Maliciously Secure Multi-Client ORAM

Matteo Maffei[1], Giulio Malavolta[2], Manuel Reinert[3](✉), and Dominique Schröder[2]

[1] TU Wien, Wien, Austria
matteo.maffei@tuwien.ac.at
[2] Friedrich-Alexander Universität Erlangen-Nürnberg, Nürnberg, Germany
{giulio.malavolta,dominique.schroeder}@fau.de
[3] CISPA, Saarland University, Saarbrücken, Germany
reinert@cs.uni-saarland.de

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 [22] and Estonia [20], collaborative platforms (e.g., Google Docs), and credit score

D. Gollmann et al. (Eds.): ACNS 2017, LNCS 10355, pp. 645–664, 2017.
DOI: 10.1007/978-3-319-61204-1_32

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 [28,39], 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* (ORAM): while the first constructions were highly inefficient [23], recent groundbreaking research paved the way for a tremendous efficiency boost, exploiting ingenious tree-based constructions [2,3,8,14,15,24,32,39,44,45,47], server side computations [26,35], and trusted hardware [5,27,31,42,48].

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 [30]. 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.

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.

Work	**MC**	**MD**	**PI**	**CS**	**Pr**	**AC**	**P**	**S comp.**	**C comp.**	**Comm.**
Franz *et al.* [21]	✓	✗	✓	✗	✗	✓	✗	$O(\sqrt{n})$	$O(\sqrt{n})$	$O(\sqrt{n})$
GORAM [33]	✗	✗	✗	✓	✗	✓	✗	$O(\log(n))$	$O(\log(n))$	$O(\log(n))$
PIR-MCORAM (this work)	✓	✓	✗	✓	✗	✓	✗	$O(n)$	$O(\sqrt{n})$	$O(\sqrt{n})$
BCP-OPRAM [7]	✗	✗	✓	✓	✗	✗	✓	$\Omega(\log^3(n))$	$\Omega(\log^3(n))$	$\Omega(\log^3(n))$
CLT-OPRAM [10]	✗	✗	✓	✓	✗	✗	✓	$O(\log^2(n))$	$O(\log^2(n))$	$O(\log^2(n))$
PrivateFS [49]	✗	✗	✗	✓	✗	✗	✓	$O(\log^2(n))$	$O(1)$	$O(\log^2(n))$
Shroud [31]	✗	✗	✗	✗	✓	✗	✓	$O(\log^2(n))$	$O(1)$	$O(\log^2(n))$
TaoStore [42]	✗	✗	✗	✗	✓	✗	✓	$O(\log(n))$	$O(1)$	$O(\log(n))$
TAO-MCORAM (this work)	✓	✓	✗	✗	✓	✓	✓	$O(\log(n))$	$O(1)$	$O(\log(n))$

Franz *et al.* pioneered the line of work on multi-client ORAM, introducing the concept of delegated ORAM [21]. 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.* [33] 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 [5,31,42,48] assume a trusted proxy performing accesses on behalf of users, with TaoStore [42] 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 [42] guarantees both. Finally, instead of a trusted proxy, BCP-OPRAM [7] and CLT-OPRAM [10] rely on a gossiping protocol while PrivateFS [49] 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) [11] 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 [9] 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 [33] by one order of magnitude.
- To bypass the aforementioned lower bound, we consider the recently proposed *proxy-based Setting* [5,31,42,48,49], 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 [42] guarantees obliviousness in the malicious setting as well as access control.
- We implement PIR-MCORAM and conduct 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 [40]. 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. [41], 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 [41] 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.* [33], 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 ($\bot$), 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 [33]). *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 $j \in \{1, \ldots, n\}$:* $\mathsf{ACM}(i, j) = \bot$. *The algorithm outputs the capability for client $\mathcal{C}_i$.*

$\langle \bot, \mathcal{DB}' \rangle \leftarrow \langle \mathsf{addE}(cap_{\mathcal{O}}, \mathsf{idx}, \mathsf{data}), \mathcal{S}_{\mathsf{addE}}(\mathcal{DB}) \rangle$. *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 $i \in \{1, \ldots, k\}$:* $\mathsf{ACM}(i, \mathsf{idx}) = \bot$*, writes* data *at position* idx *in $\mathcal{DB}$, and outputs the modified database $\mathcal{DB}'$ on $\mathcal{S}$.*

