Tunably-Oblivious Memory: Generalizing ORAM to Enable Privacy-Efficiency Tradeoffs

Jonathan Dautrich
Google, Inc.
Irvine, California
jjldj@google.com

Chinya Ravishankar
Computer Science and Engineering
University of California, Riverside
ravi@cs.ucr.edu

ABSTRACT

We consider the challenge of providing privacy-preserving access to data outsourced to an untrusted cloud provider. Even if data blocks are encrypted, access patterns may leak valuable information. Oblivious RAM (ORAM) protocols guarantee full access pattern privacy, but even the most efficient ORAMs to date require roughly $\ell \log_2 N$ block transfers to satisfy an ℓ-block query, for block store capacity N.

We propose a generalized form of ORAM called Tunably-Oblivious Memory (λ-TOM) that allows a query's public access pattern to assume any of λ possible lengths. Increasing λ yields improved efficiency at the cost of weaker privacy guarantees. 1-TOM protocols are as secure as ORAM.

We also propose a novel, special-purpose TOM protocol called *Staggered-Bin TOM* (SBT), which efficiently handles large queries that are not cache-friendly. We also propose a read-only SBT variant called Multi-SBT that can satisfy such queries with only $O(\ell + \log N)$ block transfers in the best case, and only $O(\ell \log N)$ transfers in the worst case, while leaking only $O(\log \log \log N)$ bits of information per query. Our experiments show that for $N = 2^{24}$ blocks, Multi-SBT achieves practical bandwidth costs as low as 6X those of an unprotected protocol for large queries, while leaking at most 3 bits of information per query.

Categories and Subject Descriptors

H.2.7 [**Database Management**]: Database Administration—*security, integrity, and protection*

Keywords

Data privacy; Oblivious RAM; privacy tradeoff

1. INTRODUCTION

It has become common for resource-constrained clients to outsource data storage and management to *cloud* servers lying beyond their administrative control. Such outsourcing, however, raises data privacy concerns. Unfortunately, merely encrypting data does not ensure privacy, since information is leaked by access patterns on encrypted data [4,11].

Oblivious RAM (ORAM) protocols [9] can guarantee full access pattern privacy in an outsourced block store. ORAM protocols use dummy block reads and periodic oblivious data block re-shufflings to guarantee that any two access patterns of the same length are computationally indistinguishable to any outside observer, including the server itself. The added costs incurred when using ORAM for data outsourcing are generally dominated by *bandwidth cost*, which we measure as the number of actual block transfers needed to satisfy a single block *access* (read or write).

ORAM bandwidth costs range from $O(\sqrt{N \log N})$ [1] to $O(\log N)$ [6, 10, 22, 23], where N is the ORAM block capacity. Recently, there has been a push to make ORAM practically, as well as asymptotically, efficient [14,22,24,25]. The most bandwidth-efficient ORAM construction known to date [22, 23] still incurs a bandwidth cost of roughly $\log_2 N$. Other protocols [8,13,16,24,25] use less client space than [22,23], but incur higher bandwidth costs. Some such protocols target use cases such as secure coprocessors [15], where bandwidth efficiency is less critical than client space.

This $\log_2 N$ cost is particularly disappointing for multi-block read-only queries, where we might expect better performance. To achieve full access pattern indistinguishability, ORAMs must ensure that all queries generate public access patterns of roughly the same length, regardless of access locality or ORAM state. As a result, all queries must incur the same, worst-case cost.

To avoid this limitation, we build special-purpose ORAM-like protocols that leak a strictly bounded amount of access pattern information in order to obtain a bandwidth cost under $\log_2 N$ for large queries. Existing schemes that partially protect access patterns (e.g. [7,18]) start with unprotected protocols and add obfuscation mechanisms to quantifiably limit the adversary's ability to make certain inferences. However, they do not consider all possible inferences, and thus cannot assess total information leakage. In contrast, we start with a fully protected protocol (ORAM) and carefully relax its privacy requirements in order to tightly bound the total access pattern information leaked.

1.1 Our Contributions

We propose Tunably-Oblivious Memory (TOM), a new model that relaxes and generalizes the traditional ORAM model, allowing controlled trade-offs between efficiency and information leakage. TOM permits variable-length public access patterns, allowing properties such as locality to be exploited to improve efficiency. Queries are distinguishable by access pattern length, so for each query λ-TOM generates an access pattern with one of λ pre-determined lengths, limiting information leaked per query to $\log_2 \lambda$ bits. λ-TOM protocols with large λ are more flexible and efficient, but leak more information. Protocols with small λ are more rigid, but offer better privacy. 1-TOM leaks no information, and has security equivalent to a traditional ORAM.

TOM can directly improve efficiency for queries showing locality by simply enhancing ORAM with a local block cache. However, we address the more challenging problem of building a TOM that efficiently handles workloads that are *not* cache-friendly. To this end, we propose a novel, special-purpose TOM called *Staggered-Bin TOM* (SBT). We prove that SBT achieves bandwidth cost $O(\log N/\log\log N)$ for large queries with blocks chosen uniformly at random, but has worst-case cost $O(\sqrt{N})$.

We also propose three read-only SBT variants, culminating in the Multi-SBT, which combines the SBT with a traditional ORAM, storing three copies of each block. The Multi-SBT achieves bandwidth cost $O(1)$ for large uniform random block queries, and $O(\log N)$ in the worst case, while leaking only $O(\log\log\log N)$ bits per query. Thus, Multi-SBT can satisfy any ℓ-block uniform random block query using only $O(\ell + \log N)$ block transfers.

We developed a simulator to evaluate SBT and its variants, and compare practical costs of the Multi-SBT with the ORAM in [23]. We show that Multi-SBT maintains a practical bandwidth cost of roughly 6X for queries of $4\sqrt{N}$ blocks, while [23] has substantially larger costs ranging from 22X to 29X for similar parameterizations (see Table 1).

The rest of this paper is organized as follows. Section 2 covers related work in protecting access pattern privacy. Section 3 presents the TOM model and its security definition. We describe the SBT in Section 4 and its variants in Section 5, with detailed performance analyses in the Appendix. Section 6 gives experimental results from our simulator comparing SBT and its variants.

2. RELATED WORK

2.1 ORAM and PIR Protocols

We focus on showing that the Multi-SBT outperforms the *Practical ORAM* in [22, 23] because it remains the most bandwidth-efficient single-server ORAM, and incurs a similar client space cost ($O(N)$ with low constant). Both Practical ORAM [22,23] and the SBT logically partition blocks on the server. In [22,23] each partition is itself an ORAM, so the bandwidth cost remains logarithmic. In Multi-SBT, every fetch retrieves a potentially-usable block, enabling constant bandwidth cost in the best case.

ORAMs that emphasize reduced client space incur even higher bandwidth costs. Assuming 64KB blocks, Practical ORAM [22, 23] requires $\log_2 N$ bandwidth cost. Path ORAM [24] requires closer to $8\log_2 N$, and more if client space is reduced using recursion. The ORAMs in [13] and [8] both have asymptotic bandwidth cost $O(\log^2 N/\log\log N)$, and are outperformed in practice by Path ORAM [24]. Multi-cloud oblivious storage [21] achieves very low bandwidth cost (under 3X), but assumes multiple non-colluding servers.

Private Information Retrieval (PIR) techniques also support secure data outsourcing with full access pattern indistinguishability. PIR alone is generally computationally impractical [20], but progress has been made mixing ORAM with PIR to reduce bandwidth costs [5,16]. Due to the high latencies and drastic computation costs of such schemes, we do not compare the Multi-SBT with them here.

2.2 Partial Access Pattern Protection

Several efficient protocols have been proposed that *partially* protect access patterns. One example is the *Shuffle Index* [7], which uses an unchained B+ tree to store encrypted blocks. *Cover searches* provide access pattern privacy by making dummy block requests to obscure the true request. The authors quantify the adversary's ability to recognize that two given accesses correspond to the same block, but ignore other information leaks. For example, the protocol may run indefinitely without retrieving certain blocks. Since the adversary knows that such blocks are rarely requested, he can use their eventual request pattern to make additional inferences. In contrast, TOM's bounds on total information leakage hold for all inferences. Shuffle Index bandwidth cost is 16X, but drops to 4X with enough client space to store pointers to each block.

The protocol in [18] reads 2 blocks for every request, does no oblivious shuffling, and achieves a bandwidth cost as low as 4X even with limited client space. Like the Shuffle Index, it bounds the adversary's ability to correlate two accesses, but leaks even more unquantified information, via access patterns of rarely requested blocks, than the Shuffle Index.

Like TOM, the private computation protocol of [26] uses an ORAM and allows a bounded amount of access pattern information to leak in order to improve efficiency. However, the notions are otherwise fundamentally different. The protocol in [26] accesses main memory from trusted hardware via a black-box ORAM, using the additional space to enable more elaborate computations. Applications vary in the number of required ORAM fetches per computation. Leakage comes through each application's one-time maximum fetch rate choice. In contrast, TOM allows fetch counts to vary dynamically, letting the ORAM adjust fetch counts to match workloads, leaking information per query instead of per application setup. Thus, TOMs see efficiency improvements when the average number of fetches is small, even if the worst-case number is large.

