first time a formal definition of *obliviousness against malicious clients in the multi-client setting*. Intuitively, none should be able to determine which entry is read by which client. However, write operations are oblivious only with respect to the server and to those clients who cannot read the modified entry, since clients with read access can obviously notice that the entry has changed.
- We establish an insightful *computational lower bound*: in a multi-client setting where clients have *direct access* to the database, the number of operations *on the server side* has to be linear in the database size. Intuitively, the reason is that if a client does not want to access all entries in a read operation, then it must know where the required entry is located in the database. Since malicious clients can share this information with the server, the server can determine for each read operation performed by an honest client, which among the entries the adversary has access to might be the subject of the read, and which certainly not.
- We present PIR-MCORAM, the first *cryptographic construction* that ensures the obliviousness of data accesses as well as access control in a malicious multi-client setting. Our construction relies on Private Information Retrieval (PIR) [12] to achieve obliviousness and uses new accumulation technique based on an oblivious gossiping protocol to reduce the communication bandwidth in an amortized fashion. Moreover, it combines public-key cryptography and zero-knowledge proofs for access control.
- We present a novel technique based on universal pair-wise hash functions [10] in order to speed up the efficiency of Plaintext Equivalence Proofs, a computationally demanding cryptographic building block of PIR-MCORAM. This construction is generally applicable and we show that it improves solutions recently adopted in the multi-client ORAM literature [39] by one order of magnitude.
- To bypass the aforementioned lower bound, we consider the recently proposed *proxy-based Setting* [56, 37, 55, 6, 49], which assumes the presence of a trusted party mediating the accesses between clients and server. We prove, in particular, that a simple variant of TaoStore [49] guarantees obliviousness in the malicious setting as well as access control.
- We implement PIR-MCORAM and conduce an *experimental evaluation* of our schemes. PIR-MCORAM constitutes a practical solution for databases of modest size: for instance, DNA encoded in VCF files requires approximately 125MB [47]. Thus, an extended personal health record fits without problems in a 256MB database, for which a read or write operation in PIR-MCORAM takes

approximately 14 seconds amortized. TaoStore offers much better performance as well as support for parallel accesses, but it assumes a trusted proxy.

2 A Lower Bound for Maliciously Secure Multi-Client ORAM

In this section, we study how much computational effort is necessary to securely realize ORAM in the malicious multi-client setting. Our result shows that *any* construction, regardless of the underlying computational assumptions, must access the entire memory (up to a constant factor) in every operation. Our lower bound can be seen as a generalization of the result on history independence of Roche et al. [48], in the sense that they consider a "catastrophic attack" where the complete state of the client is leaked to the adversary, whereas we allow only the corruption of a certain subset of clients. Note that, while the bound in [48] concerns the *communication* complexity, our result only bounds the *computation* complexity on the server side.

Before stating our lower bound, we formalize the notion of Multi-Client ORAM in the malicious setting. We follow the definitional framework introduced by Maffei *et al.* [39], refining the obliviousness definition in order to consider malicious clients possibly colluding with the server.

2.1 Multi-Client Oblivious RAM

In a Multi-Client ORAM scheme the parties consist of the data owner $\mathcal{O}$, several clients $\mathcal{C}_1, \ldots, \mathcal{C}_k$, and the server $\mathcal{S}$. The data owner outsources its database $\mathcal{DB}$ to $\mathcal{S}$ while granting access to the clients $\mathcal{C}_1, \ldots, \mathcal{C}_k$ in a selective manner. This is expressed by an access control matrix ACM which has an entry $\mathsf{ACM}(i, \mathsf{idx})$ for every client $\mathcal{C}_i$ and every entry idx in the database, characterizing which access right $\mathcal{C}_i$ has for entry idx: either no access ($\perp$), read-only access (R), or read-write access (RW). We treat ACM as a global variable for the data owner so as to ease the presentation. Moreover, ACM is only accessible to the data owner and not to any client. We write $o \leftarrow A(\ldots)$ to denote that algorithm A on some input generates output o. Likewise, we write $\langle o_{\mathcal{C}}, o_{\mathcal{S}} \rangle \leftarrow \langle A(\ldots), \mathcal{S}_A(\ldots) \rangle$ to denote that the protocol A executed between the client and the server yields client output $o_{\mathcal{C}}$ and server output $o_{\mathcal{S}}$.

Definition 1 (Multi-Client ORAM [39])**.** *A* Multi-Client ORAM *scheme* Θ *is composed of the following (interactive)* PPT *algorithms:*

$(cap_{\mathcal{O}}, \mathcal{DB}) \leftarrow \mathsf{gen}(1^{\lambda}, n)$. *The generation algorithm initializes a database* $\mathcal{DB}$ *of size* n *and an empty access control matrix* ACM. *Finally, the algorithm returns the data owner's capability* $cap_{\mathcal{O}}$.

$cap_i \leftarrow \mathsf{addCl}(cap_{\mathcal{O}}, i)$. *The input of the add client algorithm is the data owner's capability* $cap_{\mathcal{O}}$ *and a client identifier* i. *It appends a row corresponding to* i *in* ACM *such that for all* $j \in \{1, \ldots, n\}$: $\mathsf{ACM}(i, j) = \perp$. *The algorithm outputs the capability for client* $\mathcal{C}_i$.

$\langle \perp, \mathcal{DB}' \rangle \leftarrow \langle \mathsf{addE}(cap_{\mathcal{O}}, \mathsf{idx}, \mathsf{data}), \mathcal{S}_{\mathsf{addE}}(\mathcal{DB}) \rangle$. *The add entry algorithm takes as input the data owner's capability* $cap_{\mathcal{O}}$ *, an index* idx*, and a data* data *in interaction with* $\mathcal{S}$ *that takes* $\mathcal{DB}$ *as input. It appends a column corresponding to* idx *in* ACM *such that for all* $i \in \{1, \ldots, k\}$: $\mathsf{ACM}(i, \mathsf{idx}) = \perp$, *writes* data *at position* idx *in* $\mathcal{DB}$*, and outputs the modified database* $\mathcal{DB}'$ *on* $\mathcal{S}$.

$\langle \perp, \mathcal{DB}' \rangle \leftarrow \langle \mathsf{chMode}(cap_{\mathcal{O}}, \mathsf{idx}, i, p), \mathcal{S}_{\mathsf{chMode}}(\mathcal{DB}) \rangle$. *The change mode algorithmtakes as input the data owner's capability* $cap_{\mathcal{O}}$*, some index* idx*, a client identifier* i*, and a permission* $p \in \{\mathsf{R}, \mathsf{RW}, \perp\}$ *in interaction with* $\mathcal{S}$ *that takes* $\mathcal{DB}$ *as input. It updates the entry* $\mathsf{ACM}(i, \mathsf{idx})$ *to* p *and returns the modified database* $\mathcal{DB}'$ *on* $\mathcal{S}$.

$\langle \mathsf{data}, \perp \rangle \leftarrow \langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB}) \rangle$. *The read algorithm takes as input an index* idx *and a client capability* cap_i *on the client side and the database* $\mathcal{DB}$ *on* $\mathcal{S}$ *and returns a data* data *on the client and generates no output on the server.*

$\langle \mathsf{data}', \mathcal{DB}' \rangle \leftarrow \langle \mathsf{write}(\mathsf{idx}, cap_i, \mathsf{data}), \mathcal{S}_{\mathsf{write}}(\mathcal{DB}) \rangle$. *The write algorithm takes as input an index* idx*, a client capability* cap_i*, and a data* data *on the client side and the database* $\mathcal{DB}$ *on* $\mathcal{S}$. *Let* data' *be the data stored at* idx *in* $\mathcal{DB}$. *The protocol modifies* $\mathcal{DB}$ *at index* idx *to* data. *Finally, it returns* data' *on the client side as well as the modified database* $\mathcal{DB}'$ *on* $\mathcal{S}$.

Attacker model. The data owner is assumed to be trusted, since she is interested to protect her data. We allow the server to be fully compromised and to corrupt an arbitrary subset of clients. As explained below, this attacker model is relaxed when it comes to the integrity of outsourced data, which can only

be achieved by assuming an honest-but-curious server (while still allowing for client compromise), as discussed below.

SECURITY. A Multi-Client ORAM has four fundamental security properties. The first three concern access control and are intuitively described below.

Secrecy: only users with at least read permissions on an entry can learn its content.

Integrity: only users with write permissions on an entry can change its content.

Tamper Resistance: only users with write permissions on an entry can change its content in a way that the updated entry is considered valid by honest clients.

The difference between integrity and tamper-resistance is that integrity prevents unauthorized changes and thus requires an honest-but-curious server to perform access control, while tamper resistance is a weaker property that allows clients to detect unauthorized changes a-posteriori and thus can in principle be achieved even if the server is malicious.

OBLIVIOUSNESS AGAINST MALICIOUS CLIENTS. Intuitively, a Multi-Client ORAM is secure if the server and an arbitrary subset of clients cannot get any information about the access patterns of honest clients, other than what is trivially leaked by the entries that the corrupted clients have read access to. The original obliviousness definition [39] does not allow the server to corrupt honest clients: here we extend it to handle static corruption of the clients and, in order to avoid trivial attacks, we restrict the queries of the adversary to the write oracle to indices that the set of corrupted clients cannot read.

Definition 2 (Obliviousness against Malicious Clients)**.** *A Multi-Client ORAM* Θ *is* secure against malicious clients*, if for all* PPT *adversaries* $\mathcal{A}$ *the success probability of* $\mathcal{A}$ *in the following experiment is negligibly close to* $1/2$*.*

1. $\mathcal{A}$ *commits to a set of client identifiers* ID*.*
2. *The challenger samples* $b \in \{0,1\}$*, executes* $(\mathsf{ACM}, \mathcal{DB}) \leftarrow \mathsf{gen}(1^\lambda, n)$ *and forwards* $\mathcal{DB}$ *to* $\mathcal{A}$ *and hands over the capabilities of all the clients* $\in \mathsf{ID}$ *to* $\mathcal{A}$*.*
3. *The adversary has access to the following interfaces that he can query adaptively and in any order.*
   $\mathsf{addCl}_{cap_{\mathcal{O}}}(i)$*: The challenger adds an empty row entry to* ACM *corresponding to* i*.*
   $\mathsf{addE}_{cap_{\mathcal{O}}}(\mathsf{idx}, \mathsf{data})$*: The challenger runs* $\langle \mathsf{addE}(\mathsf{idx}, \mathsf{data}, cap_{\mathcal{O}}), \mathcal{A} \rangle$ *in interaction with* $\mathcal{A}$*.*
   $\mathsf{chMode}_{cap_{\mathcal{O}}}(\mathsf{idx}, i, \{\mathsf{R}, \mathsf{RW}, \perp\})$*: The challenger runs* $\langle \mathsf{chMode}(cap_{\mathcal{O}}, \mathsf{idx}, i, \{\mathsf{R}, \mathsf{RW}, \perp\}), \mathcal{A} \rangle$ *in interaction with* $\mathcal{A}$*.*
   $\mathsf{read}(\mathsf{idx}, i)$*: The challenger runs* $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{A} \rangle$ *in interaction with* $\mathcal{A}$*.*
   $\mathsf{write}(\mathsf{idx}, i, \mathsf{data})$*: The challenger runs* $\langle \mathsf{write}(\mathsf{idx}, cap_i, \mathsf{data}), \mathcal{A} \rangle$ *in interaction with* $\mathcal{A}$*.*
   $\mathsf{query}((\mathsf{op}_0, \mathsf{op}_1), (\mathsf{idx}_0, \mathsf{idx}_1), (i_0, i_1), (\mathsf{data}_0, \mathsf{data}_1))$*: The challenger checks in case that* $\mathsf{op}_0 = \mathsf{write}$ *or* $\mathsf{op}_1 = \mathsf{write}$ *if there is an* $i \in \mathsf{ID}$ *such that* $\mathsf{ACM}(i, \mathsf{idx}_0) \neq \perp$ *and* $\mathsf{ACM}(i, \mathsf{idx}_1) \neq \perp$*, if this is the case the challenger aborts. Otherwise it executes* $\langle \mathsf{read}(\mathsf{idx}_b, cap_{i_b}), \mathcal{A} \rangle$ *(or* $\langle \mathsf{write}(\mathsf{idx}_b, cap_{i_b}, \mathsf{data}_b), \mathcal{A} \rangle$*, depending on the operation) in interaction with* $\mathcal{A}$*. In case* $\mathsf{op}_0 = \mathsf{write}$ *or* $\mathsf{op}_1 = \mathsf{write}$*, from this moment on, the queries of* $\mathcal{A}$ *to the interface* chMode *on any* $i \in \mathsf{ID}$ *and* idx_0 *or* idx_1 *are forbidden.*
4. $\mathcal{A}$ *outputs a bit* b'*, the challenger returns 1 if* $b' = b$*.*

2.2 Formal Result

In the following we state a formal lower bound on the computational complexity of any ORAM secure against malicious clients. We denote by physical addresses of a database the memory addresses associated with each storage cell of the memory. Intuitively, the lower bound says that the server has to access each entry of the dataset for any read and write operation.

Theorem 1. *Let* n *be the number of entries in the database and* Θ *be a multi-client ORAM scheme. If* Θ *accesses on average* $o(n)$ *physical addresses for each read and write operation (over the random coins of the read or write operation, respectively),* Θ *is not secure against malicious clients (see Definition 2).*

We formally prove this theorem in Appendix B.

2.3 Discussion

Given the lower bound established in the previous section, we know that any multi-client ORAM scheme that is secure against malicious clients *must* read and write a fixed constant fraction of the database

on every access. However, the bound does not impose any restriction on the required communication bandwidth. In fact, it does not exclude constructions with sublinear communication complexity, where the server performs a significant amount of computation. In particular, the aforementioned lower bound calls for the deployment of private information retrieval (PIR) [12] technologies, which allow a client to read an entry from a database without the server learning which entry has been read.

The problem of private database modification is harder. A naïve approach would be to let the client change each entry in the database $\mathcal{DB}$ upon every access, which is however too expensive. Homomorphic encryption might be a natural candidate to outsource the computation to the server and to reduce the required bandwidth: unfortunately, Ostrovsky and Skeith III [43] showed that no private database modification (or PIR writing) scheme with sublinear communication (in the worst case) can be implemented using algebraic cryptographic constructions, such as linearly homomorphic encryption schemes. This result does not apply to schemes based on fully-homomorphic encryption, which is however hardly usable in practice due to the high computation cost associated with the currently known schemes.

The following sections describe our approach to bypass these two lower bounds. First we show how to integrate non-algebraic techniques, specifically out-of-band communication among clients, in order to achieve sublinear *amortized* communication complexity (Section 3). Second, we show how to leverage a trusted proxy performing the access to the server on behalf of clients in order to reach a logarithmic overhead in communication and server-side computation, with constant client-slide computation (Section 5).

