The Minimum Bucket Capacity for Reliable Path ORAM

Yijia Chang
Department of Computing
The Hong Kong Polytechnic University
yijia.chang.crypto@gmail.com

Man Ho Au
Department of Computing
The Hong Kong Polytechnic University
man-ho-allen.au@polyu.edu.hk

Abstract

Path ORAM hides memory access patterns by storing N data blocks in a binary tree whose buckets have capacity Z, using a client-side stash for blocks that cannot be placed in the tree. Smaller buckets reduce storage and communication costs but increase the risk of stash overflow. The revised Path ORAM analysis in 2014 established an exponentially decreasing stash overflow probability for $Z \geq 5$. Experiments suggest small stashes at $Z = 4$ and substantially larger ones at $Z = 3$. However, it remained unknown whether $Z = 4$ admits a uniform exponential stash bound or whether such a bound is impossible at $Z = 3$.

We establish a sharp threshold at $Z = 4$ for binary Path ORAM with tree depth $L = \lceil \log_2 N \rceil$. For $Z = 4$, we prove that, uniformly over all database sizes and fixed access sequences, the stash size S after the final refill satisfies

$$\Pr[S > R] \leq \frac{23}{28} \left(\frac{5}{7}\right)^R$$

for every integer $R \geq 0$. For $Z = 3$, we exhibit a family of instances on which a single sequential scan of all N blocks produces a stash of size $\Omega(N)$ with probability bounded away from zero. Thus, four is the minimum bucket capacity admitting a uniform exponential stash bound: a logarithmic stash suffices for polynomial-length executions with inverse-polynomial failure probability at $Z = 4$, whereas at $Z = 3$ no sublinear stash can make the overflow probability vanish uniformly.

Our proofs connect the actual Path ORAM process with dependent node loads to an auxiliary model with independent geometric node loads. This independence enables a recursive analysis of overflow that avoids the union bound over rooted subtrees used in the previous $Z = 5$ proof. The main challenge is to transfer bounds from the auxiliary model to the actual process, which requires different techniques for the upper and lower bounds. For $Z = 4$, we develop a two-stage randomization argument that bounds the actual process's exponential overflow moment by that of the auxiliary model. For $Z = 3$, we develop a local continuous-time approximation with controlled errors that transfers an overflow surplus from the auxiliary model to the actual process, yielding the linear stash lower bound. Together, these arguments establish the sharp threshold at bucket capacity four.

Contents

1 Introduction **1**

1.1 The Bucket Capacity Selection Problem in Path ORAM 1

1.2 Our Results 2

1.3 Technical Overview 3

2 Path ORAM Description and Problem Formulation **6**

2.1 Path ORAM: Basic Setting and Operations 6

2.2 Formal Definition of the Optimal Bucket Capacity Problem 7

3 Main Theorems and Proof Sketch **8**

3.1 Main Theorems 8

3.2 Proof Sketch 9

3.2.1 Representing the Stash by Target Loads 9

3.2.2 Capacity Four: Independent Loads and a Uniform Moment Bound 10

3.2.3 Capacity Three: Local Surplus and Global Overflow 11

3.3 Corollaries 13

4 Uniform Exponential Stash Tail Bound for $Z = 4$ **13**

4.1 Proof Organization 14

4.2 Formal Proof of Theorem 1.2 14

4.3 Proof of Proposition 4.3 15

4.3.1 Exact Overflow Representation of the Stash 16

4.3.2 Comparison with a Full Scan 18

4.3.3 Conditional Poisson Comparison 22

4.3.4 Independent Geometric Loads and Completion 25

4.4 Proof of Proposition 4.4 30

5 Failure of a Uniform Exponential Stash Tail Bound for $Z = 3$ **36**

5.1 Proof Organization 36

5.2 Formal Proof of Theorem 1.3 37

5.2.1 Target Loads and Upward Packing 37

5.2.2 Aggregating Overflow from Actual Scan Regions 38

5.3 Proof of Proposition 5.3 39

5.3.1 A Geometric Surplus and the Losses in Transferring It 39

5.3.2 Realizing the Geometric Field by Continuous Arrivals 41

5.3.3 Retaining the Surplus in a Finite Window 42

5.3.4 Transferring the Finite Window to Discrete Scan Rounds 43

5.4 Exact Finite Verification of the Local Surplus 44

6 Related Work **45**

6.1 The Ordinary Process and Its Capacity Question 46
6.2 Different Eviction Rules 47
6.3 Inherited Representations and the New Comparisons 47

7 Conclusion and Discussions **48**
7.1 Further Questions 48

A General-Capacity Packing Facts **49**
A.1 From Placement Opportunities to Tree Overflow 49
A.2 Why This Packing Has the Actual Stash Count 50
A.3 Independent Labels and the Exact Scan Representation 53

B Finite Certificate Specifications **54**
B.1 Capacity-Four Upper Certificate 55
B.2 Capacity-Three Lower Certificate 56

C Arithmetic for the Lower-Bound Parameters **56**

D A Diagnostic Check of the Full Finite Process **57**

E Proofs of the Stash Corollaries **58**

1 Introduction

1.1 The Bucket Capacity Selection Problem in Path ORAM

When a client outsources its private database to an untrusted server, encrypting each data block is not enough. Prior work shows that querying values in an interval and observing which records are returned can enable reconstruction of the queried attribute across the database [KKNO16]. Oblivious RAM (ORAM), developed in the foundational work of Goldreich and Ostrovsky [GO96], addresses this leakage by hiding which records a client reads or writes. Designing practical ORAM constructions is therefore an important research problem with direct implications for privacy-preserving systems.

In practical ORAM design, a foundational contribution is Path ORAM by Stefanov, van Dijk, Shi, Chan, Fletcher, Ren, Yu, and Devadas [SvDS$^+$13, SvDS$^+$18]. It stores encrypted blocks in a binary tree of buckets, each holding at most Z blocks, and handles an access by reading and rewriting a single root-to-leaf path. The client reads the blocks on the path into a private buffer called the *stash* and then returns as many blocks as possible to the tree, with the rest left in the stash. This simple design makes Path ORAM attractive for practical systems. It has been implemented in the ZeroTrace oblivious-memory library and the Oblix encrypted-search index [SGF18, MPC$^+$18].

Despite the advantages of Path ORAM in practice, it faces a risk of *stash overflow* that affects its reliability: With stash capacity R, an access may overflow if more than R blocks remain after the client returns blocks to the tree. To avoid such a reliability problem, we need to control the probability of stash overflow. Formally, let π denote an execution of Path ORAM and $S(\pi; s)$ denote the stash size after executing π over any s access requests. We express the reliability goal as an upper bound valid for every fixed sequence of s requests:

$$\mathbb{P}[S(\pi; s) > R] \leq f(R). \tag{1}$$

We usually seek an exponential bound of the form $f(R) = C\exp(-cR)$, with positive constants C, c depending only on the bucket capacity Z. The reasons are two fold. First, such a bound ensures that a stash of $O(\log(1/\delta))$ blocks suffices to keep the overflow probability below δ. In other words, a small increase in stash capacity can provide a much stronger reliability guarantee. Second, requiring the same constants for every database size, every length of access sequence, and every fixed access sequence makes this guarantee uniform across system sizes and workloads. Hence, even as the database grows or the access pattern changes, the same stash budget can still provide the same reliability guarantee, which is desired in the complicated real-world systems.

To achieve the above uniform exponential stash tail bound, a key factor is the capacity Z of each bucket in the tree. A larger Z allows more blocks to return to the tree and can leave fewer in the stash, making the bound easier to achieve. However, it also increases the communication cost of each access, since the client must read and rewrite more blocks along the path. To balance efficiency with reliability, it is natural to seek the optimal bucket capacity Z and ask

What is the minimum bucket capacity Z for which the uniform exponential bound holds?

Despite progress in both theory and experiment, the optimal value of Z remained undetermined before this work. In 2013, the initial analysis of Path ORAM proved a uniform exponential stash tail bound for $Z = 6$ [SvDS$^+$13, Theorem 1]. A revised analysis released in 2014 reduced the proven capacity to $Z = 5$ [SvDS$^+$14, Theorem 1]. Experimental results provide some hints about the minimum bucket capacity. They reported small stashes at $Z = 4$, but substantially larger ones at $Z = 3$ [SvDS$^+$18, Sections 7.1–7.2]. However, these observations neither prove the uniform

exponential bound at $Z = 4$ nor disprove it at $Z = 3$. At $Z = 4$, finite simulations cover only some instances and may miss rare overflow events, whereas the same constants C, c must cover every allowed instance. At $Z = 3$, large observed stashes cannot rule out a larger C or smaller c. Indeed, over any fixed finite range of database sizes, a sufficiently large C makes the bound hold even if every block is in the stash. Disproving the uniform bound therefore requires a family of growing instances that defeats every fixed pair C, c.

The long-standing uncertainty about the optimal bucket capacity has left a gap between reliability guarantees and resource efficiency in Path ORAM. Users relying on existing uniform reliability guarantees must choose conservative bucket capacities, incurring higher storage and path communication costs. Choosing smaller capacities, however, leaves users without sufficiently strong theoretical reliability guarantees. This gap also exposes a limitation of existing analyses. We cannot yet determine precisely whether the additional capacity is necessary for reliable operation or merely an artifact of conservative proofs. Similar difficulties arise in the subsequent extensions of Path ORAM. For example, the Path Oblivious Heap explicitly notes that its proof requires non-root buckets of capacity at least 5, whereas it recommends capacity 2 in practice. It also identifies tightening these constants as a long-standing open challenge in this line of work [Shi20, Section 3.1, Remark 4]. Determining the minimum bucket capacity would provide a rigorous basis for parameter selection. The analysis needed to establish this threshold could also reveal why existing bounds are loose and how to sharpen them.

1.2 Our Results

In this paper, we prove the following theorem, which states that $Z = 4$ is the minimum bucket capacity for reliable Path ORAM.

Theorem 1.1 (Optimal Bucket Capacity for Reliable Path ORAM, informal)**.** *Consider initially empty binary Path ORAM with bucket capacity Z. The following statement holds if and only if $Z \geq 4$: there exist constants $C \geq 1$ and $c > 0$, depending only on Z, such that*

$$\mathbb{P}[S(\pi; s) > R] \leq Ce^{-cR}$$

for every database size, every fixed access sequence of any length s, and every stash capacity $R \geq 0$. Here π denotes the random execution of the access sequence and the quantity $S(\pi; s)$ is the number of blocks remaining in the stash after the refill of the s-th access.

The next two theorems establish sufficiency at four and insufficiency at three. Increasing bucket capacity cannot increase the stash under the same access sequence and random labels, so these results imply Theorem 1.1.

Theorem 1.2 (Sufficiency of Capacity Four, informal)**.** *For initially empty binary Path ORAM with $Z = 4$,*

$$\mathbb{P}[S(\pi; s) > R] \leq \frac{23}{28}\left(\frac{5}{7}\right)^R$$

for every database size, every fixed access sequence of any length s, and every integer stash capacity $R \geq 0$.

Theorem 1.3 (Insufficiency of Capacity Three, informal)**.** *Consider initially empty binary Path ORAM with $Z = 3$. For every $C \geq 1$ and $c > 0$, there are arbitrarily large databases and stash capacities $R \geq 0$ such that*

$$\mathbb{P}[S(\pi; s) > R] > Ce^{-cR},$$

where π is the random execution of a length-s access sequence that visits each block once in address order. The capacity R can be chosen as a fixed positive fraction of the database size. The overflow probability stays above a positive constant as the database grows.

These results identify a capacity threshold for the residual stash space needed for reliability. At capacity four, the same exponential tail bound and expectation bound $\mathbb{E}[S(\pi; s)] \leq 3/8$ hold at every fixed execution endpoint, uniformly over database sizes and fixed access sequences (Corollary 3.5). At capacity three, the scan family in Theorem 1.3 has linear expected stash. Its stash also reaches linear size with probability bounded away from zero. Along this family, no sublinear stash budget $R(N) = o(N)$ can therefore make the overflow probability tend to zero as N grows.

The capacity threshold gives a concrete communication saving. At the same depth L, reading and rewriting a path transfers $2Z(L+1)$ block slots, including dummy blocks. Reducing Z from five to four lowers this count from $10(L+1)$ to $8(L+1)$, a 20% reduction. Stash budgets for a given failure probability must still account for the different constants in the two exponential bounds.

Our proofs use an auxiliary model with independent node loads to analyze stash overflow. Compared with the real Path ORAM whose block placements are dependent to each other, the independent node loads enables us to analyze the stash overflow in a recursive manner, which avoids the union-bound overcounting in the existing $Z = 5$ proof [SvDS+18, Sections 5.3–5.6]. The main challenge left is to transfer analysis results from this auxiliary model to the actual process. For $Z = 4$, we develop a two-stage randomization technique that bounds the real process's exponential overflow moment by that of the auxiliary model. Bounding this moment in the independent model then bounds the real process's overflow probability. For $Z = 3$, the independent model reveals a local overflow surplus that can accumulate into a large stash. We use a continuous-time approximation with rigorous error bounds to transfer a local overflow surplus from the auxiliary model to the real process. Section 1.3 explains these two uses of the auxiliary model.