3. TUNABLY-OBLIVIOUS MEMORY

3.1 ORAM Review

Oblivious RAM (ORAM) techniques [9] provide a mechanism for outsourcing encrypted data while ensuring that all possible access patterns are computationally indistinguishable to all observers other than the client, including the server itself. In an ORAM protocol, the client arranges his data in N fixed-size blocks of B bits each. Each block has a unique address $a \in \{0, 1, \ldots, N-1\}$. Each of the N blocks is encrypted using a semantically secure encryption scheme and then stored on the server. Every time a block a is written to the server, it is re-encrypted using a different nonce, and assigned a new server-side ID, preventing it from being directly linked to previous encrypted versions of a.

The goal of ORAM is to define an efficient protocol that re-shuffles and re-encrypts blocks to ensure that no information is leaked about the address or contents of each block, how frequently a given block is accessed, and whether the access is a read or write. The protocol may incorporate *dummy*

Table 1: Comparison of [23] with results based on proposed Multi-SBT using the ORAM component from [23], with the parameterizations and costs given below and in [23], with 64 KB block size. Multi-SBT average cost is for uniform random queries of length $\ell = 4\sqrt{N}, \lambda = 8$. Max. cost is three times ceiling of ORAM cost.

		ORAM [23]			Multi-SBT using ORAM from [23]				
N	ORAM Capacity	Client Storage	Server Storage	Cost	Client Storage	Server Storage	Avg. Cost	Cost Upper-Bound	Leaked Bits / Access
2^{20}	64 GB	204 MB	205 GB	22.5X	604 MB	333 GB	5.4X	69X	$3 \cdot 2^{-12}$
2^{22}	256 GB	415 MB	819 GB	24.1X	1.2 GB	1.3 TB	6.0X	75X	$3 \cdot 2^{-13}$
2^{24}	1 TB	858 MB	3.2 TB	25.9X	2.6 GB	5.2 TB	6.3X	78X	$3 \cdot 2^{-14}$
2^{28}	16 TB	4.2 GB	51 TB	29.5X	13.6 GB	83 TB	5.8X	90X	$3 \cdot 2^{-16}$

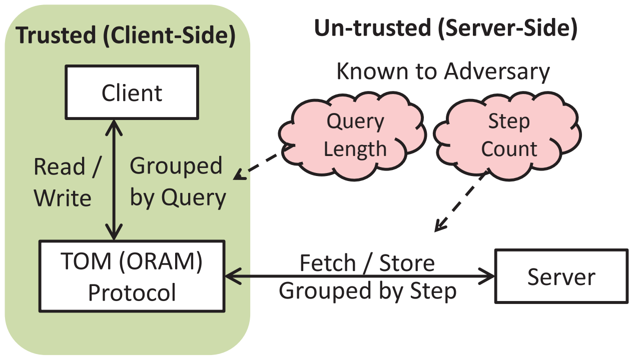

Figure 1: Client issues secret accesses (read/write) to ORAM/TOM protocol, which translates to a sequence of public accesses (store/fetch) to the server. Adversary knows query length (# requests in query) and step count (# steps needed to satisfy query).

blocks, which contain no data but are indistinguishable from encrypted data blocks.

A client interacts with the ORAM protocol as with a trusted block store (Figure 1), issuing a *secret access pattern* $\vec{S} = (s_1, \ldots, s_{|\vec{S}|})$ of block requests. Each secret access s is a triple $(type, a, data)$, where *type* is the access type (*read* or *write*), a is the local address of the block to access, and *data* is the plaintext data written to block a, if any.

The ORAM translates $\vec{S}$ into a *public access pattern* $P(\vec{S}) = (p_1, \ldots, p_{|P(\vec{S})|})$ that is generally much longer than $\vec{S}$. Each public access p is also a triple $(type, id, edata)$, where *type* denotes the access type (*store* or *fetch*), *id* denotes the server-side ID of the accessed block, and *edata* denotes the encrypted block data to be stored, if any. A fetch optionally removes the block from the server.

The term *access pattern* has been used in the literature ambiguously to refer to either $\vec{S}$ or $P(\vec{S})$. We disambiguate by calling $\vec{S}$ the *secret* access pattern and $P(\vec{S})$ the *public* access pattern. We now give the standard ORAM security definition of [23] in terms of our notation:

Definition 1. A protocol satisfies ORAM security if for every pair of secret access patterns $\vec{S_1}$ and $\vec{S_2}$ of the same length ($|\vec{S_1}| = |\vec{S_2}|$), $P(\vec{S_1})$ and $P(\vec{S_2})$ are computationally indistinguishable (to every observer other than the client).

If the ORAM block size B is reasonably large ($B \gg \log_2 N$), the communication cost is dominated by block transfers. The ORAM makes $|P(\vec{S})|$ block transfers to satisfy $\vec{S}$, while an unprotected protocol needs only $|\vec{S}|$ transfers. Thus the *bandwidth cost* of using ORAM to obscure an access pattern is given by $\frac{|P(\vec{S})|}{|\vec{S}|}$. The more efficient an ORAM, the lower its bandwidth cost.

3.2 Trading Obliviousness for Efficiency

We introduce the term *step* to refer to a discrete unit of work performed by an ORAM or TOM. Informally, each step retrieves a single encrypted *target* block from the server. Each step may also fetch and store other blocks in order to obscure the target block's identity or prepare for future requests (e.g. shuffling).

In a traditional ORAM, each secret access yields exactly one such step, and the target block is simply the block associated with the secret access. Each ORAM step is powerful in that it can obliviously retrieve any given target block from the server, but this power also makes each step expensive. Informally, the *step count* is the total number of steps needed to satisfy a given secret access pattern. The step count must match the number of secret accesses in order to satisfy ORAM's perfect privacy guarantee, so such powerful, expensive steps are mandatory.

In contrast, the TOM generalization allows the step count to vary, creating the possibility for more efficient but less powerful steps, and thus for more efficient protocols. For example, the Staggered Bin TOM (Section 4) partitions the blocks on the server into k bins (Figure 2). Each step may only retrieve a target block from a single, pre-determined bin. Each such step is thus less powerful than an ORAM step, but it is also more efficient. In the worst case, k steps are needed to satisfy a single secret access, but by carefully scheduling secret accesses from the same multi-block query, we can obtain lower overall bandwidth cost than a comparable ORAM. Allowing the step count to vary inevitably leaks some access pattern information. We show how to tightly bound such information in Sections 3.4 and 3.5.

We now define the TOM model more precisely. For each secret access pattern $\vec{S}$, a TOM generates a public access pattern $P(\vec{S})$ divided into a sequence $\sigma(\vec{S})$ of discrete *steps*. We use $|\sigma(\vec{S})|$ to denote the *step count* of $P(\vec{S})$.

Definition 2. Each *step* is a series of stores and fetches used by the TOM protocol to retrieve a single *target* block from a subset of the blocks on the server. A step is complete when the TOM is ready to retrieve another target block.

Traditional ORAMs are special cases of TOM in which each secret access generates exactly one step ($|\sigma(\vec{S})| = |\vec{S}|$). Thus ORAM does not distinguish between *secret access* and *step*, necessitating our new terminology for TOM. In ORAM, the block subset accessible during a step includes all blocks on the server, while in Staggered Bin TOM (Section 4) it only includes blocks from one bin.

We say $\vec{S}$ is *satisfied* once all the steps in $\sigma(\vec{S})$ are complete. As in ORAM, if the TOM is *stateful*, some blocks updated by $\vec{S}$ may not be stored to the server immediately. Instead, even after the step completes, they are held locally as *dirty* blocks until they are written back to the server during a subsequent step.

Definition 3. A *query* is a secret access pattern $\vec{S}$ composed of a batch of secret accesses that may be satisfied in any order.

A TOM receives multi-block *queries* from the client. Queries are handled sequentially relative to each other, but accesses within a query may be processed in any order. For security, *query* and *secret access pattern* are interchangeable.

TOM decouples *steps* from *secret accesses*, allowing query length $|\vec{S}|$ to differ from step count $|\sigma(\vec{S})|$. This approach offers better efficiency than ORAM for two reasons. First, TOM need not generate steps for accesses to cached blocks. In ORAM, a repeat access to a recently cached block must still incur the overhead of a step, else the reduced step count would reveal the repeated access. Second, TOM need not require that each step be capable of accessing any block. By reducing the power of each step, TOM makes steps more efficient, potentially reducing a query's total bandwidth cost, even though the step count may increase. The SBT and its variants (Sections 4 and 5) exploit this second advantage.

3.3 TOM Security Definition

As in ORAM, we assume that query length $|\vec{S}|$ is public. We also make the worst-case assumption that the adversary can observe precisely when each query starts and ends, and thus knows the exact step count $|\sigma(\vec{S})|$ of each query.

In ORAM, $|\sigma(\vec{S})| = |\vec{S}|$, so $|\sigma(\vec{S})|$ reveals nothing new to the adversary. In TOM, $|\vec{S}|$ and $|\sigma(\vec{S})|$ may differ, so $|\sigma(\vec{S})|$ may leak information. For example, if $|\sigma(\vec{S})| < |\vec{S}|$, the adversary may infer that $\vec{S}$ contains repeated accesses. We limit such leakage by forcing $|\sigma(\vec{S})|$ to assume one of λ *milestone* values taken from a predefined set $\mathcal{M}_{|\vec{S}|}$. More milestones improve flexibility in generating $\sigma(\vec{S})$ and thus improve efficiency, but also leak more information about $\vec{S}$.