3 PIR-MCORAM

In this section, we present a high-level description of PIR-MCORAM, a Multi-Client ORAM scheme based on PIR. The full details can be found in Appendix A. Our construction is inspired by Franz *et al.* [24], who proposed to augment the database with a stack of modified entries, which is periodically flushed into the database by the data owner. In our construction, we let each client $\mathcal{C}_i$ maintain its own temporary stack of entries S_i that is stored on the server side in addition to the regular database $\mathcal{DB}$. These stacks contain recent changes to entries in $\mathcal{DB}$ and to entries in other clients' stacks, which are not yet propagated to $\mathcal{DB}$. In contrast to the approach by Franz *et al.* [24], clients themselves are responsible to flush their stack once it is filled (i.e., after $|\mathsf{S}_i|$ many operations), *without requiring any intervention of the data owner*. An *oblivious gossiping protocol*, which can be realized using standard techniques [18, 36], allows clients to find the most up-to-date entry in the database, thereby obtaining a sublinear communication bandwith even for write operations and thus bypassing the impossibility result by Ostrovsky and Skeith III [43].

More precisely, when operating on index j, the client performs a PIR read on $\mathcal{DB}$ and on all stacks S_i, which can easily be realized since all stacks are stored on the server. Thanks to the oblivious gossiping protocol, the client knows which index is the most current one. At this point, the client appends either a dummy entry (read) or a real entry (write) to its personal stack. If the stack is full, the client flushes it. Flushing means to apply all changes in the personal stack to the database. To be oblivious, the client has to ensure that *all* entries in $\mathcal{DB}$ change. Moreover, for guaranteeing correctness, the client has to ensure that it does not overwrite entries which are more recent than those in its stack.

After explaining how to achieve obliviousness, we also need to discuss how to realize access control and how to protect the clients against the server. Data secrecy (i.e., read access control) is obtained via public-key encryption. Tamper-resistance (i.e., a-posteriori detection of illegal changes) is achieved by letting each client sign the modified entry so that others can check that this entry was produced by a client with write access. Data integrity (i.e., write access control) is achieved by further letting each client prove to the server that it is eligible to write the entry. As previously mentioned, data integrity is stronger than tamper-resistance, but assumes an honest-but-curious server: a malicious server may collude with malicious clients and thus store arbitrary information without checking integrity proofs.

3.1 Analysis

We elaborate on the communication complexity of our solution. We assume that $|\mathsf{DB}| = N$, that there M clients, and we set the stack length $len_{\mathsf{S}} = \sqrt{N}$ for every client. The worst case for an operation, hence, happens every $\sqrt{N}$-th operation for a client $\mathcal{C}_i$, meaning that besides extracting the data from

the database and adding an entry to the personal stack, $\mathcal{C}_i$ has also to flush the stack. We analyze the four algorithms independently: extracting data requires two PIR reads, one on DB and the other on the concatenation of all stacks. Thus, the overall cost is $\mathsf{PIR}(N) + \mathsf{PIR}(M\sqrt{N})$. Adding an entry to the personal stack always requires to upload one entry, independently of whether this replacement is real or dummy.

Our flushing algorithm assumes that $\mathcal{C}_i$ holds $\sqrt{N}$ entries and then down-and-uploads every entry of DB. Thus, the overall complexity is $2N+\sqrt{N}$. A similar analysis shows that if the client holds only $O(1)$ many entries, then $\mathcal{C}_i$ down-and-uploads DB but additionally performs a PIR step for every downloaded entry in its own stack to retrieve a potential replacement, resulting in a complexity of $2N+N\cdot\mathsf{PIR}(\sqrt{N})$.

To conclude, the construction achieves a worst-case complexity of $O(\mathsf{PIR}(N) + \mathsf{PIR}(M\sqrt{N}) + N)$ and $O(\mathsf{PIR}(N) + \mathsf{PIR}(M\sqrt{N}) + N\mathsf{PIR}(\sqrt{N}))$ for $O(\sqrt{N})$ and $O(1)$ client-side memory, respectively. By amortizing the flush step over $\sqrt{N}$ many operations, we achieve an amortized complexity of $O(\mathsf{PIR}(N) + \mathsf{PIR}(M\sqrt{N}) + \sqrt{N})$ or $O(\mathsf{PIR}(N) + \mathsf{PIR}(M\sqrt{N}) + \sqrt{N}\mathsf{PIR}(\sqrt{N}))$, respectively. Since our construction is parametric over the specific PIR protocol, we can leverage the progress in this field: at present, the best $\mathsf{PIR}(N)$ is $O(\log\log(N))$ [20] and, hence, the amortized cost becomes $O(\log\log(M\sqrt{N}) + \sqrt{N})$ or $O(\log\log(M\sqrt{N}) + \sqrt{N}\log\log(N))$, respectively. Since, in most scenarios, $M\sqrt{N} < 2^{2^{N/2}}$, we get $O(\sqrt{N})$ and $O(\sqrt{N}\log\log(N))$.

3.2 Discussion

The construction presented in this section leverages PIR for reading entries and an accumulated PIR writing technique to replace old entries with newer ones. Due to the nature of PIR, one advantage of the construction is its possibility to allow multiple clients to concurrently read from the database and to append single entries to their stacks. This is no longer possible when a client flushes her personal stack since the database is entirely updated, which might lead to inconsistent results when reading from the database. To overcome this drawback, we present a fully concurrent, maliciously secure Multi-Client ORAM in Section 5. Another drawback of the flush algorithm is the cost of the integrity (zero-knowledge) proofs. Since we have to use public-key encryption as the top-layer encryption scheme for every entry to allow for proving properties about the underlying plaintexts, the number of proofs to be computed, naïvely implemented, is proportional to the block size. Varying block sizes require us to split an entry into chunks and encrypt every chunk separately since the message space of public-key encryption is a constant amount of bits. The zero-knowledge proof has then to be computed on every of these encrypted chunks. To overcome this linear dependency, we present a new proof paradigm to make the number of computed zero-knowledge proofs independent of the block size in Section 4.

4 Integrity Proof Revised: The Hash-and-Proof Paradigm

In this section, we focus on the integrity proofs employed in our construction, presenting a novel and generally applicable cryptographic technique to boost their efficiency.

Plaintext Equivalence Proofs. To guarantee the integrity of the database, our construction requires extensive use of proofs the ciphertexts were correctly rerandomized. These proofs are called plaintext-equivalence-proofs (PEPs) in the literature and are the main efficiency bottleneck of our writing algorithm. Since the block size of an entry is in general *much larger* than the message space of the encryption scheme, we have to compute zero-knowledge proofs over vectors of ciphertexts. In this case, the integrity proof shows *for each* of these ciphertext vectors that they have been correctly rerandomized. The computational cost for these proofs scales linearly with the block size, which is clearly an undesirable dependency. In fact, this problem is not unique to our setting but affects any system deploying PEPs over long entries, among others verifiable secret shuffling [30, 42, 4] and mix networks [35].

In the following, we put forward a general technique to improve the computational efficiency of PEPs over ciphertext vectors. Our approach is fully black-box, non-interactive, and its proof size is *independent* of the number of ciphertexts of each entry. Thus, our technique can be used to boost the efficiency of not only PIR-MCORAM, but also any system based on PEPs. The basic idea behind our solution is to homomorphically compute a pairwise independent hash function [10] over the plaintexts of the two vectors and a PEP over the two resulting ciphertexts. Intuitively, a pairwise independent hash function is a collection of compressing functions such that the probability of two inputs to yield the same output is

negligibly small in the size of the output domain (over the random choice of the function). This property ensures that the soundness of the proof is preserved.

General Problem Description. Let $\Pi_{\mathsf{PKE}} = (\mathsf{Gen}_{\mathsf{PKE}}, \mathsf{E}, \mathsf{D}, \mathsf{Rnd})$ be a randomizable, additively homomorphic public-key encryption scheme and $(\mathcal{P}, \mathcal{V})$ be zero-knowledge proof system (ZKP) that takes as input two instances of ciphertexts $(c, b) \in \mathsf{E}(ek, m)^2$ for some $m \in \mathcal{M}$ and outputs a proof π for the statement $\exists r : b = \mathsf{Rnd}(ek, c, r)$. Construct a zero-knowledge proof system $(\mathcal{P}^*, \mathcal{V}^*)$ that takes as input two vectors of ciphertexts of length n, $(\mathbf{c}, \mathbf{b}) \in \mathsf{E}(ek, \mathbf{m})^{n \times 2}$ for some vector $\mathbf{m}$ and a vector $\mathbf{r}$ of randomnesses of the same length and outputs a proof π^* of the following statement: for all $i \in \{1, \ldots, n\}$ there exists a value r_i such that $b_i = \mathsf{Rnd}(ek, c_i, r_i)$. The efficiency goal is to make the size of the proof as well as the invocations of $(\mathcal{P}, \mathcal{V})$ independent of n. Knowing the decryption key dk, this statement is equivalent to the following one: for all $i \in \{1, \ldots, n\}$ we have $\mathsf{D}(dk, b_i) = \mathsf{D}(dk, c_i)$.

Our Solution. Let $\mathcal{M} = \mathbb{F}_p$ be the message space of Π_{PKE} for some field $\mathbb{F}_p$, such as the ElGamal or the Paillier encryption scheme [21, 44]. We describe our solution as an honest-verifier Σ-protocol which can be made non-interactive and resilient against any malicious verifier by applying the Fiat-Shamir heuristic [22]. In the following, $\mathsf{E}(ek, z_0; z_1)$ denotes the encryption of z_0 with key ek and randomness z_1.

1) $\mathcal{P}^*$ sends the vectors $(\mathbf{c}, \mathbf{b})$ to $\mathcal{V}^*$.
2) $\mathcal{V}^*$ samples a vector $\mathbf{z} \in \mathbb{F}_p^{n+2}$ uniformly at random and sends it to $\mathcal{P}^*$.
3) $\mathcal{P}^*$ computes $c' \leftarrow \mathsf{E}(ek, z_0; z_1) \bigotimes_{i=1}^{n} z_{i+1} \cdot c_i$ and $b' \leftarrow \mathsf{E}(ek, z_0; z_1) \bigotimes_{i=1}^{n} z_{i+1} \cdot b_i$ and runs $\mathcal{P}$ on inputs (c', b') to obtain π; $\mathcal{P}^*$ sends π to $\mathcal{V}^*$, who can recompute (c', b') and run $\mathcal{V}$ on $((c', b'), \pi)$. $\mathcal{V}^*$ returns the output of $\mathcal{V}$.

Security Analysis. In the following we state the formal guarantees of our techniques.

Theorem 2 (Hash-and-Proof)**.** *Let Π_{PKE} be an additively homomorphic CPA-secure public-key encryption scheme and let $(\mathcal{P}, \mathcal{V})$ be a ZKP for PEPs over Π_{PKE}. Then $(\mathcal{P}^*, \mathcal{V}^*)$ is a ZKP for PEPs over Π_{PKE}.*

Proof. The correctness of Π_{PKE} and of the ZKP $(\mathcal{P}, \mathcal{V})$ imply the *correctness* of the protocol described above. The *zero-knowledge* of the protocol follows from the zero-knowledge of $(\mathcal{P}, \mathcal{V})$. Arguing about the *soundness* requires a more accurate analysis: we define as $\mathsf{cheat}_{(\mathcal{P}^*, \mathcal{V}^*)}$ the event where a malicious $\mathcal{P}^*$ fools $\mathcal{V}^*$ into accepting a proof over a false statement. This event happens with probability

$$\begin{aligned}\Pr\left[\mathsf{cheat}_{(\mathcal{P}^*, \mathcal{V}^*)}\right] = \Pr\left[\mathsf{cheat}_{(\mathcal{P}, \mathcal{V})} \mid \mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right] \\ \cdot \Pr\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right] + \\ \Pr\left[\mathsf{cheat}_{(\mathcal{P}, \mathcal{V})} \mid \mathsf{D}(dk, c') \neq \mathsf{D}(dk, b')\right] \\ \cdot \Pr\left[\mathsf{D}(dk, c') \neq \mathsf{D}(dk, b')\right]\end{aligned}$$

where the probabilities are taken over the random coins of $\mathcal{P}^*$ and $\mathcal{V}^*$. By the soundness of $(\mathcal{P}, \mathcal{V})$ we get

$$\begin{aligned}\Pr\left[\mathsf{cheat}_{(\mathcal{P}^*, \mathcal{V}^*)}\right] &\leq 1 \cdot \Pr\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right] + \mu \cdot \Pr\left[\mathsf{D}(dk, c') \neq \mathsf{D}(dk, b')\right] \\ &\leq \mu + \Pr\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right]\end{aligned}$$

where μ is a negligible function in the security parameter. Therefore, to prove soundness, it is sufficient to show that when $\mathsf{cheat}_{(\mathcal{P}^*, \mathcal{V}^*)}$ happens $\Pr\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right]$ is a negligible function in the security parameter. We shall note that, due to the homomorphic properties of Π_{PKE}, the resulting plaintext of c' and b' are $z_0 + \sum_{i=1}^{n} z_{i+1} \mathsf{D}(dk, c_i) \in \mathbb{F}_p$, and $z_0 + \sum_{i=1}^{n} z_{i+1} \mathsf{D}(dk, b_i) \in \mathbb{F}_p$, respectively. It is easy to see that this corresponds to the computation of the *universal pair-wise hash function* $\mathsf{h}_{(\mathbf{z})}$ as described by Carter and Wegman in [10] (Proposition 8). It follows that for all $\mathbf{c} \neq \mathbf{b}$ the resulting plaintexts of c' and b' are uniformly distributed over $\mathbb{F}_p$, thus $\Pr\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right] = p^{-2}$, which is a negligible function in the security parameter. This concludes our proof. □ □

5 Proxy-based Realization

Driven by the goal of building an efficient and scaleable Multi-Client ORAM that is secure against malicious users, we explore the usage of a trusted proxy mediating accesses between clients and the

server, an approach advocated in recent parallel ORAM constructions [55, 6, 49]. In contrast to previous works, we are not only interested in parallel accesses, but also in handling access control and providing obliviousness against multiple, possibly malicious, clients.

TaoStore [49]. In a nutshell, trusted proxy-based ORAM constructions implement a single-client ORAM which is run by the trusted entity on behalf of clients, which connect to it with read and write requests in a parallel fashion. We leverage the state of the art, TaoStore [49], which implements a variant of a Path-ORAM [53] client on the proxy and allows for retrieving multiple paths from the server concurrently. More specifically, the proxy consists of the *processor* and the *sequencer*. The processor performs read and write requests to the untrusted server: this is the most complex part of TaoStore and we leave it untouched. The sequencer is triggered by client requests and forwards them to the processor which executes them in a concurrent fashion.

Our modifications. Since the proxy is trusted, it can enforce access control. In particular, we can change the sequencer so as to let it know the access control matrix and check for every client's read and write requests whether they are eligible or not. As already envisioned by Sahin *et al.* [49], the underlying ORAM construction can be further refined in order to make it secure against a malicious server, either by following the approach based on Merkle-trees proposed by Stefanov *et al.* [53] or by using authenticated encryption as suggested by Sahin *et al.* [49]. In the rest of the paper, we call the system TAO-MCORAM.

6 Security and Privacy Results

In the following we report the security results for PIR-MCORAM: those for TAO-MCORAM follow along the same lines. For the formal definition of the security properties we refer to [39]. Note that in our proofs we consider the adaptive version of each definition where the attacker is allowed to spawn and corrupt clients without restrictions. As a consequence, our instantiation requires us to fix in advance the number of clients M supported by the construction. Alternatively, one could consider the selective versions of the security definitions where the attacker is required to commit in advance to the client subset that he wants to corrupt. The full proofs for the theorems below can be found in Appendix C.

