Sameer Wagh*, Paul Cuff, and Prateek Mittal

Differentially Private Oblivious RAM

Abstract: In this work, we investigate if statistical privacy can enhance the performance of ORAM mechanisms while providing rigorous privacy guarantees. We propose a formal and rigorous framework for developing ORAM protocols with statistical security viz., a *differentially private ORAM* (DP-ORAM). We present Root ORAM, a family of DP-ORAMs that provide a tunable, multi-dimensional trade-off between the desired bandwidth overhead, local storage and system security.
We theoretically analyze Root ORAM to quantify both its security and performance. We experimentally demonstrate the benefits of Root ORAM and find that (1) Root ORAM can reduce local storage overhead by about 2× for a reasonable values of privacy budget, significantly enhancing performance in memory limited platforms such as trusted execution environments, and (2) Root ORAM allows tunable trade-offs between bandwidth, storage, and privacy, reducing bandwidth overheads by up to 2×-10× (at the cost of increased storage/statistical privacy), enabling significant reductions in ORAM access latencies for cloud environments. We also analyze the privacy guarantees of DP-ORAMs through the lens of information theoretic metrics of Shannon entropy and Min-entropy [16]. Finally, Root ORAM is ideally suited for applications which have a similar access pattern, and we showcase its utility via the application of Private Information Retrieval.

Keywords: Oblivious RAM, Differential Privacy

DOI 10.1515/popets-2018-0032
Received 2018-02-28; revised 2018-06-15; accepted 2018-06-16.

1 Introduction

Oblivious RAM (ORAM), first introduced by Goldreich and Ostrovsky [26, 27], is a cryptographic primitive which allows a client to protect its data access pattern from an untrusted server storing the data. Since its introduction, substantial progress has been made by the research community in developing novel and efficient ORAM schemes [10, 22, 25, 37, 40, 52, 53, 55]. Recent work has also shown the promise of using ORAMs as a critical component in developing protocols for Secure Multi-Party Computation [25].

ORAMs can mitigate side-channel attacks [17, 34] in two typical deployment contexts: (1) Trusted Execution Environments such as SGX-based enclaves [33], involving communications between last-level cache (LLC) and DRAM, and (2) Client-server environments, such as communications between smartphones and cloud servers. However, a key bottleneck in the practical deployment of ORAM protocols in these contexts is the performance overhead. For instance, even the most efficient ORAM protocols [37, 40, 54, 55] incur a logarithmic bandwidth overhead (>100×-200×) as well as a logarithmic local storage/stash overhead[1]. Bandwidth is considered the typical bottleneck for ORAM deployment in client-server applications, while memory is the typical bottleneck for the ORAM deployment in trusted execution environments. This lack of low-overhead ORAMs, despite considerable efforts from the security community, is an undeniable indicator for the need of a fundamentally new approach.

In this paper, we propose a novel approach for developing practical ORAM protocols. Our key idea is to trade-off performance at the cost of quantified statistical privacy. We first formalize the notion of a *differentially private ORAM* that provides statistical privacy guarantees. As the name suggests, we use the differential privacy framework developed by Dwork *et al.* [20] with its (ϵ, δ)-differential privacy modification [21]. In the current formulation of an ORAM, the output is computationally indistinguishable for any two input sequences. In a differentially private ORAM, we characterize the effect of a small change in the ORAM input to the change in the probability distribution at the output.

This formalization of a differentially private ORAM subsumes the current notion of ORAM security viz., $\epsilon = 0$ leads to the currently accepted ORAM security definition in Section 2. Yet such a formalization opens up a large underlying design space currently not considered by the community. We also present Root ORAM, a tunable family of ORAM schemes allowing variable bandwidth overheads, system security and outsourcing

***Corresponding Author: Sameer Wagh:** Princeton University, E-mail: swagh@princeton.edu
Paul Cuff: Renaissance Technonogies, E-mail: paul.cuff@gmail.com
Prateek Mittal: Princeton University, E-mail: pmittal@princeton.edu

1 Recent work such as Circuit ORAM [57] require constant local memory but increase the protocol round complexity, thereby increasing the effective bandwidth.

ratios while providing quantified privacy guarantees of differentially private ORAMs. Root ORAM is not a silver bullet for all applications (see Section 7 for enabling requirements). But we hope that this first step in the direction of statistically private ORAMs opens the door for the research community to build more efficient ORAM protocols. In Section 7, we demonstrate an application where DP-ORAM provides a promising solution to the problem of private information retrieval.

1.1 Our Contributions

Root ORAM introduces a number of paradigm shifts in the design of ORAM protocols while building on the prevailing ideas of contemporary ORAM constructions. Our main contributions are:

Formalizing *differentially private ORAMs*: We formalize the notion of a *differentially private ORAM*, which to the extent of our knowledge is the first of its kind. A differentially private ORAM bounds the information leakage from memory access patterns of an ORAM protocol. For details, refer to Section 2.

Tunable protocol family: We propose a tunable family of ORAM protocols called Root ORAM. These schemes can be tailored as per the needs and constraints of the underlying application to achieve a desirable trade-off between security, bandwidth and local storage. This serves as a key enabler for practical deployment and is discussed in more detail in Section 6.

Security and Utility: We analyze and provide theoretical guarantees for the security offered by Root ORAM schemes in the proposed differentially private ORAM framework. The proofs are general and will be useful for analyzing the security of alternative statistically private ORAM schemes in the future. We also theoretically analyze the utility benefits of using statistical privacy. These results are supported by extensive experiments using a complete implementation of Root ORAM. The central results of this paper are summarized below (for details, refer to Section 5).

- We prove that the family of Root ORAM protocols described in Section 4 satisfy the (ϵ, δ)-differential privacy guarantees and give the relation between ϵ, δ and the model parameters.
- We concretely show the benefits of using differential privacy i.e., we demonstrate how a larger value of ϵ helps reduce the protocol overheads, thereby showing an explicit security-performance trade-off.

Practical Impact and Applications: We experimentally investigate the impact of DP-ORAM in the following contexts:

- To reduce local storage requirements to run ORAM protocols in trusted hardware. Trusted execution environments such as the first generation Intel Skylake SGX processors have stringent memory constraints, with available memory for implementing programs (including the ORAM overhead) as low as 90MB [48]. For reasonable values of the privacy budget ϵ, Root ORAM reduces local storage by more than 2×, thereby enhancing compatibility with Intel SGX (Section 6.3).
- To reduce the bandwidth in embedded computing and IoT applications, where devices have limited available bandwidth. Depending on the system parameters chosen, DP-ORAM can reduce the bandwidth overhead by 2×-10× at the cost of statistical security and higher local storage.
- We also demonstrate how statistical ORAMs in conjunction with trusted hardware can be used to perform differentially private PIR queries [56]. We justify the trade-off between enhanced performance at the cost of quantified statistical security for PIR protocols in Section 7.

Root ORAM enables novel design points in developing ORAM protocols by leveraging the benefits of statistical privacy. It also supports design points with order of magnitude performance improvements over state-of-the-art protocols (at the cost of a quantified loss in security). Finally, *Root ORAM does not assume any server-side computation* and requires practical amounts of client-side storage (depending on the parameters chosen). It is also extremely simple to implement at both the client and the server side.

2 Differentially Private ORAM

The notion of statistical privacy has been around in security/privacy applications [9, 39, 45] yet it has never been previously explored in the context of ORAMs. We believe formulating such a framework would greatly expand the ability of the research community to develop novel ORAM protocols with low-bandwidth and low-client overhead, serving as an enabler for real-world deployment of this technology.

Formally, an ORAM is defined as a protocol (possibly randomized) which takes an input access sequence $\mathbf{a}$ as given below,

$$\mathbf{a} = ((\mathtt{op_M}, \mathtt{addr_M}, \mathtt{data_M}), ..., (\mathtt{op_1}, \mathtt{addr_1}, \mathtt{data_1})) \quad (1)$$

and outputs a resulting output sequence denoted by $\mathbf{o} = \mathtt{ORAM}(\mathbf{a})$. Here, M is the length of the access sequence,

$\mathtt{op_i}$ denotes whether the $\mathtt{i}^{\mathrm{th}}$ operation is a read or a write, $\mathtt{addr_i}$ denotes the address for that access, and $\mathtt{data_i}$ denotes the data (if $\mathtt{op_i}$ is a write). Denoting by $|\mathbf{a}|$ the length of the access sequence $\mathbf{a}$, the currently accepted security definition for ORAM security can be summarized as follows [55]:

Definition 1. ***(Currently accepted ORAM Security):*** *Let* $\mathbf{a}$ *as given in Eq. 1, denote an input access sequence. Let* $\mathbf{o} = \mathtt{ORAM}(\mathbf{a})$ *be the resulting randomized data request sequence of an ORAM algorithm. The ORAM guarantees that for any two sequences* $\mathbf{a}$ *and* $\mathbf{a}'$*, the resulting access patterns* $\mathtt{ORAM}(\mathbf{a})$ *and* $\mathtt{ORAM}(\mathbf{a}')$ *are computationally indistinguishable if* $|\mathbf{a}| = |\mathbf{a}'|$*, and also that for any sequence* $\mathbf{a}$ *the data returned to the client by ORAM is consistent with* $\mathbf{a}$ *(i.e the ORAM behaves like a valid RAM) with high probability.*

This framework for ORAMs is constructed with complete security at its core [25, 37, 40, 55] and there is no natural way to extend this to incorporate a statistical privacy notion. Hence, we introduce and formalize a statistically private ORAM viz., *differentially private ORAM (DP-ORAM).*

2.1 Formalizing DP-ORAM

The intuition behind a DP-ORAM is that given any two input sequences that differ in a single access, the distributions of their output sequences should be "close". In other words, similar access sequences lead to similar distributions. We formally define it as follows:

Definition 2. ***Differentially Private ORAM:*** *Let* $\mathbf{a}$*, as defined in Eq. 1, denote the input to an ORAM. Let* $\mathbf{o} = \mathtt{ORAM}(\mathbf{a})$ *be the resulting randomized data request sequence of an ORAM algorithm. We say that an ORAM algorithm is* (ϵ, δ)*-differentially private if for all input access sequences* $\mathbf{a_1}$ *and* $\mathbf{a_2}$*, which differ in at most one access, the following condition is satisfied by the ORAM,*

$$Pr[\mathtt{ORAM}(\mathbf{a_1}) \in S] \leq e^{\epsilon} Pr[\mathtt{ORAM}(\mathbf{a_2}) \in S] + \delta \qquad (2)$$

where e is the base of the natural logarithm and S *is any set of output sequences of the ORAM.*

First we note that we lose no generality by using this definition: it can capture the existing computational ORAM security paradigm using $\epsilon = 0$ and negligible δ. The formalism also does not make any assumption about the size of the output sequences in S. If the input to the ORAM is changed by a single access tuple $(\mathtt{op_i}, \mathtt{addr_i}, \mathtt{data_i})$, the output distribution does not change significantly. Given two sequences $\mathbf{a_1}$ and $\mathbf{a_2}$, the two distributions generated (the red and the blue) are close to each other in the differential privacy sense.

Differential privacy provides two important composability properties [20] viz, "composition" and "group privacy". The former (Theorem 1) refers to the degradation of privacy guarantees over multiple invocations of a differentially private mechanism and the latter (Theorem 2) refers to the privacy guarantees when neighboring databases differ by multiple entries. Together, they give privacy bounds for arbitrary sequences and provide rigorous privacy guarantees over multiple invocations or when access sequences differ in multiple accesses.

Theorem 1 (Composition for DP-ORAM)***. Invoking an*** (ϵ, δ)***-differentially private ORAM mechanism*** m ***times guarantees*** $(m\epsilon, m\delta)$***-differential privacy.***

Theorem 2 (Group privacy for DP-ORAM)***. An*** (ϵ, δ)***-differentially private ORAM is*** (ϵ', δ')***-differentially private for access sequences differing by*** m ***accesses where*** $\epsilon' = m\epsilon$ ***and*** $\delta' = me^{m-1}\delta$***. In other words, given two access sequences*** $\mathbf{a_1}$ ***and*** $\mathbf{a_2}$ ***that differ in*** m ***accesses.***

$$Pr[\mathtt{ORAM}(\mathbf{a_1}) \in S] \leq e^{\epsilon'} Pr[\mathtt{ORAM}(\mathbf{a_2}) \in S] + \delta' \qquad (3)$$

Theorem 1 holds even for adaptive queries as long as the randomness used in each mechanism is independent of each other. Together, *Theorem 1 and 2 allow us to extend differential privacy guarantees to arbitrary access sequences from the guarantees for a single invocation on access sequences that differ by a single access.* It is important to note since privacy guarantees degrade with both the number of invocations and the worst case hamming distance between access sequences, DP-ORAMs are best suited for applications where the input sequences differ in a small number of accesses. *We present a case study of such an application - Private Information Retrieval (PIR) - in Section 7.*

PIR is a cryptographic primitive for privately accessing data from a public database. ORAM schemes can be used in conjunction with trusted hardware to perform PIR queries [7, 59]. We demonstrate the utility of statistical ORAMs by showing how DP-ORAM can be used in conjunction with trusted hardware to perform efficient DP-PIR queries [56]. This application is well suited to showcase the benefits of using statistical ORAMs as *each PIR query corresponds to an access sequence of exactly one element.*

Symbol	Description
$N = 2^L$	Number of real data blocks outsourced
$0 \le k \le L+1$	Model parameter (to tune bandwidth)
$p \in [0,1]$	Model parameter (to tune security)
Z	Number of blocks in each bucket

Table 1. Notation for Root ORAM

3 Root ORAM overview

In this section, we describe our key design goals and give an overview of the Root ORAM protocol.