$\mathcal{M}_{|\vec{S}|}$ is defined up-front for each value of $|\vec{S}|$, so the milestones themselves do not leak information. Since the adversary knows $|\vec{S}|$, he already knows that $|\sigma(\vec{S})|$ will be one of the λ milestones. Thus, he only learns information through the specific choice of milestone used for $|\sigma(\vec{S})|$. Equivalently, he learns which of λ equivalence classes $\vec{S}$ belongs to, limiting information leakage by the size of λ.

We now define security for λ-TOM, which translates a secret access pattern $\vec{S}$ into a public access pattern with one of λ milestone step counts.

Definition 4. A protocol satisfies λ-TOM security if both of the following conditions hold for every possible pair of secret access patterns $\vec{S_1}$ and $\vec{S_2}$:

1. Let $\ell = |\vec{S_1}|$. If $|\vec{S_1}| = |\vec{S_2}|$ then $|\sigma(\vec{S_1})|, |\sigma(\vec{S_2})| \in \mathcal{M}_\ell$, where $\mathcal{M}_\ell$ is a set of *milestones* of cardinality $\leq \lambda$.
2. If $|\sigma(\vec{S_1})| = |\sigma(\vec{S_2})|$, then $P(\vec{S_1})$ and $P(\vec{S_2})$ are computationally indistinguishable (outside the client).

By ensuring that any two public access patterns with the same step count are indistinguishable, we guarantee that information about $\vec{S}$ only leaks through the observation of the step count $|\sigma(\vec{S})|$, which is in turn limited to one of λ milestones. We can bound the information leakage I_λ of a λ-TOM protocol by assuming the worst case, in which all milestones are equi-probable, giving:

Lemma 1. *A λ-TOM protocol leaks at most $I_\lambda \leq \log_2 \lambda$ bits per query in expectation.*

Proof. Let R be a random variable representing the choice of milestone. The expected information leaked by revealing the outcome of R is given by the entropy $H(R)$. $H(R)$ is maximized when each of the λ milestones is equi-probable, giving $H(R) = \log_2 \lambda$ bits. Thus we have that $I_\lambda \leq \log_2 \lambda$. □

When $\lambda = 1$, the leakage is $I_\lambda = 0$, which indicates that 1-TOM is as strong as ORAM. In fact, for $\lambda = 1$, we have by Condition 1 of Definition 4 that $|\vec{S_1}| = |\vec{S_2}|$ implies $|\sigma(\vec{S_1})| = |\sigma(\vec{S_2})|$, and thus by Condition 2 that $|\vec{S_1}| = |\vec{S_2}|$ implies $P(\vec{S_1})$ and $P(\vec{S_2})$ are indistinguishable. Therefore any 1-TOM protocol satisfies ORAM security (Definition 1). The reverse is also true for any ORAM with a notion of steps. In any case, we make no claim that 1-TOM is substantively more secure than ORAM, so we treat 1-TOM and ORAM as equivalent.

Since each query leaks at most I_λ bits, larger queries leak less information per access. Combining small, independent queries would reduce leakage, but may also increase latency. It is critical that no query results be released to the client until the entire query is satisfied. If the client used partial results, the partial completion time might leak, revealing additional information. Thus, query size is limited by the size of the results cache allocated to the TOM, and excessively large queries may need to be broken up. In standard ORAM, $I_\lambda = 0$, so there is no motivation to make queries larger than a single block access.

What the adversary *gains* from leaked access pattern information depends heavily on what other information the adversary holds. Other schemes that obscure access patterns (e.g. [7]) focus on quantifying the adversary's inability to make particular inferences, but do not assess holistic information loss. In contrast, we upper-bound the total access pattern information leakage, and leave it to the client to decide how much leakage is acceptable given the application.

3.4 Paddable TOM Protocols

We now show how to construct a λ-TOM for any given λ from a *paddable* TOM. Intuitively, we start by choosing λ milestones, then delay each query's completion by silently padding it with dummy steps until its step count reaches a milestone. We use S_{MAX} to denote the worst-case per-access step count.

Definition 5. A protocol is a *Paddable TOM* if it satisfies:

1. Condition 2 of Definition 4 (indistinguishable patterns)
2. It has finite upper bound $\ell \cdot S_{\text{MAX}}$ on step count $|\sigma(\vec{S})|$ generated by a secret access pattern of length $\ell = |\vec{S}|$.
3. Any $P(\vec{S})$ may be *padded* by adding any number of *dummy* steps, increasing $|\sigma(\vec{S})|$ by any amount.

We can coerce any paddable TOM into satisfying λ-TOM for any given λ. We first define appropriate milestones for $\mathcal{M}_\ell$, then instruct the protocol to pad every public access pattern with dummy steps, increasing $|\sigma(\vec{S})|$ to the smallest milestone in $\mathcal{M}_\ell$ greater than or equal to the original step count. If we trivially set $\mathcal{M}_\ell = \{\ell \cdot S_{\text{Max}}\}$, and translate every secret access pattern of length ℓ, with padding, into a public access pattern with step count $\ell \cdot S_{\text{Max}}$, we satisfy 1-TOM and thus ORAM security.

Efficient paddable protocols will generate step counts much smaller than $\ell \cdot S_{\text{Max}}$, so the padding required to reach $\ell \cdot S_{\text{Max}}$ may incur substantial bandwidth cost. Increasing λ (adding milestones) can reduce cost, but also reduces privacy. To make the best possible tradeoffs, our strategy for choosing milestones should minimize cost due to padding for any λ.

3.5 Log-Spacing for Paddable Protocols

Let $m = |\sigma(\vec{S})|$ be the original step count generated from query $\vec{S}$ of length $\ell = |\vec{S}|$. We may have $m < \ell$ if most queried blocks are cached, but such cases are too rare to merit dedicated milestones, so we assume $\ell \leq m \leq \ell \cdot S_{\text{Max}}$.

Let m' be the smallest milestone in $\mathcal{M}_\ell$ such that $m' \geq m$. In a paddable TOM, the fractional increase in step count, and thus bandwidth cost, is given by the *padding factor* m'/m. Let δ be the maximum padding factor (maximum possible value of m'/m). Given λ, we propose to minimize δ by log-spacing milestones as multiples of ℓ over $[\ell, \ell \cdot S_{\text{Max}}]$:

$$\mathcal{M}_\ell = \left\{ \left\lceil \ell \, (S_{\text{Max}})^{i/\lambda} \right\rceil \mid i \in \mathbb{Z}, 1 \leq i \leq \lambda \right\}. \quad (1)$$

This spacing strategy minimizes the maximum padding factor δ, ensuring:

$$\delta \leq \left\lceil (S_{\text{Max}})^{1/\lambda} \right\rceil. \quad (2)$$

To minimize λ for given δ, we solve $(S_{\text{Max}})^{1/\lambda} \leq \delta$ for λ:

$$\lambda \geq \frac{\log S_{\text{Max}}}{\log \delta} = \log_\delta S_{\text{Max}}. \quad (3)$$

These expressions reveal a clear tradeoff between privacy (λ) and efficiency (δ). Smaller S_{Max} can improve privacy *and* efficiency, which is unsurprising since ORAMs fix privacy at $\lambda = 1$ and seek to reduce the worst-case per-access cost.

3.6 Assessing TOM Information Leakage

Since each query in a λ-TOM protocol leaks at most a fixed I_λ bits of information, smaller queries leak more information per access. Thus TOM is best applied in scenarios where queries are large or can be easily batched.

Consider a TOM with worst-case per-access step count $S_{\text{Max}} \in O(\mathit{polylog}(N))$. The log-spacing strategy with $\delta = 2$ gives $\lambda \in O(\log \log N)$, and thus a leakage per query of only $I_\lambda \in O(\log \log \log N)$ bits per ℓ-block query. Even a less efficient TOM with $S_{\text{Max}} \in O(\sqrt{N})$ leaks only $I_\lambda \in O(\log \log N)$ bits, which is still far better than the $O(\ell \log N)$ bits leaked by an unprotected protocol.

What the adversary actually *gains* from the leaked access pattern information depends heavily on when each step count was observed and what other information the adversary holds. Other schemes that obscure access patterns (e.g. [7]) focus on quantifying the adversary's inability to make particular types of inferences, but do not address all possible inferences, and thus do not assess holistic information loss. In contrast, we upper-bound the total access pattern information leakage, and leave it to the client to decide how much leakage is acceptable given the application.

Table 2: SBT and TOM Notation

δ	Max. padding factor (padding cost increase)
H	Max. fetch queue length, before padding
All queries, strict upper bound:	
S_{Max}	Worst-case per-access step count, before padding
C_{Max}	Worst-case bandwidth cost, after padding
Large uniform rand. block queries, high-prob. bound:	
C_{HP}	High-prob. bandwidth cost, after padding

4. STAGGERED-BIN TOM

Here we present a novel λ-TOM protocol, called *Staggered-Bin TOM* (SBT), that reduces costs even for large queries that are not cache-friendly. In Section 5 we propose three read-only variants of SBT that store multiple copies of each block and reduce costs by choosing the most convenient copy to fetch. Table 2 gives some key notation, and Table 3 compares performance of SBT variants.