$\langle \bot, \mathcal{DB}' \rangle \leftarrow \langle \mathsf{chMode}(cap_{\mathcal{O}}, \mathsf{idx}, i, p), \mathcal{S}_{\mathsf{chMode}}(\mathcal{DB}) \rangle$. *The change mode algorithmtakes as input the data owner's capability $cap_{\mathcal{O}}$, some index* idx*, a client identifier i, and a permission $p \in \{\mathsf{R}, \mathsf{RW}, \bot\}$ 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 $\mathcal{S}$.*

$\langle \mathsf{data}, \bot \rangle \leftarrow \langle \mathsf{read}(\mathsf{idx}, cap_i), \mathcal{S}_{\mathsf{read}}(\mathcal{DB}) \rangle$. *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 [33] 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 $\mathcal{A}$.*

   $\mathsf{chMode}_{cap_{\mathcal{O}}}(\mathsf{idx}, i, \{\mathsf{R}, \mathsf{RW}, \bot\})$**:** *The challenger runs $\langle \mathsf{chMode}(cap_{\mathcal{O}}, \mathsf{idx}, i, \{\mathsf{R}, \mathsf{RW}, \bot\}), \mathcal{A} \rangle$ 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 $\mathsf{ACM}(i, \mathsf{idx}_0) \neq \bot$ and $\mathsf{ACM}(i, \mathsf{idx}_1) \neq \bot$, 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 a constant fraction 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 our full version [34].

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 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) [11] 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 [37] 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 (Sect. 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 (Sect. 5).

3 PIR-MCORAM

In this section, we present a high-level description of $\mathsf{PIR\text{-}MCORAM}$, a Multi-Client ORAM scheme based on PIR. The full details can be found in our full version [34]. Our construction is inspired by Franz *et al.* [21], 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.* [21], 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 [16, 30], 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 [37].

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 are 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 $O(\log\log(N))$ [17] 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 Sect. 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 Sect. 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 showing that the ciphertexts were correctly rerandomized. In the literature, these proofs are called plaintext-equivalence-proofs (PEPs) 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 [4,25,36] and mix networks [29].

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 $\mathsf{PIR\text{-}MCORAM}$, but also any system based on PEPs. The basic idea behind our solution is to homomorphically compute a pairwise independent hash function [9] 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 a 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 $i \in \{1, \dots, n\}$ 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 $i \in \{1, \dots, n\}$ 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 [18,38]. 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 [19]. 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

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

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

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

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, then the probability $\mathsf{Pr}\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right]$ 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 [9] (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 $\mathsf{Pr}\left[\mathsf{D}(dk, c') = \mathsf{D}(dk, b')\right] = p^{-2}$, 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 [5,42,48]. 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 [42]**.** 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 [42], which implements a variant of a Path-ORAM [46] 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.* [42], 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.* [46] or by using authenticated encryption as suggested by Sahin *et al.* [42]. 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 [33]. 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. We postpone full proofs to our full version [34].

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

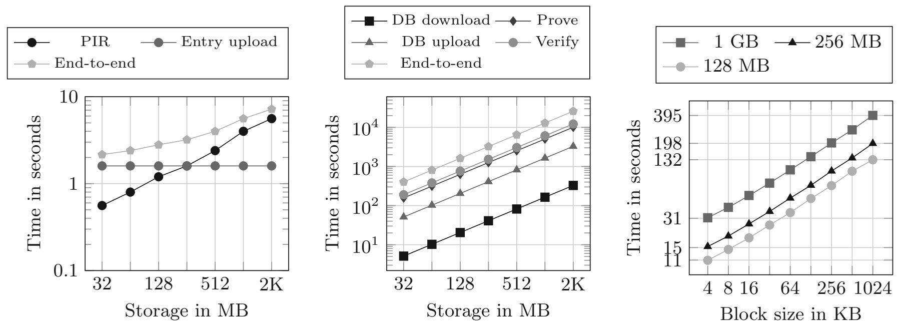
In this section, we describe our implementation and report on the experimental results. We start by reviewing the cryptographic schemes that we deploy: all of them are instantiated with a security parameter of 128 bits [6].