1.3 Technical Overview

We explain why the original analysis stops at capacity five and how our two connections to the auxiliary model with independent node loads resolve the capacity question. The common starting point is a reduction from stash overflow to packing blocks upward in a tree, which is shared by our proofs and the original analysis. Capacity four requires an upper bound for this packing overflow, whereas capacity three requires a local lower bound that can be transferred back to the real process.

Problem Reduction: Packing Random Targets Upward. Each Path ORAM block has a uniformly random leaf label and may occupy only a bucket on the corresponding root-to-leaf path or the stash. An access reads the block's assigned path, assigns it a fresh independent label, and refills the old path using blocks whose current labels permit placement there. To avoid tracing every refill, the original analysis removes bucket capacity limits [SvDS+18, Lemma 5.4]. In this auxiliary process, a block's label and refill history determine the deepest bucket it reaches, called its *target*. The number of blocks targeting a node is its target load. The reduced problem starts with all blocks at their targets and packs them upward from the leaves. Each bucket keeps up to Z blocks and passes the excess to its parent. The number passed above the root is exactly the residual stash size.

For the upper bound, it suffices to analyze a *full scan*, which accesses each block once. We adapt the original distinct-address argument for the load of one fixed rooted subtree [SvDS+18, Lemma 5.6 and Remark 5.7] to a joint construction of all targets that bounds root overflow. Section 3.2.1 summarizes these reductions. Section 4.3.1 defines the packing representation, whose

general-capacity proof appears in Appendix A. Section 4.3.2 proves the full-scan bound.

Why the Original Analysis Stops at $Z = 5$. The original proof detects overflow through a set of buckets with too many targets to store. The set contains the root and every ancestor of its nodes. Its blocks cannot move outside the set except through the root, so exceeding its capacity by more than R forces the stash to exceed R. The converse also holds [SvDS$^+$18, Lemma 5.5]. The proof bounds the overload probability for each fixed set, then applies a union bound over all possible sets. At capacity five, the bound decreases fast enough with set size to offset the exponentially growing number of sets [SvDS$^+$18, Sections 5.3–5.6]. However, at capacity four, the same estimates do not decrease fast enough to offset the number of sets.

This failure does not imply that large stash sizes are common. A single target configuration may overload many sets, causing its probability to be counted repeatedly in the sum. We therefore seek to control root overflow directly, without enumerating the overloaded sets that witness the same overflow event.

Capacity Four: Control Root Overflow through Independent Loads. We draw inspiration from *parking processes* in probability and combinatorics [ACCH23, BBJ21], where cars that find no space at a node continue toward the root. Here blocks take the place of cars, and each node has four slots. This suggests a recursive analysis: combine the two child overflows with the parent's target load, retain up to four blocks, and pass any excess upward. Repeating this calculation gives the stash size. The obstacle is dependence. Different blocks determine their targets using shared refill paths, so the two child overflows may be dependent. Their individual distributions then do not determine the distribution of their sum.[1]

Our key idea is to introduce randomness that makes the node loads independent while preserving an upper bound on an *exponential moment* of root overflow. For root overflow D and fixed $u > 1$, this moment is $\mathbb{E}[u^D]$. Since $D > R$ implies $u^D > u^R$, Markov's inequality gives $\mathbb{P}[D > R] \le \mathbb{E}[u^D]/u^R$. A constant moment bound, independent of tree height, therefore gives the desired uniform exponential bound. We need the independent model's moment to be no smaller than the real one, without requiring such an ordering of their probabilities at every threshold.

Two sources of dependence require two randomizations. Even with refill paths fixed, each block chooses only one target, so placing it at one node excludes placing it at another. We let each block make a Poisson number of target choices with mean one. Poisson splitting then makes node loads independent for those fixed paths. Their means still depend on the shared random paths.

To express these means through the refill paths, we read the full scan backward. The target of the kth block in this order depends on the first k refill paths. It targets an internal node precisely when these paths have visited one child but not the other and its final leaf label lies in the unvisited child. With the paths fixed, this label condition has probability equal to that child's fraction of all leaves. The node mean is this probability times the number of such steps within the scan. Continuing the paths independently when needed to define both first visits gives a full waiting interval, whose step count times the same probability bounds the mean.

To make these mean bounds independent, we replace the unit spacings between successive refill paths by independent exponential time spacings of mean one, independent also of the paths. Averaging over paths and these spacings gives independent child first-arrival times with the same exponential rate. Their minimum is independent of their gap: the first arrival at the parent is

[1] Suppose each child overflow is zero or one with equal probability. If the counts are always equal, their sum is zero or two. If they are always different, their sum is always one.

independent of the extra wait for the other child. We use the scaled gaps as new means and apply this decomposition recursively, keeping the gaps inside each subtree mutually independent and their entire collection independent of its first-arrival time. This yields jointly independent means, with physical leaves handled separately. Averaging the conditional Poisson loads over these means gives independent geometric node loads, with probability 2^{-k-1} of load $k \geq 0$ at each node.

The essential point is to obtain this independence without losing the upper bound. Upward packing has a convexity property: adding blocks has no smaller effect on u^D when the existing loads are larger. We use this property to show that both randomizations preserve an upper bound on the moment. Independence then turns the moment of the combined child overflows into the product of their moments. This yields a recursive bound that remains constant as the tree grows. An exact finite calculation verifies the conditions that make the bound hold at every height. Section 4 proves this connection and the resulting bound for every fixed access sequence.

Capacity Three: Local Overflow Exceeds the Storage Above It. To rule out a uniform exponential bound at capacity three, we seek full scans whose stash is proportional to the database size with probability bounded away from zero. The packing representation suggests a local way to force this: make many disjoint subtrees send more blocks upward than their ancestors can store. Cut the tree at one depth into m subtrees, called regions. Above their roots are $m - 1$ buckets, holding at most $3(m - 1)$ blocks. Thus, for a fixed $\varepsilon > 0$, an expected overflow of at least $3 + \varepsilon$ per region leaves an expected stash of at least $\varepsilon m + 3$. Independence between regions is unnecessary, because expectations add regardless of dependence.

The independent geometric model suggests how to obtain this local surplus, but the upper-bound connection cannot transfer it to the real scan. The added randomness might have created overflow that the real scan lacks. We instead need a local approximation that preserves expected overflow with only a small loss. An exact finite calculation shows that a geometric-load tree with a fixed number of levels has expected overflow greater than three. Our task is to show that small regions of a large real scan exhibit nearly the same loads.

Regions deep in the tree receive rare visits from both refill paths and final leaf labels. We read the scan backward, so the paths already seen are precisely the refill opportunities for the block currently being considered. After rescaling time, the rare path visits and label visits are approximated by independent Poisson arrival streams. This connects real scan regions to a model in which we can identify the target loads. Once a path has visited one child but not the other, labels in the unvisited child target their parent until a path reaches that child. The number of such labels is geometric. We retain a fixed number of levels at the top of each region, above its physical leaves. Over unlimited time, a joint analysis shows that the loads at these nodes are independent, giving the geometric model used in the finite calculation.

To preserve its surplus in the real scan, we control two approximation errors: observing only a finite time window and replacing rare discrete visits by continuous arrivals. We cap the overflow counted from each region at a fixed constant. A rare discrepancy can then cause only a bounded loss in expectation. The finite calculation shows that even this capped overflow leaves enough margin above three to absorb both losses. Each real region therefore still sends more than three blocks upward on average.

Choosing regions of a fixed size makes their number proportional to N, so the ancestor-capacity calculation gives an expected stash proportional to N. The stash is at most N, so this expectation cannot come only from events whose probabilities vanish. It implies a fixed positive probability of exceeding a fixed positive fraction of N. Every proposed bound Ce^{-cR} tends to zero at such

thresholds, contradicting that probability lower bound. Section 5 makes this argument quantitative.

2 Path ORAM Description and Problem Formulation

Section 2.1 specifies the storage layout, initialization, and access operations. Section 2.2 defines the random execution and the uniform exponential stash tail bound. These definitions fix the process and guarantee studied in Sections 4 and 5.

2.1 Path ORAM: Basic Setting and Operations

Suppose that a client wishes to outsource a private database to an untrusted server while retaining read and write access to its contents. The database contains at most $N \geq 1$ data blocks, with logical addresses in $[N] = \{1, \ldots, N\}$. These numbers are the *logical addresses* used in read and write requests. The client encrypts each block to protect its contents. Encryption alone, however, does not hide the access pattern. For example, repeated requests to the same location may reveal that the client keeps reading the same block. Path ORAM hides which logical blocks the client accesses [SvDS+18, Section 3]. Below we describe its memory design and operations, respectively.

Memory design. The server uses a complete binary tree, with a *root* at depth zero, two children per non-leaf node, and *leaves* at depth L. The tree has depth $L = \lceil \log_2 N \rceil$, the smallest nonnegative integer such that $2^L \geq N$. For $N = 1$, it consists only of the root. We name nodes by their paths from the root, writing 0 for a left step and 1 for a right step. For example, when $L \geq 2$, node 01 is reached by moving left and then right.

In Path ORAM, data blocks can be stored in two places. The client has a private buffer called the *stash* for blocks waiting to be placed in the tree. In the server, each node in the tree contains a storage space called a *bucket*, which can hold at most Z data blocks. Each data block is stored either in the client's stash or in one bucket on a path assigned to that block. This path starts at the root and ends at a leaf. The client keeps a private record of the assigned leaf for each logical block. We call this record the *position map* and the assigned leaf the block's *leaf label*. To find a stored block, the client simply checks its stash and reads the path of this block from the position map.

Initialization. Initially, the tree and the stash contain no real data blocks, following the original initialization rule [SvDS+18, Section 3.4]. The design reserves N logical addresses, and a block is created when its address is first accessed. As the first step, the client needs to assign each logical address a path for its first access. To do so, the client chooses an initial leaf for each address and records it in the position map. Each leaf is equally likely, and the choices for different addresses are independent, indicating that different addresses may receive the same leaf label. Once this position map is created, the client can read or write the data blocks through the ordinary access procedure as described below.

Reading or writing a block. Each access follows the four steps of Path ORAM [SvDS+18, Section 3.5 and Figure 1], with block identifiers breaking ties among eligible blocks.

1. *Read the assigned path.* Look up the requested block's leaf label in the position map and save it as the *old label*. Download and decrypt the buckets on that root-to-leaf path. Move their data blocks into the stash, leaving those buckets without data blocks. The requested block is now in the stash if it has been created before. All buckets outside this path are left unchanged.

2. *Perform the requested read or write.* If this is the first access to the address, create its block in the stash with value zero. For a read, obtain the block's value to return to the client. For a write, replace that value with the supplied data.

3. *Assign a new path.* Choose a new leaf for the accessed block and update its entry in the position map. The new label determines which buckets may hold the block after this access and which path will be read on its next access. Each leaf is equally likely, independently of all earlier choices. The new and old leaves may coincide. All other labels remain unchanged.

4. *Refill the old path.* The path read in step 1 now has empty buckets, since their blocks were moved into the stash. This step refills its buckets with blocks from the stash, processing the buckets in order from the leaf to the root. For each bucket, the client first selects blocks from the stash according to the selection rule below. It then removes the selected blocks from the stash and places them in the bucket before processing the next bucket. The selection rule is as follows: a block is eligible for a bucket if its assigned path passes through that bucket. We note that for the accessed block, this condition uses the new path from step 3. If at least Z blocks are eligible, the client selects the Z with the smallest block numbers. Otherwise, it selects all eligible blocks and pads the remaining bucket slots with dummy blocks. After processing the root bucket, any remaining blocks are left in the stash.

Example of the refilling rule. Consider a tree of depth two. Suppose the accessed block's old leaf label is 01 and its new label is 00. The client reads and refills the path through the leaf 01, node 0, and the root. The accessed block can be placed in node 0 or the root, since both also lie on its new assigned path. Nonetheless, it cannot be placed in leaf 01. If both of these buckets fill with other blocks during the refill, the accessed block remains in the stash.

2.2 Formal Definition of the Optimal Bucket Capacity Problem

Let $\mathcal{P}_{Z,N}$ denote an instantiation of the Path ORAM specified in Section 2.1, with bucket capacity Z and the maximum number of blocks N.

Modelling of Path ORAM execution. To model the real-world execution of Path ORAM, we define an *instance* of $\mathcal{P}_{Z,N}$ as a pair (s, α), where $s \geq 1$ is an integer and $\alpha = (a_1, \ldots, a_s) \in [N]^s$ is a length-s address sequence. This instance is fixed in advance, independently of all random leaf labels and the internal storage state. For an instance (s, α), let $\mathcal{P}_{Z,N}(s, \alpha)$ denote the random execution of $\mathcal{P}_{Z,N}$ on this instance, whose randomness comes from the leaf labels assigned to blocks.