As noted in Section 3.3, an ORAM is simply a 1-TOM. TOM allows us to decouple steps from secret accesses, so we could improve on ORAM performance by simply increasing λ and adding a local block cache. We could then satisfy most cached block accesses without stepping the λ-TOM (without block transfers), while leaking only $\log_2 \lambda$ bits per multi-block query. However, caching only improves performance when secret access patterns exhibit temporal locality.

4.1 SBT Architecture

An SBT contains N blocks of B bits each placed in $n+1$ logical *bins*, each with a maximum capacity of n blocks. We initialize the SBT by filling the bins with $n, n-1, \ldots, 1, 0$ blocks, respectively, and storing them on the server. The SBT always keeps n more blocks locally, for $N = n(n+3)/2$ blocks total (Figure 2).

We choose n to be the smallest integer such that $N \leq n(n+3)/2$, and add up to n extra data blocks to increase SBT capacity N to exactly $n(n+3)/2$. *No unusable dummy blocks of any kind are added*, keeping server storage overhead low. Bins are purely logical structures, so the server is free to use any physical configuration for storing blocks.

The SBT needs local (client-side) storage space for three purposes. First, it requires Bn bits for the n blocks always stored locally. Second, it needs $B\ell$ bits to cache the results of an ℓ-block query, so that all ℓ blocks can be simultaneously released to the client. Finally, as in [23], the SBT needs a small amount of space for each of the N blocks to record its server ID, containing bin's index, and a list of block addresses in each bin, for a total of roughly $2 \log_2 N$ bits per block.

In all, approximately $B(n+\ell) + 2N \log_2 N$ bits of client storage are required. Though these storage requirements may seem high, [23] and [22] note that B is large enough in practice that the space needed to store $n \approx \sqrt{2N}$ blocks is comparable to the space needed for the meta-data of all N blocks. For example, with $N = 2^{30}$ blocks, and block size $B = 64$KB, we need under 8GB for the meta-data, and up to 8GB local block storage for queries of $\ell = 3\sqrt{N}$ blocks.

Table 3: Comparison of our λ-TOM protocols, block size B. Numbers approximate; estimated average costs taken from Figures 5–7. Smaller $C_{\mathbf{Max}}$ improves privacy/efficiency tradeoff. $\lambda \leftarrow S_{\mathbf{Max}}$ for constant δ. $\lg \equiv \log_2$

Protocol	Worst-Case C_{MAX}	Uniform Rand. Block C_{HP} for $\ell \approx 4\sqrt{N}$	Bits Leaked Per ℓ-Block Query	Efficient Write	Server Storage (Bits)	Client Storage (Bits)
Unprotected	1	1	$\ell \lg N$	Yes	NB	$O(1)$
ORAM [23]	$\lg N$	$\lg N$	0	Yes	$\leq 4NB$	$3B\sqrt{N} + 1.25N \lg N$
SBT	$2\sqrt{2N}$	$O\left(\frac{\log N}{\log \log N}\right)$	$\lg \lg(\sqrt{2N})$	Yes	NB	$(\ell + \sqrt{2N})B + 2N \lg N$
2-Choice SBT	$4\sqrt{N}$	$O(1)$, $\approx$ (3–5)	$\lg \lg(2\sqrt{N})$	No	$2NB$	$(\ell + 2\sqrt{N})B + 4N \lg N$
SBT+ORAM	$3 \lg N$	$O(\log \log N)$	$\lg \lg(3 \lg N)$	No	$\leq 5NB$	$(\ell + \sqrt{2N} + 3\sqrt{N})B + 3.25N \lg N$
Multi-SBT	$3 \lg N$	$O(1)$, $\approx$ (4–7)	$\lg \lg(3 \lg N)$	No	$\leq 6NB$	$(\ell + 5\sqrt{N})B + 5.25N \lg N$

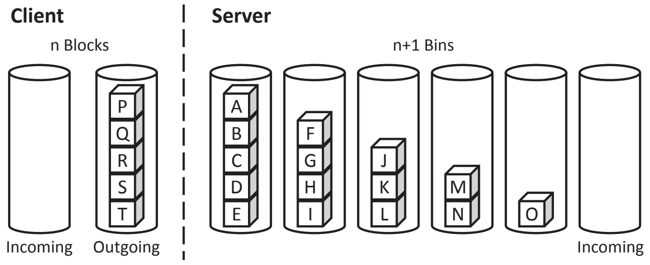

Figure 2: SBT in its initial state, with $n = 5$ blocks on the client, and $n(n+1)/2$ on the server. The empty server-side *incoming* bin will be filled in, one block at a time, by the n blocks from the client.

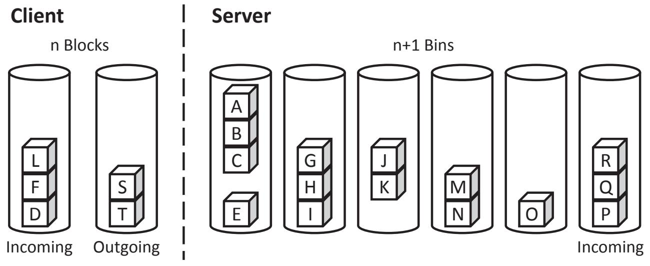

Figure 3: SBT after 3 steps. Server bins are accessed in a round-robin fashion. Blocks L, F, D have been fetched to the client-side *incoming* bin, and blocks R, Q, P stored to server-side *incoming* bin.

4.2 SBT Operation

Each step in the SBT fetches one block from and stores one block to the server. SBT operation is best described in terms of *passes* of n steps each. A pass fetches and removes one block from each of the n non-empty bins in order, and stores n blocks to the previously empty bin. After each pass, the bin load pattern rotates by 1 bin, and fetches continue round-robin (Figure 3). After each pass, the SBT re-encrypts the n fetched blocks, randomly permutes them, assigns them to the empty bin, and generates new server-side IDs to prevent linking to old copies.

Each query consists of ℓ secret accesses for distinct block addresses. A query may begin or end at any point during a pass. The SBT cannot change the one-per-bin round-robin fetch pattern, but may choose which block to fetch from each bin. Thus, a single-block query can always be satisfied in n steps, since we fetch at least one block from every non-empty bin. Similarly, any ℓ-block query takes at most ℓn steps ($S_{\text{MAX}} = n$), since we always retrieve at least one target block per pass, even if all ℓ blocks are in one bin.

The SBT maintains a *fetch queue* for each bin. To start a query, we identify the bin containing each block to be accessed, and add an appropriate fetch to that bin's fetch queue. When the SBT is ready to issue a fetch for bin i, it first checks i's fetch queue. If the queue is non-empty, the next fetch is dequeued and dispatched to the server. Otherwise, a dummy fetch is generated for a randomly chosen block from the bin. Once all fetch queues are empty and all outstanding fetches finish, the query is satisfied and results are released to the client.

All fetches must proceed in order, as must all stores. Further, to maintain n blocks on the client, a given step's store cannot begin until its fetch completes. However, stores may trail their corresponding fetches as much as necessary to ensure full network bandwidth utilization. That is, we may initially let fetches get several steps ahead of stores, so that many stores and fetches run concurrently.

4.3 SBT Security

We now show that SBT meets the Paddable TOM criteria in Definition 5. We have shown that SBT has a finite step count upper bound ℓn, so it remains to show that public access patterns with the same step count are indistinguishable (Condition 2 of Definition 4), and that public access patterns may be padded.

Theorem 1. *In the SBT, for any two public access patterns $P(\vec{S_1}), P(\vec{S_2})$, if $|\sigma(\vec{S_1})| = |\sigma(\vec{S_2})|$, then $P(\vec{S_1})$ and $P(\vec{S_2})$ are computationally indistinguishable.*

Proof. First, the order in which the SBT fetches from and stores to bins is fixed. Hence, any two public access patterns with the same step count must make fetches and stores to and from exactly the same sequence of bins.

Store Patterns: After each pass, the locally-stored bin of blocks to be sent to the server is randomly permuted and re-encrypted using a semantically secure encryption scheme and a fresh nonce. Blocks are then stored to the server in their permuted order. Re-encryption ensures that the server cannot distinguish the blocks. Random permutation ensures that blocks are always stored in a uniformly random order, independent of fetch order. Thus any two *store* patterns of the same length are computationally indistinguishable.

Fetch Patterns: Since the blocks within each bin were randomly permuted, each block's location in the bin is independent of its data and any prior accesses. Thus each fetch is indistinguishable from a uniformly random choice from the bin's remaining blocks, and any two fetches from one bin are indistinguishable. Thus any two *fetch* patterns of the same length are indistinguishable.