3.1 Design Goals

Statistically private ORAMs: We target protocols that offer performance benefits at the cost of statistical privacy which is quantified using the metric of differential privacy.

Tunable ORAM schemes: Conventional ORAM schemes operate at specific overheads with full privacy but cannot operate at lower overheads. We aim to provide an ORAM architecture that can be tuned to application requirements and can achieve privacy proportional to system resources such as the bandwidth and local storage.

Rigorous Analysis and Efficiency: We target systems amenable to rigorous security analysis. At the same time, we aim for efficient systems that can be easily implemented on both client and server side.

Finally, the design should use low storage both at the client as well as the server side. *Server side computation is not always practical and hence we do not assume any such capability.* Next we describe the key ideas of Root ORAM protocol. Over the years, several different definitions have been used to quantify ORAM bandwidth overhead.

We will use the original and straightforward definition of bandwidth as the average number of blocks transferred for one access [37].

Definition 3. *The bandwidth cost of a storage scheme is given by the average number of blocks transferred in order to read or write a single block.*

3.2 Approach Overview

Root ORAM protocol can broadly be split into four components, the storage, the access, the new mapping and the eviction. These are briefly described below. The notation for Root ORAM is illustrated in Table 1. Tree-based ORAMs make a relatively easy proof-of-concept to demonstrate the benefits of DP-ORAMs and hence we construct Root ORAM as a tree-based ORAM.

Storage: The data to be outsourced is assumed to be split into units called blocks. Blocks are stored at the server-side storage in 2^k binary trees, each with a depth of $L-k$. For simplicity of proofs, we call each of these trees as "sub-trees" as they can be thought of as subtrees of a larger virtual tree (cf Fig. 1). Each node is a bucket that can hold up to Z data blocks (Z is typically a small constant such as 4 or 5). This is represented in Fig. 1. A stash at the client is used to store a small amount of data. Each data block is mapped to a leaf and this mapping is stored recursively in smaller ORAMs.

Access: The main invariant is that any data block is along the path from the associated leaf to the corresponding sub-tree root or is in the stash (as shown in Fig. 1). Hence, to access a data block, the client looks up the mapping to find the sub-tree and the associated leaf that the data block is mapped to and then traverses the path from that leaf to the sub-tree root.

New Mapping: The data block is then read or written with the new data and then mapped to a new leaf. It is important to note that this new mapping is not uniform among the leaves. The flexibility and the choice of this non-uniform distribution is given in Section 4.

The intuition behind using a non-uniform distribution is that it provides performance benefits such as improving stash storage (refer to Theorem 4). At the same time, we theoretically quantify the security impact of non-uniform distributions using the framework of differential privacy (refer to Theorem 3).

Eviction: Finally, new randomized encryptions are generated and all the data (including some blocks from the stash) are written to the accessed path with blocks being pushed as further down the path as possible (towards the leaf). Root ORAM also uses the recursion technique developed previously [22, 54, 55] to store the mapping in smaller ORAMs.

3.3 Comparison with Path ORAM [55]

Root ORAM is a generalization of the Path ORAM protocol [55], yet there are critical differences between the two protocols. In this subsection, we highlight some of the critical differences between the two papers.

Differentially Private ORAM: Root ORAM introduces a new metric to quantify ORAM security, which extends the current formalism to include the notion of a statistically private ORAM. We bound the privacy offered by the Root ORAM (as well as Path ORAM) using this metric.

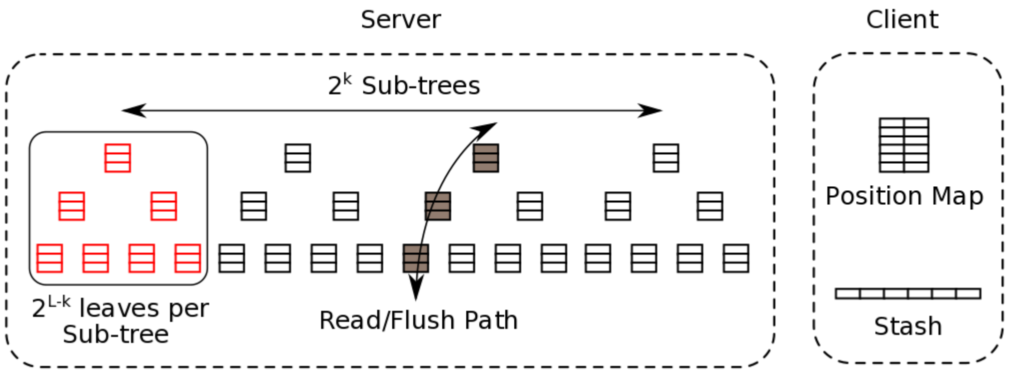


Fig. 1. The figure illustrates the client and server side storage. At the server side, there are 2^k sub-trees, each having a depth of $L-k$. An individual sub-tree is boxed and shown in red.

Tunable Statistical ORAM: Path ORAM incurs a fixed bandwidth cost that cannot be tuned. Thus, applications that cannot accommodate high bandwidth costs are unable to achieve access pattern security. Root ORAM on the other hand is tunable and applications with limited bandwidth can achieve security proportional to their resources.

Multi-dimensional design space: We demonstrate the feasibility of new design points by showing a multi-dimensional trade-off between bandwidth, security and client storage. We support a range of operating conditions by tuning the protocol parameters and demonstrate the trade-off between resource overheads and statistical privacy both theoretically and experimentally.

Note that Path ORAM is an instantiation of Root ORAM for $k = 0$. For more details, see the remark at the end of Section 6.

4 Root ORAM details

In this section, we provide the details of Root ORAM. Basic notation is given in Table 1. B denotes size of each block in bits, $P(x)$ denotes path from leaf x to the sub-tree root, $P(x, i)$ the node at level i in $P(x)$ and $x := \texttt{position}[\mathrm{a}]$ indicates data block a is currently mapped to leaf x.

4.1 Server Storage

Server Storage: The server stores data in the form of 2^k binary trees as shown in Fig. 1. Each node of the tree is a bucket containing multiple data blocks, real or dummy (a dummy block is a randomized encryption of 0). For the simplicity of analysis, we consider the roots of these sub-trees to be at level k and subsequent levels $k+1, \dots L$ where L would correspond to the leaves of each sub-tree.

Bucket structure: Each node is a bucket consisting of Z blocks, each block can either be real or dummy (encryptions of 0).

Path structure: The leaves are numbered in the set $\{0, 1, ..., 2^L - 1\}$. $P(x)$ denotes the path (set of buckets along the way) from leaf x to the sub-tree root and $P(x, i)$ denotes the bucket in $P(x)$ at level i. It is important to emphasize here that the path length in Root ORAM is $(L+1-k)$ blocks compared to the $L+1$ blocks in Path ORAM.

Dummy blocks and randomized encryption: We use the standard padding technique (fill buckets with dummy blocks when needed) along with randomized encryption to ensure indistinguishability of real and dummy blocks.

4.2 Invariants of the scheme

Main Invariant: The main invariant in Root ORAM is that each *real* data block a is mapped to a leaf $x := \texttt{position}[\mathrm{a}], x \in \{0, 1, 2, ..., 2^L - 1\}$ and at any point in the execution of the ORAM, the real block will be somewhere in a bucket $\in P(x)$ or in the local Stash. This path is from the root of a sub-tree to the leaf x and consists of $L - k + 1$ buckets. It is also important to note that the invariant does not say that the mapping of each data block is uniform over the set of leaves, as shall be clarified by the second invariant.

Secondary Invariant: We maintain the secondary invariant that after each access to a data block, its mapping changes according to a leaf dependent non-uniform distribution D (i.e., its new mapping is randomly sampled from this distribution D). There is tremendous flexibility in choosing this distribution; for our purposes, we consider a distribution in which a data block is more likely to be remapped to another leaf in the same subtree than to another sub-tree leaf. This distribution D is formally given by Eq. 4 and shown graphically in Fig. 2.

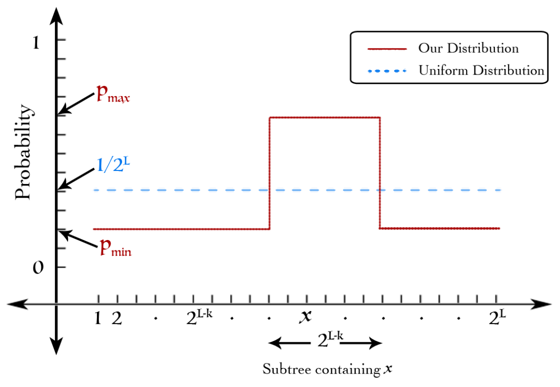


Fig. 2. New mapping of a block is more likely among the current sub-tree than any other sub-tree (red). Alternative distributions turn out to be sub-optimal for Root ORAM.

$$P_{z,x} = p_{\min} + (p_{\max} - p_{\min})\delta_{r_z r_x} \tag{4}$$

Where $P_{z,x}$ is the probability that the new mapping is leaf z given the previous mapping was leaf x, r_x denotes the root of the sub-tree of leaf x, δ_{ij} is the Kronecker delta defined as

$$\delta_{ij} = \begin{cases} 0 & \text{if } i \neq j \\ 1 & \text{if } i = j \end{cases}$$

and $p_{\max}$ and $p_{\min}$ are functions of the model parameter p and are given by:

$$\begin{aligned} p_{\max} &= \frac{1 + (2^k - 1)p}{N} \\ p_{\min} &= \frac{1 - (1 - \delta_{k0})p}{N} \end{aligned} \tag{5}$$

The reason behind using a non-uniform distribution is that it gives performance benefits such as lower stash usage, captured theoretically in Theorem 4. This particular choice of D happens to be ideal for Root ORAM as can be seen by the analysis from Section 5. Theorem 3 gives the relation between the model parameter p and the desired level of privacy (given by ϵ). In practice, the acceptable privacy budget ϵ would decide the parameter p used in the model. We refer the reader to Section 6.4 for details of choosing the parameters.

4.3 Client Storage

Position Map: The client side stores a position map which maps real data blocks to leaves of the server tree. This position map is stored recursively using smaller ORAMs. The recursion technique [22, 54, 55] aims to recursively store the ORAM position maps into subsequently smaller ORAMs. The final ORAM position map is stored locally.

Stash: The client maintains a local stash, which is a small amount of storage locally at the client which is used to store overflown data blocks locally[2].

4.4 Protocol Details

The main functions of the protocol are Access and updateMapping. In the former, we read blocks along a path of a sub-tree, try to write blocks back to the same path (with new encryptions) and if there is insufficient storage, the excess data blocks are stored locally in the Stash. The latter function generates a distribution where a data block is more likely to be remapped to another leaf in the same sub-tree than to another sub-tree leaf. We use sub-tree$_a$ to denote the sub-tree the data block a is currently mapped to, $\cup$ to denote union and $\setminus$ to denote removal from stash.

```
Access (op, a, data*) :
1: $x \leftarrow$ position[a]
2: position[a] = updateMapping(a)
3: Stash = Stash $\cup$ $P(x)$
4: data $\leftarrow$ Read(a) from Stash
5: if op = write then
6:     Stash = Stash $\setminus$(a, data) $\cup$ (a, data*)
7: end if
8: flush($x$)
9: return data
```

The updateMapping function implements the new mapping function as described in Section 4.2. The values of the probabilities are as shown in Fig. 2 (and Eq. 5).

```
updateMapping (a) :
1: $x \leftarrow$ UniformReal(0,1)
2: if $x \le N \cdot p_{\min}$ then
3:     return Uniform$(0, 1, \ldots, 2^L - 1)$
4: else
5:     return Uniform(sub–tree$_a$)
6: end if
```

The flush(x) function is implemented by writing blocks from the stash into the sub-tree, along the path from the associated leaf to the sub-tree root while writing them as low in the sub-tree as possible. The pseudocode for flush(x) is given below.