For an execution $\pi = \mathcal{P}_{Z,N}(s, \alpha)$ and an integer $0 \leq t \leq s$, let $S(\pi; t)$ be the number of real blocks in the stash after the refill of its t-th access (i.e., the dummy blocks used for padding are not counted). Since π is a random execution, $S(\pi; t)$ is a random variable with $S(\pi; 0) = 0$ and $0 \leq S(\pi; t) \leq N$. In practice, the stash has a finite capacity R, i.e., the stash can hold at most R blocks. For ease of analysis, we assume that $\mathcal{P}_{Z,N}(s, \alpha)$ uses an unbounded analytical stash. For a fixed instance (s, α) and stash capacity R, overflow after the final refill is the event

$$S(\mathcal{P}_{Z,N}(s, \alpha); s) > R.$$

Bounding the overflow probability. We seek an exponential bound in R with constants valid for every database size and instance.

Definition 2.1 (Uniform Exponential Stash Tail Bound)**.** For a fixed positive integer Z, the family $\{\mathcal{P}_{Z,N} : N \geq 1\}$ admits a *uniform exponential stash tail bound (UEST bound)* if there exist constants

$C \geq 1$ and $c > 0$, depending only on Z, such that for every integer $N \geq 1$, every instance (s, α) of $\mathcal{P}_{Z,N}$, and every integer $R \geq 0$,

$$\mathbb{P}[S(\mathcal{P}_{Z,N}(s,\alpha); s) > R] \leq Ce^{-cR}. \tag{2}$$

The following properties and consequences make this bound useful for controlling the overflow probability and choosing the stash capacity.

- The bound is *exponential* in the stash capacity, i.e., increasing R by one multiplies its right-hand side by $e^{-c} < 1$. An exponential bound is useful because it yields a logarithmic stash capacity for a prescribed failure probability. For any $0 < \delta < 1$, Inequality (2) gives $\mathbb{P}[S(\mathcal{P}_{Z,N}(s,\alpha); s) > R] \leq \delta$ whenever

$$R \geq \left\lceil \frac{\log(C/\delta)}{c} \right\rceil.$$

- The bound is *uniform* because the same pair C, c must cover all database bounds N, sequence lengths, address orders, and stash capacities. In particular, these constants cannot be chosen separately for each instance or observation time.

- Whenever Z admits this bound, the same budget suffices at each specified endpoint of every allowed instance, independently of its size and workload. This is a guarantee at a fixed endpoint. It also applies to $S(\mathcal{P}_{Z,N}(s,\alpha); t)$ at every specified $1 \leq t \leq s$: The first t accesses have the same law as the execution of the prefix instance $(t, (a_1, \ldots, a_t))$ on the same design.

The optimal bucket capacity problem. The primary goal of this work is to find the smallest positive integer Z for which the UEST bound holds.

3 Main Theorems and Proof Sketch

We now state the formal version of our main theorems, using the designs $\mathcal{P}_{Z,N}$ and instances (s, α) defined in Section 2. Section 3.1 states the three main theorems. Section 3.2 gives the proof dependencies and derives the main theorem from the capacity-four and capacity-three results. Section 3.3 gives their corollaries for finite executions and expected stash size.

3.1 Main Theorems

Theorem 1.1 (Optimal Bucket Capacity for Reliable Path ORAM)**.** *Fix a positive integer Z. The following statement holds if and only if $Z \geq 4$: there exist constants $C \geq 1$ and $c > 0$, depending only on Z, such that for every integer $N \geq 1$, every instance (s, α) of $\mathcal{P}_{Z,N}$, and every integer $R \geq 0$,*

$$\mathbb{P}[S(\mathcal{P}_{Z,N}(s,\alpha); s) > R] \leq Ce^{-cR}.$$

The next two theorems give the guarantees at capacities four and three.

Theorem 1.2 (Sufficiency of Capacity Four)**.** *For every integer $N \geq 1$, every instance (s, α) of $\mathcal{P}_{4,N}$, and every integer $R \geq 0$, the execution $\pi = \mathcal{P}_{4,N}(s, \alpha)$ satisfies*

$$\mathbb{P}[S(\pi; s) > R] \leq \frac{23}{28}\left(\frac{5}{7}\right)^R \leq \exp\bigl(-R\log(7/5)\bigr). \tag{3}$$

Theorem 1.3 (Insufficiency of Capacity Three)**.** *For every integer $L \geq 49$, let $N = 2^L$, let $\alpha = (1, \ldots, N)$, and let $\pi = \mathcal{P}_{3,N}(N, \alpha)$. Then*

$$\mathbb{E}[S(\pi; N)] > \frac{N}{2^{18}} + 3, \qquad \mathbb{P}[S(\pi; N) > N/2^{19}] > \frac{1}{2^{19} - 1}. \tag{4}$$

Consequently, the family $\{\mathcal{P}_{3,N} : N \geq 1\}$ does not admit a uniform exponential stash tail bound.

3.2 Proof Sketch

The proof has three parts: packing and full-scan reductions (Section 3.2.1), an upper comparison with independent loads (Section 3.2.2), and a local lower comparison with the real scan (Section 3.2.3). We state the intermediate conclusions and their dependencies, then combine them to prove Theorem 1.1. Executions start empty, with address sequences fixed independently of the random labels.

3.2.1 Representing the Stash by Target Loads

From Refills to Upward Packing. To analyze the stash without tracing every refill, we assign each block a target that ignores bucket capacities. Its target after access t is the deepest bucket shared by its current assigned path and at least one refill path since its most recent access, including the refill at that access. Let $A_v(t)$ count blocks targeting bucket v, and write $A(t) = (A_v(t))_v$. Starting with these blocks at their targets, pack them upward: each bucket retains up to Z blocks and passes its excess to its parent. For a load vector A, the resulting overflow satisfies

$$D_v^{(Z)}(A) = \left(A_v + D_{v0}^{(Z)}(A) + D_{v1}^{(Z)}(A) - Z\right)^+, \qquad D^{(Z)}(A) = D_\varepsilon^{(Z)}(A),$$

where $x^+ = \max\{x, 0\}$ and missing children contribute zero.

The overflow above the root is exactly the actual stash size:

$$S(\pi; t) = D^{(Z)}(A(t)). \tag{5}$$

Every block is stored at an ancestor of its target, and all buckets from its target up to, but excluding, its actual location are full. These refill invariants and conservation of blocks give the same overflow recurrence for the actual placement and upward packing (Lemma 5.1). The recurrence also shows that deleting targets or increasing Z cannot increase overflow. Moving a target to an ancestor cannot decrease overflow, because it removes storage opportunities below that ancestor (Lemma 4.8).

Reducing the Upper Bound to a Full Scan. Fix $\pi = \mathcal{P}_{4,N}(s, \alpha)$, put $L = \lceil \log_2 N \rceil$, and let $\widehat{N} = 2^L$. Order the blocks present after the execution by decreasing last access time. The kth block has at least k refill opportunities since its last access. Retaining only its k most recent opportunities can move its target only upward. Adding blocks to reach $\widehat{N}$ can also only increase overflow. Any additional path labels and labels for added blocks are sampled uniformly and independently of one another and of the original execution. The resulting target field A^* has the joint distribution of a full scan of $\widehat{N}$ distinct blocks on the same tree. This gives a coupling under which

$$S(\pi; s) \leq D^{(4)}(A^*). \tag{6}$$

The old path labels and final block labels of the original execution are jointly independent and uniform: they are a deterministic permutation of the labels sampled during initialization and

accesses (Lemma 4.9). Together with the additional samples, they provide the independent uniform labels required by a full scan. In reverse scan order, write the path and final block labels as X_k and Y_k, respectively. Block k targets the deepest node among $\{\mathrm{lcp}(Y_k, X_j) : 1 \le j \le k\}$, where lcp denotes the longest common prefix. Lemma 4.12 proves the coupling.

3.2.2 Capacity Four: Independent Loads and a Uniform Moment Bound

We seek a uniform bound on $\mathbb{E}[u^{S(\pi;s)}]$ for a fixed $u > 1$. The full scan's target loads remain dependent. Two randomization stages remove this dependence while upper-bounding the moment.

First Randomization: Independent Loads Conditional on Paths. Fix $1 < u < 2$ and write $F(A) = u^{D^{(4)}(A)}$. The function F is increasing and directionally convex: adding load has no smaller effect when the existing loads are larger (Lemma 4.13). Fix the refill paths $X = (X_k)_k$. Each block chooses one target. Letting it instead make an independent Poisson(1) number of independent target choices with the same probabilities cannot decrease the expected cost, by directional convexity and Jensen's inequality. Poisson splitting makes the node loads independent, with means $\lambda_v(X) = \mathbb{E}[A_v^* \mid X]$. For independent $P_v \sim \mathrm{Poisson}(\lambda_v)$, define $\Psi(\lambda) = \mathbb{E}[F(P)]$. Lemma 4.14 gives

$$\mathbb{E}[F(A^*) \mid X] \le \Psi(\lambda(X)). \tag{7}$$

This independence is conditional on X, which still determines all node means.

Second Randomization: Independent Means. We next remove dependence among the means by randomizing the waiting intervals used to bound them. For an internal node v, let Δ_v count the reverse refill steps between its children's first visits, and let p_v be either child's fraction of all leaves. Continue the path sequence independently if needed to define both visits. The target rule gives $\lambda_v(X) \le p_v \Delta_v$. Replacing unit spacings by independent mean-one exponential spacings, independent also of X, preserves the conditional expected interval length. Averaging over paths and spacings gives independent child first-arrival times with the same exponential rate p_v.

The minimum of these two times is independent of their gap, and scaling the gap by p_v gives an Exp(1) variable. Applying this decomposition recursively makes all scaled gaps jointly independent. The induction maintains that the entire gap vector in each subtree is independent of its first-arrival time (Lemmas 4.16 and 4.17). Jensen's inequality, applied one clock at a time, ensures that the randomization cannot decrease the expected cost Ψ (Lemma 4.15). Each physical-leaf mean is at most one. Raising it to one and then applying Jensen with a fresh independent Exp(1) mean preserves the upper bound. The resulting vector $\Lambda = (\Lambda_v)_v$ has independent Exp(1) entries.

Conditional on Λ, take independent Poisson(Λ_v) loads. Averaging over Λ gives independent geometric loads with $\mathbb{P}[G_v = k] = 2^{-k-1}$ (Lemma 4.18). Let Q_h be root overflow in this capacity-four geometric-load tree with h bucket levels, with $Q_0 = 0$. Section 4.3.4 proves

$$\mathbb{E}_X[\Psi(\lambda(X))] \le \mathbb{E}[\Psi(\Lambda)] = \mathbb{E}[u^{Q_{L+1}}]. \tag{8}$$

Combining (6), (7), and (8) yields the global comparison (Proposition 4.3):

$$\mathbb{E}[u^{S(\pi;s)}] \le \mathbb{E}[u^{Q_{L+1}}], \qquad 1 < u < 2. \tag{9}$$

We bound exponential moments without ordering tail probabilities at every threshold.

From a Finite Verification to Every Height. Independence gives $Q_{h+1} \stackrel{\mathrm{d}}{=} (G+Q_h^{(0)}+Q_h^{(1)}-4)^+$, where G has the geometric law above and the two child overflows are independent copies of Q_h, independent also of G. Fix $u = 7/5$, put $M_h = \mathbb{E}[u^{Q_h}]$, and let $a = u^{-4}\mathbb{E}[u^G]$. Lemma 4.19 gives

$$M_0 = 1, \qquad M_{h+1} = aM_h^2 + B_h, \tag{10}$$

where $B_h \geq 0$ accounts for replacing negative overflow by zero. The same lemma shows that M_h is nondecreasing and B_h is nonincreasing. Lemma 4.20 uses these facts to reduce the all-height bound to three conditions at one height H:

$$M_H \leq r, \qquad B_H \leq b, \qquad ar^2 + b \leq r.$$

Monotonicity bounds all earlier heights by r. For later heights, $B_h \leq b$ makes $M_h \leq r$ imply $M_{h+1} \leq ar^2 + b \leq r$. Lemma 4.21 verifies these conditions by exact integer arithmetic at $H = 12$ with $r = 23/20$. Hence $M_h \leq 23/20$ for every h (Proposition 4.4). Together with (9), this gives

$$\mathbb{E}[(7/5)^{S(\pi;s)}] \leq \frac{23}{20}, \qquad \mathbb{P}[S(\pi;s) > R] \leq \frac{23}{28}\left(\frac{5}{7}\right)^R,$$

where the second inequality uses Markov at the integer threshold $R+1$. This proves Theorem 1.2. The case $N = 1$ is immediate because the stash is empty.