Cryptographic Instantiations. We deploy ElGamal encryption [18] in a hybrid fashion to construct an entry in the database. Using the hybrid technique, we decrease the entry size from $O(MB)$ to $O(M + B)$ since the data is encrypted only once 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 [43] using the Cramer-Shoup encryption scheme [13]. We use XPIR [1], the state of the art in computational PIR.

Finally, in order to construct integrity proofs, we use an OR-proof [12] over a conjunction of plaintext-equivalence proofs [29] (PEP) on the ElGamal ciphertexts forming one entry and a standard discrete logarithm proof [43] 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 20 MB cache for the client and server experiments. We performed micro-benchmarks for PIR-MCORAM while varying the storage size from 32 MB to 2 GB and the block size from 4 KB to 1 MB, 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 [33] 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.



(a) Access without flush, 1 MB block size. (b) Flush, 1 MB block size. (c) Amortized time, varying block size.

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

Discussion. Figures 1 and 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 Fig. 1b depicts the worst case running time (i.e., with flush operation). For the example of the medical record which usually fits into 128 MB (resp. 256 MB for additional files such as X-ray images), the amortized times per access range from 11 (resp. 15) seconds for 4 KB up to 131 (resp. 198) seconds for 1 MB sized entries (see Fig. 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 Sect. 3 and then with the integrity proof based on the universal homomorphic hash (see Sect. 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 1 MB 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 [33], 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).

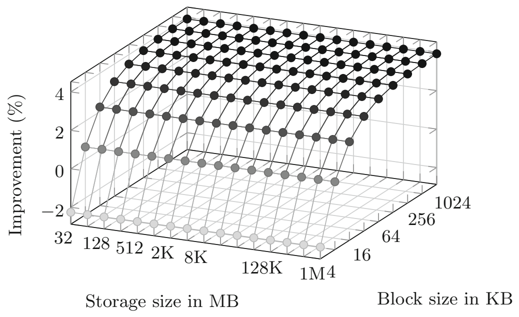


Fig. 2. The improvement in percent when comparing the combined proof computation time on the client and proof verification time on the server for varying storage and block sizes, once without and once with the universal homomorphic hash.

Table 2. Comparison of GORAM [33] with batched shuffle proofs and GORAM instantiated with our homomorphic hash (HH) variant for 10 users, 1 GB storage, and 8 KB block size.

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**

Finally, our solution TAO-MCORAM only adds access control to the actual computation of TaoStore's trusted proxy [42]. 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 [42].

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 [42], 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.

Acknowledgements. This research is based upon work supported by the German research foundation (DFG) through the collaborative research center 1223, by the German Federal Ministry of Education and Research (BMBF) through the Center for IT-Security, Privacy and Accountability (CISPA), and by the state of Bavaria at the Nuremberg Campus of Technology (NCT). NCT is a research cooperation between the Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU) and the Technische Hochschule Nürnberg Georg Simon Ohm (THN). Dominique Schröder is supported by the German Federal Ministry of Education and Research (BMBF) through funding for the project PROMISE. Finally, we thank the reviewers for their helpful comments.

References

1. Aguilar-Melchor, C., Barrier, J., Fousse, L., Killijian, M.O.: XPIR : private information retrieval for everyone. In: Proceedings of the Privacy Enhancing Technologies Symposium (PETS 2016), pp. 155–174. De Gruyter (2016)
2. Ajtai, M.: Oblivious RAMs without cryptographic assumptions. In: Proceedings of ACM Symposium on Theory of Computing (STOC 2010), pp. 181–190. ACM (2010)
3. Apon, D., Katz, J., Shi, E., Thiruvengadam, A.: Verifiable oblivious storage. In: Krawczyk, H. (ed.) PKC 2014. LNCS, vol. 8383, pp. 131–148. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54631-0_8
4. Bayer, S., Groth, J.: Efficient zero-knowledge argument for correctness of a shuffle. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 263–280. Springer, Heidelberg (2012). doi:10.1007/978-3-642-29011-4_17
5. 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: Proceedings of the Conference on Computer and Communications Security (CCS 2015), pp. 837–849. ACM (2015)
6. BlueKrypt: Cryptograhpic Key Length Recommendation. www.keylength.com
7. Boyle, E., Chung, K.-M., Pass, R.: Oblivious parallel RAM and applications. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016 Part II. LNCS, vol. 9563, pp. 175–204. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_7
8. Carbunar, B., Sion, R.: Regulatory compliant oblivious RAM. In: Zhou, J., Yung, M. (eds.) ACNS 2010. LNCS, vol. 6123, pp. 456–474. Springer, Heidelberg (2010). doi:10.1007/978-3-642-13708-2_27
9. Carter, J.L., Wegman, M.N.: Universal classes of hash functions (extended abstract). In: Proceedings of the ACM Symposium on Theory of Computing (STOC 1977), pp. 106–112. ACM (1977)
10. Chen, B., Lin, H., Tessaro, S.: Oblivious parallel RAM: improved efficiency and generic constructions. In: Kushilevitz, E., Malkin, T. (eds.) TCC 2016. LNCS, vol. 9563, pp. 205–234. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49099-0_8