2 The stash can be stored on the server at the cost of an increase in the bandwidth whereas in our approach we provide a way to reduce the stash without impacting the system bandwidth and hence store the stash locally.

```
flush $(x)$ :
1: for $l = L : k$ do
2:     $y \leftarrow P(x, l)$
3:     $S' = \{(\mathrm{a}', \mathrm{d}') \in \text{Stash s.t. } P(\texttt{position}[\mathrm{a}'], l) = y\}$
4:     $S' = \min(|S'|, Z)$ blocks from $S'$
5:     Stash = Stash $\setminus S'$
6:     writeToBucket$(y, S')$
7: end for
```

5 Theoretical evaluation

5.1 Notation

We begin by developing some notation to present the central results of this paper. We fix $N = 2^L$ to be the total number of outsourced blocks. We denote by $\mathbb{O}^Z_{k,p}$, the Root ORAM protocol with bucket size Z and model parameters k, p. We define the sequence of load/store operations by $\mathbf{s} = (\mathbf{a}, \mathbf{x}, \mathbf{y})$ where $\mathbf{a} = \{a_i\}_{i=1}^M$ are the logical block addresses loaded/stored, $\mathbf{x} = \{x_i\}_{i=1}^M$ is the sequence of leaf labels seen by the server and $\mathbf{y} = \{y_i\}_{i=1}^M$ is the sequence of new leaf labels.

Let $\texttt{st}\left[\mathbb{O}^Z_{k,p}(\mathbf{s})\right]$ denote the random variable which equals the number of real data blocks in the stash after a sequence of load/store operations $\mathbf{s}$, $\mathbb{O}^\infty_{k,p}$ denote ∞-Root ORAM[3] and $\texttt{st}^Z[\mathbb{O}^\infty_{k,p}(\mathbf{s})]$ denote the stash usage after greedy post-processing. The greedy post-processing takes an ∞-Root ORAM and reassigns blocks so that each bucket has no more than Z blocks (for details refer to [55]). The main results of this paper are split into two main categories, Section 5.2 which states and proves the security results and Section 5.3 which states and proves the performance results.

5.2 Security Results

Theorem 3 (Differentially Private Protocols). ***The Root ORAM protocol with parameters*** k, p ***is*** (ϵ, δ)***-differentially private for the following choice of*** ϵ ***and*** δ

$$\epsilon = 2\log\left(\frac{1 + (2^k - 1)\cdot p}{1 - (1 - \delta_{k0})p}\right)$$
$$\delta = M \cdot \left(\frac{1 + (2^k - 1)\cdot p}{N}\right)^M \tag{6}$$

where δ_{k0} ***is the Kronecker delta,*** M ***is the size of the access sequence and*** $M >$ ***total stash size.***

Proof of Theorem 3: Using a conservative security analysis, we prove the bounds on Root ORAM protocols given in Theorem 3. The proof is split into two components, viz., the ϵ bound and the δ bound. For the ϵ bound, we first set up the differential privacy framework, then a model to evaluate probabilities of a given input sequence leading to a specific output sequence and finally compute the maximum change that could result from a change in the input. For the δ bound, we first demonstrate the significance and need for δ in the security bound and then proceed to conservatively prove the δ bound.

5.2.1 The ϵ bound:

We follow the notation described in Section 5.1. We consider two input sequences $\mathbf{a_1}$ and $\mathbf{a_2}$ that differ in only one access, say a_j, for some $j \in \{1, 2, \ldots, M\}$. We know that the server sees a sequence $\mathbf{x}$ given by

$$\mathbf{x} = (\texttt{position}_M[a_M], \ldots, \texttt{position}_1[a_1])$$

where $x_i := \texttt{position}_i[a_i]$ is the position of address a_i for the i^{th} load/store operation, along with the associated path to the root of the sub-tree. Now, we need to compute the ratio of probabilities that $\texttt{ORAM}(\mathbf{a_1})$ and $\texttt{ORAM}(\mathbf{a_2})$ both lead to the same observed sequence $\mathbf{x}$ at the server. In other words, we compute

$$\Pr[\texttt{ORAM}(\mathbf{a}) = \mathbf{x}]$$

for any set of input sequence and observed leaf sequence $\mathbf{a}$ and $\mathbf{x}$ respectively, of a given fixed size M.

We evaluate the above probability by invoking the secondary invariant viz., after each access the mapping of that data block changes randomly according to a fixed distribution D given in Eq. 4. Under this invariant, the probability that a sequence of load/store operations $\mathbf{a}$ leads to a particular observed sequence $\mathbf{x}$ can be computed according to the rules below. Since the position map of each location changes independently and randomly according to D, we can compute the probability that the input sequence $\mathbf{a}$ leads to the output sequence $\mathbf{x}$ ($\Pr[\texttt{ORAM}(\mathbf{a}) = \mathbf{x}]$) by simply multiplying the probabilities of each individual access. This is shown graphically in Table. 2.

1. If the block is accessed for the first time, its location is random and hence the probability is $1/2^L$.
2. If the block a_k was accessed previously at a_i, then the probability is $p_{\max}$ or $p_{\min}$ depending on whether $\texttt{position}_k[a_k]$ and $\texttt{position}_i[a_i]$ belong to the same sub-tree.
3. Finally, we multiply all the above probabilities for access 1 to M.

If at any point during the above enumeration the stash size exceeds S, we set the probability to 0. Refer to

3 Analogous to the ∞-ORAM in [55], refer Section 5.3.

Obs. Seq.	a'	b	a	c	a	b	d
Real Seq.	x	y	$\boxed{x}$	z	y	z	x
Probability	$\frac{1}{N}$	$\frac{1}{N}$	$p_{\max}$	$\frac{1}{N}$	$p_{\min}$	$p_{\min}$	$p_{\min}$

Table 2. Demonstration of probabilities given real and observed access patterns a, o respectively. Different symbols for real and observed access patterns are merely for the sake of clarity. Primed symbols are used to denote leaves belonging to the same sub-tree (ex: a, a'). Only the blue symbols affect the probability of the boxed data block. The red elements show the previous and next access of the boxed data block.

Section 5.2.2 for details. The probability that the stash size exceeds S is bounded by Theorem 5.

Next, we compute the maximum change in probabilities over two neighboring access sequences $\mathbf{a_1}$ and $\mathbf{a_2}$ that differ in the i^{th} access. Let the logical address accessed in the two sequences be $\mathtt{a}$ and $\mathtt{b}$ respectively i.e., the i^{th} access in $\mathbf{a_1}$ is $\mathtt{a}$ and in $\mathbf{a_2}$ is $\mathtt{b}$. Since $\mathbf{a_1}$ and $\mathbf{a_2}$ agree in all other locations, let the previous location of access of block $\mathtt{a}$ be l_{pa} (leaf pa) and the next location be l_{na}. Similarly, let l_{pb} denote the previous location of access of $\mathtt{b}$ and l_{nb} the next location. If any of these 4 do not exist i.e., the symbol was never accessed before or was never accessed afterwards, we define that leaf to be 0 for the sake of clarity of the equations (if data element $\mathtt{a}$ was never accessed after the location of access change, then $l_{na} = 0$). Let l be the location of the access of the i^{th} access. Note that $l_{pa}, l_{na}, l_{pb}, l_{nb}, l$ all are specific leaves from $\mathbf{x}$ and hence are same for $\mathbf{a_1}$ and $\mathbf{a_2}$.

It is easy to see that the probabilities can differ in at most 3 places viz., l, l_{na} and l_{nb} (probabilities for l_{pa} and l_{pb} depend on the previous access and hence do not change). To make the equations crisp, we define the following extension to the Kronecker delta function,

$$\delta_{ij} = \begin{cases} 0 & \text{if } r_i \neq r_j \\ 1 & \text{if } r_i = r_j \\ \frac{1/N - p_{\min}}{p_{\max} - p_{\min}} & \text{if } j = 0 \end{cases}$$

where r_x is the root of the sub-tree associated with leaf x. This modification of the Kronecker delta is for the simplicity of the equations. The modification ensures that if a symbol is accessed for the first time, then its probability evaluates to $1/N$, as it should.

Now if $\Pr[\mathtt{ORAM}(\mathbf{a_1}) = \mathbf{x}] > 0$ and $\Pr[\mathtt{ORAM}(\mathbf{a_2}) = \mathbf{x}] > 0$ i.e., both the ratios are well-defined, we can calculate the ratio of the probabilities as:

$$\frac{\Pr[\mathtt{ORAM}(\mathbf{a_1}) = \mathbf{x}]}{\Pr[\mathtt{ORAM}(\mathbf{a_2}) = \mathbf{x}]} = \frac{P_{l,l_{pa}} \cdot P_{l_{na},l} \cdot P_{l_{nb},l_{pb}}}{P_{l_{na},l_{pa}} \cdot P_{l,l_{pb}} \cdot P_{l_{nb},l}}$$

After observing that $\frac{1/N}{p_{\max}} \geq \frac{p_{\min}}{p_{\max}}$, we can see that this maximum value of the ratio of probabilities occurs when l_{na}, l, l_{pa} belong to the same sub-tree and l_{pb}, l_{nb} belong to a different sub-tree. In this case, the ratio is given by,

$$\frac{p_{\max} \cdot p_{\max} \cdot p_{\max}}{p_{\max} \cdot p_{\min} \cdot p_{\min}} = \left(\frac{p_{\max}}{p_{\min}}\right)^2$$

Evaluating this in terms of our parameters, $p_{\max}$ and $p_{\min}$ given by Eq. 5 and plugging this into the differential privacy equation:

$$\max_{\substack{\mathbf{a_1},\mathbf{a_2} \\ |\mathbf{a_1}-\mathbf{a_2}|=1}} \frac{\Pr[\mathtt{ORAM}(\mathbf{a_1}) = \mathbf{x}]}{\Pr[\mathtt{ORAM}(\mathbf{a_2}) = \mathbf{x}]} \leq \left(\frac{p_{\max}}{p_{\min}}\right)^2 = \left(\frac{1 + (2^k - 1)p}{1 - (1 - \delta_{k0})p}\right)^2$$

It is important to note that the above equation holds for all observed access sequences $\mathbf{x}$. And hence, we can see that Root ORAM guarantees $\epsilon = 2 \log \left(\frac{1+(2^k-1)p}{1-(1-\delta_{k0})p}\right)$. This completes the ϵ bound.[4]

5.2.2 The δ bound

In this subsection, we show the need for δ in quantifying the security. We demonstrate this necessity using generic tree-based ORAM constructions. We assume that the total stash size is S. For demonstration purpose, we construct a minimal working example. Let:

$$\begin{aligned} \mathbf{a} &= ((r, \mathit{1}, \cdot), (r, \mathit{1}, \cdot), ..., (r, \mathit{1}, \cdot)) \text{ and} \\ \mathbf{a}' &= ((r, \mathit{1}, \cdot), (r, \mathit{2}, \cdot), ..., (r, S + \mathit{1}, \cdot)) \end{aligned} \tag{7}$$

where r denotes the read operation and $\cdot$ denotes data which is not important for the demonstration. In words, one access sequence consists of S accesses to the same element and the second access sequence consists of $S+1$ different accesses to elements $1, 2, ..., S + 1$.

It can be seen that the sequence $1, 1, ..., 1$ is a possible output sequence of $\mathtt{ORAM}(\mathbf{a})$. It is not hard to see that the same sequence $1, 1, ..., 1$ can never occur as $\mathtt{ORAM}(\mathbf{a}')$. The reason for this is simply because if there are more than $S+1$ data blocks mapped to the same leaf, the tree ORAM invariant is broken. Hence the $S + 1$ accesses to the same location cannot all be different elements.

To demonstrate this further, we consider a situation where a program is using a tree-based ORAM protocol

4 Another important point to note here is that the above analysis is a worst case analysis and hence it only depends on two probabilities in the distribution D viz., the largest and the smallest probabilities. The same proof goes through for other probability distributions leading to $\epsilon = 2 \log \left(\frac{p_{\max}}{p_{\min}}\right)$.

to hide its access pattern. We also assume that the program has the following traits,

$$\text{Access Pattern} = \begin{cases} 1,1,1,...,1 & \text{if Secret} = 1 \\ 1,2,3,..,S+1 & \text{if Secret} = 0 \end{cases}$$

If $\mathbf{a}$ is the input access pattern, *and we observe a sequence of* $S+1$ *or more access made to the same location in* $\texttt{ORAM}(\mathbf{a})$*, we can immediately infer that Secret = 1.* It is important to note that the probability of an observed sequence can suddenly jump from a non-zero value to 0 with the change of a single accessed block. We quantify this by the δ in the (ϵ, δ)-differential privacy framework for ORAMs.