Theorem 3 (Secrecy)**.** *Let* Π_{PKE} *be a CPA-secure encryption scheme, then PIR-MCORAM achieves secrecy.*

Theorem 4 (Integrity)**.** *Let* Π_{DS} *be an existentially unforgeable digital signature scheme,* ZKP *be a zero-knowledge proof of knowledge protocol, and* Π_{PKE} *be a CCA-secure encryption scheme, then PIR-MCORAM achieves integrity.*

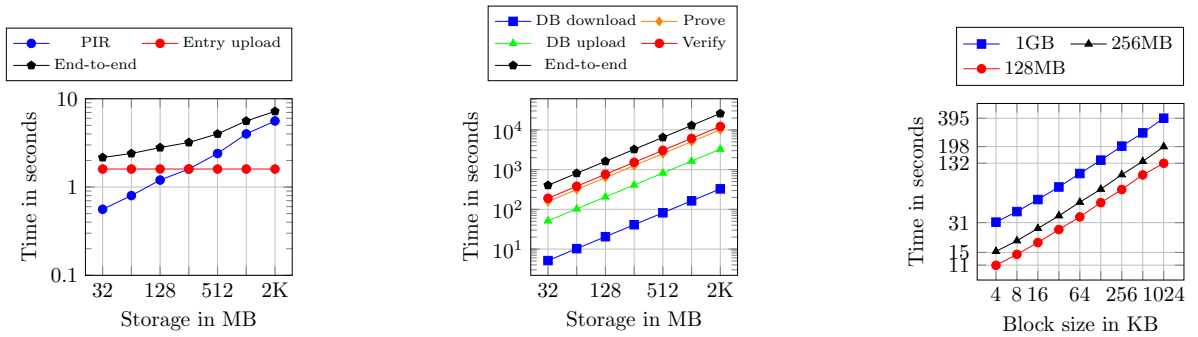
Theorem 5 (Tamper Resistance)**.** *Let* Π_{DS} *be an existentially unforgeable digital signature scheme and let* Π_{PKE} *be a CCA-secure encryption scheme, then PIR-MCORAM achieves tamper resistance.*

Theorem 6 (Obliviousness against mal. clients)**.** *Let* PIR *be a private information retrieval scheme, let* Π_{PKE} *be a CPA-secure encryption scheme, let* Π_{DS} *be an existentially unforgeable digital signature scheme, and let* ZKP *be a zero-knowledge proof of knowledge protocol, then PIR-MCORAM is secure against malicious clients.*

7 Evaluation

In this section, we describe our implementation and report on the experimental results. We start by reviewing the cryptographic schemes that we deploy: all of them are instantiated with a security parameter of 128bits [7].

Cryptographic instantiations. We deploy ElGamal encryption [21] in an hybrid fashion to construct an entry in the database (cf. $\mathbf{c}_{\mathsf{Data}}$ and $\mathbf{c}_{\mathsf{BrCast}}$ in Appendix A). Using the hybrid technique, we decrease the entry size from $O(MB)$ to $O(M + B)$ since the data is encrypted only one and the corresponding secret key is encrypted for all clients with read access. In contrast, we encrypt the signing keys of the Schnorr signature scheme [50] (cf. $\mathbf{c}_{\mathsf{Auth}}$ in Appendix A) using the Cramer-Shoup encryption scheme [14]. We use XPIR [1], the state of the art in computational PIR.



(a) Access without flush, 1MB block size.

(b) Flush, 1MB block size.

(c) Amortized time, varying block size.

Figure 1: The end-to-end running time of an operation in PIR-MCORAM.

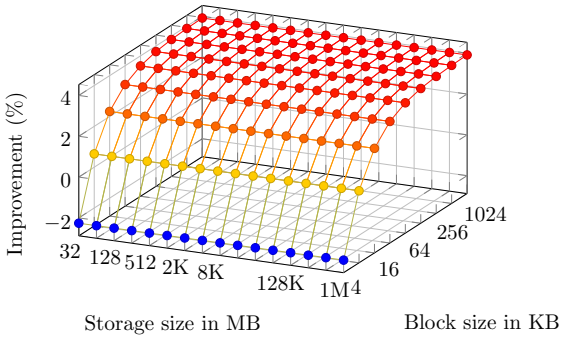


Figure 2: The improvement in percent when comparing the combined proof computation time on the client and proof verification time on the server for varying storage and block sizes, once without and once with the universal homomorphic hash.

Finally, in order to construct integrity proofs in (cf. lines 4.20 and 4.26 in Appendix A), we use an OR-proof [13] over a conjunction of plaintext-equivalence proofs [35] (PEP) on the ElGamal ciphertexts forming one entry and a standard discrete logarithm proof [50] showing that the client knows the signing key corresponding to the authenticated verification key. In the homomorphic hash version, the conjunction of PEPs reduces to the computation of the homomorphic hash plus one PEP. As a matter of fact, since the public components necessary to verify a proof (the new and old ciphertexts and the verification key) and the secret components necessary to compute the proof (the randomness used for rerandomization or the signing key) are independent of the number of clients, all deployed proofs solely depend on the block size.

Implementation and experiments. We implemented the cryptographic components of PIR-MCORAM in Java and we use a wrapper to GMP to speed up computations.

We used an Intel Xeon E5-4650L with 2.60 GHz, 8 cores, and 20MB cache for the client and server experiments. We performed micro-benchmarks for PIR-MCORAM while varying the storage size from 128MB to 2GB and the block size from 4KB to 1MB, both for the solution with and without the homomorphic hash computation. We measured partial computation times as well as the end-to-end access time where we assume a network with 100 Mbit/s downstream and 10 Mbit/s upstream. In order to show the efficiency of our homomorphic hash construction and demonstrate its generic applicability, we also compare GORAM [39] with batched shuffle proofs, as originally presented, with a variant thereof where we replace the batched shuffle proofs with our homomorphic hash plus one shuffle proof.

Discussion. Figure 1 and Figure 2 report the results for PIR-MCORAM. Figure 1a shows the end-to-end and partial running times of an access to the ORAM when the flush algorithm is not executed, whereas Figure 1b depicts the worst case running time (i.e., with flush operation). For the example of the medical record which usually fits into 128MB (resp. 256MB for additional files such as X-ray images),

the amortized times per access range from 11 (resp. 15) seconds for 4KB up to 131 (resp. 198) seconds for 1MB sized entries (see Figure 1c).

Figure 2 shows the improvement as we compare the combined proof computation and proof verification time in the flush algorithm of PIR-MCORAM, first as described in Section 3 and then with the integrity proof based on the universal homomorphic hash (see Section 4). We observe that our expectations are fulfilled: the larger the block size, the more effect has the universal hash computation since the number of proofs to compute decreases. Concretely, with 1MB block size we gain a speed-up of about 4% for flush operations with respect to the construction without homomorphic hash.

To demonstrate its general applicability, we instantiate our proof technique into GORAM [39], which uses so-called batched shuffle proofs, achieving much better results. In GORAM, clients have to compute integrity proofs, which are proofs of shuffle correctness–a much more expensive primitive than PEPs. To overcome the efficiency problem, the authors have developed batched shuffle proofs: the idea is to homomorphically sum up half of the columns of the database matrix at random and to perform a shuffle proof on the resulting list of ciphertexts. To achieve soundness, the protocol has to be repeated $k = 128$ times. We observe that we can replace batched shuffle proofs by our protocol: clients compute the homomorphic hash on the old and the new ciphertexts and then one shuffle proof on the resulting lists of ciphertexts. As shown in Table 2, this modification speeds up GORAM by one order of magnitude (14x on the client and 10.8x on the server).

Finally, our solution TAO-MCORAM only adds access control to the actual computation of TaoStore's trusted proxy [49]. Interestingly enough, TaoStore's bottleneck is not computation, but communication. Hence, our modifications do not cause any noticeable slowdown on the throughput of TaoStore. Consequently, we end up with a throughput of about 40 operations per second when considering an actual deployment of TAO-MCORAM in a cloud-based setting [49].

Construction	**Client time**	**Server time**
GORAM with $k = 128$	91.315 s	39.213 s
GORAM with HH	5.980 s	3.384 s
Improvement	**14x**	**10.8x**

Table 2: Comparison of GORAM [39] with batched shuffle proofs and GORAM instantiated with our homomorphic hash (HH) variant for 10 users, 1GB storage, and 8KB block size.

8 Conclusion

This work studies the problem of obliviousness in multi-client outsourced storage. We establish a lower bound on the server-side computational complexity, showing that any secure realization has to involve at least $\Omega(n)$ computation steps. We further present a novel cryptographic instantiation, which achieves an amortized communication overhead of $O(\sqrt{n})$ by combining private information retrieval technologies, a new accumulation technique, and an oblivious gossiping protocol. Access control is enforced by efficient integrity proofs, which leverage a new construction for Plaintext Equivalence Proofs based on a homomorphic universal pair-wise hash function. Finally, we showed how to bypass our lower bound by leveraging a trusted proxy [49], thereby achieving logarithmic communication and server side computational complexity.

This work opens up a number of interesting research directions. Among those, it would be interesting to prove a lower bound on the communication complexity. Furthermore, we would like to relax the obliviousness property in order to bypass the lower bound established in this paper, coming up with more efficient constructions and quantifying the associated privacy loss.

References

[1] Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: XPIR : Private Information Retrieval for Everyone. In: Proc. Privacy Enhancing Technologies Symposium (PETS'16). pp. 155–174. De Gruyter (2016)

[2] Ajtai, M.: Oblivious RAMs Without Cryptographic Assumptions. In: Proc. ACM Symposium on Theory of Computing (STOC'10). pp. 181–190. ACM (2010)

[3] Apon, D., Katz, J., Shi, E., Thiruvengadam, A.: Verifiable Oblivious Storage. In: Proc. Practice and Theory in Public Key Cryptography (PKC'14). pp. 131–148. LNCS, Springer Verlag (2014)

[4] Bayer, S., Groth, J.: Efficient Zero-Knowledge Argument for Correctness of a Shuffle. In: Proc. Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'12). pp. 263–280. LNCS, Springer Verlag (2012)

[5] Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations Among Notions of Security for Public-Key Encryption Schemes. In: Proc. Advances in Cryptology (CRYPTO'98). pp. 26–45. LNCS, Springer Verlag (1998)

[6] Bindschaedler, V., Naveed, M., Pan, X., Wang, X., Huang, Y.: Practicing Oblivious Access on Cloud Storage: The Gap, the Fallacy, and the New Way Forward. In: Proc. Conference on Computer and Communications Security (CCS'15). pp. 837–849. ACM (2015)

[7] BlueKrypt: Cryptograhpic Key Length Recommendation, online at www.keylength.com

[8] Boyle, E., Chung, K.M., Pass, R.: Oblivious Parallel RAM and Applications. In: Proc. Theory of Cryptography (TCC'16). LNCS, Springer Verlag (2016)

[9] Carbunar, B., Sion, R.: Regulatory Compliant Oblivious RAM. In: Proc. Applied Cryptography and Network Security (ACNS'10). pp. 456–474. LNCS, Springer Verlag (2010)

[10] Carter, J.L., Wegman, M.N.: Universal Classes of Hash Functions (Extended Abstract). In: Proc. ACM Symposium on Theory of Computing (STOC'77). pp. 106–112. ACM (1977)

[11] Chen, B., Lin, H., Tessaro, S.: Oblivious Parallel RAM: Improved Efficiency and Generic Constructions. In: Proc. Theory of Cryptography (TCC'16). LNCS, Springer Verlag (2016)

[12] Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private Information Retrieval. Journal of the ACM 45(6), 965–981 (Nov 1998)

[13] Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In: Proc. Advances in Cryptology (CRYPTO'94). pp. 174–187. LNCS, Springer Verlag (1994)

[14] Cramer, R., Shoup, V.: A Practical Public Key Cryptosystem Provably Secure Against Adaptive Chosen Ciphertext Attack. In: Proc. Advances in Cryptology (CRYPTO'98). pp. 13–25. LNCS, Springer Verlag (1998)

[15] Culnane, C., Schneider, S.: A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In: Proc. Symposium on Computer Security Foundations (CSF'14). pp. 169–183. IEEE Press (2014)

[16] Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly Secure Oblivious RAM Without Random Oracles. In: Proc. Theory of Cryptography (TCC'11). pp. 144–163. LNCS, Springer Verlag (2011)

[17] Dautrich, J., Stefanov, E., Shi, E.: Burst ORAM: Minimizing ORAM Response Times for Bursty Access Patterns. In: Proc. USENIX Security Symposium (USENIX'14). pp. 749–764. USENIX Association (2014)

[18] Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic Algorithms for Replicated Database Maintenance. In: Proc. Symposium on Principles of Distributed Computing (PODC'87). pp. 1–12. ACM (1987)

[19] Diffie, W., Hellman, M.: New Directions in Cryptography. IEEE Transactions on Information Theory 22(6), 644–654 (Sep 2006)

[20] Dong, C., Chen, L.: A Fast Single Server Private Information Retrieval Protocol with Low Communication Cost. In: Proc. European Symposium on Research in Computer Security (ESORICS'14). LNCS, vol. 8712, pp. 380–399. Springer Verlag (2014)

[21] El Gamal, T.: A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In: Proc. Advances in Cryptology (CRYPTO'84). pp. 10–18. LNCS, Springer Verlag (1985)

[22] Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: Proc. Advances in Cryptology (CRYPTO'86). pp. 186–194. Springer Verlag (1987)

[23] Foundation, E.H.: e-TERVIS, online at http://www.e-tervis.ee

[24] Franz, M., Carbunar, C., Sion, R., Katzenbeisser, S., Sotakova, M., Williams, P., Peter, A.: Oblivious Outsourced Storage with Delegation. In: Proc. Financial Cryptography and Data Security (FC'11). pp. 127–140. Springer Verlag (2011)

[25] GmbH, E.: ELGA, online at https://www.elga.gv.at

[26] Goldreich, O., Ostrovsky, R.: Software Protection and Simulation on Oblivious RAMs. Journal of the ACM 43(3), 431–473 (May 1996)

[27] Goldwasser, S., Micali, S.: Probabilistic Encryption & How To Play Mental Poker Keeping Secret All Partial Information. In: Proc. ACM Symposium on Theory of Computing (STOC'82). pp. 365–377. ACM (1982)

[28] Goldwasser, S., Micali, S., Rivest, R.L.: A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks. SIAM Journal on Computing 17(2), 281–308 (Apr 1988)

[29] Goodrich, M.T., Mitzenmacher, M.: Privacy-Preserving Access of Outsourced Data via Oblivious RAM Simulation. In: Proc. International Conference on Automata, Languages and Programming (ICALP'11). pp. 576–587. LNCS, Springer Verlag (2011)

[30] Groth, J.: A Verifiable Secret Shuffle of Homomorphic Encryptions. In: Proc. Practice and Theory in Public Key Cryptography (PKC'03). pp. 145–160. LNCS, Springer Verlag (2003)

[31] Heather, J., Lundin, D.: The Append-Only Web Bulletin Board. In: Proc. USENIX Conference on File and Storage Technologies (FAST'09). pp. 242–256. Springer Verlag (2009)

[32] Huang, Y., Goldberg, I.: Outsourced Private Information Retrieval with Pricing and Access Control. In: Proc. Annual ACM Workshop on Privacy in the Electronic Society (WPES'13). ACM (2013)