3.2.3 Capacity Three: Local Surplus and Global Overflow

The Local Surplus Needed for a Large Stash. For capacity three, we need a lower bound in the actual scan. The upper-bound moment comparison cannot supply it, because the added randomness may have created overflow absent from the real process. Instead, cut the tree at depth d into $m = 2^d$ disjoint subtrees, called regions. Their $m-1$ ancestor buckets hold only $3(m-1)$ blocks. It therefore suffices to make each region's expected overflow exceed three by a fixed positive amount and choose m proportional to the database size. Any excess beyond ancestor storage must remain in the stash.

Fix the full scan $\pi = \mathcal{P}_{3,N}(N, (1, \ldots, N))$ with $N = 2^L$. In a region rooted at v, retain targets from the first n blocks in reverse scan order only when they lie in its first H bucket levels, where $1 \leq H \leq L - d$ and $1 \leq n \leq N$. Write A_v^{disc} for these retained loads, and set

$$F_v = \min\{K, D^{(3)}(A_v^{\mathrm{disc}})\}, \qquad K \geq 1.$$

Deleting targets and capping overflow can only reduce it, so F_v is a lower bound on the region's actual overflow. The cap also ensures that an approximation failing with probability η loses at most $K\eta$ in expected overflow.

Transferring Geometric Surplus to an Actual Region. A path or final label enters a depth-d region with probability $q = 2^{-d}$ per round. Giving each reverse round duration q makes the first n rounds span time $T = nq$. We approximate the rare visits by two independent rate-one Poisson arrival streams, one for paths and one for labels. Each arrival has an independent uniform H-bit mark specifying its branch below the retained levels. A label targets its deepest match with a preceding path. Only targets in the first H levels are retained, and labels with no preceding path contribute none. Conditional on the path stream, labels targeting distinct nodes occupy disjoint regions of time and marks, giving independent Poisson counts. Over unlimited time, their means

are the scaled child-arrival gaps from Lemma 4.16. Averaging over these independent exponential means gives independent geometric loads (Lemma 5.6).

Two errors separate this model from the finite real scan. The probability that retained loads change after time T is at most $2^H e^{-T/2^H}$ (Lemma 5.7). A Bernoulli–Poisson coupling gives discrete-approximation error at most $3Tq$, including mismatches in the order of a path and a label in the same round (Lemma 5.8). The cap K converts these probabilities into expected-overflow losses. If $Q_H^{(3)}$ denotes overflow in the H-level geometric-load tree with capacity three, Lemma 5.4 gives

$$\mathbb{E}[F_v] \geq \mathbb{E}[\min\{K, Q_H^{(3)}\}] - K2^H e^{-T/2^H} - 3KTq. \tag{11}$$

This is the local connection needed for a lower bound in the real scan.

The finite certificate in Lemma 5.5 gives $\mathbb{E}[\min\{32, Q_{14}^{(3)}\}] > 11/2$. For $L \geq 49$, choose $H = 14$, $K = 32$, $d = L - 19$, and $n = N$. Then $T = 2^{19}$, the retained levels lie above the physical leaves, and the two losses in (11) sum to less than $1/2$. Thus every region satisfies

$$\mathbb{E}[F_v] > 5 \qquad \text{(Proposition 5.3)}.$$

Section 5.3 verifies these parameter choices.

Aggregating the Surplus. With $m = 2^d = N/2^{19}$ regions, subtracting their ancestors' total capacity gives the pathwise inequality

$$S(\pi; N) \geq \sum_{|v|=d} F_v - 3(m-1) \qquad \text{(Lemma 5.2)}.$$

Taking expectations yields

$$\mathbb{E}[S(\pi; N)] > 2m + 3 = \frac{N}{2^{18}} + 3.$$

This step uses linearity of expectation and requires no independence between regions. To obtain a probability lower bound, use $0 \leq S(\pi; N) \leq N$:

$$\mathbb{E}[S(\pi; N)] \leq m + (N - m)\mathbb{P}[S(\pi; N) > m].$$

Combining these inequalities gives

$$\mathbb{P}[S(\pi; N) > N/2^{19}] > \frac{m+3}{N-m} > \frac{1}{2^{19}-1},$$

proving Theorem 1.3.

Completing the Capacity Threshold.

Proof of Theorem 1.1. Increasing the bucket capacity cannot increase the stash under the same address sequence and random labels (Lemma 5.1). The capacity-four bound therefore holds for every $Z \geq 4$, with $C = 1$ and $c = \log(7/5)$. For $Z = 3$, a uniform exponential bound at $R = N/2^{19}$ would give

$$\mathbb{P}[S(\pi; N) > N/2^{19}] \leq Ce^{-cN/2^{19}} \longrightarrow 0 \qquad (N = 2^L \longrightarrow \infty),$$

contradicting the constant lower bound above. For $Z < 3$, the same scans leave at least as many blocks in the stash, so they also rule out a uniform exponential bound. □

3.3 Corollaries

As the extensions of the main theorems, the next two corollaries quantify the stash space required for finite executions and in expectation. Their proofs appear in Appendix E.

Stash Budgets for a Finite Execution. In practice, the stash must suffice after every access in an execution. The capacity-four bound gives a concrete budget once the number of accesses and the acceptable failure probability are specified.

Corollary 3.4 (Stash Budget for a Finite Execution)**.** *Fix an instance (s, α) of $\mathcal{P}_{4,N}$ and let $\pi = \mathcal{P}_{4,N}(s, \alpha)$. For every integer $R \geq 0$,*

$$\mathbb{P}\left[\max_{1 \leq t \leq s} S(\pi; t) > R\right] \leq \min\left\{1, \frac{23s}{28}(5/7)^R\right\}. \tag{12}$$

For any target failure probability $0 < \delta < 1$, choosing

$$R \geq \max\left\{0, \left\lceil \frac{\log(23s/(28\delta))}{\log(7/5)} \right\rceil\right\} \tag{13}$$

keeps the probability of overflow after any of these s refills at most δ.

For fixed s and δ, this budget does not depend on the database size. It grows only logarithmically with the execution length and with $1/\delta$. These statements concern residual stash space. Reading a path requires additional temporary working memory. No fixed finite budget is asserted to prevent overflow throughout an infinite execution.

Expected Stash Size. The capacity-four proof also bounds the expected number of blocks left in the stash after an access.

Corollary 3.5 (Expected Stash Size)**.** *For every instance (s, α) of $\mathcal{P}_{4,N}$, let $\pi = \mathcal{P}_{4,N}(s, \alpha)$. Then*

$$\mathbb{E}[S(\pi; s)] \leq \frac{3}{8}.$$

This expectation bound concerns the stash after refill and holds uniformly over all database sizes and fixed access sequences in the model.

4 Uniform Exponential Stash Tail Bound for $Z = 4$

We consider the binary Path ORAM design $\mathcal{P}_{4,N}$ of Section 2.1, with depth $L = \lceil \log_2 N \rceil$. An instance (s, α) specifies a positive number of accesses and a sequence $\alpha \in [N]^s$ fixed independently of all random leaf labels. Write $\pi = \mathcal{P}_{4,N}(s, \alpha)$ for the complete random execution from the empty tree and stash. All initial and newly assigned labels are independent and uniform. We use an unbounded analytical stash and measure $S(\pi; s)$ after the final refill, as in Section 2.2. Our goal is to prove that capacity four admits a uniform exponential stash tail bound.

Theorem 1.2 (Sufficiency of Capacity Four, Restated)**.** *For every integer $N \geq 1$, every instance (s, α) of $\mathcal{P}_{4,N}$, and every integer $R \geq 0$, the execution $\pi = \mathcal{P}_{4,N}(s, \alpha)$ satisfies*

$$\mathbb{P}[S(\pi; s) > R] \leq \frac{23}{28}\left(\frac{5}{7}\right)^R \leq \exp(-R\log(7/5)). \tag{3}$$

By Theorem 1.2, the family $\{\mathcal{P}_{4,N} : N \geq 1\}$ satisfies Definition 2.1 with $C = 1$ and $c = \log(7/5)$.

4.1 Proof Organization

Section 4.2 reduces the theorem to two propositions. The first bounds the actual stash moment by the root-overflow moment of an independent geometric-load tree. The second bounds that moment uniformly over all tree heights. Markov's inequality then gives the stated uniform exponential bound.

Section 4.3 proves the first proposition. It uses the packing identity proved in Appendix A, constructs a full scan that bounds the actual stash, and introduces independent loads in two stages. The first stage randomizes each block's number of target choices. The second randomizes the waiting intervals that bound the node means. The examples accompanying these constructions explain their effect before the corresponding inequalities are proved.

Section 4.4 proves the second proposition. A moment recurrence and its monotonicity properties reduce the all-height bound to three conditions at one finite height. An exact calculation verifies these conditions at height twelve, completing the proof.

4.2 Formal Proof of Theorem 1.2

Our first observation is that the following exponential moment bound is sufficient for Theorem 1.2.

Proposition 4.1 (Uniform Stash Moment)**.** *For every integer $N \geq 1$ and every instance (s, α) of $\mathcal{P}_{4,N}$, let $\pi = \mathcal{P}_{4,N}(s, \alpha)$. Then*

$$\mathbb{E}\big[(7/5)^{S(\pi;s)}\big] \leq \frac{23}{20}. \tag{14}$$

Proof of Theorem 1.2. For an integer $R \geq 0$, the event $S(\pi; s) > R$ implies $(7/5)^{S(\pi;s)} \geq (7/5)^{R+1}$. Markov's inequality and Proposition 4.1 give

$$\mathbb{P}[S(\pi; s) > R] \leq \frac{\mathbb{E}[(7/5)^{S(\pi;s)}]}{(7/5)^{R+1}} \leq \frac{23}{20}(7/5)^{-(R+1)} = \frac{23}{28}\left(\frac{5}{7}\right)^{R}.$$

Since $23/28 < 1$, the final inequality in (3) follows as well. □

It remains to prove Proposition 4.1. We introduce an auxiliary tree with independent geometric node loads. Independence lets us analyze its two child subtrees separately and combine their overflow moments recursively.[2] We will prove that the actual stash moment is bounded above by the exponential moment of this tree's root overflow (Proposition 4.3). A uniform upper bound for the auxiliary moment then gives the same upper bound for the actual stash moment. The geometric law is derived in the comparison construction and is not an additional assumption on the actual Path ORAM loads. We now define this auxiliary tree. Here *height* counts bucket levels, whereas *depth* counts edges from the root to a leaf. Thus a nonempty tree of depth L has height $h = L + 1$. For example, a single-node tree has depth zero and height one.

Definition 4.2 (Independent Geometric-Load Tree)**.** A complete binary tree of height $h \geq 1$ (every node before the last level has two children) is called an *independent geometric-load tree* if its node loads are mutually independent and each has the distribution of a nonnegative geometric random variable G satisfying

$$\mathbb{P}[G = k] = 2^{-k-1} \quad (k \geq 0). \tag{15}$$

[2] Recursive exponential-moment bounds for upward packing also appear in BULKOR's analysis of blocks assigned to random leaves [LLG24, Section 4.3 and Appendix B]. Here the comparison handles the dependent loads produced by access histories.

For an independent geometric-load tree of height h, let Q_h denote the random number of blocks passed above its root after upward packing with capacity four. Set $Q_0 = 0$ for the empty tree. Consider an independent geometric-load tree of height $h+1$. Let $Q_h^{(0)}$ and $Q_h^{(1)}$ denote the overflow counts of its left and right child subtrees, respectively. The two overflow counts are independent and have the same distribution as Q_h. Then, for $h \geq 0$,

$$Q_{h+1} \stackrel{\mathrm{d}}{=} \left(G + Q_h^{(0)} + Q_h^{(1)} - 4\right)^+, \qquad x^+ = \max\{x, 0\}, \tag{16}$$

where G, $Q_h^{(0)}$, and $Q_h^{(1)}$ are mutually independent.

Based on the recursion above, we state the following two propositions, whose proofs are deferred to Sections 4.3 and 4.4, respectively. Proposition 4.3 bounds the real stash's exponential moment by that of the independent geometric-load tree, and Proposition 4.4 bounds the exponential moment of the independent geometric-load tree (i.e., $\mathbb{E}[(7/5)^{Q_h}]$) by 23/20 uniformly over $h \geq 0$. Next, we state these propositions and show that together they imply Proposition 4.1.