Thus, since $P(\vec{S_1})$ and $P(\vec{S_2})$ have the same step count, and there is exactly one fetch and store per step, their fetch and store patterns each have the same length and are indistinguishable. Since both fetches and stores are indistinguishable, and the pattern of when to issue fetches and stores is predetermined, $P(\vec{S_1})$ and $P(\vec{S_2})$ are themselves computationally indistinguishable. □

Theorem 2. *Any public access pattern generated by SBT may be* padded *by adding any number of dummy steps.*

Proof. We can pad any public access pattern in SBT with any number d of additional steps by issuing d *dummy* fetches for randomly chosen blocks from each of the next d bins, along with their corresponding stores. □

Theorems 1 and 2 establish that SBT is a Paddable TOM, as per Definition 5. Thus we can coerce SBT into satisfying λ-TOM for any λ. In particular, we apply the log-spacing strategy of Section 3.5 to choose the λ milestones in $\mathcal{M}_\ell$. The smaller our choice of λ, the greater our privacy but the poorer our performance. By Equation 2, for a given λ, we have a maximum padding factor:

$$\delta \leq \left\lceil (S_{\text{MAX}})^{1/\lambda} \right\rceil = \left\lceil n^{1/\lambda} \right\rceil \leq (2N)^{1/2\lambda}. \tag{4}$$

Similarly, by Equation 3, for a δ, we get a minimum milestone count λ given by:

$$\lambda \geq \log_\delta S_{\text{MAX}} = \log_\delta n. \tag{5}$$

4.4 SBT Performance

The upper-bound on the SBT's per-access step count is given by $S_{\text{MAX}} = n$, so the upper-bound bandwidth cost is given by $C_{\text{MAX}} = 2n \leq 2\sqrt{2N}$. We now determine the bandwidth cost C_{HP} that holds with high probability for large, uniform random block queries, which are queries composed of ℓ block addresses chosen uniformly at random, without replacement.

The size of each fetch queue decreases by at most one during a given pass, so the number of passes needed to satisfy a query depends on the initial length of the longest fetch queue. We use H to denote the maximum length of the longest fetch queue. The query generates step count roughly nH without padding. In the best case, each fetch queue is nearly the same length, and in the worst case all fetches are in the same queue, so we know that $\lceil \ell/n \rceil \leq H \leq \ell$.

Theorem 3. *Let $\ell \geq n$ (large queries). With high probability for the SBT with uniform random block queries:*

$$H \in O\left(\frac{\ell}{n}\frac{\log n}{\log\log n}\right).$$

We prove this theorem in Appendix A, using the observation that we can bound H by bounding the maximum urn height in the well-known *balls and urns* problem [12, 19], where balls are thrown into urns uniformly at random, with replacement. Thus, for uniform random block queries with $\ell \geq n$, the bandwidth cost, with high probability, is:

$$C_{\text{HP}} \in O\left(\delta\frac{n}{\ell}\frac{\ell}{n}\frac{\log n}{\log\log n}\right) \subseteq O\left(\frac{\delta \log N}{\log\log N}\right), \tag{6}$$

with constant δ for at least $\lambda \in \Omega(\log N)$ milestones. Thus SBT is able to satisfy large queries that are not cache-friendly with a lower asymptotic cost than the best existing ORAM protocols (cost $O(\log N)$) while leaking only $I_\lambda \in O(\log\log N)$ bits per ℓ-block query.

5. SBT VARIANTS

We now propose three read-only SBT variants: *2-Choice SBT*, *SBT+ORAM*, and *Multi-SBT* (a combination of 2-Choice SBT and SBT+ORAM). These variants store multiple copies of each block and fetch the most convenient copy available, reducing bandwidth cost to as little as $C_{\text{MAX}} = 3\log_2 N$ in the worst case, and $C_{\text{HP}} \in O(1)$ for large uniform random block queries (see Table 3).

Read-only means that the client cannot update the contents of any of his blocks. However, blocks must still be re-encrypted and stored back to the server to preserve privacy. Writes *can* be supported, but would require all copies of a block to be updated, making writes substantially more expensive than reads.

5.1 The 2-Choice SBT Variant

We construct the 2-Choice SBT by creating two copies of each of the N data blocks, and adding them all to a single SBT with capacity $2N$, which treats both copies as independent blocks. The key difference from SBT is that when the 2-Choice SBT needs to read block a, it may choose to fetch the block from either of 2 bins. It is possible that both copies of a will be in the same bin, but this state is rare and transient, persisting only until either copy is fetched.

For a given query, each of the ℓ secret block accesses yields a fetch that is assigned to one of two bins' fetch queues. We want to optimize the assignment of fetches to bins, reducing the maximum queue length H. Since we know the entire query and the block-bin mapping, the optimization resembles the *optimal multi-choice allocation* [2], and *offline Cuckoo hashing* [17] problems.

5.1.1 Random Round Robin Algorithm

We optimize fetch assignments using the iterative *Random Round Robin* (*RRR*) algorithm proposed in [2]. We describe RRR briefly, replacing *balls* with block *fetches* and *bins* with fetch *queues*.

We first guess a target maximum queue length H', starting with the minimum $H' = \lceil \ell/n \rceil$. We then run RRR to try to find an assignment of fetches to queues with actual $H \leq H'$. If the attempt fails, we increment H' and repeat. In practice, we rarely expect more than two iterations [2], so we fix a maximum iteration count $r = 5$, after which we return the best available result. The iterative RRR runs efficiently, requiring time and space in $O(r(n + \ell))$.

For each RRR iteration, each fetch starts out *uncommitted*: assigned to the queues of both bins containing its block. When we *commit* a fetch to a queue, we irreversibly remove it from its other queue. We identify any queue q with length at most H', and commit to q all its uncommitted fetches. The intuition is that since q's length is at most H' and cannot increase, it should accept its current assignment, freeing as many fetches as possible from other queues. Any time we remove a fetch from a queue, we repeat this check.

We continue by stepping through all remaining queues with uncommitted fetches, for each queue randomly choosing one uncommitted fetch to commit to the queue, followed by the length check. We continue stepping through queues until all fetches are committed. If any queues still have more than H' fetches, the RRR iteration is declared a failure.

5.1.2 2-Choice SBT Security

To an observer, the 2-Choice SBT behaves just like the SBT, except for its higher capacity. Thus, 2-Choice SBT's security follows from the arguments for SBT security in Section 4.3. In particular, 2-Choice SBT meets the criteria of Definition 5 for a Paddable TOM Protocol with worst-case per-access step count $S_{\text{Max}} = n \leq 2\sqrt{N}$. Thus, it can be coerced to λ-TOM using the log-spacing strategy. By Equation 2, the maximum padding factor δ is given by $\delta \leq \left\lceil n^{1/\lambda} \right\rceil \leq (4N)^{1/(2\lambda)}$. For a given δ, Equation 3 gives $\lambda \geq \log_\delta n$.

5.1.3 2-Choice SBT Performance

Since $S_{\text{Max}} \leq 2\sqrt{N}$, we have $C_{\text{Max}} \leq 4\sqrt{N}$.

Conjecture 1. *With high probability, the 2-Choice SBT with uniform random block queries gives $H \in O(\ell/n + 1)$.*

For a related balls and urns problem, the authors in [2] show empirically that *RRR* yields maximum urn height in $O(\ell/n+1)$, with performance nearly indistinguishable from the more complex *Selfless Algorithm*, which is proven to have maximum height $O(\ell/n + 1)$ with high probability. While we cannot provide a formal proof of Conjecture 1, we give a detailed argument supporting it in Appendix A, and show in Section 6 that it is borne out by our experiments.

Assuming Conjecture 1, the bandwidth cost of the 2-Choice SBT used on uniform random block queries with $\ell \geq n$ is, with high probability:

$$C_{\text{HP}} \in O\left(\delta \frac{n}{\ell}\left(\frac{\ell}{n} + 1\right)\right) \subseteq O(\delta), \qquad (7)$$

with constant δ for $\lambda \in \Omega(\log N)$. Thus, for large uniform random block queries, the 2-Choice SBT is highly efficient, and leaks only $I_\lambda \in O(\log\log N)$ bits/query.

Relative to SBT, the 2-Choice SBT doubles the storage space required for the server $(2N)$, and increases required client block storage from $\sqrt{2N}$ to $2\sqrt{N}$ blocks and client index space from about $2N\log_2 N$ to nearly $4N\log_2 N$ bits.

5.2 The SBT+ORAM Variant

We construct the SBT+ORAM by merging a SBT with any efficient ORAM. We store one copy of each block in the SBT and in the ORAM, and run both protocols in parallel. To read a block, we either fetch it using the SBT, or read it using a single ORAM step. For now we use the practical ORAM of [23] due to its low bandwidth cost of roughly $\log_2 N$ block transfers per secret access.

For each query, we first assign all fetches to the SBT component's fetch queues and let it run normally. After every $\log_2 N$ SBT steps, we remove one fetch from the current longest fetch queue and re-assign it to the ORAM.

5.2.1 SBT+ORAM Security

The ORAM component advances one step for every $\log_2 N$ SBT steps. Thus, in the worst case where we rely strictly on the ORAM, we need $\ell(1 + \log_2 N)$ total steps to satisfy a query of length ℓ, so the per-access step count is bounded by $S_{\text{Max}} = 1 + \log_2 N$. We now show that SBT+ORAM satisfies the indistinguishability and paddability conditions of a Paddable *TOM*.