[33] Iliev, A., Smith, S.W.: Protecting Client Privacy with Trusted Computing at the Server. IEEE Security and Privacy 3(2), 20–28 (Mar 2005)

[34] Islam, M., Kuzu, M., Kantarcioglu, M.: Access Pattern Disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In: Proc. Annual Network & Distributed System Security Symposium (NDSS'12). Internet Society (2012)

[35] Jakobsson, M., Juels, A.: Millimix: Mixing in Small Batches. Tech. Rep. 99-33, DIMACS (1999)

[36] Kim, B.H., Lie, D.: Caelus: Verifying the Consistency of Cloud Services with Battery-Powered Devices. In: Proc. IEEE Symposium on Security & Privacy (S&P'15). pp. 880–896. IEEE Press (2015)

[37] Lorch, J.R., Parno, B., Mickens, J., Raykova, M., Schiffman, J.: Shroud: Ensuring Private Access to Large-scale Data in the Data Center. In: Proc. USENIX Conference on File and Storage Technologies (FAST'13). pp. 199–214. USENIX Association (2013)

[38] Maas, M., Love, E., Stefanov, E., Tiwari, M., Shi, E., Asanovic, K., Kubiatowicz, J., Song, D.: PHANTOM: Practical Oblivious Computation in a Secure Processor. In: Proc. Conference on Computer and Communications Security (CCS'13). pp. 311–324. ACM (2013)

[39] Maffei, M., Malavolta, G., Reinert, M., Schröder, D.: Privacy and Access Control for Outsourced Personal Records. In: Proc. IEEE Symposium on Security & Privacy (S&P'15). IEEE Press (2015)

[40] Maffei, M., Malavolta, G., Reinert, M., Schröder, D.: Maliciously Secure Multi-Client ORAM. In: Proc. Applied Cryptography and Network Security (ACNS'17). LNCS, Springer Verlag (2017), to appear

[41] Mayberry, T., Blass, E.O., Chan, A.H.: Efficient Private File Retrieval by Combining ORAM and PIR. In: Proc. Annual Network & Distributed System Security Symposium (NDSS'14). Internet Society (2013)

[42] Neff, C.A.: A Verifiable Secret Shuffle and Its Application to e-Voting. In: Proc. Conference on Computer and Communications Security (CCS'01). pp. 116–125. ACM (2001)

[43] Ostrovsky, R., III, W.E.S.: Algebraic Lower Bounds for Computing on Encrypted Data. Electronic Colloquium on Computational Complexity (ECCC) 14(022) (2007)

[44] Paillier, P.: Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In: Proc. Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'99). pp. 223–238. LNCS, Springer Verlag (1999)

[45] Pinkas, B., Reinman, T.: Oblivious RAM Revisited. In: Proc. Advances in Cryptology (CRYPTO'10). pp. 502–519. LNCS, Springer Verlag (2010)

[46] Rivest, R.L., Shamir, A., Adleman, L.: A Method for Obtaining Digital Signatures and Public-key Cryptosystems. Communications of the ACM 21(2), 120–126 (Feb 1978)

[47] Robinson, R.J.: How big is the human genome? Online at https://medium.com/precision-medicine/how-big-is-the-human-genome-e90caa3409b0

[48] Roche, D.S., Aviv, A., Choi, S.G.: A Practical Oblivious Map Data Structure with Secure Deletion and History Independence. In: Proc. IEEE Symposium on Security & Privacy (S&P'16). IEEE Press (2016)

[49] Sahin, C., Zakhary, V., Abbadi, A.E., Lin, H.R., Tessaro, S.: TaoStore: Overcoming Asynchronicity in Oblivious Data Storage. In: Proc. IEEE Symposium on Security & Privacy (S&P'16). IEEE Press (2016)

[50] Schnorr, C.P.: Efficient Identification and Signatures for Smart Cards. In: Proc. Advances in Cryptology (CRYPTO'89). pp. 239–252. LNCS, Springer Verlag (1989)

[51] Shi, E., Chan, T.H.H., Stefanov, E., Li, M.: Oblivious RAM With $O((\log n)^3)$ Worst-Case Cost. In: Proc. International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'11). pp. 197–214. LNCS, Springer Verlag (2011)

[52] Stefanov, E., Shi, E., Song, D.: Towards Practical Oblivious RAM. In: Proc. Annual Network & Distributed System Security Symposium (NDSS'12). Internet Society (2012)

[53] Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: An Extremely Simple Oblivious RAM Protocol. In: Proc. Conference on Computer and Communications Security (CCS'13). ACM (2013)

[54] Stefanov, E., Shi, E.: Multi-Cloud Oblivious Storage. In: Proc. Conference on Computer and Communications Security (CCS'13). pp. 247–258. ACM (2013)

[55] Stefanov, E., Shi, E.: ObliviStore: High Performance Oblivious Cloud Storage. In: Proc. IEEE Symposium on Security & Privacy (S&P'13). pp. 253–267. IEEE Press (2013)

[56] Williams, P., Sion, R., Tomescu, A.: PrivateFS: A Parallel Oblivious File System. In: Proc. Conference on Computer and Communications Security (CCS'12). pp. 977–988. ACM (2012)

A PIR-MCORAM Construction

A.1 Preliminaries and Data Structures

In the following we introduce the cryptographic primitives used in our constructions.

Public-key encryption. A *public-key encryption scheme* is a tuple of PPT algorithms $\Pi_{\mathsf{PKE}} = (\mathsf{Gen}_{\mathsf{PKE}}, \mathsf{E}, \mathsf{D})$ for *key-generation* $((ek, dk) \leftarrow \mathsf{Gen}_{\mathsf{PKE}}(1^\lambda))$, *encryption* $(c \leftarrow \mathsf{E}(ek, m))$, and *decryption* $(m \leftarrow \mathsf{D}(dk, c))$, respectively. We require Π_{PKE} to be additively homomorphic and rerandomizable: for all messages $(m, n) \in \mathcal{M}^2$, where $\mathcal{M}$ is the message space of Π_{PKE}, there exists an operator $\otimes$ such that $\mathsf{D}(dk, \mathsf{E}(ek, m) \otimes \mathsf{E}(ek, n)) = m + n$. We denote the product of a ciphertext $\mathsf{E}(ek, m)$ with a scalar α by $\alpha \cdot \mathsf{E}(ek, m) = \mathsf{E}(ek, m \cdot \alpha)$ and the public rerandomization function by Rnd. We require Π_{PKE} to be IND-CPA secure [27]. Additionally, we deploy a standard Π_{PKE} which must be IND-CCA secure [5].

Digital signatures. A *digital signature scheme* [19, 46] is a tuple of PPT algorithms $\Pi_{\mathsf{DS}} = (\mathsf{Gen}_{\mathsf{DS}}, \mathsf{Sign}, \mathsf{Vrfy})$ for *key-generation* $((vk, sk) \leftarrow \mathsf{Gen}_{\mathsf{DS}}(1^\lambda))$, *signing* $(\sigma \leftarrow \mathsf{Sign}(sk, m))$, and *verification* $(\{\top, \bot\} \leftarrow \mathsf{Vrfy}(vk, \sigma, m))$. We require Π_{DS} to be existentially unforgeable [28].

Private information retrieval. A *private information retrieval (PIR)* protocol [12] is a tuple of PPT algorithms $\mathsf{PIR} = (\mathsf{prepRead}, \mathsf{execRead}, \mathsf{decodeResp})$ where $q \leftarrow \mathsf{prepRead}(i)$ generates a query q on input an index i; $r \leftarrow \mathsf{execRead}(q)$ computes an encoded response r when executing q on the underlying database $\mathcal{DB}$; and $d \leftarrow \mathsf{decodeResp}(r)$ decodes the response r and returns the result d. Intuitively, a PIR guarantees privacy if an adversary executing queries on a database cannot tell which of two known indices is being queried.

Zero-knowledge proofs. A *zero-knowledge proof system* ZKP is a proof system between a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with three fundamental properties: *correctness*, *soundness*, and *zero-knowledge*. Correctness means that a correctly generated proof will always verify. Soundness means that it is not possible to generate a proof for a wrong statement that successfully verifies. Finally, zero-knowledge means that the proof reveals nothing else but the validity of the proven statement. A zero-knowledge proof of knowledge additionally guarantees that the prover knows the witnesses to the statement. Finally, a zero-knowledge proof (of knowledge) is non-interactive if it consists of a single message from $\mathcal{P}$ to $\mathcal{V}$. We write $PK\{(\vec{x}) : F\}$ to denote a non-interactive zero-knowledge proof of knowledge (NIZK) of the statement F where the variables in $\vec{x}$ are existentially quantified, i.e., these variables are hidden by the proof.

System Assumptions. $\mathcal{DB}$ stores up to N entries of size B each, hence the maximum capacity of $\mathcal{DB}$ is BN. The number of clients with access to $\mathcal{DB}$ is at most M. We assume that the server $\mathcal{S}$ has a storage capacity of $O(BN + \sum_{i=1}^{k} B\,|\mathsf{S}_i|)$ for the database and the client stacks; each $\mathcal{C}_i$ has a storage capacity of $O(|\mathsf{S}_i|\,B + N)$ to store the personal stack and a partial position map I, which is used to find the most current version of each entry; finally, $\mathcal{O}$ has a storage capacity of $O(N + B)$ to store the full position map and the access control matrix. While the position map is in general $O(N)$, this is usually much less than the storage size of $O(NB)$ [6] and can also be decreased to $O(1)$ by storing it in recursive ORAMs [53].

The database $\mathcal{DB}$ is accompanied by a private access control matrix ACM that lets $\mathcal{O}$ manage the per-client permissions for each entry in $\mathcal{DB}$. The possible access rights are R (read-only), RW (read-write), and $\bot$ (no access).

We assume authenticated broadcast channels among clients so as to gossip position map updates using standard techniques[1] [18, 36].

Database Layout. We represent the logical database $\mathcal{DB}$ as a list of entries $\mathsf{DB} = e_1, \ldots, e_N$ and a list of stacks $\mathsf{S}_1, \ldots, \mathsf{S}_M$, one stack for every client. Both the database and the stacks are stored on the server. A stack is an entry list $\mathsf{S}_i = e_{j+1}, \ldots, e_{j+|\mathsf{S}_i|}$ where $j = \sum_{k=1}^{i-1} |\mathsf{S}_k|$. We denote by $\mathsf{S}_i(\ell)$ the ℓ-th entry of S_i. We write $\mathsf{S}_1 || \ldots || \mathsf{S}_M$ to express the list of entries in all stacks. Similarly, we count from 1 to $\sum_{i=1}^{M} |\mathsf{S}_i|$ to index an entry in $\mathsf{S}_1 || \ldots || \mathsf{S}_M$.

Client capabilities. We assume that every client $\mathcal{C}_i$ holds a key pair $(ek_i^{\mathsf{CPA}}, dk_i^{\mathsf{CPA}})$ for a CPA-secure encryption scheme as well as a key pair $(ek_i^{\mathsf{CCA}}, dk_i^{\mathsf{CCA}})$ for a CCA-secure encryption scheme where $(ek_i^{\mathsf{CPA}}, ek_i^{\mathsf{CCA}})$ are publicly known and $(dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$ are $\mathcal{C}_i$'s private keys. Moreover, each client stores a position map I and version numbers $(vrs, vrs_{\mathcal{O}})$ for every entry it holds permissions on. These version numbers are necessary to prevent roll-back attacks (intuitively, the former on data, the latter

[1] Gossiping is necessary since we do not trust the server for consistency.

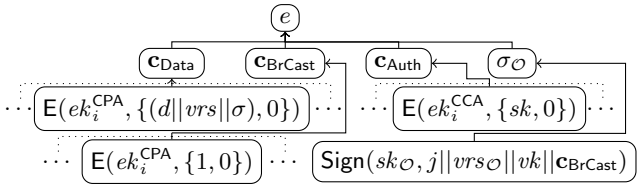


Figure 3: The entry structure of an entry in the main database. If an entry resides on the stack, it contains only the part $\mathbf{c}_{\mathsf{Data}}$.

on the access control matrix) and they are broadcast together with new indices for an entry upon write operations or whenever the policy is changed. Finally, every client stores a mapping which maps stack positions to entry indices: we keep this mapping implicit.

Entry Structure. An entry in the database DB has the form $e = (\mathbf{c}_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ where $\mathbf{c}_{\mathsf{Data}}$, $\mathbf{c}_{\mathsf{BrCast}}$, and $\mathbf{c}_{\mathsf{Auth}}$ are vectors of ciphertexts of length M and $\sigma_{\mathcal{O}}$ is a signature of the data owner $\mathcal{O}$. We describe each component and its functionality in the following (see also Figure 3).

$\underline{\mathbf{c}_{\mathsf{Data}}}$. The ciphertext $\mathbf{c}_{\mathsf{Data}}$ regulates read accesses. Specifically, it encrypts the data d of e for every client[2] with at least R permissions or a zero string for the others:

$$c^i_{\mathsf{Data}} = \begin{cases} \mathsf{E}(ek_i^{\mathsf{CPA}}, d||vrs||\sigma) & \text{if } \mathsf{ACM}(j,i) \neq \perp \\ \mathsf{E}(ek_i^{\mathsf{CPA}}, 0^{|d|+|vrs|+|\sigma|}) & \text{otherwise} \end{cases} \tag{1}$$

where in addition to d, the ciphertext also contains the data version number vrs as well as a signature σ such that $\top = \mathsf{Vrfy}(vk, d||vrs)$. By vk we denote a verification key of a signature scheme which is different for every entry. An entry is valid if the verification of σ with vk outputs $\top$ and vk is the key authenticated by the data owner $\mathcal{O}$ in $\sigma_{\mathcal{O}}$, and invalid otherwise.

$\underline{\mathbf{c}_{\mathsf{BrCast}}}$. The ciphertext $\mathbf{c}_{\mathsf{BrCast}}$ is needed in the $\mathsf{Broadcast}$ protocol (described below), which is used to obliviously propagate a new entry index and new version numbers to other clients with read access. Specifically, it encrypts either 1 for every client with at least R permissions or zero for the others:

$$c^i_{\mathsf{BrCast}} = \begin{cases} \mathsf{E}(ek_i^{\mathsf{CPA}}, 1) & \text{if } \mathsf{ACM}(j,i) \neq \perp \\ \mathsf{E}(ek_i^{\mathsf{CPA}}, 0) & \text{otherwise} \end{cases} \tag{2}$$

$\underline{\mathbf{c}_{\mathsf{Auth}}}$. This ciphertext contains the signing key corresponding to vk for those clients with RW permissions or the zero string for the others. The exact form is

$$c^i_{\mathsf{Auth}} = \begin{cases} \mathsf{E}(ek_i^{\mathsf{CCA}}, sk) & \text{if } \mathsf{ACM}(j,i) = \mathsf{RW} \\ \mathsf{E}(ek_i^{\mathsf{CCA}}, 0^{|sk|}) & \text{otherwise} \end{cases} \tag{3}$$

$\underline{\sigma_{\mathcal{O}}}$. The signature $\sigma_{\mathcal{O}}$ is created by $\mathcal{O}$ on the entry index j, a version number $vrs_{\mathcal{O}}$, the verification key vk, and $\mathbf{c}_{\mathsf{BrCast}}$.