11. Chor, B., Kushilevitz, E., Goldreich, O., Sudan, M.: Private information retrieval. J. ACM **45**(6), 965–981 (1998)
12. Cramer, R., Damgård, I., Schoenmakers, B.: Proofs of partial knowledge and simplified design of witness hiding protocols. In: Desmedt, Y.G. (ed.) CRYPTO 1994. LNCS, vol. 839, pp. 174–187. Springer, Heidelberg (1994). doi:10.1007/3-540-48658-5_19
13. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Krawczyk, H. (ed.) CRYPTO 1998. LNCS, vol. 1462, pp. 13–25. Springer, Heidelberg (1998). doi:10.1007/BFb0055717
14. Damgård, I., Meldgaard, S., Nielsen, J.B.: Perfectly secure oblivious RAM without random oracles. In: Ishai, Y. (ed.) TCC 2011. LNCS, vol. 6597, pp. 144–163. Springer, Heidelberg (2011). doi:10.1007/978-3-642-19571-6_10
15. Dautrich, J., Stefanov, E., Shi, E.: Burst ORAM: minimizing ORAM response times for bursty access patterns. In: Proceedings of the USENIX Security Symposium (USENIX 2014), pp. 749–764. USENIX Association (2014)
16. 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: Proceedings of the Symposium on Principles of Distributed Computing (PODC 1987), pp. 1–12. ACM (1987)
17. Dong, C., Chen, L.: A fast single server private information retrieval protocol with low communication cost. In: Kutyłowski, M., Vaidya, J. (eds.) ESORICS 2014 Part I. LNCS, vol. 8712, pp. 380–399. Springer, Cham (2014). doi:10.1007/978-3-319-11203-9_22
18. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. In: Blakley, G.R., Chaum, D. (eds.) CRYPTO 1984. LNCS, vol. 196, pp. 10–18. Springer, Heidelberg (1985). doi:10.1007/3-540-39568-7_2
19. Fiat, A., Shamir, A.: How to prove yourself: practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987). doi:10.1007/3-540-47721-7_12
20. The EH Foundation. http://www.e-tervis.ee
21. Franz, M., Williams, P., Carbunar, B., Katzenbeisser, S., Peter, A., Sion, R., Sotakova, M.: Oblivious outsourced storage with delegation. In: Danezis, G. (ed.) FC 2011. LNCS, vol. 7035, pp. 127–140. Springer, Heidelberg (2012). doi:10.1007/978-3-642-27576-0_11
22. GmbH, E.: ELGA. https://www.elga.gv.at
23. Goldreich, O., Ostrovsky, R.: Software protection and simulation on oblivious RAMs. J. ACM **43**(3), 431–473 (1996)
24. Goodrich, M.T., Mitzenmacher, M.: Privacy-preserving access of outsourced data via oblivious RAM simulation. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011. LNCS, vol. 6756, pp. 576–587. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22012-8_46
25. Groth, J.: A verifiable secret shuffe of homomorphic encryptions. In: Desmedt, Y.G. (ed.) PKC 2003. LNCS, vol. 2567, pp. 145–160. Springer, Heidelberg (2003). doi:10.1007/3-540-36288-6_11
26. Huang, Y., Goldberg, I.: Outsourced private information retrieval with pricing and access control. In: Proceedings of the Annual ACM Workshop on Privacy in the Electronic Society (WPES 2013). ACM (2013)
27. Iliev, A., Smith, S.W.: Protecting client privacy with trusted computing at the server. IEEE Secur. Priv. **3**(2), 20–28 (2005)