Theorem 4. *In SBT+ORAM, for any public access patterns $P(\vec{S_1})$, $P(\vec{S_2})$, if $|\sigma(\vec{S_1})| = |\sigma(\vec{S_2})|$, then $P(\vec{S_1})$ and $P(\vec{S_2})$ are computationally indistinguishable.*

Proof. Since ORAM uses exactly one step per secret access, two public access patterns with the same step count have the same secret access pattern length. Thus, by Definition 1, any two public access patterns with the same step count generated by the ORAM are indistinguishable. By Theorem 2, any two public access patterns with the same step count generated by the SBT are also indistinguishable. Since the public access patterns generated by both protocols are indistinguishable, and the pattern of when to issue fetches from the SBT and the ORAM is predetermined, the SBT+ORAM's combined public access patterns $P(\vec{S_1})$ and $P(\vec{S_2})$ are indistinguishable. □

Using the log-spacing strategy gives $\delta \leq \lceil (S_{\text{Max}})^{1/\lambda} \rceil = \lceil (1 + \log_2 N)^{1/\lambda} \rceil$ and $\lambda \geq \log_\delta S_{\text{Max}} = \log_\delta(1 + \log_2 N)$ (Equations 2, 3). Since S_{Max} is smaller for SBT+ORAM than SBT, the privacy/efficiency tradeoff is more favorable. In particular, to limit padding to $\delta = 2$, we need only $\lambda \approx \log_2 \log_2 N$ milestones.

5.2.2 SBT+ORAM Performance

We incur $\log_2 N$ block transfers for each ORAM step and 2 transfers for each SBT step. In the worst-case, we make ℓ ORAM steps and $\ell \log_2 N$ SBT steps for an ℓ-block query, giving $C_{\text{Max}} \leq 3\log_2 N$.

We know from Theorem 3 that for large uniform random block queries, the SBT has maximum fetch queue length in $O(\ell \log N/n \log\log N)$. However, the expected queue length is only ℓ/n, so we rightly expect that relatively few queues have such large lengths. Though the ORAM runs slowly, focusing it on the largest queues first asymptotically reduces the final maximum queue length H.

Theorem 5. *Let $\ell \geq n$ and $N \geq 32$. With high probability for the SBT+ORAM with uniform random block queries we have:*

$$H \in O\left(\frac{\ell}{n}\log\log N\right).$$

In the full version of the paper [3] we present a proof for Theorem 5 based on a novel balls and urns analysis. By Theorem 5, the bandwidth cost of the SBT+ORAM used on uniform random block queries with $\ell \geq n, N \geq 32$ is, with high probability:

$$C_{\text{HP}} \in O\left(\delta \frac{n}{\ell}\frac{\ell}{n}\log\log N\right) \subseteq O\left(\delta \log\log N\right), \qquad (8)$$

with constant δ for only $\lambda \in \Omega(\log\log N)$ milestones. Thus, for large uniform random block queries, the SBT+ORAM is more efficient than the SBT. At the same time, it leaks only $I_\lambda \in O(\log\log\log N)$ bits per query, yielding better privacy than 2-Choice SBT, but slightly higher C_{HP}.

The server storage costs of [23] are reported at roughly $4BN$ bits, and we estimate that client storage is $1.25\log_2 N + 3B\sqrt{N}$ bits, based on results in Table 2 of [23]. Thus, the SBT+ORAM has a total server storage cost of roughly $5BN$, and client storage $(\ell + \sqrt{2N} + 3\sqrt{N})B + 3.25N\log_2 N$ bits.

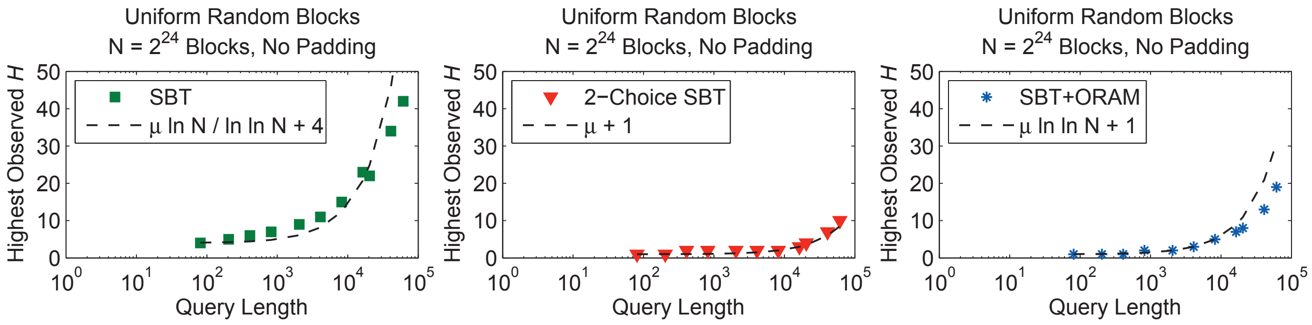

Figure 4: Analysis confirmation. Max. observed H asymptotically dominated by analytic predictions. $\mu = \ell/n$

5.3 The Multi-SBT Variant

The Multi-SBT replaces the SBT in a SBT+ORAM with a 2-Choice SBT. Thus, the Multi-SBT stores a total of three copies of each block. Its security follows directly from the security of 2-Choice SBT and SBT+ORAM.

The Multi-SBT inherits SBT+ORAM's excellent worst-case per-access step count $S_{\text{MAX}} = 1+\log_2 N$ and bandwidth cost $C_{\text{MAX}} = 3\log_2 N$. For large random block queries, it also inherits 2-Choice SBT's high-probability bandwidth cost:

$$C_{\text{HP}} \in O(\delta), \tag{9}$$

while requiring only $\lambda \in \Omega(\log\log N)$ milestones for constant δ, and leaking only $I_\lambda \in O(\log\log\log N)$ bits per query. Thus, for uniform random block queries of any size, the Multi-SBT requires only $O(\ell + \log_2 N)$ block transfers!

Multi-SBT combines the best performance and privacy characteristics of 2-Choice SBT and SBT+ORAM, and can easily outperform both. Even in worst cases, the Multi-SBT incurs at most 3 times the bandwidth cost of SBT+ORAM, or 1.5 times the cost of SBT. Multi-SBT requires total server storage of roughly $6BN$, and client storage roughly $B\sqrt{N}(3+\sqrt{2}) + 5.25N\log_2 N$ bits.

6. EVALUATION

We implemented prototypes for SBT and its variants to estimate actual bandwidth costs for various query types. The prototypes simulate secure transfers of blocks between the client and server, tracking each block's location at all times.

6.1 Maximum Queue Length Measurements

Theorems 3, 5 and Conjecture 1 give high-probability asymptotic bounds on H for large, uniform random block queries. We validated these bounds by running simulations for the corresponding SBT variants without padding, and measuring the highest observed H over $4N/\ell$ queries for various ℓ. Our results are shown in Figure 4 along with plots of concrete functions consistent with our bounds.

6.2 Simulator Details

We implemented our simulator in Java, fully modeling SBT behavior. The simulator accommodates padding, waiting to release query results until the step count reaches one of the milestones. Our simulator is synchronous, since asynchronous behavior is not needed to measure bandwidth cost.

On a single thread, the simulator requires 0.5 to 1.5μs per simulated block transfer, depending on the specific protocol and number of blocks. For the sake of speed, we do not manipulate actual block contents. Thus, we're able to efficiently evaluate SBT bandwidth costs for larger block counts and longer runs without the expense of actually performing network transfers, disk IO, and encryption.

We assume a fully de-amortized black-box ORAM with $\log_2 N$ bandwidth cost per step, based on the ORAM in [23]. When simulating the protocols with ORAM components, we step the SBT component $\log_2 N$ times, then step the ORAM once, retrieving the previous step's result.

6.3 Bandwidth Cost Experiments

Figures 5–13 give our experimental results measuring bandwidth cost for three types of queries and varying three parameters (N, ℓ, λ). Recall that bandwidth cost is given by the total number of block transfers (fetches and stores counted individually), divided by the number of secret accesses (reads or writes) ℓ.

All the experiments used a 64KB block size and allow 8GB of client space, which includes the SBT's block-ID map, space for recently fetched blocks, and space for the ORAM component, if any. Any leftover client space is used as a local block cache. Different block sizes alter storage capacity and client space, but leave bandwidth costs largely unchanged. During a trial, we run $4N/\ell$ queries of fixed length ℓ, requesting each stored block four times on average.

Each experiment varies one of: block count N (Figures 5, 8, 11), query length ℓ (Figures 6, 9, 12), or milestone count λ (Figures 7, 10, 13). Our default block count $N = 2^{24}$ yields a 1TB TOM storage capacity. Our default query length $\ell = 4\sqrt{N}$ represents a 2^{14} block (1GB) query for the default N. Our default milestone count $\lambda = 8$ leaks at most $I_\lambda = 3$ bits per ℓ-block query.

6.3.1 *Uniform Random Block Queries (Figures 5–7)*

The *Uniform Random Block* queries are the best suited to the SBT protocols. For each query, we choose ℓ distinct blocks uniformly at random from all N blocks. We used the same type of query to derive our analytic bandwidth cost predictions. All SBT variants outperform ORAM for large uniform random block queries, with costs as low as 5X for the Multi-SBT (Figure 6).