Note that one cannot store the signing key sk in the entry $\mathbf{c}_{\mathsf{Data}}$. The reason is that whenever an entry is updated, the client needs to update all entries in the vector. However, for all entries except for its own, it does not know the private decryption key dk_i and thus, neither the corresponding private signing key nor the access rights for that entry. To update these entries, we exploit the homomorphic properties of the underlying encryption scheme, as explained below.

Update. Entries residing on a client's stack consist only of $\mathbf{c}_{\mathsf{Data}}$ in modified form where the old payload $D = d||vrs||\sigma$ has been replaced with $D' = d'||vrs'||\mathsf{Sign}(sk, d'||vrs')$. Indeed, leveraging the homomorphic property and the structure of $\mathbf{c}_{\mathsf{BrCast}}$ (note that $\mathbf{c}_{\mathsf{BrCast}}$ is like $\mathbf{c}_{\mathsf{Data}}$, where D is replaced with 1) it is possible to generate $\mathbf{c}'_{\mathsf{Data}}$ as follows: choose r_i uniformly at random and compute

$$c^{i'}_{\mathsf{Data}} = \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^i_{\mathsf{BrCast}} \cdot D', r_i). \tag{4}$$

[2]This notation simplifies the presentation, but in the implementation we use of course hybrid encryption (cf. Section 7)

Multiple data owners in one ORAM. The entry structure and database layout of PIR-MCORAM can be easily extended in order to support multiple data owners storing their files in the same ORAM instance (think, e.g., of multiple patients storing their health record in the same ORAM), which is important to enhance user privacy (as the server does not even learn the owner of the accessed data). First, the signature $\sigma_{\mathcal{O}}$ is obviously constructed by the data owner to which the entry belongs. Most importantly, every entry might have a different set of potential readers and writers (e.g., not every patient visits the same doctors or pharmacies). As a consequence, an important invariant to maintain is that $\mathbf{c}_{\mathsf{Data}}$, $\mathbf{c}_{\mathsf{BrCast}}$, and $\mathbf{c}_{\mathsf{Auth}}$ are of equal length for every entry (i.e., the number of encryption keys used to construct them are the same), which can be easily achieved by padding. Otherwise, trivial entry-size based attacks against obliviousness are possible.

Obliviously broadcasting new indices. We propagate the updates of the entries to the clients with read access via broadcast. That is, if $e = (\mathbf{c}_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ with an old index j in the database $\mathcal{DB}$ and the new index ℓ on a stack or in $\mathcal{DB}$, then we broadcast a message to all clients that can only be decrypted by clients having access to that entry. To this end, we leverage the same idea as in Equation (4), that is, we add the new index information to it. Clients compute $\mathbf{c}'_{\mathsf{BrCast}}$ with $c^{i'}_{\mathsf{BrCast}} = \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^i_{\mathsf{BrCast}} \cdot (j||\ell||vrs'), r_i)$ for some random values r_i and a new version number vrs', and broadcast $\mathbf{c}'_{\mathsf{BrCast}}$ to all clients. We call this operation $\mathsf{Broadcast}((j||\ell||vrs'), \mathbf{c}_{\mathsf{BrCast}})$.

Clients having read access update their position map as follows. Upon receiving such a message c, the client $\mathcal{C}_i$ tries to decrypt the component corresponding to her identity with her private key dk_i^{CPA}. If the result is $(j||\ell||vrs')$ and not 0 (which means that it has R access at least), then $\mathcal{C}_i$ updates its partial position map with the result. Otherwise it ignores the message. This protocol is oblivious since it is deterministically executed upon each operation and only clients with R access (which, as previously discussed, are excluded by the definition of obliviousness) can extract knowledge from the received ciphertext thanks to the CPA-security of the underlying encryption scheme.

Since malicious clients could potentially send wrong gossip messages about entries, e.g., claiming that an entry is residing in a different place than it actually is, we require that clients upload their broadcast messages also onto an append-only log, e.g., residing on the cloud, which is accessible by everyone. If a client does not find an entry using the latest index information, due to the malicious behavior of another client, then it just looks up the previous index and tries it there, and so on. Such append-only logs can be realized both in centralized [31] and decentralized [15] fashion in a secure way.

A.2 Algorithmic Description

Setup. The input to the setup algorithm is a list of data $d_1, \ldots, d_N$ and a list of clients $\mathcal{C}_1, \ldots, \mathcal{C}_M$ with an access control matrix ACM which has an entry for every entry-client pair. The data owner first generates her own signing key pair $(vk_{\mathcal{O}}, sk_{\mathcal{O}}) \leftarrow \mathsf{Gen}_{\mathsf{DS}}(1^\lambda)$ and generates two encryption key pairs $(ek_j^{\mathsf{CPA}}, dk_j^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}^{\mathsf{CPA}}_{\mathsf{PKE}}(1^\lambda)$ and $(ek_j^{\mathsf{CCA}}, dk_j^{\mathsf{CCA}}) \leftarrow \mathsf{Gen}^{\mathsf{CCA}}_{\mathsf{PKE}}(1^\lambda)$ for every client $\mathcal{C}_j$. Second, the data owner prepares every entry separately as follows: she generates a fresh signing key pair $(vk, sk) \leftarrow \mathsf{Gen}_{\mathsf{DS}}(1^\lambda)$ and sets up $\mathbf{c}_{\mathsf{Data}}$ as in Equation (1) using d_j and a version number 0, attaching a signature $\sigma = \mathsf{Sign}(sk, d_j||0)$. $\mathbf{c}_{\mathsf{BrCast}}$ is generated as in Equation (2). Next, $\mathbf{c}_{\mathsf{Auth}}$ is generated as in Equation (3) using the just generated sk. Finally, using a data owner version number 0, $\mathcal{O}$ attaches $\sigma_{\mathcal{O}} = \mathsf{Sign}(sk_{\mathcal{O}}, j||0||vk||\mathbf{c}_{\mathsf{BrCast}})$. $\mathcal{O}$ uploads all entries to $\mathcal{S}$ and broadcasts the client capabilities $cap_i = (\mathsf{I}_i, \mathbf{ek}, vk_{\mathcal{O}}, i_{\mathsf{S}}, len_{\mathsf{S}}, dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$ where I_i is the full position map I restricted to those entries on which $\mathcal{C}_i$ holds at least R permissions, $\mathbf{ek}$ is a list of all clients' encryption keys, $i_{\mathsf{S}} = 0$ is $\mathcal{C}_i$'s current stack pointer, and len_{S} is the corresponding stack length. Notice that initially, I is the identity mapping on the domain $\{1, \ldots, N\}$ since all entries reside in the main database and the stacks are empty.

Reading and Writing. To read or write to the database, clients have to perform two steps: extracting the data (Algorithm 1) and appending an entry to the personal stack (Algorithm 2 for writing and Algorithm 3 for reading).

To extract the payload, the client performs two PIR queries: one on DB for the desired index j and one on the concatenation of all stacks for either a more current version of j or an arbitrary one (lines 1.1–1.8): this is crucial to hide from the server the information on whether or not the client is retrieving a previously modified entry. It then checks the entry's authenticity as provided by $\sigma_{\mathcal{O}}$ and retrieves the verification key used for further verification (line 1.9). The client extracts henceforth the overall payload (line 1.11) from the most current entry (either in DB or on a stack (line 1.10)) and verifies its validity

Algorithm 1 $\{d, \mathsf{deny}\} \leftarrow \langle \mathcal{C}_{\mathsf{extData}}(cap_i, j), \mathcal{S}_{\mathsf{extData}}(\mathcal{DB}) \rangle$

Input: the client capability cap_i and the desired index j to extract the data from
Output: the data d stored at j or deny in case of failure

1: Parse cap_i as $(\mathsf{I}, \mathbf{ek}, vk_{\mathcal{O}}, i_{\mathsf{S}}, len_{\mathsf{S}}, dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$
2: $j' \leftarrow \mathsf{I}(j) - N$ if $\mathsf{I}(j) > N$, otherwise choose j' uniformly at random from $\{1, \ldots, \sum_{i=1}^{M} |\mathsf{S}_i|\}$
3: $q \leftarrow \mathsf{prepRead}(\mathsf{DB}, j)$
4: $q' \leftarrow \mathsf{prepRead}(\mathsf{S}_1 || \ldots || \mathsf{S}_m, j')$
5: Send q, q' to $\mathcal{S}$
6: Receive r, r' from $\mathcal{S}$
7: $e \leftarrow \mathsf{decodeResp}(r)$, $e' \leftarrow \mathsf{decodeResp}(r')$
8: Parse e as $(\mathbf{c}_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ and e' as $(\mathbf{c}'_{\mathsf{Data}})$
9: Abort **if** $\perp = \mathsf{Vrfy}(vk_{\mathcal{O}}, \sigma_{\mathcal{O}}, (j||vrs_{\mathcal{O}}||vk||\mathbf{c}_{\mathsf{BrCast}}))$ or $vrs_{\mathcal{O}}$ is not current
10: $\overline{\mathbf{c}_{\mathsf{Data}}} \leftarrow \mathbf{c}_{\mathsf{Data}}$ **if** $\mathsf{I}(j) \leq N$ and $\mathbf{c}'_{\mathsf{Data}}$ otherwise
11: $(d||vrs||\sigma) \leftarrow \mathsf{D}(dk_i^{\mathsf{CPA}}, \overline{c_{\mathsf{Data}}}^i)$
12: Abort **if** $\perp = \mathsf{Vrfy}(vk, \sigma, d||vrs)$ or vrs is not current
13: **if** $i_{\mathsf{S}} > len_{\mathsf{S}}$ **then**
14: $\quad \mathsf{flush}(cap_i)$, $i_{\mathsf{S}} = 1$
15: **end if**
16: Increment i_{S}
17: **return** d if $d \neq \perp$ and deny otherwise

Algorithm 2 $\langle \mathcal{C}_{\mathsf{repl}}(cap_i, j, vrs, d', \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}), \mathcal{S}_{\mathsf{repl}}(\mathcal{DB}) \rangle$

Input: the client capability cap_i, the desired index j to operate on, the old version number vrs of the j-th entry, the new data d', the broadcast ciphertext $\mathbf{c}_{\mathsf{BrCast}}$, and the authorization ciphertext $\mathbf{c}_{\mathsf{Auth}}$ of the j-th entry

1: Parse cap_i as $(\mathsf{I}, \mathbf{ek}, vk_{\mathcal{O}}, i_{\mathsf{S}}, len_{\mathsf{S}}, dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$
2: $sk \leftarrow \mathsf{D}(dk_i^{\mathsf{CCA}}, c^i_{\mathsf{Auth}})$
3: Increment vrs, $\sigma \leftarrow \mathsf{Sign}(sk, d'||vrs)$
4: Select $\mathbf{r}$ uniformly at random
5: $c^{i'}_{\mathsf{Data}} \leftarrow \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^i_{\mathsf{BrCast}} \cdot (d'||vrs||\sigma), r_i)$ for $i \in \{1, \ldots, M\}$
6: Send $(i, i_{\mathsf{S}}, \mathbf{c}'_{\mathsf{Data}})$ to $\mathcal{S}$
7: $\mathsf{Broadcast}((j||(N + \sum_{k=0}^{i-1} |\mathsf{S}_k| + i_{\mathsf{S}}))||vrs), \mathbf{c}_{\mathsf{BrCast}})$

(line 1.12). Before returning the extracted data (line 1.17), the client flushes the personal stack if it is full (lines 1.13–1.16). We explain this algorithm in the next paragraph. We stress that data extraction is performed independently of whether the client reads or writes. Note that up to this point, since the server only sees PIR queries, it cannot distinguish read and write.

The next step (i.e., adding an entry to the stack), however, requires more care in order to retain obliviousness. In particular, when writing, the client appends an entry to its personal stack that replaces the actual entry in DB (see Algorithm 2). In order to make read and write indistinguishable, when reading, the client appends an entry to its stack which is indistinguishable from a real entry since it is an entry on which no-one holds any permissions (see Algorithm 3). Finally, the client broadcasts the modified index information in write or a zero string in read.

Flushing the stack (Algorithm 4). The flush algorithm pushes the elements in the stack that are up-to-date to DB[3]. In particular, the client first builds an index structure that contains all elements that are up-to-date (ϕ, lines 4.2–4.9) based on the mapping of stack indices to real indices that the client stores implicitly. The client then downloads the stack (line 4.10) and changes every entry of DB (PIR writing). To this end, it downloads and uploads every entry $e_j \in \mathsf{DB}$ (lines 4.12, 4.21, and 4.27).

If the currently downloaded entry is outdated, the client takes the locally stored data from S_i and rerandomizes it (lines 4.14–4.18). Then it computes an *integrity proof* (technically, a NIZK) P that shows the following OR statement: either it is eligible to write the entry by proving that it knows the signing key (line 4.19) corresponding to the verification key (line 4.13) which is authenticated by the data owner, or it only rerandomized the data part (line 4.20). In that notation, the underscore _ refers to hidden variables in the proof that the client does not know.

In case there is no entry in the stack that is more recent, it rerandomizes the current entry in DB (line 4.25) and creates an integrity proof with the same statement as in the previous case, just that now

[3]Some elements may be outdated, since a different user may have the most recent version in its stack.