Proposition 4.3 (Comparison with Geometric Loads)**.** *For every integer $N \geq 2$, every instance (s, α) of $\mathcal{P}_{4,N}$, and every $1 < u < 2$, write $\pi = \mathcal{P}_{4,N}(s, \alpha)$ and $L = \lceil \log_2 N \rceil$. Then*

$$\mathbb{E}[u^{S(\pi;s)}] \leq \mathbb{E}[u^{Q_{L+1}}]. \tag{17}$$

Proposition 4.4 (Uniform Geometric Moment Bound)**.** *Let G be a random variable with $\mathbb{P}[G = k] = 2^{-k-1}$ for integers $k \geq 0$. Set $Q_0 = 0$ and, for every integer $h \geq 0$, define*

$$Q_{h+1} \stackrel{\mathrm{d}}{=} \left(G + Q_h^{(0)} + Q_h^{(1)} - 4\right)^+,$$

where $x^+ = \max\{x, 0\}$, and $Q_h^{(0)}$ and $Q_h^{(1)}$ are independent copies of Q_h that are also independent of G. Then, for every integer $h \geq 0$,

$$\mathbb{E}\left[(7/5)^{Q_h}\right] \leq \frac{23}{20}. \tag{18}$$

Proof of Proposition 4.1. If $N = 1$, then $L = 0$ and the only real block fits in the root bucket. Thus $S(\pi; s) = 0$, and the claimed moment bound holds. For $N \geq 2$, apply Proposition 4.3 with $u = 7/5$ and then Proposition 4.4 with $h = L + 1$. This gives

$$\mathbb{E}\left[(7/5)^{S(\pi;s)}\right] \leq \mathbb{E}\left[(7/5)^{Q_{L+1}}\right] \leq \frac{23}{20}.$$

□

The stash moment bound is now reduced to the two propositions above. We first establish the comparison for arbitrary fixed address sequences, and then prove the all-height moment bound.

4.3 Proof of Proposition 4.3

Proposition 4.3 (Comparison with Geometric Loads, Restated)**.** *For every integer $N \geq 2$, every instance (s, α) of $\mathcal{P}_{4,N}$, and every $1 < u < 2$, write $\pi = \mathcal{P}_{4,N}(s, \alpha)$ and $L = \lceil \log_2 N \rceil$. Then*

$$\mathbb{E}[u^{S(\pi;s)}] \leq \mathbb{E}[u^{Q_{L+1}}]. \tag{17}$$

Throughout this subsection, fix an instance (s, α) of $\mathcal{P}_{4,N}$ with $N \geq 2$, write $\pi = \mathcal{P}_{4,N}(s, \alpha)$, and set $L = \lceil \log_2 N \rceil$ and $\widehat{N} = 2^L$, with $1 < u < 2$. The proof follows the four subsubsections below. Section 4.3.1 recalls the packing identity and monotonicity properties proved in Appendix A. Section 4.3.2 uses independent labels to construct a full scan whose overflow bounds the actual stash. Section 4.3.3 bounds the scan's moment by that of Poisson node loads that are independent given the refill paths. Section 4.3.4 randomizes their means to obtain independent geometric loads and completes the comparison.

4.3.1 Exact Overflow Representation of the Stash

We state the packing identity and monotonicity properties needed for the full-scan construction, using the general-capacity proofs in Appendix A. The exact stash identity and maximum overload representation correspond to Lemmas 5.4 and 5.5 of the original Path ORAM analysis [SvDS+18]. Our appendix proves them directly from target loads and also covers nonnegative real loads, as needed for the later convexity argument. Below we specialize these results to $Z = 4$.

Targets record the deepest placement opportunities. We define each block's target from the paths read since its most recent access, then count how many blocks have each target. Write $\mathcal{V}_L$ for the nodes of the depth-L tree, ε for its root, and $|v|$ for the depth of node v. The leaves are the strings in $\{0,1\}^L$. Write $v \preceq w$ when v is a prefix of w, so that v is an ancestor of w, possibly equal to it. The children of an internal node v are $v0, v1$, and $\mathrm{path}(y)$ is the path from the root to leaf y.

Let a_j be the address accessed at time j. At this access, the old label x_j determines the read and refill path, and the accessed block receives a fresh label U_j before refill. After t accesses, the created blocks have identifiers $I_t = \{a_1, \ldots, a_t\}$, with $I_0 = \varnothing$. For $b \in I_t$, let $\tau_t(b)$ be its last access time at or before t, and let y_b be its current leaf label. This label is unchanged from access $\tau_t(b)$ through time t. The node $\mathrm{lcp}(y, x)$ is the longest common prefix of leaves y, x, or equivalently the deepest node shared by their paths. A refill on $\mathrm{path}(x_j)$ can place b no deeper than $\mathrm{lcp}(y_b, x_j)$. We call this node a placement opportunity, whether or not a free slot is available there. Define the target of b by

$$d_t(b) = \text{the deepest node among } \{\mathrm{lcp}(y_b, x_j) : \tau_t(b) \leq j \leq t\}. \tag{19}$$

All these nodes lie on $\mathrm{path}(y_b)$, so the deepest one is unique. The interval includes b's own most recent access because its new label is assigned before that access refills the old path. Define the node loads and their vector by

$$A_v(t) = \big|\{b \in I_t : d_t(b) = v\}\big|, \qquad A(t) = (A_v(t))_{v \in \mathcal{V}_L}.$$

We call $A(t)$ the target-load vector at time t. Thus $d_t(b)$ is one block's target, whereas $A_v(t)$ counts all blocks with target v. Both depend only on the labels and access history.

Upward packing defines the overflow recursion. We define the overflow of any node-load vector by placing its load at the nodes and passing excess upward. For integer loads, process the nodes from leaves to root. A node keeps as many incoming blocks as possible, up to four, and sends the rest to its parent. The incoming blocks consist of its local load and the excess from its children. For any $A \in [0, \infty)^{\mathcal{V}_L}$, this rule is expressed by

$$D_v^{(4)}(A) = \Big(A_v + D_{v0}^{(4)}(A) + D_{v1}^{(4)}(A) - 4\Big)^+, \qquad D^{(4)}(A) = D_\varepsilon^{(4)}(A), \tag{20}$$

where $x^+ = \max\{x, 0\}$ and missing children contribute zero. Here $D_v^{(4)}(A)$ is the excess above v, and $D^{(4)}(A)$ is the excess above the root. The formula also defines these functions for nonnegative real loads, which will be needed for the convexity arguments. This packing is an auxiliary calculation, not another operation of the ORAM client.

Real refill gives the same residual count. The following identity lets us analyze target loads instead of tracing all intermediate bucket contents.

Lemma 4.5 (Exact Residual Stash)**.** *For the fixed unit-block execution π, every realization of its labels, and every $t \in \{0, \dots, s\}$,*

$$S(\pi; t) = D^{(4)}(A(t)). \tag{21}$$

The identity follows from two invariants of completed refill, stated below and proved for general capacity in Appendix A.2. At a completed access time, let B_v be the set of blocks in bucket v, and let $\mathcal{S}$ be the stash, viewed as a location strictly above the root. Each block of I_t occurs in exactly one of these locations, $|B_v| \le 4$, and $S(\pi; t) = |\mathcal{S}|$. A block in B_v must be eligible for that bucket, meaning $v \preceq y_b$. The subtree of v consists of v and all its descendants.

Lemma 4.6 (Greedy Packing Invariants)**.** *For every label realization, after each completed access t:*

1. *every block stored in a bucket is at an ancestor of its target;*
2. *for every block $b \in I_t$ and node $v \preceq d_t(b)$, if b is stored strictly above v, then $|B_v| = 4$.*

Proof. Apply Lemma A.2 with $Z = 4$. □

Proof of Lemma 4.5. Apply Lemma A.1 with $Z = 4$. Its proof uses the two invariants above and conservation of blocks at each bucket to obtain recursion (20). □

The identity compares counts, not the identifiers assigned to individual buckets. The argument uses exhaustive bottom-up refill; we keep the specified smallest-identifier rule throughout the paper.

Maximum overload implies the two monotonicity properties. The maximum-overload formula below makes the effect of adding load or moving it upward explicit. A node set is ancestor-closed if it contains every ancestor of each of its nodes. For example, including 01 requires including 0 and the root, but does not require including 1.

Lemma 4.7 (Overflow as Maximum Overload)**.** *For every $A \in [0, \infty)^{\mathcal{V}_L}$,*

$$D^{(4)}(A) = \max\left\{0, \max_{T} \sum_{v \in T} (A_v - 4)\right\}, \tag{22}$$

where T ranges over nonempty ancestor-closed node sets containing the root.

Proof. This is Lemma A.3 with $Z = 4$, including its extension to nonnegative real loads. □

Lemma 4.8 (Monotonicity of Overflow)**.** *For nonnegative real loads on the fixed tree, neither of the following operations decreases $D^{(4)}$:*

1. *adding nonnegative load at any nodes;*

2. moving load from a node to an ancestor, with all resulting loads nonnegative.

Proof. These are the first two conclusions of Lemma A.4 with $Z = 4$. □

The next subsubsection applies these operations to the exact stash representation: deleting opportunities moves targets upward, and adding blocks adds target loads.

4.3.2 Comparison with a Full Scan

In this subsubsection, we prove that the actual execution π and a full scan can be constructed using shared random labels so that

$$S(\pi; s) \le D^{(4)}(A^*),$$

where A^* is the final target-load vector of a scan that accesses $\widehat{N} = 2^L$ distinct addresses once each on the same depth-L tree. Thus the full scan's root overflow bounds the actual stash count for each realization in this joint construction, with each process retaining its own distribution. The proof has two stages. First, we recall the independent label law and exact scan representation proved in Appendix A.3. Second, we use this representation to replace an arbitrary access sequence by a full scan without decreasing overflow.

Random labels and their joint independence. We specify the initial and new leaf labels and state the joint independence needed to construct the full scan. Let $U_{0,b}$ be the initial label of address b, and let U_t be the new label drawn at access t. All these choices are mutually independent and uniform over the leaves. Initially $y_b = U_{0,b}$. At access t, let a_t be the address being accessed and save its old label $x_t = y_{a_t}$ before setting $y_{a_t} = U_t$. Both the read and the refill use path(x_t), the path from the root to the leaf x_t. For a fixed address sequence, these initial and new labels are the only random inputs to placement.

Repeated accesses appear to make the path labels dependent, because an old path uses a label selected at an earlier access. For a fixed address sequence, however, each independent random label is used once in exactly one of two roles: an old path label, or a final block label. Relabeling these same random inputs exposes the independence we need.

Lemma 4.9 (Joint Independence of Old and Final Labels)**.** *Fix a deterministic address sequence $\alpha = (a_1, \dots, a_s) \in [N]^s$. Let x_t be the leaf label of address a_t immediately before access t, and let y_b^{fin} be the label of address b after all s accesses, retaining its initial label if b is never accessed. If the initial labels $U_{0,1}, \dots, U_{0,N}$ and the new labels $U_1, \dots, U_s$ are mutually independent and uniform on $\{0,1\}^L$, then the $N + s$ random variables*

$$x_1, \dots, x_s, \quad y_1^{\mathrm{fin}}, \dots, y_N^{\mathrm{fin}}$$

are mutually independent and uniform on $\{0,1\}^L$. The same conclusion holds after any permutation of these coordinates that depends only on α.

Proof. Lemma A.5 identifies these variables as a deterministic permutation of the independent initial and fresh labels. A further permutation depending only on α preserves their joint law. □

Corollary 4.10 (Joint Independence of Reverse-Order Full-Scan Labels)**.** *Let $\widehat{N} = 2^L$ and consider the full scan $\mathcal{P}_{4,\widehat{N}}(\widehat{N}, (1, \dots, \widehat{N}))$ on the depth-L tree. Let $X_1, \dots, X_{\widehat{N}}$ be the old path labels in reverse chronological order, and let $Y_1, \dots, Y_{\widehat{N}}$ be the final block labels in reverse access order. Then these $2\widehat{N}$ labels are jointly independent and uniform on $\{0,1\}^L$.*

Proof. Apply Lemma 4.9 to the full scan and reverse the two coordinate lists. □

An exact target representation of the full scan. In a full scan, each address is accessed exactly once, so the kth block in reverse access order has exactly k placement opportunities. The exact scan representation from Appendix A.3 therefore takes the following form.

Lemma 4.11 (Exact Target Representation of the Full Scan)**.** *Let $\widehat{N} = 2^L$ and consider the full scan $\mathcal{P}_{4,\widehat{N}}(\widehat{N}, (1, \ldots, \widehat{N}))$ on the depth-L tree. Let $X_1, \ldots, X_{\widehat{N}}$ be the old path labels in reverse chronological order, and let $Y_1, \ldots, Y_{\widehat{N}}$ be the final block labels in reverse access order. For every $1 \le k \le \widehat{N}$, the target of block k in reverse access order is the deepest node among*

$$\{\mathrm{lcp}(Y_k, X_j) : 1 \le j \le k\},$$

where $\mathrm{lcp}(y, x)$ denotes the node given by the longest common prefix of leaf labels y and x, equivalently the deepest node shared by their root-to-leaf paths. Let δ_k denote the deepest node specified above. Then, for every node v in the tree, the full scan's final target-load vector A^* satisfies

$$A^*_v = \sum_{k=1}^{\widehat{N}} \mathbf{1}_{\{\delta_k = v\}}.$$

Proof. Apply Lemma A.6 with $N = \widehat{N}$ and $Z = 4$. The block's own access contributes an opportunity because relabeling precedes refill. Counting the targets at v gives A^*_v. □

Replacing an arbitrary access sequence by a full scan. We prove that an arbitrary fixed access sequence can be replaced by a full scan whose root overflow bounds the actual stash count. To construct such a full scan, we will use both the independent label law in Corollary 4.10 and the target formula in Lemma 4.11. To prove the bound, we retain only selected placement opportunities for each existing block, then add blocks until their number is $\widehat{N} = 2^L$. Removing opportunities moves targets only upward, and adding blocks increases the loads. Neither operation can decrease overflow by Lemma 4.8.