We compute the maximum probability for a sequence such that some neighboring sequence (i.e., differing in one access) has zero probability. In particular we choose the following two sequences:

$$\mathbf{a_1} = (1,2,3,...,M)$$
$$\mathbf{a_2} = (1,1,1,...,1)$$

If $\Pr[\texttt{ORAM}(\mathbf{a_i}) = \mathbf{x}] > 0$ for $i = 1,2$, then we have already shown the ϵ bound and hence $\delta = 0$ ($\delta = 0$ if $M \leq S$). So it remains to find the maximum δ when one of these is 0. Let us assume that sequence $\mathbf{x}$ has zero probability when the input sequence is $\mathbf{a_1}$ (i.e., $\Pr[\texttt{ORAM}(\mathbf{a_1}) = \mathbf{x}] = 0$). In this case, δ is simply the maximum value of $\Pr[\texttt{ORAM}(\mathbf{a_2}) = \mathbf{x}]$. Then, a conservative upper bound on δ can be found by noting the following: at each location, the associated probability is either $p_{\max}$ or $p_{\min}$ or $1/N$. Since $p_{\max}$ is the largest of these, we can get a upper bound on δ as

$$\delta \leq p_{\max}^M \tag{8}$$

Finally, to complete the proof, we note that the above δ bound should hold for each possible output access sequence $\mathbf{x}$. To extend this to all possible subsets $\mathbb{S}$, say the support of $\mathbb{S}$ contains $\mathbf{x_1}, \mathbf{x_2} \ldots \mathbf{x_s}$ and $s \leq M$. Hence,

$$\begin{aligned} \Pr[\texttt{ORAM}(\mathbf{a_1}) \in \mathbb{S}] &= \sum_{k=1}^{s} \Pr[\texttt{ORAM}(\mathbf{a_1}) = \mathbf{x_k}] \\ &\leq \sum_{k=1}^{s} (e^\epsilon \cdot \Pr[\texttt{ORAM}(\mathbf{a_2}) = \mathbf{x_k}] + \delta) \\ &= e^\epsilon \cdot \left(\sum_{k=1}^{s} \Pr[\texttt{ORAM}(\mathbf{a_2}) = \mathbf{x_k}] \right) + s \cdot \delta \\ &\leq e^\epsilon \cdot \Pr[\texttt{ORAM}(\mathbf{a_2}) \in \mathbb{S}] + M \cdot \delta \end{aligned}$$

This shows that Root ORAM is (ϵ, δ)-differentially private for $\delta = M \cdot p_{\max}^M$. This completes the δ bound. ■

5.3 Performance Results

Theorem 4 (Security-Performance Trade-off)**.** ***Let*** $\mathbb{A}_{k,p}^Z$ ***and*** $\mathbb{B}_{k,q}^Z$ ***be two Root ORAM protocols***[5] ***with privacy parameters*** ϵ_1 ***and*** ϵ_2 ***respectively, with*** $\epsilon_1 \geq \epsilon_2$***. For any given access sequence*** a***, let*** R_1 ***and*** R_2 ***denote the random variables for the stash usage after a sequence*** s ***of load/store accesses using*** $\mathbb{A}_{k,p}^Z$ ***and*** $\mathbb{B}_{k,q}^Z$ ***respectively. Then,***

$$\mathbb{E}[R_1] \leq \mathbb{E}[R_2] \tag{9}$$

where the expectation is taken over randomness of $\boldsymbol{x}, \boldsymbol{y}$ ***in*** $\boldsymbol{s} = (\boldsymbol{a}, \boldsymbol{x}, \boldsymbol{y})$***.***

Theorem 5 (Stash Bounds)**.** ***The probability that the stash size of the DP-ORAM protocol with parameters*** $k, Z = 5$ ***exceeds*** $R + Z \cdot 2^k$ ***for*** $R \geq 1$ ***is bounded by*** $14(0.6002)^R$

Theorem 6 (Bandwidth)**.** ***The bandwidth of the Root ORAM protocol with parameters*** k, p, Z ***is*** $2 \times Z(L + 1 - k)$ ***blocks per real access.***

We defer the proofs of Theorem 4, 5, and 6 to Appendix A due to space constraints.

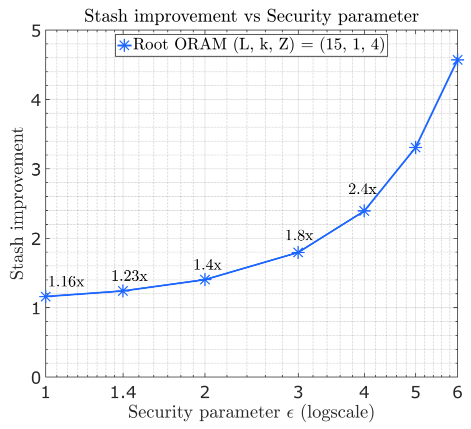
6 Systems evaluation

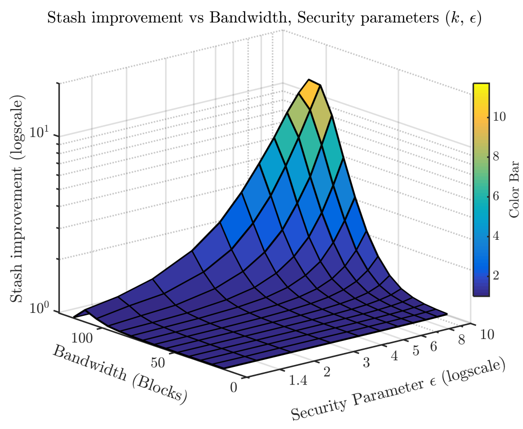
In the previous sections, we have established the design space made possible by formalizing DP-ORAM, a tunable protocol construction, and theoretical security and performance analysis. In this section, we demonstrate the multi-dimensional trade-off between bandwidth, stash storage, and security using a complete systems implementation of Root ORAM.

6.1 Details of the implementation

Root ORAM is built entirely in C++. All experiments were performed on a 1.4 GHz Intel processor with 4GB of RAM. For the Amazon EC2 experiments, remote servers were set-up and latency measurements were performed over a TCP connection for reliable data downloads. For experiments measuring access latency for applications with bandwidth constraints, we cap both the upload and download bandwidth to a value γ. The values of γ used were $\{10, 30, 100, 300, 1000\}$ KB/s. We used the *trickle* application to constrain the bandwidth at the client machines to these desired values.

5 The p-parameters for the two ORAMs are different because the p-parameters are linked to the corresponding security parameters ϵ_1, ϵ_2 as given by Theorem 3.





(a) **(b)**

Fig. 3. Stash improvement as a function of ϵ and k. Fig. 3a shows the improvement in stash usage compared to a baseline of $\epsilon = 0$ for $(L, k, Z) = (15, 1, 4)$. Fig. 3b shows the security-performance trade-off relative to $\epsilon = 0$ for $(L, Z) = (15, 4)$.

Finally, we use the worst case linear access pattern for the simulations.

We study the effect of system parameters on the performance of Root ORAM. In particular, we study the inter-dependence between local stash required, bandwidth, and security (given by ϵ). We also study the access latency of Root ORAM protocols in two different settings (1) We measure the access latency over remote Amazon EC2 servers varying the protocol bandwidth parameter k (and consequently the bandwidth itself) (2) We limit the bandwidth at the client end to a specific value γ (to emulate constrained bandwidth environments) and measure the access latency of Root ORAM protocols. In light of the recent paper by Bindschaedler *et al.* [11], we base our experimental evaluation by giving due importance to the constants involved in the overheads of the system.

6.2 Evaluation results

Bandwidth, Security and Stash Trade-offs: Fig. 3a shows how statistical privacy reduces stash sizes. Note that increasing values of ϵ lead to lower stash values, the improvement of which is captured by the y-axis of Fig. 3a. While Theorem 4 shows that relaxing the security improves performance, Fig. 3a *empirically* shows these performance improvements for concrete values of the security parameter ϵ. For instance, Root ORAM provides a 16% improvement in stash usage for $\epsilon = 1$, a 40% improvement for $\epsilon = 2$ and about 80% improvement for $\epsilon = 3$ ($\delta \approx M \cdot 2^{-14M}$ where M is the access sequence length). As shown in Appendix B, the loss in Shannon entropy of the output sequence is small for moderate values of ϵ. For instance, for $L = 20$, an $\epsilon = 3$ results in a loss in entropy of roughly 0.3 bits and for $\epsilon = 2$ the loss is less than 0.1 bits (compared to 20 bits without any security). Furthermore, as seen in Section 7.3, in the context of the Private Information Retrieval application the use of anonymous communication channels can further reduce the effective privacy values ϵ by multiple orders of magnitude. Similar parameter values for differentially private systems are being increasingly adopted by the research community [56] as well as seen in deployed systems such as RAPPOR [23] ($\epsilon = \ln 3$), Apple Diagnostics [1, 2] ($\epsilon = 2$ for Health information types, $\epsilon = 4$ for Lookup Hints and Safari crash domain detection and $\epsilon = 8$ for Auto-play intent detection) and US census data release [3–5] ($\epsilon = 8.9$ for OnTheMap LEHD Origin-Destination Employment Statistics (LODES)). Research works which extensively explore the problem of setting privacy budgets state that the adopted privacy budget values range from 0.01 to 10 (refer to Table 1 from Hsu *et al.* [32] or Fig. 2 from Lee *et al.* [38]).

Fig. 3b depicts the trade-off between the stash improvement (relative to $\epsilon = 0$), security and bandwidth (parameter k) for the Root ORAM protocol. We can see the significant performance gains in the high bandwidth regime. Note that the stash size of the Root ORAM protocol (as in Theorem 5) can be split into two components viz., exponential component (bounded by $Z \cdot 2^k$) and the randomness component (bounded by $14(0.6002)^R$). The former dominates the latter for small values of bandwidth i.e., large values of k and hence the stash-security trade-off is less significant in those regimes, which agrees with the results in Fig. 3b. Fig. 3a and Fig. 3b thus capture the effect of varying the security (ϵ) on the performance by the reduction in stash size (compared to a baseline of $\epsilon = 0$) and show

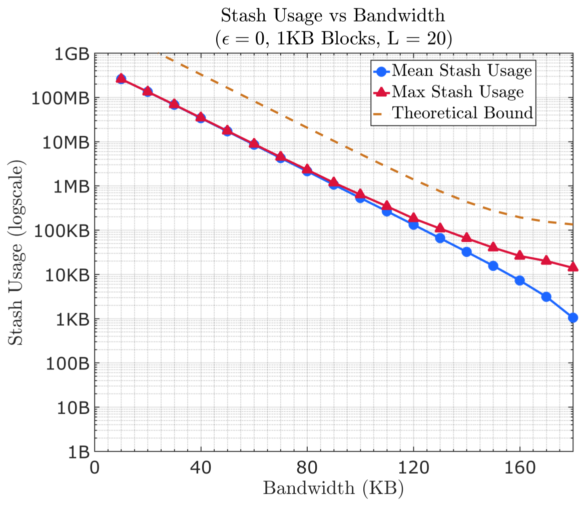


Fig. 4. Absolute values of stash usage for $(L, Z) = (20, 5)$. The theoretical values are plot for a failure probability of 2^{-80}.

that *statistical privacy can be used to improve the performance of ORAM schemes.*

Absolute Stash Values: Fig. 4 shows the absolute values of the stash size (in Bytes) as a function of the bandwidth. The stash roughly grows exponentially with reducing bandwidth, which serves as an experimental validation of Theorem 5. We can see that the required stash values are low enough to be practical in most systems today. For instance, we can achieve a 10× outsourcing ratio at a bandwidth of about 20KB (for 1GB of outsourced data and local storage of 100MB). Similarly, we can achieve 100× outsourcing ratio with a bandwidth of 60KB and an outsourcing ratio of 1000× with a bandwidth of 90KB.

Real-world implementation: We compute the latency overhead of a memory accesses as a function of the bandwidth parameter k as well as the constrained application bandwidth γ. Fig. 5a depicts the access latency as a function of the bandwidth (varying k). We can see how Root ORAM provides a spectrum of acceptable bandwidth-latency choices compared to a single design point for Path ORAM. In Fig. 5b, we compare the access latency when the application bandwidth is limited (for a fixed value of γ). We find the latency as a function of the constrained application bandwidth γ (constrained at the client side) for a few different values of k. The bandwidth for a given value of k can be computed using Theorem 6 as $10 * (21 - k)$ blocks. We find that for limited application bandwidth, the system parameters significantly affect the access latency. Hence applications with constrained bandwidths can greatly benefit from using Root ORAM. For instance, in a scenario where the application bandwidth is limited to 10KB/s (γ = 10KB/s), we can improve the access latency by roughly 10× using Root ORAM.