6.3.2 *Fixed Sequence Queries (Figures 8–13)*

For fixed sequence queries, we divide the N blocks into $s = N/\ell$ non-overlapping fixed sequences of ℓ distinct blocks each before permuting the blocks and storing them on the server. Each query consists of exactly one of these fixed sequences, simulating a file system in which each query re-

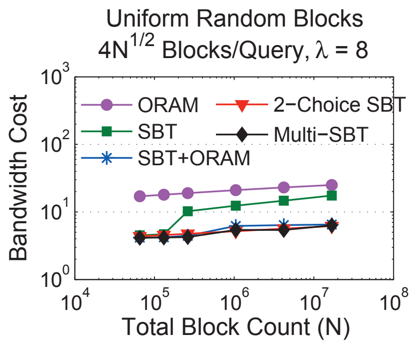

Figure 5: Uniform random block queries, varying N

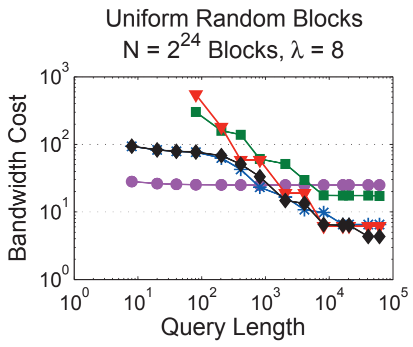

Figure 6: Uniform random block queries, varying ℓ

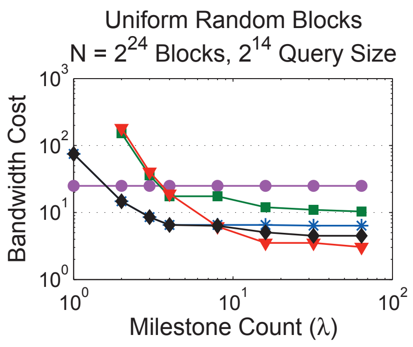

Figure 7: Uniform random block queries, varying λ

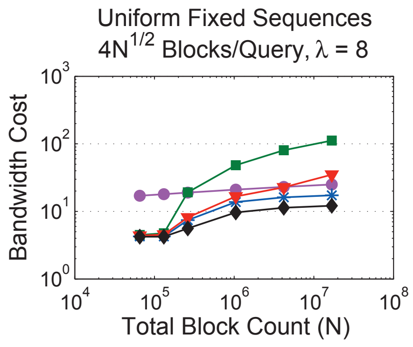

Figure 8: Uniform fixed sequence queries, varying N

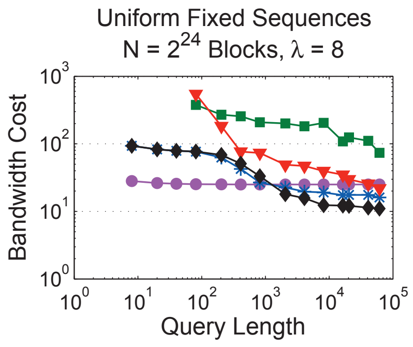

Figure 9: Uniform fixed sequence queries, varying ℓ

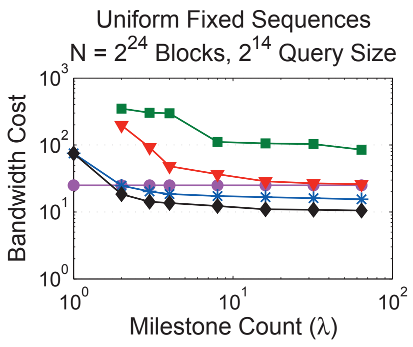

Figure 10: Uniform fixed sequence queries, varying λ

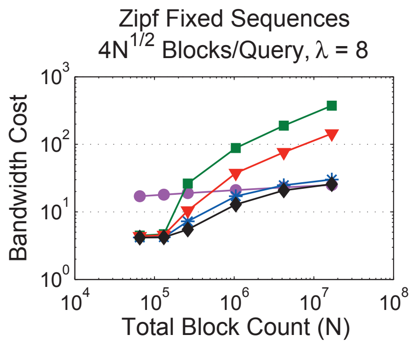

Figure 11: Zipf fixed sequence queries, varying N

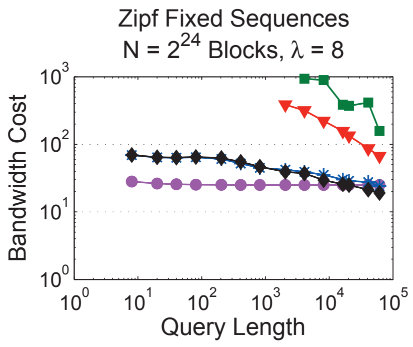

Figure 12: Zipf fixed sequence queries, varying ℓ

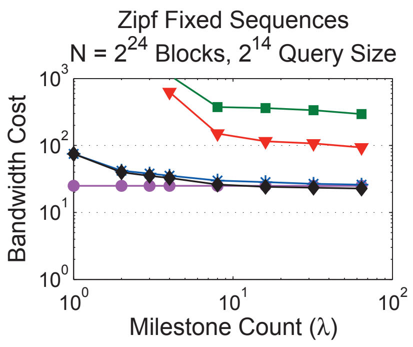

Figure 13: Zipf fixed sequence queries, varying λ

quests an entire file. *Uniform* fixed sequence experiments choose sequences uniformly at random, while *Zipf* experiments choose sequences from a power law distribution in which the ith most common sequence is chosen with probability H_s/i, where H_s is the sth harmonic number.

There are relatively few (N/ℓ) possible distinct fixed sequence queries, compared to the many (N choose ℓ) uniform random block queries. As a result, fixed sequence queries are far more likely to repeat, leading to poor SBT performance (Section 4.4). Zipf fixed sequence queries repeat frequently, so that ORAM nearly always outperforms the SBT variants (Figures 11–13). Uniform fixed sequence queries repeated less often, so several variants still outperform ORAM (Figures 8–10). We reiterate that SBT is a *special-purpose* TOM protocol. The more varied the query block distribution, the better SBT performs.

6.3.3 Other Observations

For small queries, SBTs with ORAMs converge to a worst-case cost $3\log_2 N$, while others converge to a much larger cost of n (Figures 6, 9, 12). Figures 7, 10, 13 show that we can improve performance by leaking more information (increasing λ) up to $\lambda \approx 32$ ($I_\lambda = 5$ bits). At this point padding costs become negligible, leaving the raw cost of the protocol. Since protocols with ORAM components have smaller worst-case costs, the milestones are packed more tightly, so padding effects become negligible sooner ($\lambda \approx 8$).

7. CONCLUSION

We presented a novel ORAM generalization called Tunably-Oblivious Memory (λ-TOM), which permits a privacy/efficiency tradeoff controlled via milestone count λ. We introduced the log-spacing strategy for choosing milestones to minimize padding costs, and strictly bounded the information leaked by each λ-TOM query. We also developed the special-purpose *Staggered-Bin TOM* protocol, and several read-only variants, including the Multi-SBT. We showed analytically and empirically that the Multi-SBT is highly efficient for large queries that are not cache-friendly, achieving bandwidth costs as low as 6X compared to the 22X-29X costs of the best existing ORAM protocols, while leaking at most 3 bits per query. We believe that the TOM model can be used in future work to build other highly secure special-purpose protocols, like SBT, that outperform current ORAM techniques on a variety of workloads.

8. ACKNOWLEDGEMENTS

This work was completed while the first author was a student at UC Riverside. The work was supported in part by the National Physical Science Consortium Graduate Fellowship and by grant N00014-07-C-0311 from the Office of Naval Research.

9. REFERENCES

[1] D. Boneh, D. Mazieres, and R. A. Popa. Remote oblivious storage: Making oblivious RAM practical. Manuscript, `http://dspace.mit.edu/bitstream/handle/1721.1/62006/MIT-CSAIL-TR-2011-018.pdf`, 2011.

[2] J. A. Cain, P. Sanders, and N. Wormald. The random graph threshold for k-orientiability and a fast algorithm for optimal multiple-choice allocation. In *Proc. SODA*, pages 469–476. Society for Industrial and Applied Mathematics, 2007.

[3] J. Dautrich. *Achieving Practical Access Pattern Privacy in Data Outsourcing*. PhD thesis, University of California, Riverside, 2014.

[4] J. Dautrich and C. Ravishankar. Compromising privacy in precise query protocols. In *Proc. EDBT*, 2013.

[5] J. Dautrich and C. Ravishankar. Combining oram with pir to minimize bandwidth costs. In *CODASPY*, 2015.

[6] J. Dautrich, E. Stefanov, and E. Shi. Burst ORAM: Minimizing ORAM response times for bursty access patterns. In *USENIX Security*, 2014.

[7] S. De Capitani di Vimercati, S. Foresti, S. Paraboschi, G. Pelosi, and P. Samarati. Efficient and private access to outsourced data. In *Proc. ICDCS*, 2011.

[8] C. Gentry, K. Goldman, S. Halevi, C. Julta, M. Raykova, and D. Wichs. Optimizing ORAM and using it efficiently for secure computation. In *PETS*, 2013.

[9] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[10] M. T. Goodrich, M. Mitzenmacher, O. Ohrimenko, and R. Tamassia. Privacy-preserving group data access via stateless oblivious RAM simulation. In *Proc. SODA*, pages 157–167. SIAM, 2012.