Lemma 4.12 (Full-Scan Overflow Bounds the Actual Stash)**.** *Let $L = \lceil \log_2 N \rceil$ and $\widehat{N} = 2^L$. Consider an execution $\pi = \mathcal{P}_{4,N}(s, \alpha)$ with a fixed access sequence α, and a full scan of $\widehat{N}$ distinct addresses on the same tree. Let A^* be the full scan's final target-load vector. These two executions can be constructed using shared random labels, each retaining its prescribed distribution, so that*

$$S(\pi; s) \le D^{(4)}(A^*) \tag{23}$$

for every realization of the random labels in this shared construction.

Proof. We start with the actual execution, then construct the full scan from shared labels, and finally compare their overflow counts. The actual execution is left unchanged throughout this proof.

Step 1: identify the existing blocks and the labels to be shared.

(a) Identify and order the existing blocks. In the actual execution π, let $I_s = \{a_1, \ldots, a_s\}$ and $m = |I_s|$, the number of distinct addresses accessed by α. The initially empty process creates one block at the first access to each address, so I_s is exactly the set of blocks present after the s accesses. In particular, $m \le N \le \widehat{N}$. List these block identifiers as $b_1, \ldots, b_m$ in decreasing order of their last access times, and write

$$t_k = \tau_s(b_k), \qquad t_1 > t_2 > \cdots > t_m,$$

where $\tau_s(b)$ denotes the last time address b is accessed among accesses $1, \ldots, s$.

(b) Record the labels shared with the actual execution. Let Y_k be the final leaf label of block b_k, and set $X_j = x_{s-j+1}$ for $1 \le j \le s$, where x_t is the old path label at access t. Thus X_1 labels the last read path, X_2 the second-to-last read path, and so on. By Lemma 4.9, these old path labels and final block labels are jointly independent and uniform: their reordering depends only on the fixed address sequence α.

(c) Add the labels needed to construct the full scan. We now add the remaining labels needed to construct the full scan. Extend the path-label sequence by drawing $X_{s+1}, X_{s+2}, \ldots$, and supply labels $Y_{m+1}, \ldots, Y_{\widehat{N}}$ for the blocks to be added. Sample the added labels independently and uniformly from the leaves, independently also of π's random inputs.

Thus $X_1, X_2, \ldots, Y_1, \ldots, Y_{\widehat{N}}$ are jointly independent uniform leaf labels, with $X_1, \ldots, X_s$ and $Y_1, \ldots, Y_m$ taken from the actual execution. These labels provide the shared random inputs for constructing the full scan.

Step 2: reduce each existing block's opportunities to the full-scan rule. For each existing block b_k, we keep only its k most recent placement opportunities and discard any earlier ones. This gives the block the full-scan target rule from Lemma 4.11. We now prove that retaining only these opportunities cannot decrease the resulting root overflow.

(a) Show that the k opportunities to be retained actually exist. The retained paths have labels $X_1, \ldots, X_k$. To show that all of them were read at or after b_k's last access, recall that the blocks are ordered by decreasing last access time. The blocks $b_1, \ldots, b_{k-1}$ have distinct last accesses later than t_k. Thus at least $k-1$ accesses follow b_k's last access, giving

$$w_k := s - t_k + 1 \ge k.$$

From t_k through the end of the actual execution, b_k keeps its final label Y_k. Its full opportunity window consists of the paths read at times $t_k, \ldots, s$, whose labels in reverse chronological order are $X_1, \ldots, X_{w_k}$. Since $w_k \ge k$, the paths we retain all belong to this window.

(b) Show that deleting opportunities leaves the target unchanged or moves it upward. By the target definition in (19), the actual final target $d_s(b_k)$ is the deepest node in

$$\{\mathrm{lcp}(Y_k, X_j) : 1 \le j \le w_k\}.$$

For every $1 \le k \le \widehat{N}$, define δ_k to be the deepest node in

$$\{\mathrm{lcp}(Y_k, X_j) : 1 \le j \le k\}.$$

For an existing block $k \le m$, the latter set is a subset of the former. Every node in both sets lies on the path from the root to Y_k. Removing some candidates cannot make the deepest remaining node deeper. Consequently, δ_k equals $d_s(b_k)$ or is an ancestor of it.

(c) Apply monotonicity to show that root overflow cannot decrease. The original load vector $A(s)$ places one unit at $d_s(b_k)$ for each existing block b_k. Moving that unit to δ_k for every $1 \le k \le m$ gives the load vector

$$\widetilde{A}_v = \sum_{k=1}^{m} \mathbf{1}_{\{\delta_k = v\}} \qquad \text{for every node } v \text{ in the tree.}$$

By part (b), each such move is to the same node or an ancestor. Applying the upward-move property of Lemma 4.8 to these moves gives

$$D^{(4)}(A(s)) \le D^{(4)}(\widetilde{A}).$$

This proves the bound when each block b_k retains only its k most recent opportunities.

Step 3: add blocks until there are $\widehat{N}$ targets. Add a comparison block with target δ_k for each $k = m+1, \ldots, \widehat{N}$, and define

$$A_v^* = \sum_{k=1}^{\widehat{N}} \mathbf{1}_{\{\delta_k = v\}}.$$

These added blocks belong only to the comparison construction; they do not change the actual execution. At every node,

$$A_v^* - \widetilde{A}_v = \sum_{k=m+1}^{\widehat{N}} \mathbf{1}_{\{\delta_k = v\}} \geq 0.$$

The load-addition property in Lemma 4.8 therefore gives $D^{(4)}(\widetilde{A}) \leq D^{(4)}(A^*)$.

Step 4: generate a full scan with exactly this target-load vector. First, we specify the scan's access order and label assignments. Before the scan begins, give addresses $1, \ldots, \widehat{N}$ the initial labels $X_{\widehat{N}}, X_{\widehat{N}-1}, \ldots, X_1$, respectively. Then access addresses $1, \ldots, \widehat{N}$ once each, in that order. At these accesses, assign the new labels $Y_{\widehat{N}}, Y_{\widehat{N}-1}, \ldots, Y_1$, respectively.

Second, we show that this construction produces a full scan with the prescribed distribution and that its final target-load vector is A^*. By Step 1, all initial and new labels used in this construction are jointly independent and uniform leaves. These are the prescribed random inputs of a full scan, so the construction has the required execution distribution. In particular, it has the reverse-order label law stated in Corollary 4.10.

Address i is accessed only once, so its old path label is its initial label $X_{\widehat{N}-i+1}$, and its final label is the new label $Y_{\widehat{N}-i+1}$. Hence the old path labels in reverse chronological order are exactly $X_1, \ldots, X_{\widehat{N}}$, and the final block labels in reverse access order are exactly $Y_1, \ldots, Y_{\widehat{N}}$.

We can now apply Lemma 4.11 on the same depth-L tree with these labels. The lemma gives target δ_k for reverse block k, since its target formula uses exactly the label values defining δ_k in Step 2. The final load at every node v is therefore

$$\sum_{k=1}^{\widehat{N}} \mathbf{1}_{\{\delta_k = v\}} = A_v^*,$$

by the definition in Step 3. Thus the constructed full scan's final target-load vector is A^* for every realization of the shared labels.

Step 5: combine the count comparisons. Lemma 4.5 identifies the actual stash with $D^{(4)}(A(s))$. Steps 2 and 3 then give

$$S(\pi; s) = D^{(4)}(A(s)) \leq D^{(4)}(\widetilde{A}) \leq D^{(4)}(A^*).$$

Step 4 identifies A^* as the final target-load vector of the constructed full scan. The displayed comparison chain holds for every realization of the shared random labels, proving the lemma. □

The output is a pointwise upper comparison with the full scan field. It uses $N \leq 2^L$ precisely when adding the missing targets. We next bound the expected exponential of this scan's root overflow by that induced by independent node loads.

4.3.3 Conditional Poisson Comparison

To analyze the two child subtrees separately later, we seek to bound the full scan's overflow exponential moment by that of a tree with independent node loads. Here we construct a tree model whose node loads are independent given the old path sequence $X = (X_j)_{j\geq 1}$. We prove that its conditional overflow exponential moment is no smaller than the full scan's when each node has a Poisson load with the same conditional mean as the corresponding scan load. Specifically, let $P = (P_v)_v$ have independent coordinates conditional on X, with

$$P_v \mid X \sim \text{Poisson}(\mathbb{E}[A_v^* \mid X])\,.$$

We prove

$$\mathbb{E}\big[u^{D^{(4)}(A^*)} \mid X\big] \leq \mathbb{E}\big[u^{D^{(4)}(P)} \mid X\big].$$

We first establish the convexity properties needed for Poisson replacement, then replace the blocks' placements one at a time by Poisson counts to prove this inequality.

Convexity tools justify the Poisson replacement. We prove that replacing a single block's placement by independent Poisson counts with the same node means cannot decrease the expected exponential overflow. For a load vector A, define its exponential overflow cost as $F(A) = u^{D^{(4)}(A)}$. For a deterministic nonnegative mean vector λ, define

$$\Psi(\lambda) = \mathbb{E}[F(P)], \qquad P = (P_v)_{v\in\mathcal{V}_L}, \quad P_v \sim \text{Poisson}(\lambda_v) \text{ independently.} \tag{24}$$

F takes a load vector as its input, while Ψ takes a Poisson mean vector and averages over the corresponding independent counts. The notation Poisson(λ) used above denotes the distribution with probabilities $e^{-\lambda}\lambda^k/k!$, $k = 0, 1, \ldots$. We first establish the property of F needed for a single replacement, then prove the replacement inequality.

(a) Prove that adding the same load has a greater effect on F when the existing load is larger. We express this property by the inequality

$$f(z + h + k) - f(z + k) \geq f(z + h) - f(z) \qquad (z, h, k \geq 0). \tag{25}$$

The left-hand side is the increase in cost from adding h when the background load already includes k; the right-hand side is the increase from adding h to the original background z. Thus, increasing the background load cannot reduce the marginal cost of adding the same load. A function f on nonnegative vectors is called increasing and directionally convex here if it is coordinatewise increasing and satisfies (25) [BY09, Definition 2.1].

Recall the recursive definition of $D^{(4)}$ in (20):

$$D_v^{(4)}(A) = (A_v + D_{v0}^{(4)}(A) + D_{v1}^{(4)}(A) - 4)^+, \qquad D^{(4)}(A) = D_\varepsilon^{(4)}(A),$$

where ε denotes the root and missing children contribute zero. At a leaf, this reduces to $D_v^{(4)}(A) = (A_v - 4)^+$. For each node v, the function $A \mapsto A_v$ has the stated property, and both summation and composition with the increasing convex function $x \mapsto (x - 4)^+$ preserve it. The recursion therefore passes the property from the two child overflow functions to their parent's overflow function. Induction from the leaves establishes it at every node, including the root, where the overflow is $D^{(4)}$. Composing with the increasing convex function $x \mapsto u^x$ then establishes it for F. This property provides the convexity needed to justify the subsequent Poisson replacement. We state and prove these claims below, together with a growth bound that ensures the required expectations are finite.

Lemma 4.13 (Load Convexity and Growth Bound)**.** *On $[0,\infty)^{\mathcal{V}_L}$, both $D^{(4)}$ and F are increasing and directionally convex. Moreover,*

$$1 \le F(A) \le u^{\sum_{v\in\mathcal{V}_L} A_v}. \tag{26}$$

Proof. For each node v, the function $A \mapsto A_v$ is increasing and satisfies (25) with equality. Sums preserve both monotonicity and this inequality. We first show that composition with an increasing convex scalar function ϕ also preserves them. Suppose g has both properties. Write

$$a = g(z), \quad b = g(z+k), \quad \delta = g(z+h) - g(z), \quad \delta' = g(z+h+k) - g(z+k).$$

Monotonicity gives $b \ge a$ and $\delta, \delta' \ge 0$; (25) gives $\delta' \ge \delta$. For an increasing convex ϕ, the increment $\phi(a+\delta) - \phi(a)$ is increasing in both a and $\delta \ge 0$. Hence

$$\phi(b+\delta') - \phi(b) \ge \phi(a+\delta) - \phi(a),$$

which is (25) for $\phi \circ g$.