Algorithm 3 $\langle \mathcal{C}_{\mathsf{addDummy}}(cap_i, \mathbf{c}_{\mathsf{BrCast}}), \mathcal{S}_{\mathsf{addDummy}}(\mathcal{DB}) \rangle$

Input: the client capability cap_i and the broadcast ciphertext $\mathbf{c}_{\mathsf{BrCast}}$

1: Parse cap_i as $(\mathsf{I}, \mathbf{ek}, vk_{\mathcal{O}}, i_{\mathsf{S}}, len_{\mathsf{S}}, dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$
2: Uniformly sample a vector $\mathbf{c}'_{\mathsf{Data}}$ of encryptions of 0
3: Send $(i, i_{\mathsf{S}}, \mathbf{c}'_{\mathsf{Data}})$ to $\mathcal{S}$
4: $\mathsf{Broadcast}(0, \mathbf{c}_{\mathsf{BrCast}})$

Algorithm 4 $\langle \mathcal{C}_{\mathsf{flush}}(cap_i), \mathcal{S}_{\mathsf{flush}}(\mathcal{DB}) \rangle$

Input: the client capability cap_i

1: Parse cap_i as $(\mathsf{I}, \mathbf{ek}, vk_{\mathcal{O}}, i_{\mathsf{S}}, len_{\mathsf{S}}, dk_i^{\mathsf{CPA}}, dk_i^{\mathsf{CCA}})$
2: Initialize $\phi = [\,]$
3: $off \leftarrow |\mathsf{DB}| + \sum_{l=1}^{i-1} |\mathsf{S}_l|$
4: **for** $\ell = len_{\mathsf{S}}$ down to 1 **do**
5: $\quad j \leftarrow$ index of $\mathsf{S}_i(\ell)$ in DB
6: $\quad$ **if** $\phi(j) = \perp$ and $\mathsf{I}(j) = \ell + off$ **then**
7: $\quad\quad \phi \leftarrow \phi[j \mapsto \ell]$, $\mathsf{I} \leftarrow \mathsf{I}[j \mapsto j]$
8: $\quad$ **end if**
9: **end for**
10: Download S_i from $\mathcal{S}$
11: **for** $j = 1$ to $|\mathsf{DB}|$ **do**
12: $\quad$ Download $e_j = (\mathbf{c}_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ from $\mathcal{S}$
13: $\quad$ Extract vk from $\sigma_{\mathcal{O}}$
14: $\quad l \leftarrow \phi(j)$
15: $\quad$ **if** $l \neq \perp$ **then**
16: $\quad\quad$ Parse $\mathsf{S}_i(l)$ as $\mathbf{c}'_{\mathsf{Data}}$
17: $\quad\quad$ Select $\mathbf{r}$ uniformly at random
18: $\quad\quad c^{i^*}_{\mathsf{Data}} \leftarrow \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^{i'}_{\mathsf{Data}}, r_i)$ for $i \in \{1, \ldots, M\}$
19: $\quad\quad sk \leftarrow \mathsf{D}(dk_i^{\mathsf{CCA}}, c^i_{\mathsf{Auth}})$
20: $\quad\quad P \leftarrow PK \left\{ \begin{array}{l} (sk, \_) : \\ \quad (vk, sk) \text{ valid key pair } \vee \\ \quad \forall i.\ c^{i^*}_{\mathsf{Data}} = \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^i_{\mathsf{Data}}, \_) \end{array} \right\}$
21: $\quad\quad$ Send $P, e'_j = (\mathbf{c}^*_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ to $\mathcal{S}$
22: $\quad\quad \mathsf{Broadcast}((j||j||vrs), \mathbf{c}_{\mathsf{BrCast}})$
23: $\quad$ **else**
24: $\quad\quad$ Select $\mathbf{r}$ uniformly at random
25: $\quad\quad c^{i^*}_{\mathsf{Data}} \leftarrow \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^i_{\mathsf{Data}}, r_i)$ for $i \in \{1, \ldots, M\}$
26: $\quad\quad P \leftarrow PK \left\{ \begin{array}{l} (\_, \mathbf{r}) : \\ \quad (vk, \_) \text{ valid key pair } \vee \\ \quad \forall i.\ c^{i^*}_{\mathsf{Data}} = \mathsf{Rnd}(ek_i^{\mathsf{CPA}}, c^{i'}_{\mathsf{Data}}, r_i) \end{array} \right\}$
27: $\quad\quad$ Send $P, e'_j = (\mathbf{c}^*_{\mathsf{Data}}, \mathbf{c}_{\mathsf{BrCast}}, \mathbf{c}_{\mathsf{Auth}}, \sigma_{\mathcal{O}})$ to $\mathcal{S}$
28: $\quad\quad \mathsf{Broadcast}(0, \mathbf{c}_{\mathsf{BrCast}})$ for random r
29: $\quad$ **end if**
30: **end for**

the second part of the disjunction is true (line 4.26). In any case, the client broadcasts the new indices of all updated entries to all clients (line 4.22 for a real update and line 4.28 for a dummy update). We stress that the two proofs created in lines 4.20 and 4.26 are indistinguishable by the zero-knowledge property and hence do not reveal to the server whether the entry is updated or left unchanged, which is crucial for achieving the obliviousness of data accesses.

Adding new clients. In order to grant a new client $\mathcal{C}_i$ access to entries in $\mathcal{DB}$, $\mathcal{O}$ prepares a client capability cap_i as described above in the setup phase. In general, if not all capabilities are created initially, every entry has to be adapted when adding a new client as well as every client's capability. More precisely, for each entry, $\mathcal{O}$ adds a ciphertext to $\mathbf{c}_{\mathsf{Data}}$, $\mathbf{c}_{\mathsf{BrCast}}$ and $\mathbf{c}_{\mathsf{Auth}}$ and every client needs to learn ek_i^{CPA} and ek_i^{CCA}.

Adding new entries. To add a new entry to the database, $\mathcal{O}$ prepares it according to the entry structure and sends it to $\mathcal{S}$. Finally, $\mathcal{O}$ propagates the corresponding information with the respective

index to all clients.

CHANGING ACCESS PERMISSIONS. To change access permissions of a certain entry, $\mathcal{O}$ modifies $\mathbf{c}_{\mathsf{Data}}$, $\mathbf{c}_{\mathsf{BrCast}}$ and/or $\mathbf{c}_{\mathsf{Auth}}$ as well as $\sigma_{\mathcal{O}}$ (with a new version number $vrs_{\mathcal{O}}$) accordingly.

B Proof of the Lower Bound

NOTATION. Without loss of generality we assume a binary database $\mathcal{DB}$ that consists of n entries and each entry consists of exactly one bit data. We denote by idx the identifier of each entry and by $\mathsf{data}_{\mathsf{idx}}$ the value of idx. Note that a single entry can be simultaneously stored on multiple physical addresses in the memory of the database (e.g., the database may maintain multiple copies of the same entry or secret-share some entries across several locations), for this reason we define a predicate that maps identifiers to phisical addresses of the memory of the database. The predicate $\mathsf{loc} : \mathsf{idx} \to (\ell_1, \ldots, \ell_t)$ takes as input an identifier and returns a set of physical memory addresses, for some $t \geq 1$. Note that such a predicate may depend on the database architecture and on the ORAM scheme and it may change whenever some *write* operation is performed on the database. We say that an algorithm *accesses* some physical address if its content is either modified or read during the execution of the algorithm. Throughout the following presentation, we denote by L a given set of physical addresses and by m the amount of physical addresses of the database. We observe that, by definition, we have that $m \geq n$. Finally, for a positive integer n, we let $[n]$ denote the set $\{1, \ldots, n\}$. We also extend this notation to sets of indexed variables; we write, e.g., $[cap_k]$ to denote the set $\{cap_1, \ldots, cap_k\}$.

READ CORRECTNESS. We recall the correctness definition for the read operation in a Multi-Client ORAM, while for the other operations we refer to [39].

Definition 3 (Read Correctness)**.** *The read of a Multi-Client ORAM scheme Θ is* correct*, if for all λ and n, for all $\mathsf{idx} \in [n]$ and $cap_i \in [cap_k]$ such that $\mathsf{ACM}(i, \mathsf{idx}) \neq \bot$, there exists a negligible function $\mu(\cdot)$ such that:*

$$\mathsf{Pr}\left[\mathsf{data}_{\mathsf{idx}} \leftarrow \langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle\right] \geq (1 - \mu(\lambda)).$$

We now introduce some helpful facts which simplify the proof of the main theorem. Note that in the following analysis we only consider the operation read, since read and write must be indistinguishable it is easy to see that the same result extends to write. The first lemma formalizes that in any Multi-Client ORAM scheme, a client who wants to read a certain index necessarily has to access at least one of the physical addresses associated to that index.

Lemma 1. *Let Θ be a Multi-Client ORAM scheme. Then for all $\mathsf{idx} \in [n]$ and $cap_i \in [cap_k]$ with $\mathsf{ACM}(i, \mathsf{idx}) \neq \bot$ there exists a negligible function $\mu(\cdot)$ such that:*

$$\begin{aligned}\mathsf{Pr}\big[&\mathsf{data}_{\mathsf{idx}} \leftarrow \langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle \\ &\textit{accesses } L \wedge L \cap \mathsf{loc}(\mathsf{idx}) \neq \emptyset\big] \geq (1 - \mu(\lambda))\end{aligned}$$

where the probability is taken over the random coins of $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle$.

Proof. The proof follows from the correctness of Θ. Assume towards contradiction that $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle$ does not access any of the physical addresses in $\mathsf{loc}(\mathsf{idx})$ with non-negligible probability $\epsilon(\lambda)$. In that case, the algorithm $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle$ can only guess the bit of idx, and hence returns the correct bit with probability at most $1/2$, which means that $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle$ returns $\mathsf{data}_{\mathsf{idx}}$ with probability at most $(1 - \epsilon(\lambda)/2)$. Since $\epsilon(\lambda)/2$ is non-negligible, we derived a contradiction to Θ's read correctness. □

The following lemma captures the fact that in a Multi-Client ORAM the probability of a random physical address to be accessed during a certain read is proportional to the amount of entries accessed (on average) upon each read.

Lemma 2. *Let Θ be a Multi-Client ORAM scheme whose read operation accesses on average ℓ many addresses (for some $\ell \in [m]$), over the random coins of the read operation. Then for a physical memory address $x \in [m]$ sampled uniformly at random and for all read access sequences $\vec{y}$ of length p, for all $q \in [p]$ it holds that:*

$$\mathsf{Pr}\left[\vec{y}_q \textit{ accesses } x\right] \leq \frac{\ell}{m}$$

where $\vec{y}_q$ denotes the q-th read *operation of $\vec{y}$ and the probability is taken over the random choice of x and the random coins of $\vec{y}$.*

Proof. The proof follows from the fact that x is sampled independently and uniformly at random. Since on average $\langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB})\rangle$ accesses ℓ many memory locations, the probability that x belongs to that set is exactly ℓ/m. □

The following lemma formalizes the intuition that in a secure (against malicious clients) Multi-Client ORAM scheme the probability of accessing a certain memory address upon each read must be independent from the index queried by the client.

Lemma 3. *Let Θ be a Multi-Client ORAM scheme, then for all pairs of* read *access sequences $(\vec{y}, \vec{z})$ of length p, for all $q \in [p]$, and for all memory locations $x \in [m]$ there exists a negligible function $\mu(\cdot)$ such that:*

$$|\mathsf{Pr}\left[\vec{y}_q \text{ accesses } x\right] - \mathsf{Pr}\left[\vec{z}_q \text{ accesses } x\right]| \leq \mu(\lambda)$$

where the probability is taken over the random coins of $\langle \mathsf{read}(\cdot,\cdot), \mathcal{S}(\mathcal{DB})\rangle$.

Proof. Assume towards contradiction that there exists a pair of access sequences $(\vec{y}, \vec{z})$ and a physical memory address x such that at step q:

$$|\mathsf{Pr}\left[\vec{y}_q \text{ accesses } x\right] - \mathsf{Pr}\left[\vec{z}_q \text{ accesses } x\right]| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\cdot)$. This implies that we can construct an adversary $\mathcal{A}$ that non-deterministically samples $(\vec{y}, \vec{z}, x, q)$, does not corrupt any client, queries the pair $(\vec{y}, \vec{z})$ to the query interface of the challenger and returns 1 if x was accessed at step q and 0 otherwise. To analyze the success probability of $\mathcal{A}$ we shall note that:

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] = \mathsf{Pr}\left[\vec{y}_q \text{ accesses } x\right]$$
$$\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right] = \mathsf{Pr}\left[\vec{z}_q \text{ accesses } x\right].$$

By our initial assumption it follows that:

$$|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow| \mid b = 1\right]| \geq \epsilon(\lambda),$$

which is a contradiction to the *security against malicious clients* of Θ. □

We are now ready to formally prove Theorem 1. Note that, even though the security definition for Multi-Client ORAM (see Definition 2) allows for active corruption, the result still holds in case of passive corruption (i.e., the adversary does not impersonate the corrupted instances but receives transcripts of honestly executed operations).

Proof. Assume towards contradiction that there exists a Multi-Client ORAM scheme Θ that accesses on average $o(n)$ physical memory addresses and is secure. Then we can construct an adversary $\mathcal{A}$ as follows.

$\mathcal{A}$ commits to a random client identifier i and it receives a local copy of the database $\mathcal{DB}$. $\mathcal{A}$ then samples a pair of entries $(\mathsf{idx}, \mathsf{idx}') \leftarrow [n]^2$ uniformly at random and assigns i with read permissions on idx via the interface chMode. Furthermore, $\mathcal{A}$ samples a memory location x uniformly at random, an integer $q \in \{1, \ldots, \mathrm{poly}(\lambda)\}$, and a sequence $\vec{w}$ of read operations uniformly at random of length $q-1$. $\mathcal{A}$ then initializes $\vec{y} := \vec{w}||(\mathsf{read}, \mathsf{idx}, i', \perp)$ and $\vec{z} := \vec{w}||(\mathsf{read}, \mathsf{idx}', i', \perp)$ for some client identifier $i' \neq i$ such that $\mathsf{ACM}(i', \mathsf{idx}) = \mathsf{ACM}(i', \mathsf{idx}') = \mathsf{R}$. If such an i' does not exists, then $\mathcal{A}$ generates a new non-corrupted client i' with the appropriate read permissions via the interfaces addCl and chMode. $\mathcal{A}$ queries $(\vec{y}, \vec{z})$ to the query interface. After the $(q-1)$-th step, the adversary corrupts client i and locally simulates $\mathsf{read}(\mathsf{idx}, cap_i)$: if x is not accessed, then $\mathcal{A}$ interrupts the simulation and outputs a random guess. Otherwise $\mathcal{A}$ observes the blocks accessed during the execution of the q-th operation: if x is accessed $\mathcal{A}$ returns 1, otherwise he returns 0.

For the analysis, it is easy to see that $\mathcal{A}$ is efficient. To analyze the success probability of $\mathcal{A}$ we define guess to be the event when the sampled block x belongs to the physical locations of the entry idx and it is accessed in both the simulated read and $\vec{y}$. More formally, guess is the event where $x \in \mathsf{loc}(\mathsf{idx})$ and $x \in L$ and $x \in L'$, where L and L' are the set of physical addresses accessed by $\mathsf{read}(\mathsf{idx}, cap_i)$, $\mathsf{read}(\mathsf{idx}, cap_{i'})$

respectively. Then we can express the probability that $\mathcal{A}$ outputs 1 given that the challenger is executing $\vec{y}$ as:

$$\begin{aligned}\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] = \mathsf{Pr}\left[\vec{y} \text{ accesses } x \mid \mathsf{guess}\right] \mathsf{Pr}\left[\mathsf{guess}\right] \\ + \mathsf{Pr}\left[\vec{y} \text{ accesses } x \mid \overline{\mathsf{guess}}\right] \mathsf{Pr}\left[\overline{\mathsf{guess}}\right].\end{aligned}$$

By definition of guess we have

$$\begin{aligned}\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] = 1 \cdot \mathsf{Pr}\left[\mathsf{guess}\right] \\ + \mathsf{Pr}\left[\vec{y} \text{ accesses } x \mid \overline{\mathsf{guess}}\right] \mathsf{Pr}\left[\overline{\mathsf{guess}}\right].\end{aligned} \tag{5}$$

Now we express the probability of the adversary to output 1 given that the challenger is executing $\vec{z}$:

$$\begin{aligned}\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right] = \mathsf{Pr}\left[\vec{z} \text{ accesses } x \mid \mathsf{guess}\right] \mathsf{Pr}\left[\mathsf{guess}\right] \\ + \mathsf{Pr}\left[\vec{z} \text{ accesses } x \mid \overline{\mathsf{guess}}\right] \mathsf{Pr}\left[\overline{\mathsf{guess}}\right].\end{aligned}$$

We shall note that the memory block x is uniformly distributed over the memory of the database, in particular it holds that x and idx' are independently and uniformly sampled. Thus by Lemma 2 we can rewrite:

$$\begin{aligned}\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right] \leq \frac{\ell}{m} \cdot \mathsf{Pr}\left[\mathsf{guess}\right] \\ + \mathsf{Pr}\left[\vec{z} \text{ accesses } x \mid \overline{\mathsf{guess}}\right] \mathsf{Pr}\left[\overline{\mathsf{guess}}\right].\end{aligned}$$