28. Islam, M., Kuzu, M., Kantarcioglu, M.: Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS 2012). Internet Society (2012)
29. Jakobsson, M., Juels, A.: Millimix: mixing in small batches. Technical report, pp. 99–33. DIMACS (1999)
30. Kim, B.H., Lie, D.: Caelus: verifying the consistency of cloud services with battery-powered devices. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P 2015), pp. 880–896. IEEE Press (2015)
31. Lorch, J.R., Parno, B., Mickens, J., Raykova, M., Schiffman, J.: Shroud: ensuring private access to large-scale data in the data center. In: Proceedings of the USENIX Conference on File and Storage Technologies (FAST 2013), pp. 199–214. USENIX Association (2013)
32. 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: Proceedings of the Conference on Computer and Communications Security (CCS 2013), pp. 311–324. ACM (2013)
33. Maffei, M., Malavolta, G., Reinert, M., Schröder, D.: Privacy and access control for outsourced personal records. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P 2015). IEEE Press (2015)
34. Maffei, M., Malavolta, G., Reinert, M., Schröder, D.: Maliciously Secure Multi-Client ORAM. Cryptology ePrint Archive, Report 2017/329 (2017). eprint.iacr.org
35. Mayberry, T., Blass, E.O., Chan, A.H.: Efficient private file retrieval by combining ORAM and PIR. In: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS 2014). Internet Society (2013)
36. Neff, C.A.: A verifiable secret shuffle and its application to e-voting. In: Proceedings of Conference on Computer and Communications Security (CCS 2001), pp. 116–125. ACM (2001)
37. Ostrovsky, R., III, W.E.S.: Algebraic Lower Bounds for Computing on Encrypted Data. In: Electronic Colloquium on Computational Complexity (ECCC), vol. 14, no. 022 (2007)
38. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). doi:10.1007/3-540-48910-X_16
39. Pinkas, B., Reinman, T.: Oblivious RAM revisited. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 502–519. Springer, Heidelberg (2010). doi:10.1007/978-3-642-14623-7_27
40. Robinson, R.J.: How big is the human genome? https://medium.com/precision-medicine/how-big-is-the-human-genome-e90caa3409b0
41. Roche, D.S., Aviv, A., Choi, S.G.: A practical oblivious map data structure with secure deletion and history independence. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P 2016). IEEE Press (2016)
42. Sahin, C., Zakhary, V., Abbadi, A.E., Lin, H.R., Tessaro, S.: TaoStore: overcoming asynchronicity in oblivious data storage. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P 2016). IEEE Press (2016)
43. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Quisquater, J.-J., Vandewalle, J. (eds.) EUROCRYPT 1989. LNCS, vol. 434, pp. 688–689. Springer, Heidelberg (1990). doi:10.1007/3-540-46885-4_68
44. Shi, E., Chan, T.-H.H., Stefanov, E., Li, M.: Oblivious RAM with $O((\log N)^3)$ worst-case cost. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 197–214. Springer, Heidelberg (2011). doi:10.1007/978-3-642-25385-0_11

45. Stefanov, E., Shi, E., Song, D.: Towards practical oblivious RAM. In: Proceedings of the Annual Network & Distributed System Security Symposium (NDSS 2012). Internet Society (2012)
46. Stefanov, E., van Dijk, M., Shi, E., Fletcher, C., Ren, L., Yu, X., Devadas, S.: Path ORAM: an extremely simple oblivious RAM protocol. In: Proceedings of the Conference on Computer and Communications Security (CCS 2013). ACM (2013)
47. Stefanov, E., Shi, E.: Multi-cloud oblivious storage. In: Proceedings of the Conference on Computer and Communications Security (CCS 2013), pp. 247–258. ACM (2013)
48. Stefanov, E., Shi, E.: ObliviStore: high performance oblivious cloud storage. In: Proceedings of the IEEE Symposium on Security & Privacy (S&P 2013), pp. 253–267. IEEE Press (2013)
49. Williams, P., Sion, R., Tomescu, A.: PrivateFS: a parallel oblivious file system. In: Proceedings of the Conference on Computer and Communications Security (CCS 2012), pp. 977–988. ACM (2012)