6.3 Practical Impact

Next, we consider the significance of local storage and bandwidth improvements offered by Root ORAM in the typical deployment contexts of (1) trusted execution environments and (2) client-server/cloud settings.

Local storage: Trusted Execution Environments, such as enclaves created using Intel SGX-Processors, have severe memory constraints, with total local memory of only 94MB [48], which is a significant bottleneck for ORAM deployment[6]. Fig. 3b shows 16% stash improvement for $\epsilon = 1$ values and 1.8× for $\epsilon = 3$. Hence, for 1KB block size and ORAM parameters $(L, k, Z) = (20, 11, 5)$, a 2× improvement would reduce the stash usage from 532KB (Fig. 4) to about 266KB. This significantly frees up the limited memory for trusted computing operations. For larger block sizes such as 256KB (considered in [11]), a 2× stash performance improvement would reduce stash overhead from 133MB to 66.5MB, *enabling compatibility with current architectures of trusted processors such as the Intel Skylake.*

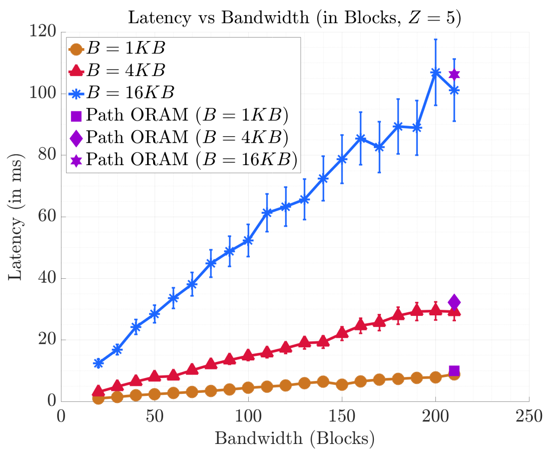
Even in the context of smartphone applications, our results indicate that for 1TB of outsourced data, Root ORAM can bring down the local storage overhead (extending Fig. 4 results for 1MB block sizes and $(L, k, Z) = (20, 11, 5)$) from 500MB to less than 250MB (as low as 100MB for higher ϵ).

Bandwidth: Root ORAM allows tunable trade-offs between bandwidth, storage, and privacy. In many embedded computing and IoT applications, bandwidth is a significant bottleneck for ORAM deployment. Root ORAM can reduce bandwidth overhead by up to 2×-10× (at the cost of increased local storage and statistical privacy), providing dramatic gains in network access latency as shown in Fig. 5b.

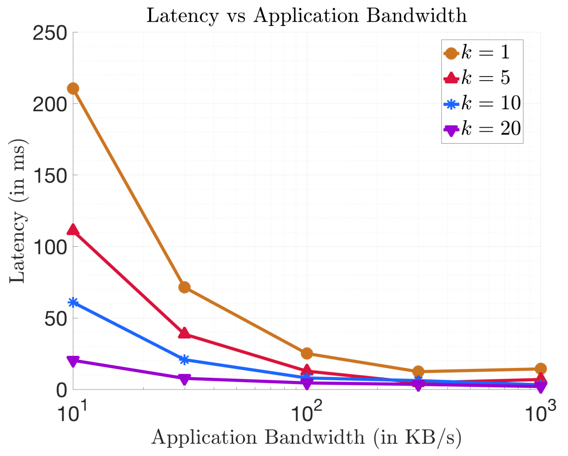
6.4 Choosing parameters

To use Root ORAM as a system, we require a lower bound on the number of accesses M (to bound the worst case δ leakage). If this is unknown, M is set to $S + 1$ (one more than the total stash size). Typical to differentially private systems, a privacy budget is set i.e., an upper bound ϵ_{budget} is set for the system use. For the particular application we take into account the worst case hamming distance between access patterns. If this distance is too large, we recommend using $\epsilon_{\text{budget}} = 0$.

6 Recent work such as Circuit ORAM [57] require constant local memory but increase the protocol round complexity, thereby increasing the effective bandwidth.



(a)



(b)

Fig. 5. Real-world implementations over Amazon EC2. Fig. 5a shows the Root ORAM latency as a function of the bandwidth for $(L, Z) = (20, 5)$ and different block sizes viz., 1 KB, 4 KB and 16 KB. Fig. 5b shows the latency as a function of the constrained/limited application bandwidth for 4 KB block sizes and $(L, Z) = (20, 5)$. Note the significant difference between the access latency across different k values for constrained bandwidth applications.

Once the privacy budget is set, using the results of Section 5 and Section 6.2, Root ORAM parameters can be chosen using acceptable values of bandwidth, stash and security parameters (k, S and ϵ). Two of the three parameters can be set to desired values independently viz., two among security parameter ϵ, the bandwidth parameter (k) and stash size (S) can be chosen independently. The third parameter is determined by the choice of the other two and the optimal trade-off choice would be determined by the specific application requirements. Finally, depending on the application under consideration and the effect of different block sizes on the bandwidth and storage overhead, an optimal block size can be chosen. For instance, in the application of PIR-Tor [46], Tor clients query about 4MB of data from Tor directory servers to retrieve information about Tor relays (refer to Section 7 connection between the ORAMs and PIR). This can be accomplished by using an ORAM with 4MB block size or a smaller block size ORAM with multiple invocations. The different performance overhead of such choices in system design are quantified in Theorem 3 and 5, and the resulting security is quantified via composition theorems from Section 2.

Remark: It is important to note that when $k = 0$, the storage structure in Root ORAM reduces to a single sub-tree. Hence, the non-uniform distribution in Root ORAM reduces to a uniform distribution over all the leaves. Another sanity check is that both $p_{\max}$ and $p_{\min}$ equal $1/N$ when $k = 0$. At the same time, since $k = 0$, no levels in the tree are cached. *Hence Root ORAM when $k = 0$ instantiates exactly into the Path ORAM protocol.*

7 Applications: Efficient Private Information Retrieval

In this section, we demonstrate how DP-ORAM in conjunction with trusted hardware can be used to perform *differentially private* Private Information Retrieval (DP-PIR) queries. The idea of using ORAM in conjunction with trusted hardware has been previously explored by the research community [7, 8, 47, 59]. An important line of research is in developing faster PIR protocols using a combination of trusted hardware and ORAM [7, 59].

7.1 Private Information Retrieval (PIR)

Private Information Retrieval is a cryptographic primitive that provides privacy to a database user. Specifically, the protocol allows the user to hide his/her queries when accessing a public database from the database holder. The critical difference between the PIR and ORAM problem settings is that one assumes a public database (PIR) and the other assumes a private database (ORAM). In a Differentially Private-PIR scheme (DP-PIR), the PIR privacy guarantees are relaxed and quantified using differential privacy.

7.2 Differentially Private-PIR schemes (DP-PIR)

Differentially Private PIR has been proposed by Toledo *el. al.* in [56]. The definition relies on an indistinguishability game between the adversary and a number of honest users as follows:

7.2.1 DP-PIR indistinguishability game

Among the set of honest users U, one is identified by the adversary as the target user U_t. The adversary provides the target user U_t two queries Q_i, Q_j and provides all other users a single query Q_0. The target user selects one of the two queries and then all users use a PIR system to retrieve records. The adversary observes all the transmitted information including all the information from corrupt servers. The privacy of a DP-PIR protocol is formulated as follows (from Toledo *et al.* [56]):

Definition 4. ***Differentially Private PIR:*** *A protocol provides (ϵ, δ)-private PIR if there are non-negative constants ϵ and δ, such that for any possible adversary-provided queries Q_i, Q_j, and Q_0, and for all possible adversarial observations O in the observation space Ω we have that*

$$\forall Q_i, Q_j, \text{ and } Q_0 \qquad \Pr(O|Q_i) \leq e^{\epsilon} \cdot \Pr(O|Q_j) + \delta \quad (10)$$

The security of DP-PIR schemes translates to the privacy of the underlying queries. Hence, the privacy guarantees of DP-PIR are easier to interpret as they directly relate to the "program secret" i.e., the PIR query.

7.2.2 DP-PIR construction from DP-ORAM

To construct a DP-PIR protocol using Root ORAM, we assume the PIR database is on a server with a trusted processor such as Intel SGX [33] or 4765 cryptographic co-processor by IBM [6]. DP-ORAM based DP-PIR operates on a public database (as required by any PIR application) but is encrypted by the trusted hardware to hide memory accesses. Different users of the DP-PIR application use the same underlying DP-ORAM. The DP-ORAM protocol is run within the trusted hardware which also stores the ORAM stash and hence is common across different users and multiple ORAM invocations. The DP-ORAM block size is set equal to the PIR database block size. To perform a DP-PIR query, a client does the following:

– **Step 1 (Initialization):** In the initialization step, the client and the trusted hardware set up an authenticated encrypted channel (AEC) for communication (with or without an anonymous communication channel). The trusted hardware also initializes the ORAM storage structure with the entries of the PIR database. The ORAM is initialized with block size equal to the PIR block size. Other parameters are chosen according to application constraints (refer to Section 6.4).
– **Step 2 (Send Query):** The client sends his PIR query (some database index i) to the trusted hardware through the AEC set up in Step 1 (over an anonymous channel or directly over the network).
– **Step 3 (DP-ORAM):** The trusted hardware decrypts the PIR query to get the decrypted index i and initiates a DP-ORAM query using this index.
– **Step 4 (Receive Response):** The trusted hardware retrieves the PIR block with index i using the DP-ORAM protocol from the untrusted memory. It sends this block over the AEC to the client.

We show that the above constructed PIR protocol satisfies the guarantees of DP-PIR protocols from Definition 4. More formally,

Theorem 7 (DP-ORAM ⇒ DP-PIR)**.** ***The PIR protocol described above completed using a*** (ϵ, δ)***-DP-ORAM is*** (ϵ, δ)***-DP-PIR.***

We defer the proof of Theorem 7 to Appendix A.

7.3 Application Requirements and Multiple Queries:

Application Requirements: Next we compare the application requirements for various DP-PIR protocols. The 4 DP-PIR protocols from Toledo *et al.* [56] all rely on the use of multiple servers and 2 of the 4 schemes rely on the use of anonymous communication channels. DP-ORAM based DP-PIR described in Section 7.2.2 is a DP-Computational PIR scheme in contrast with the DP-Information-Theoretic PIR schemes in Toledo *et al.* [56]. Our DP-PIR protocol requires a single server and the use of anonymous channels is optional, though the existence of the latter improves the performance of our proposed protocol as discussed later in this Section. Our protocols require the use of trusted hardware but this results in significant performance improvements as discussed in Section 7.4.

Single Queries: DP-PIR protocols, as formalized in Section 7.2.1, quantify the privacy for a single PIR query. In Theorem 7, we quantify the privacy of our proposed DP-PIR scheme for a single query. Performance benefits of DP-ORAM directly enhance the performance of the PIR protocol (cf Section 7.4) and showcase the benefits of DP-ORAMs[7]. We further analyze the single query mode of operation into two categories:

7 DP-PIR is well suited to showcase the benefits of using statistical ORAMs as neighboring sequences in this application directly map to "program secrets" and differ by a single access.

- **Without anonymous channels (ACs):** Without access to ACs, Theorem 7 gives the privacy guarantees of our DP-PIR protocol.
- **With anonymous channels:** If ACs are available, they can be used to boost the performance of our DP-PIR protocol by leveraging the additional privacy offered by the communication channel. This leads to significant performance benefits which we summarize in the following theorem:

Theorem 8 (DP-PIR with Anonymous Channels). ***The composition of a (ϵ, δ)-differentially private PIR mechanism with a perfect anonymity system used by u users, for sufficiently large number of users[8], yields a (ϵ', δ')-differentially private PIR mechanism for each user where:***

$$\begin{aligned} \epsilon' &= \frac{e^{2\epsilon}}{u} \\ \delta' &= \min\{1, u \cdot \delta + neg(u)\} \end{aligned} \quad (11)$$

where $neg(u)$ is a negligible function[9] of u.

We defer the proof of Theorem 8 to Appendix A. These bounds significantly enhance the privacy values of when using a DP-PIR protocol in composition with a anonymous communication channel. For instance, assuming $u = 10^3$ users use a $(2, 2^{-80})$-DP-PIR, each user is effectively using a $(0.05, 2^{-70})$-DP-PIR protocol[10].