Since we assumed Θ to be oblivious by Lemma 3 we have

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right] \lesssim \frac{\ell}{m} \cdot \mathsf{Pr}\left[\mathsf{guess}\right] + \mathsf{Pr}\left[\vec{y} \text{ accesses } x \mid \overline{\mathsf{guess}}\right] \mathsf{Pr}\left[\overline{\mathsf{guess}}\right] \tag{6}$$

where $\lesssim$ is like $\leq$ but neglects additive negligible terms. If we consider the difference of the two probabilities, Equations (5) and (6) cancel each other out and we are left with:

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right]\right| \gtrsim \left|\mathsf{Pr}\left[\mathsf{guess}\right] - \frac{\ell}{m}\mathsf{Pr}\left[\mathsf{guess}\right]\right|.$$

Since by definition the two terms are both non-negative we have:

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right]\right| \gtrsim \mathsf{Pr}\left[\mathsf{guess}\right]\left(1 - \frac{\ell}{m}\right). \tag{7}$$

We now observe that

$$\mathsf{Pr}\left[\mathsf{guess}\right] = \mathsf{Pr}\left[x \in L \wedge x \in L' \mid x \in \mathsf{loc}(\mathsf{idx})\right] \mathsf{Pr}\left[x \in \mathsf{loc}(\mathsf{idx})\right]$$

Since the local simulation of $\mathcal{A}$ and the simulation of the challenger have independent random coins, we can rewrite

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{guess}\right] = (\mathsf{Pr}\left[x \in L \mid x \in \mathsf{loc}(\mathsf{idx})\right] \cdot \\ \mathsf{Pr}\left[x \in L' \mid x \in \mathsf{loc}(\mathsf{idx})\right]) \mathsf{Pr}\left[x \in \mathsf{loc}(\mathsf{idx})\right]^2\end{aligned}$$

For all logical indices idx, we have that the probability that $x \in \mathsf{loc}(\mathsf{idx})$ is $\frac{|\mathsf{loc}(\mathsf{idx})|}{m}$, thus

$$\begin{aligned}\mathsf{Pr}\left[\mathsf{guess}\right] = (\mathsf{Pr}\left[x \in L \mid x \in \mathsf{loc}(\mathsf{idx})\right] \cdot \\ \mathsf{Pr}\left[x \in L' \mid x \in \mathsf{loc}(\mathsf{idx})\right]) \left(\frac{|\mathsf{loc}(\mathsf{idx})|}{m}\right)^2\end{aligned}$$

By Lemma 1 we have that $\mathsf{Pr}\left[x \in L \mid x \in \mathsf{loc}(\mathsf{idx})\right] \geq \frac{(1-\mu(\lambda))}{|\mathsf{loc}(\mathsf{idx})|}$ over the random choice of x (the same holds for $\mathsf{Pr}\left[x \in L' \mid x \in \mathsf{loc}(\mathsf{idx})\right]$). Therefore

$$\mathsf{Pr}\left[\mathsf{guess}\right] \geq \left(\frac{(1-\mu(\lambda))}{|\mathsf{loc}(\mathsf{idx})|}\right)^2 \left(\frac{|\mathsf{loc}(\mathsf{idx})|}{m}\right)^2 = \frac{(1-\mu(\lambda))^2}{m^2}$$

We can now substitute to equation (7)

$$\begin{aligned}&|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right]| \\ &\quad \gtrsim \tfrac{(1-\mu(\lambda))^2}{m^2} \cdot \left(1 - \tfrac{\ell}{m}\right) \gtrsim \tfrac{1}{m^2} \cdot \left(1 - \tfrac{\ell}{m}\right).\end{aligned}$$

By assumption $\ell = o(n)$, since $m \geq n$ then $\frac{\ell}{m}$ is non-negligibly smaller than 1, since $\frac{1}{m^2}$ is also a non-negligible positive value it follows that the difference of probabilities is non-negligible. This is a contradiction with respect to the security against malicious clients of Θ and it concludes our proof. □

C Security Proofs

In the following we prove the theorems reported in Section 6. Note that in our proofs we consider the adaptive version of each definition where the attacker is allowed to spawn and corrupt clients without restrictions. As a consequence our instantiation requires us to fix in advance the number of clients M supported by the construction. Alternatively, one could consider the selective versions of the security definitions where the attacker is required to commit in advance to the client subset that he wants to corrupt.

Proof of Theorem 3. Assume towards contradiction that there exists an adversary $\mathcal{A}$ that wins the secrecy game with probability non-negligibly greater than $1/2$ for some non-negligible function $\epsilon(\lambda)$, then we can construct the following reduction against the CPA-security of Π_{PKE}. Note that the CPA-security notion that we consider allows the adversary to query one message pair (m_0, m_1) and to receive the encryption of m_b, depending on the random coin of the challenger, under polynomially-many *independent* public keys. Such an experiment can be proven to be equivalent to the textbook CPA-security notion by a standard hybrid argument. The reduction is elaborated below.

$\mathcal{R}(1^\lambda, (ek_1^*, \ldots, ek_q^*))$. The reduction receives q-many public keys $(ek_1^*, \ldots, ek_q^*)$ and samples a string $\mathbf{m} \in \{0,1\}^M$ uniformly at random. For each client $i \in \{1, \ldots, M\}$ the reduction fixes $ek_i^{\mathsf{CPA}} = ek_i^*$ if $\mathbf{m}_i = 1$, otherwise it samples a key pair $(ek_i^{\mathsf{CPA}}, dk_i^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}_{\mathsf{PKE}}^{\mathsf{CPA}}(1^\lambda)$ and associates the client capability with ek_i^{CPA}. Afterwards the reduction simulates faithfully the operations queried by the adversary on the clients i such that $\mathbf{m}_i = 0$. For the clients i where $\mathbf{m}_i = 1$, the reduction performs the same steps as dictated by the protocol except that it skips the decryption procedure in $\mathcal{C}_{\mathsf{extData}}$. At some point of the execution the adversary outputs $(\mathsf{data}_0, \mathsf{data}_1, j)$ and the reduction returns the pair $(\mathsf{data}_0, \mathsf{data}_1)$ to the challenger, who replies with $(c_1^*, \ldots, c_q^*)$. The reduction writes $\mathbf{c}_{\mathsf{Data}}$ of entry j as follows:

$$c_{\mathsf{Data}}^i = \begin{cases} c_i^* & \text{if } \mathbf{m}_i = 1 \\ \mathsf{E}(ek_i^{\mathsf{CPA}}, 0^{|d||vrs||\sigma|}) & \text{otherwise} \end{cases}$$

At some point of the execution the adversary returns a bit b' that the reduction forwards to the challenger.

The reduction is obviously efficient. Assume that the string $\mathbf{m}$ denotes whether the client i is corrupted depending on the bit $\mathbf{m}_i$. Then, whenever the reduction correctly guesses (event that we denote by guess) the subset of corrupted clients, the simulation perfectly reproduces the inputs that the adversary is expecting since the algorithm $\mathcal{C}_{\mathsf{extData}}$ does not produce any output visible to the server. Note that this event happens with probability at least 2^{-M}. Therefore, whenever the challenger coin is $b = 0$ then the reduction resembles the secrecy game for $b = 0$ and the same holds for $b = 1$. In particular,

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] \mathsf{Pr}\left[\mathsf{guess}\right]$$

and

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right] \mathsf{Pr}\left[\mathsf{guess}\right].$$

Whenever the string $\mathbf{m}$ is not correctly sampled (i.e., $\overline{\mathsf{guess}}$), then we can bind the success probability of the reduction to $1/2$. Therefore we have that

$$\begin{aligned}&|\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right]| = \\ &|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid b = 1\right]| \cdot \mathsf{Pr}\left[\mathsf{guess}\right] \geq \\ &\epsilon(\lambda) \cdot 2^{-M}.\end{aligned}$$

Since M is a fixed constant, this represents a contradiction to the CPA-security of Π_{PKE} and it concludes our proof. □

Proof of Theorem 4. The proof works by gradually modifying the experiment via game hops, in the following we provide the reader with an outline of our formal argument.

$\mathsf{Exp}_0^{\mathcal{A}}(\lambda)$. Resembles exactly the integrity game.

$\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$. For each entry on which only non-corrupted clients have write permissions, we modify $\mathbf{c}_{\mathsf{Auth}}$ to encrypt $0^{|sk|}$.

$\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$. We substitute all the verification algorithms on signatures from the data owner with a default rejection if the data was not previously signed by the challenger itself. The challenger keeps track of the signed data via book-keeping.

$\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$. The honestly generated common reference string for the zero-knowledge proof system is substituted with the trapdoor common reference string.

The proof for each step is elaborated below. Note that we consider only the integrity of the main database and not of the personal stack of each client.

$\mathsf{Exp}_0^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_1^{\mathcal{A}}(\lambda)$. Assume towards contradiction that there exists an adversary such that

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_0^{\mathcal{A}}(\lambda)\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right]\right| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. Then we can construct the following reduction against the CCA-security of Π_{PKE}. Note that we allow the reduction to query polynomially-many message pairs under polynomially-many independent public keys, such a game is equivalent to the textbook notion of CCA-security by standard hybrid argument.

$\mathcal{R}(1^\lambda, (ek_1^*, \ldots, ek_q^*))$. The reduction receives q-many public keys $(ek_1^*, \ldots, ek_q^*)$ and samples a string $\mathbf{m} \in \{0,1\}^M$ uniformly at random. For each client $i \in \{1, \ldots, M\}$ the reduction fixes $ek_i^{\mathsf{CCA}} = ek_i^*$ if $\mathbf{m}_i = 1$, otherwise it samples a key pair $(ek_i^{\mathsf{CCA}}, dk_i^{\mathsf{CCA}}) \leftarrow \mathsf{Gen}_{\mathsf{PKE}}^{\mathsf{CCA}}(1^\lambda)$ and associates the client capability with ek_i^{CCA}. The reduction simulates then all of the operations as specified in the protocol except that the calls of the decryption algorithm for ciphertext encrypted for keys of non-corrupted clients are substituted with queries to the decryption oracle provided by the challenger. Additionally, if an entry can be written only from clients such that for all i $\mathbf{m}_i = 1$, then the reduction sends as many $(sk, 0^{|sk|})$ to the challenger, who replies with c_i^*. The entry $\mathbf{c}_{\mathsf{Auth}}$ is then constructed as follows:

$$c_{\mathsf{Auth}}^i = \begin{cases} c_i^* & \text{if } \mathbf{m}_i = 1 \\ \mathsf{E}(ek_i^{\mathsf{CCA}}, 0^{|sk|}) & \text{otherwise} \end{cases}$$

At some point of the execution the adversary returns a bit b' that the reduction forwards to the challenger.

The reduction is clearly efficient. Also whenever the reduction guesses correctly the subset of corrupted clients (which we denote by guess), it is easy to see that it correctly resembles the inputs of $\mathsf{Exp}_0^{\mathcal{A}}$ when the coin of the challenger is $b = 0$ and $\mathsf{Exp}_1^{\mathcal{A}}$ otherwise. It follows that

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_0^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right]$$

and

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right].$$

Whenever the string $\mathbf{m}$ is not correctly sampled (i.e., $\overline{\mathsf{guess}}$), then we can bind the success probability of the reduction to $1/2$. Therefore we can rewrite

$$\begin{aligned} &\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right]\right| = \\ &\quad \Big|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_0^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right] \\ &\qquad - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right]\Big| \geq \epsilon(\lambda) \cdot 2^{-M} \end{aligned}.$$

Since M is a fixed constant, this represents a contradiction to the CCA-security of Π_{PKE}.

$\underline{\mathsf{Exp}_1^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_2^{\mathcal{A}}(\lambda)}$. Assume towards contradiction that there exists an adversary such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right]\right| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. Then we can construct the following reduction against the existential unforgeability of Π_{DS}.

$\mathcal{R}(1^\lambda, vk)$. The reduction takes as input the verification key vk and sets the verification key of the data owner to be $vk_{\mathcal{O}} = vk$. Then it guesses an index $i \in \{1, \ldots, q\}$, where q is an upper-bound on the number of queries of the adversary to any interface, and starts the simulation of the game. The interfaces are executed as specified in $\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$ except that whenever the algorithm Sign is run on the secret key of the data owner on some message m_i, the reduction queries the oracle provided by the challenger instead. At the q-th query of the adversary the reduction parses the content of the query and checks whether it contains some valid pair $(m, \mathsf{Sign}(sk_{\mathcal{O}}, m))$ such that m was not queried to the signing oracle, if this is the case than it returns $(m, \mathsf{Sign}(sk_{\mathcal{O}}, m))$ to the challenger. The simulation of the adversary is interrupted after the q-th query.

It is easy to see that the reduction is efficient and that it perfectly simulates the input that the adversary is expecting in $\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$. We shall note that the only difference between $\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$ and $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$ is when the adversary is able to output a message-signature pair that is valid under the verification key of the data owner such that the message was not signed by the challenger. We denote this event by forge. By assumption we have that

$$\begin{aligned}\Pr[\mathsf{forge}] &= \left|\Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right]\right| \\ &\geq \epsilon(\lambda).\end{aligned}$$

Since the reduction guesses the query where forge happens with probability $\frac{1}{q}$ and the adversary cannot re-use any previously signed messages (due to the corresponding version number), $\mathcal{R}$ returns a valid forgery with probability $\frac{1}{q} \cdot \epsilon(\lambda)$, which is still non-negligible. Thus, we have derived a contradiction to the existential unforgeability of Π_{DS}.

$\underline{\mathsf{Exp}_2^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_3^{\mathcal{A}}(\lambda)}$. Assume towards contradiction that there exists an adversary such that

$$\left|\Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right] - \Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_3^{\mathcal{A}}(\lambda)\right]\right| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. Then we can construct the following reduction against the extractability of ZKP: the reduction receives either the honestly generated string or the trapdoor one and plugs it in the public parameters of the scheme, the rest of the simulation proceeds as in $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$. It is easy to see that in the former case the reduction perfectly simulates $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$ while in the latter it reproduces the inputs in $\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$. Therefore the advantage of the adversary carries over to the reduction, in particular

$$\Pr[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid \mathsf{crs} \leftarrow \{0,1\}^*] = \Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right]$$

and

$$\Pr\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda)\right] = \Pr\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_3^{\mathcal{A}}(\lambda)\right].$$

Thus we have that

$$\left|\begin{array}{l}\Pr[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid \mathsf{crs} \leftarrow \{0,1\}^*] - \\ \quad \Pr\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{E}(1^\lambda)\right]\end{array}\right| \geq \epsilon(\lambda).$$

Which is a contradiction to the extractability of ZKP.

$\underline{\Pr\left[\mathsf{Exp}_3^{\mathcal{A}}(\lambda) = 1\right] \leq \mu(\lambda)}$. In the previous steps we showed that for all adversaries $\mathsf{Exp}_0^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_3^{\mathcal{A}}(\lambda)$, therefore it is enough to show that the success probability in $\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$ is negligible for any PPT machine $\mathcal{A}$. We note that the adversary can only modify the content of the main database through the $\mathcal{C}_{\mathsf{flush}}$ algorithm, therefore it is enough to argue about the integrity of this operation to bind the success probability of the adversary. Loosely speaking, for changing the content of an entry the adversary must

prove the possession of a signing key, which means that any malicious attempt to circumvent the write access control would necessarily result in the disclosure of a hidden signing key. More formally, assume towards contradiction that there exists an adversary $\mathcal{A}$ such that

$$\mathsf{Pr}\left[\mathsf{Exp}_3^{\mathcal{A}}(\lambda) = 1\right] \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$, then we can build the following reduction against the existential unforgeability of Π_{DS}.