[11] M. Islam, M. Kuzu, and M. Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*, 2012.

[12] N. L. Johnson and S. Kotz. *Urn models and their application: an approach to modern discrete probability theory*. Wiley New York, 1977.

[13] E. Kushilevitz, S. Lu, and R. Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme. In *Proc. SODA*, pages 143–156. SIAM, 2012.

[14] J. R. Lorch, B. Parno, J. W. Mickens, M. Raykova, and J. Schiffman. Shroud: Ensuring private access to large-scale data in the data center. *FAST*, pages 199–213, 2013.

[15] M. Maas, E. Love, E. Stefanov, M. Tiwari, E. Shi, K. Asanovic, J. Kubiatowicz, and D. Song. PHANTOM: Practical oblivious computation in a secure processor. In *ACM CCS*, 2013.

[16] T. Mayberry, E.-O. Blass, and A. H. Chan. Efficient private file retrieval by combining ORAM and PIR. In *NDSS*, 2014.

[17] M. Mitzenmacher. Some open questions related to cuckoo hashing. In *Algorithms-ESA 2009*, pages 1–10. Springer, 2009.

[18] Y. Nakano, C. Cid, S. Kiyomoto, and Y. Miyake. Memory access pattern protection for resource-constrained devices. In *Smart Card Research and Advanced Applications*, pages 188–202. Springer, 2013.

[19] M. Raab and A. Steger. Balls into bins – a simple and tight analysis. In *Randomization and Approximation Techniques in Computer Science*, pages 159–170. Springer, 1998.

[20] R. Sion. On the computational practicality of private information retrieval. In *Proc. NDSS*, 2007.

[21] E. Stefanov and E. Shi. Multi-Cloud Oblivious Storage. In *CCS*, 2013.

[22] E. Stefanov and E. Shi. ObliviStore: High performance oblivious cloud storage. In *IEEE Symposium on Security and Privacy*, 2013.

[23] E. Stefanov, E. Shi, and D. Song. Towards practical oblivious RAM. NDSS, 2012.

[24] E. Stefanov, M. van Dijk, E. Shi, C. Fletcher, L. Ren, X. Yu, and S. Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In *ACM CCS*, 2013.

[25] P. Williams, R. Sion, and A. Tomescu. PrivateFS: A parallel oblivious file system. In *CCS*, 2012.

[26] X. Yu, C. W. Fletcher, L. Ren, M. v. Dijk, and S. Devadas. Generalized external interaction with tamper-resistant hardware with bounded information leakage. In *Proc. ACM CCSW*, pages 23–34. ACM, 2013.

APPENDIX

A. PERFORMANCE ANALYSES / PROOFS

Here we prove Theorem 3 and argue for Conjecture 1. The proof of Theorem 5 is deferred to the extended version of the paper [3]. In each case, our goal is to upper-bound the maximum fetch queue length H — the maximum number of blocks that must be fetched from any one bin by the

SBT component to satisfy a query. Equivalently, H is the maximum number of SBT *passes* needed to satisfy a query.

For simplicity, we assume the SBT is at the start of a pass, so we are given n bins filled with $1, 2, \ldots, n$ blocks each.[1] Each query requests ℓ distinct blocks chosen uniformly at random, without replacement, from the set of all N blocks. We call such queries *uniform random block queries*. Every block has a unique *location*. Of the $N = n(n+3)/2$ blocks, n are located somewhere in the local cache, and the remaining $n(n+1)/2$ are located somewhere in one of the n bins. Requests for cached blocks are satisfied instantly.

We assign each queried block a unique index i between 1 and ℓ. Let $\Pr_B(i,j)$ be the maximum probability that block i will be in bin j, given any possible arrangement of the remaining queried blocks. The maximum $\Pr_B(i,j)$ occurs when j is the n-block bin, and the other $\ell - 1$ blocks are located in bins other than j. In this case, i has $N - \ell + 1$ possible locations, n of which are in bin j, giving:

$$\Pr_B(i,j) \leq \frac{n}{N-\ell+1} \leq \frac{n}{N-\ell}. \tag{10}$$

A great deal of work has been done on the closely-related *balls and urns* problem (e.g. [12, 19]), in which balls are thrown independently into one of several urns chosen uniformly at random (with replacement).[2] There are well-known bounds on the resulting maximum urn occupancy. To use these bounds, we first reduce our blocks and bins problem to a larger balls and urns problem.

A.1 Problem Transformation

Consider the balls and urns problem with 3ℓ balls, where 3 distinct balls are given each index $1 \leq i \leq \ell$. We throw these 3ℓ balls independently and uniformly at random into n urns. Let $\Pr_U(i,j)$ be the probability that at least one ball with label i will appear in urn j, which is given by:

$$\Pr_U(i,j) = 1 - \left(\frac{n-1}{n}\right)^3 = \frac{3n^2 - 3n + 1}{n^3} \tag{11}$$

Intuitively, if $\Pr_B(i,j) \leq \Pr_U(i,j)$, then a ball labeled i is at least as likely to be placed in urn j as block i is to be located in bin j, and so the number of blocks found in bin j should be no larger than the number of balls in urn j. If we can show that $\Pr_B(i,j) \leq \Pr_U(i,j)$ for every i, j, then any upper-bound on the maximum urn occupancy in the balls and bins problem should hold for the maximum queue length H in the blocks and bins problem.

Lemma 2. *$\Pr_B(i,j) \leq \Pr_U(i,j)$ holds for all $\ell \leq n^2/6$.*

Proof. Substituting $N = n(n+3)/2$, we get:

$$\Pr_B(i,j) \leq \frac{n}{N-\ell} = \frac{2}{n+3-2\ell/n}.$$

Thus we have

$$\Pr_B(i,j) \leq \Pr_U(i,j) \iff \frac{2}{n+3-2\ell/n} \leq \frac{3n^2-3n+1}{n^3}$$

$$\iff 2n^3 \leq 3n^3 + 6n^2 - 8n + 3 - \ell(6n - 6 + 2/n)$$

$$\iff \ell\,(6n - 6 + 2/n) \leq n^3 + 6n^2 - 4n + 3$$

$$\Longleftarrow \ell \leq n^2/6$$

□

[1]We can force the SBT to the start of a pass for each query, which increases H by at most 1 and thus does not affect our asymptotic analysis.

[2]Such problems are commonly referred to as *balls and bins* problems, but since we use *bin* in our SBT construction, we use *urn* here for clarity.

A.2 SBT Analysis

We are now ready to prove Theorem 3.

Theorem 3. *Let $\ell \geq n$ (large queries). With high probability for the SBT with uniform random block queries, we have:*

$$H \in O\left(\frac{\ell}{n}\frac{\log n}{\log\log n}\right).$$

Proof. It is well known (e.g. [19]) that if we throw n balls independently and uniformly at random into n urns, we get a maximum urn occupancy in $O(\log n/\log\log n)$ with high probability. Thus, if we throw $m \geq n$ balls, we get a maximum height $O((m\log n)/n\log\log n)$. By Lemma 2, when $\ell \leq n^2/6$, an upper-bound on the maximum urn occupancy for $m = 3\ell$ balls and n urns applies to H, giving:

$$H \in O\left(\frac{3\ell\log n}{n\log\log n}\right) \subseteq O\left(\frac{\ell}{n}\frac{\log n}{\log\log n}\right), \text{ for } \frac{n}{3} \leq \ell \leq \frac{n^2}{6}$$

Further, since $H \leq n$, for $\ell > n^2/6$ we also have, trivially, that: $H \in O\left(\frac{\ell}{n}\right) \subseteq O\left(\frac{\ell}{n}\frac{\log n}{\log\log n}\right)$. □

A.3 2-Choice SBT Analysis

Conjecture 1. *With high probability for the 2-Choice SBT with uniform random block queries, we have:*

$$H \in O(\ell/n + 1).$$

Authors in [2] analyze the *Selfless Algorithm* for allocating m balls to n urns, where each ball may be placed in either of two urns chosen uniformly at random. They show, analytically, that the Selfless Algorithm yields a maximum final urn occupancy $U' \in O(\lceil m/n \rceil) \subseteq O(m/n+1)$ with high probability. They also show empirically that the simpler *Random Round Robin* algorithm, which we use for the 2-Choice SBT, has nearly equivalent performance.

As we did for SBT, we can think of the 2-Choice SBT's blocks and bins problem as a balls and urns problem where we throw $m = 3\ell$ balls into n urns. However, since each block now belongs to two bins, and can thus be added to either of two fetch queues, the corresponding ball may be placed in either of two urns, but need not be placed in both. Though Lemma 2 no longer holds, we appeal to the intuition that a bound on the maximum urn occupancy H' is likely to hold for the maximum fetch queue height H as well.

We therefore contend that $H \approx U' \in O(m/n+1) \subseteq O(\ell/n+1)$. Clearly, this argument is far from a proof, both because *Random Round Robin* has not been fully analyzed, and because of the different models used for the two-choice blocks and bins and two-choice balls and urns problems. However, we observe empirically that 2-Choice SBT does in fact appear to follow $H \in O(\ell/n+1)$, as evidenced by Figure 4 in Section 6.