Multiple Queries: An important consideration in the use of DP-PIR schemes is the effect of multiple queries on the security of the scheme. Multiple invocations of the DP-PIR scheme results in a privacy loss. We extend Theorem 1 to prove Theorem 9 that bounds the privacy of DP-PIR schemes under multiple invocations. Consequently, the privacy of multiple DP-PIR invocations can be found by composing Theorem 9 with the bounds from Theorem 7 or Theorem 8 depending on the availability of anonymous communication channels.

Theorem 9 (DP-PIR Composition Theorem). ***m invocations of a (ϵ, δ)-DP-ORAM based DP-PIR protocol guarantees an overall $(m\epsilon, m\delta)$-DP-PIR protocol.***

We defer the proof of Theorem 9 to Appendix A.

8 Sufficiently large number of users: $u \gg \max\{1, e^{2\epsilon}\}$.

9 A negligible function $neg(n)$ is a function $neg \colon \mathbb{N} \to \mathbb{R}$ such that $\forall c \in \mathbb{Z}^+ \quad \exists N_c \in \mathbb{Z}^+$ such that $\forall x \geq N_c$, $|neg(n)| < n^{-c}$

10 Ignoring terms negligible in u

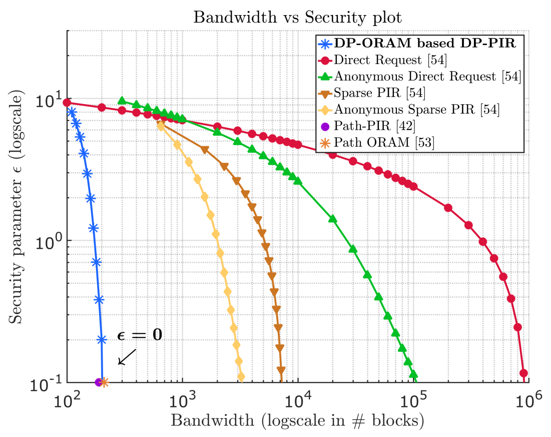


Fig. 6. Security-Bandwidth trade-offs for DP-PIR protocols (Toledo *et al.* [56], Path-PIR [43], and Path ORAM [55]).

7.4 Comparison with Prior Work

Next we compare the performance of our DP-PIR scheme with (1) DP-PIR schemes from [56] (2) Path-PIR construction [43]. We begin by briefly describing the 4 DP-PIR schemes from Toledo *et al.* [56]:

- **Direct Requests:** For each real query, the client sends $p - 1$ other dummy queries spread across d identical databases. d_a of the databases are assumed to be adversarial.
- **Anonymous Direct Requests:** This protocol assumes the use of anonymous communication channels (ACs) and performs the above mentioned Direct request protocol in conjuction with the AC. The increased privacy occurs from the fact that each user sends p requests yet derives privacy among $u \cdot p$ requests (where u is the number of users).
- **Sparse-PIR:** This protocol is based on Chor's PIR protocol [15]. Instead of generating random vectors for the servers, the client generates biased (hence sparse) random vectors using i.i.d Bernoulli trials with parameter θ.
- **Anonymous Sparse-PIR:** Similar to anonymous direct requests, this protocol is the composition of the Sparse-PIR protocol with an anonymity system.

We compare our protocols with the exact same setup as in [56]. Different parameters are set to the following values: (1) Database with $n = 10^6$ blocks (2) Number of databases $d = 10^2$ (3) Number of adversarial databases $d_a = 0.1 \times d = 10$ (this showcases the most optimistic version of the results of [56]). Anonymous direct request has the same parameters as the direct request protocol with an additional assumption of number of users $u = 10^3$. Sparse-PIR and Anonymous Sparse-PIR protocols ignore the communication cost from the client side. This communcication overhead is information theoretically lower bounded by $a \cdot h(\theta)$ where a is the size of vector to be sent and $h(\cdot)$ is the binary entropy function. Since the overhead for encoding the random vectors is linear (and hence very large) in Sparse PIR and Anonymous Sparse PIR protocols, we assume they are based on the 2D variant of Chor's protocol [15].

Bandwidth comparison: As seen in Fig. 6, our DP-PIR protocol provides orders of magnitude performance improvements over state-of-the-art DP-PIR protocols from [56]. The performance gains come from the logarithmic overhead of ORAM schemes compared to linear overhead of PIR schemes. Path-PIR does not provide statistical security and hence is seen as a single data-point in Fig. 6. Path-PIR also achieves logarithmic overhead yet suffers from (1) heavy computation requirements at the client and the server due to the use of underlying homomorphic encryptions (2) large storage overhead due to logarithmic bucket sizes (3) scalability i.e., is better suited for small databases (or large block sizes).

Other comparisons: As discussed before, our DP-PIR protocol requires the use of trusted hardware but results in significant performance improvements. At the same time, our DP-PIR protocol requires a single server in contrast with multiple servers required for Toledo *et al.* [56]. It is interesting to note that our DP-ORAM based DP-PIR (described in Section 7.2.2) is a DP-Computational PIR scheme in contrast with the DP-Information-Theoretic PIR schemes from Toledo *et al.* [56]. Our protocol as well as protocols from Toledo *et al.* [56] benefit from the use of anonymous channels. Computational costs for our DP-PIR protocol are $O(\log N)$ whereas they are $O(p), O(d \cdot N \cdot \theta)$ for various schemes in Toledo *et al.* [56]. On the contrary, additional storage costs for our protocol are given by Theorem 5 ($O(\log N) + Z * 2^k$) but are 0 for schemes in Toledo *et al.* [56]. Finally, we remark that the set-up cost for our protocol includes a *one time* ORAM database initialization[11] phase (where the PIR database is stored in the ORAM database) which does not exist for other DP-PIR protocols.

7.5 Other Applications

Our discussion above focused on the PIR, which itself is a fundamental privacy technology that can enable numerous applications, including PIR-Tor [46], PIR for e-commerce [31], PIR for MIX Nets [35]. Benefits of DP-ORAM extend to other applications as well. For instance, Gentry *et al.* [25] demonstrate the use of ORAMs as building blocks for secure computation. The benefits of DP-ORAM can be extended to such applications of ORAM protocols for improving performance. In fact, the use of differential privacy to boost the performance of secure computation is already gaining attention in the research community with work by He *et al.* [30]. Finally, DP-ORAM can be used in systems such as Dropbox and Google Drive to privately retrieve data at low network overheads and local storage.

11 DP-ORAM initialization is done once and hence can be done using an $\epsilon = 0$ DP-ORAM to preserve privacy budget for future queries.

8 Related work

Oblivious RAMs were first formalized in a seminal paper by Goldreich and Ostrovsky [27]. Since then, the research community has made substantial progress in making ORAMs practical [28, 29, 40, 50, 53, 55, 60]. Hierarchical constructions such as [29, 36, 50] were proposed building on [27] and tree based ORAM schemes such as [25, 37, 40, 51, 53–55] were proposed building on Shi *et. al.* [22]. A recent benchmark for ORAMs has been the Path ORAM protocol [55] that gives theoretical bounds on the local memory usage. Tessaro *et. al.* [14] build on [55] and extend it for multiple clients by level caching in tree based ORAM schemes. Root ORAM generalizes the construction of [55] to provide a tunable framework offering DP-ORAM guarantees. Root ORAM gets around the Goldreich-Ostrovsky lower bound by using (1) Statistical security, which voids the proof of the lower bound [27] (2) Stash storage that is not a constant (which is what gives the logarithmic GO-lower bound). Our work opens up new opportunities for rethinking lower bounds for statistical ORAMs.

Gentry *et. al.* [25] have shown the promise of using ORAMs as a building block in developing protocols for Secure Multi-Party Computation. This work is among the first in the line of research using ORAMs as critical component of building other cryptographic primitives. Recently, there has been a number of works using ORAMs for private information retrieval [8, 43, 47, 59], for private ad recommendation [7] and secure computation and machine learning [25, 41, 49, 58].

Several optimizations have been proposed to reduce the overhead of tree-based ORAMs. Recently, Ring ORAM [40] reduced the bandwidth using the XOR technique leveraging server-side computation. The XOR technique is orthogonal to the ideas explored in this work and can be extended to Root ORAM, further influencing the protocol design space. Two optimizations for Shi *et. al.* [22] were proposed by Gentry *et al.* [25]. First, they reduce the storage overhead by a multiplicative factor and second, they reduce the time complexity of the protocol. They explore the benefits of using a multiple fan-out tree structure instead of a conventional binary tree. ORAM has also been implemented at a chip level in prototypes such as the Ascend architecture [24] and the Phantom architecture [42].

Recently, Circuit ORAM [57] proposed a novel protocol to reduce the complexity of the eviction protocol in Path ORAM when implementing on a small private memory. This is ideally suited for secure computation environments and is the state-of-the-art protocol when implementing ORAMs in trusted hardware. Though Circuit ORAM works with a constant memory, it increases the protocol complexity which leads to a higher bandwidth usage. Burst ORAM [37] builds on ObliviStore [53] by level caching and optimizing the online bandwidth (formalized in [12]) for bursty (realistic) access pattern. Onion ORAM [18] “breaks” the ORAM lower bound by leveraging server side computation and additively homomorphic encryptions and achieving constant bandwidth overhead.

Floram [19] is a state-of-the-art ORAM construction which constructs an ORAM protocol in the Distributed ORAM model (DORAM). In the DORAM model, the ORAM memory is split across multiple servers. Whereas in the conventional ORAM setting two logical access sequences of the same length produce indistinguishable physical access sequences, in a DORAM, only the physical access sequences observed by a *single server* are indistinguishable. It is possible to augment our work with Floram to further boost its performance.

In summary, Root ORAM is the first protocol that demonstrates a trade-off between performance and statistical privacy (quantified with differential privacy). The tunable security-bandwidth-outsourcing ratio construction and the formalization of differentially private ORAMs differentiates our work from prior approaches.

9 Limitations

In this work, we enable the design of practical ORAM schemes for applications with stringent bandwidth constraints and small local storage. For some applications, it might be acceptable to trade-off statistical privacy for better performance and Root ORAM demonstrates the first step in this direction by introducing a tunable framework that provides differential privacy guarantees. Though Theorem 1 - 2 help us bound the privacy leakage for arbitrary access sequences, we acknowledge that Root ORAM is currently better suited for similar access sequences. For example, our approach is ideally suited for applications such as PIR (Section 7). The formalization of DP-ORAM opens up a number of research directions such as optimal security-performance trade-offs, rethinking lower bounds for statistically private ORAMs, as well as better performance improvement results. Our work has already inspired other researchers to rethink research ideas at the intersection of differential privacy and conventional cryptography [13, 44]. Finally, we note that the ideas developed in this work are orthogonal yet applicable to more recent works such as Ring ORAM, Onion ORAM, and Burst ORAM [18, 37, 40]. Similarly, DP-ORAM constructions for non-tree based ORAMs would be interesting for future work.

10 Conclusions

To summarize, we introduce and formalize the notion of a differentially private ORAM, which to our knowledge is the first of its kind. We present Root ORAM, a tunable family of ORAM protocols which provide a multi-dimensional trade-off between security, bandwidth and local storage requirements. We evaluate the protocol using theoretical analysis, simulations, and real world implementation on Amazon EC2. We analyze the benefits of statistical ORAMs in (1) trusted execution environments and (2) server-client settings and demonstrate how statistical ORAMs can improve the performance of existing ORAMs. Finally, we showcase the utility of Root ORAM via the application of Private Information Retrieval.

11 Acknowledgments

We would like to thank the anonymous PETS reviewers for insightful feedback on the paper and the following funding agencies: Army Research Office YIP Award, National Science Foundation (CNS-1409415, CIF-1617286, CNS-1553437), and Faculty research awards from Google, Intel, and Cisco.

References

[1] Apple Differential Privacy Technical Overview. https://images.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

[2] Apple is using Differential Privacy to help discover the usage patterns of a large number of users without compromising individual privacy. https://discussions.apple.com/thread/7664417?start=0&tstart=0. Published: 2016-06-13.

[3] The Challenge of Scientific Reproducibility and Privacy Protection for Statistical Agencies. https://www2.census.gov/cac/sac/meetings/2016-09/2016-abowd.pdf.

[4] US Census Data Release. http://reports.opendataenterprise.org/2016opendataroundtables.pdf. Published: Summer 2016.

[5] Why the Census Bureau Adopted Differential Privacy for the 2020 Census of Population. https://privacytools.seas.harvard.edu/why-census-bureau-adopted-differential-privacy-2020-census-population.

[6] IBM systems cryptographic hardware products. http://www-03.ibm.com/security/cryptocards/, 2016.

[7] Michael Backes, Aniket Kate, Matteo Maffei, and Kim Pecina. Obliviad: Provably secure and practical online behavioral advertising. In *IEEE Symposium on Security and Privacy (S&P)*, 2012.