$\mathcal{R}(1^\lambda, vk)$. The reduction takes as input a verification key vk, then it guesses an index $i \in \{1, \ldots, q\}$, where q is an upper-bound on the number of queries of the adversary to the interface for adding entries and changing access permission, and an entry index $j \in \{1, \ldots, N\}$. When the q-th query happens, the reduction sets vk as the verification key associated to the target entry and continues the simulation. Anytime that a signature is required on some message m under the verification key vk, the reduction queries the signing oracle provided by the challenger instead. Afterwards, when the reduction executes $\mathcal{C}_{\mathsf{flush}}$ in interaction with the adversary, it runs the extractor $\mathcal{E}$ to obtain the witness w of the proof of knowledge produced by the adversary on the entry j. The reduction samples a message m not queried to the signing oracle yet and computes $\sigma \leftarrow \mathsf{Sign}(w, m)$. The reduction returns (m, σ) to the challenger and interrupts the execution.

The reduction runs only polynomially bound algorithms, therefore it is efficient. Assume for the moment that the reduction correctly guesses the entry j that the adversary outputs in the challenge phase and the query q that last modifies the entry j in the query phase. Then we note that the simulation does not need to know the secret key associated to vk since it is never encrypted in $\mathbf{c}_{\mathsf{Auth}}$ when only non-corrupted clients have access to it (see hop 1). Also, it must be the case that the subsequent proofs of knowledge for entry j are computed against vk (if not they are automatically rejected, see hop 2). Therefore, due to the correctness of the extractor (that the reduction can execute from hop 3), we have that $\mathcal{R}$ extracts the valid key (and thus returns a valid forgery) with the same probability as the adversary returns a valid proof for a modified entry that no corrupted party has write access to. By assumption this happens with probability $\frac{1}{q} \cdot \frac{1}{N} \cdot \epsilon(\lambda)$, which is non-negligible in the security parameter. This is a contradiction to the existential unforgeability of Π_{DS}, so we can bind

$$\mathsf{Pr}\left[\mathsf{Exp}_3^{\mathcal{A}}(\lambda) = 1\right] \leq \mu(\lambda).$$

This concludes our proof. □

Proof of Theorem 5. The proof is developed by applying the same modifications to the experiment as outlined in the proof for Theorem 4 up to $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$. The indistinguishability arguments follow along the same lines. In the following we prove that the success probability probability of any adversary in the modified version of the tamper resistance game (which we denote by $\mathsf{Exp}_{\mathsf{t}}^{\mathcal{A}}(\lambda)$) is negligible in the security parameter. Since this game is indistinguishable from the original, the theorem holds true. Assume towards contradiction that there exists an adversary $\mathcal{A}$ such that

$$\mathsf{Pr}\left[\mathsf{Exp}_{\mathsf{t}}^{\mathcal{A}}(\lambda) = 1\right] \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$, then we can build the following reduction against the existential unforgeability of Π_{DS}.

$\mathcal{R}(1^\lambda, vk)$. The reduction takes as input a verification key vk, then it guesses an index $i \in \{1, \ldots, q\}$, where q is an upper-bound on the number of queries of the adversary to the interface for adding entries and changing access permission. When the q-th query happens, the reduction sets vk as the verification key associated to the target entry and continues the simulation. Anytime that a signature is required on some message m under the verification key vk, the reduction queries the signing oracle provided by the challenger instead. The rest of the simulation proceeds as specified in $\mathsf{Exp}_{\mathsf{t}}^{\mathcal{A}}(\lambda)$. When the adversary queries the challenge interface on the entry j, the reduction forwards the corresponding $(d||vrs, \sigma)$ to the challenger and interrupts the simulation.

The reduction runs only polynomially bound algorithms, therefore it is efficient. We note that the simulation does not need to know the secret key associated with vk since it is never encrypted in $\mathbf{c}_{\mathsf{Auth}}$

when only non-corrupted clients have access to it (see hop 1). Also it must be the case that the validity of the entry j is checked against vk until another query to the interface for changing access permissions (if not the data is automatically rejected, see hop 2). Thus we have that whenever $\mathcal{R}$ successfully guesses the number of the last query that modifies the entry j that is later on sent from the adversary as a challenge, $\mathcal{R}$ returns a valid forgery with the same probability as the adversary returns a valid entry different from the one in the database maintained by the reduction. By assumption this happens with probability $\frac{1}{q} \cdot \epsilon(\lambda)$, which is non-negligible in the security parameter. This is a contradiction to the existential unforgeability of Π_{DS}, so we can bind

$$\mathsf{Pr}\left[\mathsf{Exp}_{\mathsf{t}}^{\mathcal{A}}(\lambda) = 1\right] \leq \mu(\lambda).$$

This concludes our proof. □

Proof of Theorem 6. The proof of obliviousness for the read protocol follows directly from the privacy of the PIR scheme: it is easy to see that $\mathcal{C}_{\mathsf{extData}}$ is essentially a PIR query, while $\mathcal{C}_{\mathsf{addDummy}}$ is completely independent from the index read. Arguing about the obliviousness of the write protocol requires a more careful analysis. In the following we gradually modify the experiment of obliviousness against malicious clients to obtain a simulation where the write algorithm is indistinguishable from the read (for the entries that the attacker is not allowed to access). The validity of the theorem follows. We hereby outline the modifications that we apply on the simulation:

$\mathsf{Exp}_0^{\mathcal{A}}(\lambda)$. Resembles exactly the experiment of obliviousness against malicious client.

$\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$. We modify the public parameters to include a trapdoor common reference string, that is used later on by the challenger to simulate all of the zero-knowledge proofs during the execution of $\mathcal{C}_{\mathsf{flush}}$.

$\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$. We record all of the data signed with the key of the data owner in a list maintained by the challenger. We then substitute all the verification algorithms on signatures from the data owner with a default rejection if the data does not belong to the list.

$\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$. For all entries that no corrupted client can read, we change the $\mathcal{C}_{\mathsf{repl}}$ algorithm to compute $\mathbf{c}'_{\mathsf{Data}}$ and $\mathbf{c}_{\mathsf{BrCast}}$ as vectors of encryptions of 0.

$\underline{\mathsf{Exp}_0^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_1^{\mathcal{A}}(\lambda)}$. Recall that the zero-knowledge of ZKP guarantees the existence of a simulator $\mathcal{S}$ that generates a common reference string crs and a trapdoor td such that crs is indistinguishable from an honestly generated reference string. The knowledge of td allows for simulating a proof for any statement without knowing the witness. Under these premises we can formally prove our claim. Assume towards contradiction that there exists an adversary $\mathcal{A}$ such that

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_0^{\mathcal{A}}(\lambda)\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right]\right| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. Then we can construct the following reduction against the zero-knowledge of ZKP.

$\mathcal{R}(1^\lambda, \mathsf{crs})$. The reduction plugs the common reference string crs in the public parameters of the scheme and starts the simulation as specified by the original protocols. Whenever the reduction has to compute a proof over a statement stmt_i and a witness w_i, it sends (stmt_i, w_i) to the challenger who replies with a non-interactive proof π_i. The reduction forwards π_i to the adversary and proceeds with the simulation. At some point of the execution the adversary returns a bit b that the reduction forwards to the challenger.

The reduction is obviously efficient. Furthermore it is easy to see that whenever crs and the proofs π_i are honestly generated, the reduction perfectly simulates $\mathsf{Exp}_0^{\mathcal{A}}(\lambda)$, while when they are generated by the simulator $\mathcal{S}$, then the inputs are identically distributed as in $\mathsf{Exp}_1^{\mathcal{A}}(\lambda)$. It follows that

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid \mathsf{crs} \leftarrow \{0,1\}^*\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_0^{\mathcal{A}}(\lambda)\right]$$

and

$$\mathsf{Pr}\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{S}(1^\lambda)\right] = \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_1^{\mathcal{A}}(\lambda)\right].$$

By the initial assumption we have that

$$\left|\begin{array}{l}\mathsf{Pr}\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid \mathsf{crs} \leftarrow \{0,1\}^*\right] - \\ \quad \mathsf{Pr}\left[1 \leftarrow \mathcal{R}(\mathsf{crs}) \mid (\mathsf{crs}, \mathsf{td}) \leftarrow \mathcal{S}(1^\lambda)\right]\end{array}\right| \geq \epsilon(\lambda).$$

This is a contradiction to the zero-knowledge of ZKP.

$\underline{\mathsf{Exp}_1^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_2^{\mathcal{A}}(\lambda)}$. The proof for the indistinguishability of the two experiments follows along the same lines of the second step in the proof of Theorem 4.

$\underline{\mathsf{Exp}_2^{\mathcal{A}}(\lambda) \approx \mathsf{Exp}_3^{\mathcal{A}}(\lambda)}$. Assume towards contradiction that there exists an adversary $\mathcal{A}$ such that

$$\left|\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_3^{\mathcal{A}}(\lambda)\right]\right| \geq \epsilon(\lambda)$$

for some non-negligible function $\epsilon(\lambda)$. Then we can construct the following reduction against the CPA-security of Π_{PKE}. We stress that we allow the reduction to query polynomially-many message pairs under polynomially-many independent public keys; a standard hybrid argument is enough to show that this experiment is equivalent to the textbook notion of CPA-security.

$\mathcal{R}(1^\lambda, (ek_1^*, \ldots, ek_q^*))$. The reduction receives q-many public keys $(ek_1^*, \ldots, ek_q^*)$ and samples a string $\mathbf{m} \in \{0,1\}^M$ uniformly at random. For each client $i \in \{1, \ldots, M\}$ the reduction fixes $ek_i^{\mathsf{CPA}} = ek_i^*$ if $\mathbf{m}_i = 1$, otherwise it samples a key pair $(ek_i^{\mathsf{CPA}}, dk_i^{\mathsf{CPA}}) \leftarrow \mathsf{Gen}_{\mathsf{PKE}}^{\mathsf{CPA}}(1^\lambda)$ and associates the client capability with ek_i^{CPA}. The reduction simulates then the protocols as specified by the construction, ignoring the decryption procedure in the $\mathcal{C}_{\mathsf{extData}}$ algorithm (note that no output is displayed to the server anyway). Additionally, the $\mathcal{C}_{\mathsf{repl}}$ algorithm is modified as follows: for all entries that can only be read by clients i such that for all i $\mathbf{m}_i = 1$, then the reduction sends to the challenger the tuples $(d'||vrs||\sigma, 0)$ and $(j||\ell||vrs, 0)$, where $(j, d', vrs, \sigma, \ell)$ are the index, the data, the version number, the signature, and the new index associated with the target entry. The challenger answers with $(c_{i,0}^*, c_{i,1}^*)$ and the reduction construct the vectors $\mathbf{c}'_{\mathsf{Data}}$ and $\mathbf{c}_{\mathsf{BrCast}}$ as follows:

$$c_{\mathsf{Data}}^{i,\prime} = \begin{cases} c_{i,0}^* & \text{if } \mathbf{m}_i = 1 \\ \mathsf{E}(ek_i^{\mathsf{CPA}}, 0) & \text{otherwise} \end{cases}$$

and

$$c_{\mathsf{BrCast}}^{i} = \begin{cases} c_{i,1}^* & \text{if } \mathbf{m}_i = 1 \\ \mathsf{E}(ek_i^{\mathsf{CPA}}, 0) & \text{otherwise} \end{cases}$$

At some point of the execution the adversary returns a bit b' that the reduction forwards to the challenger.

As it runs only a polynomially bounded algorithm, the reduction is clearly efficient. Assume for the moment that the reduction correctly guesses the subset of clients that the adversary is going to corrupt throughout the execution of the experiment. Then we argue that whenever the challenger samples $b = 0$ the reduction resembles the inputs that the adversary is expecting in $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$, while when $b = 1$ the reduction perfectly reproduces $\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$. In order to see that we point out that the reduction does not need to know the randomness of $\mathbf{c}'_{\mathsf{Data}}$ to compute the proof as it is computed using the simulator and the trapdoor (see hop 1). Also, we observe that in $\mathsf{Exp}_2^{\mathcal{A}}(\lambda)$ any valid pair of vectors $\mathbf{c}'_{\mathsf{Data}}$ and $\mathbf{c}_{\mathsf{BrCast}}$ is always composed by encryptions under the initial set of public keys (otherwise they are automatically rejected, due to hop 2), therefore we can assess that the inputs that the reduction provides to the adversary are correctly distributed. It follows that whenever $\mathbf{m}$ is correctly sampled (guess) we have that

$$\begin{aligned} &\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \\ &\qquad \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right] \end{aligned}$$

and

$$\begin{aligned} &\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right] \mathsf{Pr}\left[\mathsf{guess}\right] = \\ &\qquad \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_3^{\mathcal{A}}(\lambda)\right] \mathsf{Pr}\left[\mathsf{guess}\right]. \end{aligned}$$

Whenever the string $\mathbf{m}$ is not correctly sampled (i.e., $\overline{\mathsf{guess}}$), then we can bind the success probability of the reduction to $1/2$. Therefore we can rewrite

$$\begin{aligned} &|\mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 0\right] - \mathsf{Pr}\left[1 \leftarrow \mathcal{R} \mid b = 1\right]| = \\ &\left|\mathsf{Pr}\left[\mathsf{guess}\right] \left(\mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_2^{\mathcal{A}}(\lambda)\right]\right.\right. \\ &\qquad \left.\left. - \mathsf{Pr}\left[1 \leftarrow \mathcal{A} \mid \mathsf{Exp}_3^{\mathcal{A}}(\lambda)\right]\right)\right| \geq \epsilon(\lambda) \cdot 2^{-M} \qquad . \end{aligned}$$

Since M is a fixed constant, this represents a contradiction to the CPA-security of Π_{PKE}.

$\underline{\mathsf{Exp}_0^{\mathcal{A}}(\lambda) \approx \cdots \approx \mathsf{Exp}_3^{\mathcal{A}}(\lambda)}$. The argument above shows that for all PPT adversaries $\mathcal{A}$ the original experiment for obliviousness against malicious clients is indistinguishable from $\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$. It follows that the success probabilities in the two experiments must be the same (up to a negligible factor in the security parameter). Therefore in order to prove our theorem it is enough to observe that in $\mathsf{Exp}_3^{\mathcal{A}}(\lambda)$, for entries that no corrupted client can read, the algorithm $\mathcal{C}_{\mathsf{repl}}$ is identical to $\mathcal{C}_{\mathsf{addDummy}}$ and that the execution of $\mathcal{C}_{\mathsf{flush}}$ is completely independent of the content of the entries in the client's personal stack. This implies that in this context the read and write operations are indistinguishable, which we initially proved to be oblivious. Since obliviousness must hold only for entries that are not readable by corrupted clients (see Definition 2), the obliviousness of the construction follows as the advantage of any adversary is bound to a negligible function in the security parameter. This concludes our analysis. □