In recursion (20), each node adds its own coordinate to two child overflow functions and applies the increasing convex function $x \mapsto (x-4)^+$. Induction from the leaves proves the claims for $D^{(4)}$. Applying the same composition argument to $x \mapsto u^x$, which is increasing and convex since $u > 1$, proves them for F. Finally, no overflow is negative or exceeds the total load, giving (26). □

(b) Prove that a Poisson replacement cannot decrease expected cost. We prove that replacing a single block's node choice by independent Poisson counts with the same node means cannot decrease expected cost. For example, suppose one block targets either v or w, each with probability $1/2$. A single choice puts one block at exactly one node. Instead, draw $K \sim \text{Poisson}(1)$ and make K independent choices with the same probabilities. If $K = 0$, neither node receives a block. If $K = 2$, both blocks reach v or both reach w, each with probability $1/4$, or one reaches each node with probability $1/2$. After averaging over K, the two node loads are independent Poisson$(1/2)$ variables by Poisson splitting [LP18, Chapter 5]. The next lemma proves this property for arbitrary target probabilities and shows that the expected exponential overflow cannot decrease. A single placement is a coordinate unit vector; allowing the zero vector also covers the possibility of no placement.

Lemma 4.14 (Categorical Poissonization)**.** *Let $p_1, \dots, p_d \ge 0$ with $\sum_i p_i \le 1$, and let V equal the ith coordinate unit vector with probability p_i and zero with probability $1 - \sum_i p_i$. Let $C = (C_1, \dots, C_d)$ have independent coordinates with $C_i \sim \text{Poisson}(p_i)$. For every increasing directionally convex f and deterministic $z \ge 0$,*

$$\mathbb{E}[f(z+V)] \le \mathbb{E}[f(z+C)].$$

The same inequality holds for a random background independent of the placements, with V and C each sampled independently of that background.

Proof. We introduce an intermediate process that makes a Poisson number of independent placement choices. We first show that randomizing the number of choices cannot decrease the expected cost, then show that the resulting node counts have the law of C.

Take independent copies $V_1, V_2, \dots$ of V, and independently take $K \sim \text{Poisson}(1)$. For integers $k \ge 0$, define

$$g(k) = \mathbb{E}\left[f\left(z + \sum_{j=1}^{k} V_j \right) \right].$$

This is the expected cost after k choices; in particular, $g(1) = \mathbb{E}[f(z+V)]$. We show that each additional choice has a greater expected marginal cost as the number of choices increases. Let $Z_k = z + \sum_{j=1}^{k} V_j$, and let W, W' be independent copies of V, independent of Z_k. Then

$$\begin{aligned} &g(k+2) - 2g(k+1) + g(k) \\ &\quad = \mathbb{E}\big[f(Z_k + W + W') - f(Z_k + W) - f(Z_k + W') + f(Z_k)\big] \geq 0. \end{aligned}$$

The equality uses the identical distributions of $Z_k + W$ and $Z_k + W'$; the inequality is (25), applied to each realization. Hence the linear interpolation of g between consecutive integers is convex. Since $\mathbb{E}[K] = 1$, Jensen's inequality gives

$$\mathbb{E}[f(z+V)] = g(1) = g(\mathbb{E}[K]) \leq \mathbb{E}[g(K)].$$

Thus replacing one choice by K choices cannot decrease the expected cost.

It remains to show that these K choices produce independent Poisson node counts. Write $B = \sum_{j=1}^{K} V_j$, so B_i counts the choices that place a unit at node i. Since K is independent of the placement choices, conditioning on $K = k$ gives

$$\mathbb{E}[f(z+B) \mid K = k] = \mathbb{E}\left[f\left(z + \sum_{j=1}^{k} V_j\right)\right] = g(k).$$

The law of total expectation therefore yields $\mathbb{E}[f(z+B)] = \mathbb{E}[g(K)]$. For $r_1, \ldots, r_d \in [0,1]$, the joint probability generating function is

$$\begin{aligned} \mathbb{E}\left[\prod_i r_i^{B_i}\right] &= \mathbb{E}\left[\left(1 + \sum_i p_i(r_i - 1)\right)^K\right] \\ &= \exp\left(\sum_i p_i(r_i - 1)\right) = \prod_i \exp(p_i(r_i - 1)). \end{aligned}$$

The first equality conditions on K and uses independence of the choices; the second uses the generating function of Poisson(1). The final expression is the joint generating function of independent Poisson(p_i) variables. Thus B has the law of C, including when a choice contributes zero, and

$$\mathbb{E}[f(z+V)] \leq \mathbb{E}[g(K)] = \mathbb{E}[f(z+B)] = \mathbb{E}[f(z+C)].$$

For an independent random background, condition on that background and then average this inequality. In our application to F, the growth bound (26) and Poisson exponential moments make all expectations in this argument finite. □

Replacing the scan gives the conditional exponential-moment bound. We apply the single-block replacement in Lemma 4.14 to the entire scan: first fix X, then replace the blocks' placements one at a time, and finally sum the resulting Poisson counts. Write δ_k for reverse block k's target from Lemma 4.11, and define

$$p_{k,v} = \mathbb{P}[\delta_k = v \mid X] \qquad (1 \leq k \leq \widehat{N})$$

for every node v in the tree. The shared-label construction in the proof of Lemma 4.12 makes $Y_1, \ldots, Y_{\widehat{N}}$ mutually independent and independent of the entire sequence X. Given X, each δ_k

depends only on Y_k, so the targets are conditionally independent. *Replace the placements one block at a time.* Let e_v denote the coordinate unit vector for node v. Conditional on X, the scan load is $A^* = \sum_{k=1}^{\widehat{N}} e_{\delta_k}$, a sum of independent single-placement vectors. Replace the kth vector by Poisson counts at the nodes with means $p_{k,v}$. Take these counts independent over both k and v, conditional on X. At each replacement the unreplaced targets and previously introduced Poisson counts are an independent background, so Lemma 4.14 applies to F by Lemma 4.13. The expected cost cannot decrease. After all replacements, the count P_v is the sum of independent Poisson($p_{k,v}$) counts, hence

$$P_v \mid X \sim \text{Poisson}(\lambda_v(X)), \qquad \lambda_v(X) = \sum_{k=1}^{\widehat{N}} p_{k,v} = \mathbb{E}[A_v^* \mid X],$$

and the coordinates of P are independent conditional on X. Definition (24) now gives

$$\mathbb{E}[F(A^*) \mid X] \le \mathbb{E}[F(P) \mid X] = \Psi(\lambda(X)). \tag{27}$$

This proves the main conditional comparison. The node counts are independent given X; their means still depend on the shared sequence X, so unconditional independence has not been established.

Averaging this inequality over X by the law of total expectation gives

$$\mathbb{E}[F(A^*)] \le \mathbb{E}[F(P)] = \mathbb{E}_X[\Psi(\lambda(X))].$$

This completes the conditional Poisson step. The next subsubsection calculates and bounds the random means $\lambda(X)$, then handles their dependence through X.

4.3.4 Independent Geometric Loads and Completion

We prove that the Poisson model from the preceding subsubsection satisfies

$$\mathbb{E}_X[\Psi(\lambda(X))] \le \mathbb{E}[F(G)] = \mathbb{E}[u^{Q_{L+1}}],$$

where $G = (G_v)_{v \in \mathcal{V}_L}$ has independent coordinates with $\mathbb{P}[G_v = k] = 2^{-k-1}$, $k \ge 0$. We first calculate the conditional target probabilities and Poisson means using the paths' first-hit indices. Next, we establish the monotonicity and convexity of Ψ needed to change these means. We then bound them by internal-node waiting intervals and constant leaf means. Finally, random clocks and a separate leaf-mean step give independent exponential means, whose Poisson mixtures produce the independent geometric loads G. We combine these expected-cost inequalities to complete the comparison on the same finite tree.

Conditional target probabilities determine the Poisson means. We calculate the probabilities $p_{k,v}$ from the preceding subsubsection in order to evaluate and bound the Poisson means $\lambda_v(X) = \sum_{k=1}^{\widehat{N}} p_{k,v}$. Write δ_k for the target of reverse block k, as in Lemma 4.11. With X fixed, δ_k depends only on Y_k, so the targets of different blocks are conditionally independent. This does not make node loads independent: one block has only one target and cannot contribute to two nodes.

For every node v, define its first path-hit index

$$T_v = \min\{j \ge 1 : v \preceq X_j\}.$$

Every node has positive hit probability, so all these indices are finite almost surely on the finite tree. For an internal node v, let $c(v)$ be its later-hit child, and set $p_v = 2^{-|v|-1}$, the fraction of leaves below that child. The earlier child's hit index is T_v. The conditional target probabilities satisfy

$$p_{k,v} = p_v \mathbf{1}_{\{T_v \le k < T_{c(v)}\}} \quad (v \text{ internal}), \qquad p_{k,y} = 2^{-L} \mathbf{1}_{\{k \ge T_y\}} \quad (y \text{ a leaf}).$$

For an internal v, the target equals v precisely when some of the first k paths have reached one child, none have reached the other, and Y_k lies in the unvisited child. The first two conditions give $T_v \le k < T_{c(v)}$. Conditional on X, the last has probability p_v because Y_k is uniform and independent of X. If neither child has been reached, a target cannot reach v; if the child containing Y_k has been reached, the deepest match is below v. For a leaf y, the target is y exactly when $Y_k = y$ and some of the first k paths equal y. These conditions give the displayed leaf probability, since an exact leaf match cannot be improved further.

Mean-cost convexity justifies changing the Poisson means. We prove the properties of Ψ that justify increasing its means and later applying Jensen to random clocks and leaf means.

Lemma 4.15 (Convexity of the Poisson-Averaged Cost)**.** *The function Ψ in (24) is finite, continuous, increasing, and directionally convex on $[0,\infty)^{\mathcal{V}_L}$. In particular, for fixed $\lambda, a \ge 0$, the function $t \mapsto \Psi(\lambda + ta)$ is convex on $t \ge 0$.*

Proof. For fixed λ, the total count $T = \sum_v P_v$ is Poisson with mean $\sum_v \lambda_v$. Thus (26) gives

$$\Psi(\lambda) \le \mathbb{E}[u^T] = \exp\big((u-1)\textstyle\sum_v \lambda_v\big) < \infty.$$

For nonnegative vectors λ, h, k, take independent Poisson load vectors P, H, K with these respective mean vectors, all mutually independent. Sums of independent Poisson variables add their means, so

$$\begin{aligned}&\Psi(\lambda+h+k) - \Psi(\lambda+h) - \Psi(\lambda+k) + \Psi(\lambda)\\ &\quad = \mathbb{E}\big[F(P+H+K) - F(P+H) - F(P+K) + F(P)\big] \ge 0\end{aligned}$$

by Lemma 4.13. The same construction with only H proves monotonicity.

To prove continuity, restrict the means to a range with total at most $m < \infty$. For any $w > u$ and integer $K_0 \ge 0$,

$$\mathbb{E}[u^T \mathbf{1}_{\{T > K_0\}}] \le (u/w)^{K_0} \mathbb{E}[w^T] \le (u/w)^{K_0} \exp(m(w-1)).$$

This tends to zero uniformly in that range as $K_0 \to \infty$. For $T \le K_0$ there are finitely many load vectors, and each Poisson probability is continuous in its means. The uniform tail bound and (26) therefore prove continuity of Ψ. Finally, equal additions in (25) give midpoint convexity along every nonnegative direction; continuity gives convexity on the whole corresponding ray. No joint convexity in arbitrary directions is asserted or needed. □

Waiting intervals bound the Poisson means. We enlarge the internal-node means to complete waiting intervals and the leaf means to one, obtaining the upper bound used by the random-clock construction below. The conditional probability formulas give

$$\lambda_v(X) = p_v \, \#\{k \in \{1, \dots, \widehat{N}\} : T_v \le k < T_{c(v)}\} \le p_v (T_{c(v)} - T_v)$$

for an internal node v. The inequality counts the entire half-open integer interval, dropping the upper restriction $k \le \widehat{N}$. For example, suppose the two children of v are first visited at reverse

indices one and three, so $T_v = 1$ and $T_{c(v)} = 3$. Only reverse blocks $k = 1, 2$ can target v, each with conditional probability p_v. Their total contribution to the mean is $\lambda_v(X) = 2p_v$, equal to the full-interval bound. At a leaf y, each of the $\widehat{N}$ blocks contributes probability at most 2^{-L}, so $\lambda_y(X) \le \widehat{N}2^{-L} = 1$. Thus, defining the enlarged internal means by the following formula, we have

$$\begin{aligned} &\lambda_v(X) \le \bar{\lambda}_v(X) := p_v(T_{c(v)} - T_v) && (v \text{ internal}), \\ &\lambda_y(X) \le 1 && (y \text{ a physical leaf}). \end{aligned} \tag{28}$$

Let $\lambda^+(X) = (\bar{\lambda}_{\mathrm{int}}(X), 1_{\mathrm{leaf}})$. Here int restricts a vector to internal nodes, and 1_{leaf} is the vector assigning one to each physical leaf. We have $\lambda(X) \le \lambda^+(X)$ coordinatewise. Monotonicity in Lemma 4.15, followed by (27), gives

$$\mathbb{E}[F(A^*) \mid X] \le \Psi(\lambda(X)) \le \Psi(\lambda^+(X)).$$

Average the mean comparison over the old paths. Averaging over X by the law of total expectation gives

$$\mathbb{E}[F(A^*)] \le \mathbb{E}_X[\Psi(\lambda(X))] \le \mathbb{E}_X[\Psi(\bar{\lambda}_{\mathrm{int}}(X), 1_{\mathrm{leaf}})].$$

We next introduce random clocks for the internal means in the last expression. Leaf means require separate treatment because a physical leaf has no children and hence no child-hit waiting interval.