[8] Sumeet Bajaj and Radu Sion. Trusteddb: a trusted hardware based database with privacy and data confidentiality. In *ACM SIGMOD International Conference on Management of Data*, 2011.

[9] Mihir Bellare, Shafi Goldwasser, Carsten Lund, and Alexander Russell. Efficient probabilistically checkable proofs and applications to approximations. In *ACM Symposium on Theory of Computing (STOC)*, 1993.

[10] Benjamin Benjamin Fuller, Mayank Varia, Arkady Yerukhimovich, Emily Shen, Ariel Hamlin, Vijay Gadepally, Richard Shay, John Darby Mitchell, and Robert K. Cunningham. Sok: Cryptographically protected database search. In *IEEE Symposium on Security and Privacy (S&P)*, 2017.

[11] Vincent Bindschaedler, Muhammad Naveed, Xiaorui Pan, XiaoFeng Wang, and Yan Huang. Practicing oblivious access on cloud storage: The gap, the fallacy, and the new way forward. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[12] Dan Boneh, David Mazieres, and Raluca Ada Popa. Remote oblivious storage: Making oblivious RAM practical. 2011.

[13] TH Hubert Chan, Kai-Min Chung, Bruce Maggs, and Elaine Shi. Foundations of differentially oblivious algorithms. https://eprint.iacr.org/2017/1033.pdf, 2018.

[14] Binyi Chen, Huijia Lin, and Stefano Tessaro. Oblivious parallel ram: improved efficiency and generic constructions. In *Theory of Cryptography Conference (TCC)*, 2016.

[15] Benny Chor, Eyal Kushilevitz, Oded Goldreich, and Madhu Sudan. Private information retrieval. *J. ACM*, 45(6), 1998.

[16] Paul Cuff and Lanqing Yu. Differential privacy as a mutual information constraint. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[17] Jonathan L Dautrich Jr and Chinya V Ravishankar. Compromising privacy in precise query protocols. In *International Conference on Extending Database Technology*, 2013.

[18] Srinivas Devadas, Marten van Dijk, Christopher W Fletcher, Ling Ren, Elaine Shi, and Daniel Wichs. Onion ORAM: A constant bandwidth blowup oblivious ram. In *Theory of Cryptography Conference (TCC)*, 2016.

[19] Jack Doerner and Abhi Shelat. Scaling oram for secure computation. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[20] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming*. Springer, 2006.

[21] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology—EUROCRYPT*. 2006.

[22] Shi Elaine, Chan T-H Hubert, Stefanov Emil, and Li Mingfei. Oblivious ram with $o((\log n)^3)$ worst-case cost. In *Advances in Cryptology—ASIACRYPT*, 2011.

[23] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[24] Christopher W Fletcher, Marten van Dijk, and Srinivas Devadas. A secure processor architecture for encrypted computation on untrusted programs. In *ACM Workshop on Scalable Trusted Computing*, 2012.

[25] Craig Gentry, Kenny A Goldman, Shai Halevi, Charanjit Julta, Mariana Raykova, and Daniel Wichs. Optimizing ORAM and using it efficiently for secure computation. In *Privacy Enhancing Technologies Symposium (PETS)*, 2013.

[26] Oded Goldreich. Towards a theory of software protection and simulation by oblivious rams. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[27] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3), 1996.

[28] Michael T. Goodrich and Michael Mitzenmacher. Privacy-preserving access of outsourced data via oblivious RAM simulation. In *ICALP (2)*, 2011.

[29] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko, and Roberto Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *ACM-SIAM Symposium on Discrete Algorithms*, 2012.

[30] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[31] Ryan Henry, Femi Olumofin, and Ian Goldberg. Practical pir for electronic commerce. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

[32] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C Pierce, and Aaron Roth. Differential privacy: An economic method for choosing epsilon. In *Computer Security Foundations Symposium (CSF), 2014 IEEE 27th*, 2014.

[33] Intel Corp. Software guard extensions programming reference, 2013. No. 329298-001.

[34] MS Islam, Mehmet Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[35] Dogan Kesdogan, Mark Borning, and Michael Schmeink. Unobservable surfing on the world wide web: is private information retrieval an alternative to the mix based approach? In *Privacy Enhancing Technologies Symposium (PETS)*, 2002.

[36] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2012.

[37] Dautrich Jr Jonathan L, Stefanov Emil, and Shi Elaine. Burst oram: Minimizing oram response times for bursty ac-

cess patterns. In *USENIX Security Symposium*, 2014.

[38] Jaewoo Lee and Chris Clifton. How much is enough? choosing ε for differential privacy. In *International Conference on Information Security*, 2011.

[39] Yingbin Liang, H Vincent Poor, and Shlomo Shamai. Information theoretic security. *Foundations and Trends® in Communications and Information Theory*, 2009.

[40] Ren Ling, Fletcher Christopher W, Kwon Albert, Stefanov Emil, Shi Elaine, Van Dijk Marten, and Devadas Srinivas. Constants count: Practical improvements to oblivious ram. In *USENIX Security Symposium*, 2015.

[41] Chang Liu, Xiao Shaun Wang, Kartik Nayak, Yan Huang, and Elaine Shi. ObliVM: A programming framework for secure computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[42] Martin Maas, Eric Love, Emil Stefanov, Mohit Tiwari, Elaine Shi, Krste Asanovic, John Kubiatowicz, and Dawn Song. Phantom: Practical oblivious computation in a secure processor. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[43] Travis Mayberry, Erik-Oliver Blass, and Agnes Hui Chan. Efficient private file retrieval by combining oram and pir. In *Symposium on Network and Distributed System Security (NDSS)*, 2014.

[44] Sahar Mazloom and S Dov Gordon. Differentially private access patterns in secure computation. https://eprint.iacr.org/2017/1016.pdf, 2017.

[45] Prateek Mittal and Nikita Borisov. Information leaks in structured peer-to-peer anonymous communication systems. *ACM Transactions on Information and System Security (TISSEC)*, 2012.

[46] Prateek Mittal, Femi G Olumofin, Carmela Troncoso, Nikita Borisov, and Ian Goldberg. Pir-tor: Scalable anonymous communication using private information retrieval. In *USENIX Security Symposium*, pages 31–31, 2011.

[47] Pedro Moreno-Sanchez, Aniket Kate, Matteo Maffei, and Kim Pecina. Privacy preserving payments in credit networks. In *Symposium on Network and Distributed System Security (NDSS)*, 2015.

[48] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, 2016.

[49] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, 2016.

[50] Benny Pinkas and Tzachy Reinman. Oblivious RAM revisited. In *Advances in Cryptology—CRYPTO*, 2010.

[51] Ling Ren, Christopher W Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In *USENIX Security Symposium*, 2015.

[52] Ling Ren, Xiangyao Yu, Christopher W Fletcher, Marten Van Dijk, and Srinivas Devadas. Design space exploration and optimization of path oblivious RAM in secure processors. In *ACM SIGARCH Computer Architecture News*, 2013.

[53] Emil Stefanov and Elaine Shi. Oblivistore: High performance oblivious cloud storage. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[54] Emil Stefanov, Elaine Shi, and Dawn Song. Towards practical oblivious RAM. In *Symposium on Network and Distributed System Security (NDSS)*, 2012.

[55] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher W. Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: an extremely simple oblivious RAM protocol. In *ACM Conference on Computer and Communications Security (CCS)*, 2013.

[56] Raphael R Toledo, George Danezis, and Ian Goldberg. Lower-cost ϵ-private information retrieval. *Privacy Enhancing Technologies Symposium (PETS)*, 2016.

[57] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the Goldreich-Ostrovsky lower bound. In *ACM Conference on Computer and Communications Security (CCS)*, 2015.

[58] Xiao Shaun Wang, Yan Huang, T-H. Hubert Chan, Abhi Shelat, and Elaine Shi. SCORAM: Oblivious RAM for secure computation. In *ACM Conference on Computer and Communications Security (CCS)*, 2014.

[59] Peter Williams and Radu Sion. Usable pir. In *Symposium on Network and Distributed System Security (NDSS)*, 2008.

[60] Peter Williams, Radu Sion, and Bogdan Carbunar. Building castles out of mud: practical access pattern privacy and correctness on untrusted storage. In *CCS*, 2008.

A Theorem Proofs

Proof of Theorem 4: We use two key concepts viz., ∞-ORAM and the greedy post-processing algorithm from prior works [55, 57] in proving the above result. We begin by briefly describing these concepts and then prove an equivalence between a greedily post-processed ∞-ORAM and Root ORAM (Lemma 1 and Lemma 2). Finally, we complete the argument by proving the effectiveness of using a non-uniform distribution in reducing the stash usage in ∞-ORAM, thereby showing its effectiveness in Root ORAM. Next, we briefly discuss the concepts of ∞-ORAM and the greedy post-processing algorithm and refer the reader to [55] for more details. We follow the notation from Section 5.1.

∞-ORAM: This is an imaginary ORAM, used as a mathematical abstraction to facilitate proofs about Root ORAM. The ∞-ORAM has all parameters identical to the Root ORAM except it has an infinitely large bucket size ($Z \to \infty$). This allows the ∞-ORAM to store as many blocks in a bucket as possible.

Greedy post-processing: This is an algorithm that post processes the stash and the buckets in an ∞-ORAM such that after a sequence of $\mathbf{s}$ load/store operations, the distribution of the real blocks over the buckets and stash is exactly the same as that of the Root ORAM after being accessed using $\mathbf{s}$. It is easy to see that the ∞-ORAM starts with an empty stash. The greedy post processing algorithm mentioned below processes the ∞-ORAM until the tree has no buckets with more than Z blocks.

- Select any block in a bucket that stores more than Z blocks. Suppose that the bucket is at level h and P is the path from the bucket to the root.
- Find the highest level (closer to the root) $i \leq h$ such that the bucket at level i on path P stores less than Z blocks. If such a bucket exists, move the block to level i else move it to the stash.

Next, we state Lemma 1 and Lemma 2 and omit their proofs due to their similarity with [55] as well as space constraints.

Lemma 1. *The stash usage in the post processed ∞-Root ORAM is the same as Root ORAM protocol with the same parameters.*

$$\mathtt{st}^Z[\mathbb{O}^{\infty}_{k,p}(\boldsymbol{s})] = \mathtt{st}[\mathbb{O}^{Z}_{k,p}(\boldsymbol{s})] \tag{12}$$

For the sake of analysis, we combine the 2^k binary sub-trees by appending a binary tree of depth $k-1$ above the sub-trees. This creates an extended binary tree of height L which contains the original sub-trees at its bottom. We look at the bucket usage over rooted sub-trees of this extended binary tree (rooted sub-tree is a sub-tree which contains the root of the extended tree). We denote by T a generic rooted sub-tree. We use $n(T)$ to denote the total number of buckets in T and $\mathrm{u}^T(\mathbb{O}^{\infty}_{k,p}[s])$ for the number of real blocks in T for an ∞-Root ORAM after a sequence s operations.

Lemma 2. *The stash usage $\mathtt{st}^Z[\mathbb{A}^{\infty}_{k,p}(\boldsymbol{s})]$ in post-processed ∞-Root ORAM is $>$ R if and only if there exists a sub-tree T in ∞-Root ORAM such that $\mathrm{u}^T\left(\mathbb{A}^{\infty}_{k,p}[s]\right) > n(T)\cdot Z + R$*

Let $\mathbb{A}^Z_{k,p}$ and $\mathbb{B}^Z_{k,q}$ be two Root ORAM protocols with security parameters ϵ_1 and ϵ_2 respectively.

Suppose S denotes the set of leaves of the extended binary tree and S' the set of leaves of the currently mapped sub-tree. The probability distribution functions for the updateMapping function in $\mathbb{A}^Z_{k,p}$ and $\mathbb{B}^Z_{k,q}$ differ only in the following way: some probability mass m (for some $m \geq 0$) moves from leaves $S - S'$ to S'.

Thus, with probability mass $(1-m)$, the randomized mapping for both protocol $\mathbb{A}^Z_{k,p}$ and protocol $\mathbb{B}^Z_{k,q}$ behave identically. However, with probability mass m, the data block will be mapped to a leaf in S' in protocol $\mathbb{A}^Z_{k,p}$ but to a leaf in $S - S'$ in protocol $\mathbb{B}^Z_{k,q}$. Hence, in the ∞-ORAM, the data block will be placed on a level less than $k-1$ (higher up in the tree) in $\mathbb{B}^{\infty}_{k,q}$ whereas in $\mathbb{A}^{\infty}_{k,q}$ it will be placed the same sub-tree i.e., on a level greater than $k-1$ (lower down in the tree). Hence for any subtler T, if the data block in $\mathbb{A}^{\infty}_{k,q}$ was placed in a bucket in T, then so will the data block in $\mathbb{B}^{\infty}_{k,q}$. Hence,

$$\mathrm{u}^T(\mathbb{A}^{\infty}_{k,p}[s]) \leq \mathrm{u}^T(\mathbb{B}^{\infty}_{k,q}[s])$$

Hence, for any given sub-tree T, we have:

$$\begin{aligned}&\left(\mathrm{u}^T(\mathbb{A}^{\infty}_{k,p}[s]) > n(T)\cdot Z + R\right)\\ \Rightarrow &\left(\mathrm{u}^T(\mathbb{B}^{\infty}_{k,q}[s]) > n(T)\cdot Z + R\right)\end{aligned}$$

Using the above condition over all rooted sub-trees T, we have

$$\begin{aligned}&\Pr\left[\exists T\left(\mathrm{u}^T(\mathbb{A}^{\infty}_{k,p}[s]) > n(T)\cdot Z + R\right)\right]\\ &\quad\leq \Pr\left[\exists T\left(\mathrm{u}^T(\mathbb{B}^{\infty}_{k,q}[s]) > n(T)\cdot Z + R\right)\right]\end{aligned}$$

Hence,

$$\begin{aligned}\Pr\left[\mathtt{st}[\mathbb{A}^Z_{k,p}(\mathbf{s})] > R\right] &= \Pr\left[\mathtt{st}^Z[\mathbb{A}^{\infty}_{k,p}(\mathbf{s})] > R\right]\\ &= \Pr\left[\exists T\left(\mathrm{u}^T(\mathbb{A}^{\infty}_{k,p}[s]) > n(T)\cdot Z + R\right)\right]\\ &\leq \Pr\left[\exists T\left(\mathrm{u}^T(\mathbb{B}^{\infty}_{k,q}[s]) > n(T)\cdot Z + R\right)\right]\\ &= \Pr\left[\mathtt{st}^Z[\mathbb{B}^{\infty}_{k,q}(\mathbf{s})] > R\right]\\ &= \Pr\left[\mathtt{st}[\mathbb{B}^{Z}_{k,q}(\mathbf{s})] > R\right]\end{aligned}$$

Finally, to complete the argument, we use the following result from basic information theory:

Lemma 3. *Let X be a discrete random variable that takes on only non-negative integer values. Then*

$$\mathbb{E}[X] = \sum_{i=1}^{\infty} \Pr(X \geq i) \tag{13}$$

Summing Eq. 13 for $R = 0, 1, 2\cdots$ (which corresponds to $R_1, R_2 \geq 1$) we get that,

$$\begin{aligned}\mathbb{E}[R_1] &= \sum_R \Pr\left[\mathtt{st}[\mathbb{A}^Z_{k,p}(\mathbf{s})] > R\right]\\ &\leq \sum_R \Pr\left[\mathtt{st}[\mathbb{B}^Z_{k,p}(\mathbf{s})] > R\right]\\ &= \mathbb{E}[R_2]\end{aligned}$$

Which completes the proof for Theorem 4. ■

Proof of Theorem 5: Using Theorem 4, we know that the stash usage is "lower" for non-zero values of ϵ. Hence, it suffices to give stash bounds when $\epsilon = 0$. As in the proof for Theorem 4, we conceptually extend the server storage to contain a complete binary tree with height L, where the sub-trees form the lower levels of the extended binary tree. We can see that for $\epsilon = 0$, the Root ORAM protocol with the additional storage reduces to the Path ORAM protocol and hence the stash size of Root ORAM can bounded as:

$$\Pr\left[\mathtt{st}[\mathbb{O}^Z_{k,p}(\mathbf{s})] > R + Z \cdot 2^k\right] \leq 14(0.6002)^R \qquad (14)$$

This completes the proof of the stash bounds[12]. ■

Proof of Theorem 6: The proof follows by noting that the depth of each sub-tree is equal to $(L + 1 - k)$ and hence number of blocks are transferred per access is $2Z$ times the depth. ■

Proof of Theorem 7: The proof follows directly from the setup and the definition of DP-PIR. Given any two adversarial queries Q_i, Q_j for database records, we consider these as ORAM input access sequences, each with only single access. Since these access sequences differ in a single access, for any output observation O:

$$\Pr[O|Q_i] \leq e^{\epsilon} \Pr[O|Q_j] + \delta \qquad (15)$$

which is the privacy guarantee for DP-PIR. ■

Proof of Theorem 8: δ captures the failure probability of our system and hence we can union bound this failure probability across the u users. Across u users, the failure probability δ' can be bounded as: $\delta' \leq u \cdot \delta$ $(\delta' \leq 1)$. With probability $1 - \delta'$, the composite system is now ϵ-differentially private and we can use the Composition Lemma[13] to get the following bound on ϵ':

$$\begin{aligned}\epsilon' &= \ln\left(e^{2\epsilon} + u - 1\right) - \ln u \\ &= \ln\left(\frac{e^{2\epsilon}}{u} + \frac{u-1}{u}\right) \\ &\leq \ln\left(\frac{e^{2\epsilon}}{u} + 1\right) \leq \frac{e^{2\epsilon}}{u}\end{aligned}$$

Where the last two inequalities follow from (1) $u \gg 1$ (2) $u \gg e^{2\epsilon}$ and $\ln(1+x) \leq x$ when $x > -1$. Finally, since the Composition Lemma from Toledo *et al.* [56] holds with a failure probability $1 - neg(u)$, we can incorporate this failure into the delta bound (union bound) to complete the proof of Theorem 8. ■

12 Line 4 in Flush protocol in Root ORAM ensures that buckets never store more than Z blocks. Hence, capturing the stash failure probability suffices.

13 Since $u \gg 1$, the law of large numbers holds for the proof of the Composition Lemma from Toledo *et al.* [56].

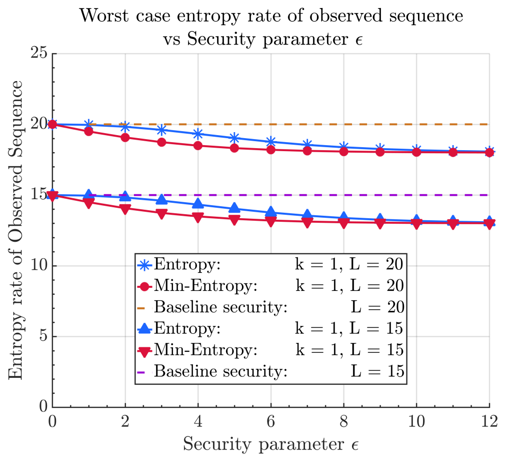


Fig. 7. Entropy rate for worst case access sequence for $L = 20$ and 15 and $k = 1$.

Proof of Theorem 9: The proof follows naturally from the results of Theorem 1. DP-PIR protocol relies on a DP-ORAM protocol and hence the privacy of m (ϵ, δ)-DP-PIR queries sent to the trusted hardware by Theorem 1 can be bound by a single DP-PIR protocol with privacy $(m\epsilon, m\delta)$-DP-PIR guarantees. ■

B Entropy Calculation for DP

Next, we provide an interpretation of the privacy guarantees of the protocol in terms of entropy [16]. Specifically, we find the worst case entropy of the observed access sequence for any given input access sequence. This entropy reflects the adversaries' uncertainty in the observed access sequence given any input sequence. We compute this entropy as follows:

Let $X_1, X_2, \ldots, X_M$ denote the random variables indicating the accessed location for the given access sequence $\mathbf{a}$ of logical block addresses (i.e., X_i is the random variable for y_i where $\mathbf{y} = \{y_i\}_{i=1}^{M}$ as in Section 5). Let $\pi(\cdot)$ denote a function which maps an index in $\{1, 2, \ldots, M\}$ to the location of the previous access of the same data block. In other words, if some data block a was accessed at the i^{th} $(x_i = a)$ and then at the j^{th} $(x_j = a)$ location, then $\pi(j) = i$.[14]

Let $H(\cdot)$ denote the Shannon Entropy. If we have perfect security, $H(X_1, X_2, \ldots, X_M) = M \cdot \log N$ and hence the *entropy rate* is $\log N$ (entropy per access). Using DP-ORAM reduces this entropy and we

14 Formally, we can define $\pi(\cdot) : \{1, 2, \ldots, M\} \to \{1, 2, \ldots, M - 1\} \cup \phi$ as $\pi(j) = \max i$ s.t. $x_i = x_j$ and $i < j$ and ϕ otherwise.

$$\pi(j) = \begin{cases} \max i & \text{s.t. } x_i = x_j \text{ and } i < j \\ \phi & \text{otherwise} \end{cases}$$

compute this loss in entropy below. We know that $P(X_i) = \text{Uniform}(\{0, 1, 2, \dots, 2^L - 1\})$ if $\pi(i) = \phi$ and $P(X_i | X_{\pi(i)}) = D$, where D is the distribution specified in Eq. 4. Hence, we can compute the entropy of the complete sequence as follows:

$$\begin{aligned} H(X_1, X_2, \dots X_M) &= H(X_1) + \sum_{i=2}^{M} H(X_i | X_{i-1} \dots X_1) \\ &\geq H(X_1) + \sum_{i=2}^{M} H(X_i | X_{i-1} \dots X_1, \pi(i)) \\ &= H(X_1) + \sum_{i=2}^{M} H(X_i | X_{\pi(i)}) \\ &= \alpha \cdot \log N + (M - \alpha) \cdot H(D) \\ &\geq M \cdot H(D) \end{aligned}$$

Where α is the number of accesses such that $\pi(\cdot) = \phi$ i.e., accessed for the first time. We know that $1 \leq \alpha \leq N$ and the entropy rate is $\frac{1}{M} H(X_1, X_2, \dots H_M) \geq H(D)$.

For the chosen distribution D, we can compute $H(D)$ as:

$$H(D) = -(N - 2^{L-k}) p_{\min} \log p_{\min} - 2^{L-k} p_{\max} \log p_{\max} \tag{16}$$

since there are $N - 2^{L-k}$ leaves with probability $p_{\min}$ and 2^{L-k} leaves with probability $p_{\max}$. The entropy rate is hence lower bounded by the expression in Eq. 16. In a similar manner, we compute the min-entropy of the observed sequence $H_\infty(X_1, X_2, \dots, X_M)$[15].

$$\begin{aligned} H_\infty(X_1, X_2, \dots H_M) &= -\log\left[(1/N)^\alpha \cdot p_{\max}^{M-\alpha}\right] \\ &\geq -M \cdot \log p_{\max} \end{aligned} \tag{17}$$

Fig. 7 plots the lower bound on the entropy rate of the observed access sequence as a function of the ϵ (numerically). We can see that the entropy rate decreases as ϵ increases but the decrease is small for moderate values of ϵ. For instance, for $L = 20$, using $\epsilon = 3$ results in a loss of roughly 0.3-bits of entropy and for $\epsilon = 2$ the loss is 0.1 bits (contrast this with the performance improvements presented in Section 6.2 and Fig. 3a). Even at larger values of ϵ, the loss is less than 2-bits.

The entropy loss analysis bounds the *reduction in the entropy of the next access observed by the adversary given access to an arbitrary number of previous accesses (hence the asymptotic analysis).* This can be used to argue about the entropy loss of the entire sequence as a function of the number of accesses (using the bounds from Fig. 7 and Eq. 16,17). In other words, given a number of accesses M, the baseline of perfect security would guarantee an uncertainty of the observed sequence to be $M \cdot \log N = 20 \cdot M$-bits (considering $L = \log N = 20$) whereas using DP-ORAM will reduce this entropy to $M \times$entropy per access. To put these in perspective, for an access sequence of length 10^3 the entropy loss for $\epsilon = 2$ will reduce the entropy of the observed sequence from 20,000-bits to 19,900-bits (by about 0.5%).

15 Min-entropy is defined as $H_\infty(X) = -\log \max_i p(x_i)$.

C Advanced Composition theorem - DP

Advanced composition theorem bounds the privacy guarantees of a mechanism under *k-fold adaptive composition* (for details refer to Chapter 3 of [20]). Formally it is stated as:

Theorem 10 (Advanced composition). ***For all*** $\epsilon, \delta, \delta' \geq 0$***, the class of*** (ϵ, δ)***-differentially private mechanisms satisfies*** $(\epsilon', k\delta + \delta')$***-differential privacy under k-fold adaptive composition for:***

$$\epsilon' = \sqrt{2k \ln(1/\delta')}\epsilon + k\epsilon(e^\epsilon - 1) \tag{18}$$

The above result provides a guideline for setting the privacy budget under composition. Using the advanced composition theorem, we can see that the privacy budget scales sub-linearly with the number of queries (Example 3.7 in [20]).