Random clocks cannot decrease the conditional expected cost. We replace the discrete waiting intervals in $\bar{\lambda}$ by random time intervals and prove the comparison while conditioning on X. For $r > 0$, write $\mathrm{Exp}(r)$ for the exponential law with survival probability e^{-rt}, $t \ge 0$. Take independent $W_1, W_2, \ldots \sim \mathrm{Exp}(1)$, independent also of the old path sequence X. Give old path X_j arrival time $C_j = \sum_{i=1}^{j} W_i$, and put $\mathcal{T}_v = C_{T_v}$ for the first arrival at node v. For every internal node, define

$$H_v = p_v(\mathcal{T}_{c(v)} - \mathcal{T}_v) = p_v \sum_{j=T_v+1}^{T_{c(v)}} W_j. \tag{29}$$

Thus H_v is the time between the first hits of the two children, scaled by their common leaf fraction p_v. In the example $T_v = 1$ and $T_{c(v)} = 3$, the bound $2p_v$ becomes $H_v = p_v(W_2 + W_3)$. These clocks measure the intervals from path one to path two and from path two to path three, respectively. For fixed paths, their mean-one laws give $\mathbb{E}_W[H_v \mid X] = 2p_v$. The construction uses the same clock sequence at every node, so we must justify its effect on all node means together.

Fix a path sequence X for which all first-hit indices are finite. This event has probability one by the first-hit argument above. Let $n_X = \max_{v \in \mathcal{V}_L} T_v$ and define internal-node vectors a_j by

$$(a_j)_v = p_v \mathbf{1}_{\{T_v < j \le T_{c(v)}\}} \qquad (1 \le j \le n_X).$$

These coefficients are fixed and nonnegative after conditioning on X, and (29) gives

$$H_{\mathrm{int}} = \sum_{j=1}^{n_X} a_j W_j, \qquad \mathbb{E}_W[H_{\mathrm{int}} \mid X] = \sum_{j=1}^{n_X} a_j = \bar{\lambda}_{\mathrm{int}}(X).$$

For fixed values of all clocks except W_j, the map

$$w \longmapsto \Psi\left(\sum_{i \ne j} a_i W_i + a_j w, 1_{\mathrm{leaf}}\right)$$

is convex on $w \geq 0$ by Lemma 4.15: its direction is the nonnegative vector a_j on internal nodes and zero on leaves. Since the clocks are independent and have mean one, one-dimensional Jensen permits replacing each W_j by one without increasing the expected value. Applying this to the finitely many clocks yields

$$\Psi(\bar{\lambda}_{\mathrm{int}}(X), 1_{\mathrm{leaf}}) \leq \mathbb{E}_W[\Psi(H_{\mathrm{int}}, 1_{\mathrm{leaf}}) \mid X]. \tag{30}$$

The conditional notation emphasizes that only the clocks are averaged here. We have not assumed the H_v independent given X, or used joint convexity of Ψ. Nonnegative extended expectations suffice for this inequality; the final geometric comparison will give finiteness.

Scaled child-arrival gaps have an independent joint law. We prove a tree-level independence statement that will identify the joint law of H after both paths and clocks are averaged.

Lemma 4.16 (Dyadic Exponential Gaps)**.** *Let a finite complete binary tree have depth $\ell \geq 1$, and assign independent* $\mathrm{Exp}(\rho)$ *variables E_y to its leaves, for a common $\rho > 0$. For each node v, let E_v be the minimum of these variables in its descendant leaves. For every internal node, define*

$$r_v = \rho\, 2^{\ell - |v| - 1}, \qquad \Gamma_v = r_v |E_{v0} - E_{v1}|.$$

The variables Γ_v are jointly independent with law $\mathrm{Exp}(1)$*, and their entire vector is independent of E_ε.*

Proof. Step 1: calculate subtree first-arrival laws. A node v has $2^{\ell-|v|}$ descendant leaves. By their independence,

$$\mathbb{P}[E_v > t] = \prod_{y \text{ below } v} \mathbb{P}[E_y > t] = e^{-\rho 2^{\ell-|v|} t}.$$

Thus $E_v \sim \mathrm{Exp}(\rho 2^{\ell-|v|})$. The two children of an internal node depend on disjoint leaf sets, so their first times are independent and each has rate r_v.

Step 2: separate a pair's minimum and gap. Take independent $E_0, E_1 \sim \mathrm{Exp}(r)$, and set $m = \min(E_0, E_1)$, $g = |E_0 - E_1|$. Let b indicate which variable is smaller. Ties have probability zero. For each fixed value of b, the change of variables to (m, g) has absolute Jacobian one, so its density is

$$r^2 e^{-2rm - rg} = (2re^{-2rm})(re^{-rg})/2 \qquad (m, g \geq 0). \tag{31}$$

The factorization gives independent variables $m \sim \mathrm{Exp}(2r)$, $g \sim \mathrm{Exp}(r)$, and a uniform binary choice b. In particular, $rg \sim \mathrm{Exp}(1)$ and is independent of m.

Step 3: prove joint independence by induction up the tree. We prove the stronger claim that the scaled gaps inside each subtree are independent $\mathrm{Exp}(1)$ variables and that their entire vector is independent of that subtree's first time. This is vacuous at a leaf. Assume the claim for both children of a node. The two subtrees are functions of disjoint independent leaf variables. Their two gap vectors are therefore jointly independent and, by the induction hypotheses, are jointly independent of the pair of child first times. By Step 1 the child first times are independent exponentials of the same rate. Step 2 transforms this pair into the parent's first time and its scaled gap, which are independent of one another and of both existing gap vectors. This proves the stronger claim for the parent. At the root it gives both assertions of the lemma. □

Lemma 4.17 (Joint Law of the Clock Means)**.** *Under the joint randomness of the independent uniform paths X_j and independent clocks $W_j \sim \mathrm{Exp}(1)$, the internal-node means H_v in* (29) *are jointly independent* $\mathrm{Exp}(1)$ *variables.*

Proof. The exponential spacings W_j give a rate-one Poisson arrival process. Mark its jth arrival by the independent uniform leaf X_j. We first verify that the resulting leaf processes are independent, each of rate $1/\widehat{N}$. For an interval of length t, its arrival count is Poisson(t). Conditional on that count, the marks are independent uniform leaves. If C_y denotes the number marked by leaf y in this interval, the same generating-function calculation used in Lemma 4.14 gives

$$\mathbb{E}\left[\prod_y r_y^{C_y}\right] = \exp\left(\frac{t}{\widehat{N}}\sum_y (r_y - 1)\right) = \prod_y \exp\left(\frac{t}{\widehat{N}}(r_y - 1)\right).$$

Hence the leaf counts in this interval are independent Poisson variables with mean $t/\widehat{N}$. The original Poisson process has independent increments and all marks are independent, so the same statement holds jointly over disjoint intervals. It follows that the leaf arrival processes are independent Poisson processes of rate $1/\widehat{N}$. Their first-arrival times are therefore independent $\mathrm{Exp}(1/\widehat{N})$ variables.

Apply Lemma 4.16 with $\ell = L$ and $\rho = 1/\widehat{N}$. The first arrival at node v is the minimum of the first arrivals at its descendant leaves, namely $\mathcal{T}_v$. The child rate in that lemma is

$$r_v = \widehat{N}^{-1} 2^{L-|v|-1} = 2^{-|v|-1} = p_v.$$

The earlier child first time is $\mathcal{T}_v$, and the later one is $\mathcal{T}_{c(v)}$. Consequently $\Gamma_v = H_v$ at every internal node, and the joint-independence conclusion of the lemma applies. This argument averages over both X and W; it makes no assertion of independence conditional on X. □

Independent exponential means give independent geometric loads. We identify the count distribution obtained by averaging independent Poisson loads over independent exponential means.

Lemma 4.18 (Independent Exponential Means Give Geometric Counts)**.** *Let $(\Lambda_v)_{v\in V}$ be a finite vector of independent* Exp(1) *variables. Conditional on the entire vector Λ, let $(C_v)_{v\in V}$ be independent with $C_v \mid \Lambda \sim \mathrm{Poisson}(\Lambda_v)$. Then the C_v are jointly independent, and $\mathbb{P}[C_v = k] = 2^{-k-1}$ for each v and integer $k \geq 0$.*

Proof. For one coordinate, integrate its conditional Poisson probability against the exponential density:

$$\mathbb{P}[C_v = k] = \int_0^\infty e^{-z}\frac{e^{-z}z^k}{k!}\,dz = \frac{1}{k!}\frac{k!}{2^{k+1}} = 2^{-k-1}.$$

The integral $\int_0^\infty e^{-2z}z^k\,dz = k!/2^{k+1}$ follows by integration by parts, starting with $k = 0$. For any fixed tuple $(k_v)_{v\in V}$, conditional independence gives

$$\mathbb{P}[C_v = k_v \text{ for every } v] = \mathbb{E}\left[\prod_{v\in V} e^{-\Lambda_v}\frac{\Lambda_v^{k_v}}{k_v!}\right].$$

Independence of the means factors this expectation into the product of the one-coordinate integrals, giving $\prod_v 2^{-k_v-1}$. This proves joint independence as well as the individual laws. □

Leaf randomization and the comparison chain complete the proof. We give the remaining leaf comparison and combine all preceding inequalities to prove Proposition 4.3.

Proof of Proposition 4.3. Step 1: average the conditional comparisons. By Lemma 4.12, the actual stash is at most $D^{(4)}(A^*)$ in a shared-label construction that preserves both prescribed distributions.

Since $u > 1$, taking exponential costs and expectations preserves this inequality. The complete chain is

$$\begin{aligned}
\mathbb{E}[u^{S(\pi;s)}] &\le \mathbb{E}[F(A^*)] && \text{by Lemma 4.12 and } u > 1,\\
&= \mathbb{E}_X[\mathbb{E}[F(A^*) \mid X]] && \text{by total expectation},\\
&\le \mathbb{E}_X[\Psi(\lambda(X))] && \text{by (27)},\\
&\le \mathbb{E}_X[\Psi(\bar{\lambda}_{\text{int}}(X), 1_{\text{leaf}})] && \text{by (28) and Lemma 4.15},\\
&\le \mathbb{E}_{X,W}[\Psi(H_{\text{int}}, 1_{\text{leaf}})] && \text{by (30)}.
\end{aligned}$$

At the last line both paths and clocks have been averaged, so Lemma 4.17 identifies the internal mean vector as independent Exp(1) coordinates.

Step 2: replace the constant leaf means. Take fresh $J_y \sim \text{Exp}(1)$ at the physical leaves, independent of each other and of (X, W). With H_{int} and all other leaf means fixed, Ψ is convex in the mean at y by Lemma 4.15. Jensen and $\mathbb{E}[J_y] = 1$ therefore permit replacing that leaf's mean one by J_y without decreasing expected cost. Applying this to each leaf yields

$$\mathbb{E}_{X,W}[\Psi(H_{\text{int}}, 1_{\text{leaf}})] \le \mathbb{E}_{X,W,J}[\Psi(H_{\text{int}}, J_{\text{leaf}})].$$

The full mean vector $(H_{\text{int}}, J_{\text{leaf}})$ now consists of jointly independent Exp(1) variables. This step uses the actual physical leaves and does not add nodes below the finite tree.

Step 3: integrate the means and identify the root overflow. By definition (24), counts are independent Poisson variables conditional on this full mean vector. Lemma 4.18 applies because the mean coordinates are also independent. Thus their unconditional count vector has the law of G, and

$$\mathbb{E}_{X,W,J}[\Psi(H_{\text{int}}, J_{\text{leaf}})] = \mathbb{E}[F(G)] = \mathbb{E}[u^{Q_{L+1}}].$$

The last equality uses Definition 4.2: this tree has depth L and hence height $L + 1$, and its capacity-four upward overflow is Q_{L+1}. Combining Steps 1–3 proves the proposition.

To verify finiteness, the root overflow is at most the total geometric load by (26). For one node,

$$\mathbb{E}[u^{G_v}] = \sum_{k\ge 0} u^k 2^{-k-1} = (2-u)^{-1} \qquad (1 < u < 2).$$

Independence on the fixed finite tree therefore bounds $\mathbb{E}[F(G)]$ by $(2-u)^{-|\mathcal{V}_L|} < \infty$. All earlier expected costs are nonnegative and bounded above by this quantity through the proved comparisons, so they too are finite. □

The comparison is now proved for every fixed physical depth. The next subsection bounds the geometric moment uniformly over all heights.

4.4 Proof of Proposition 4.4

Proposition 4.4 (Uniform Geometric Moment Bound, Restated)**.** *Let G be a random variable with $\mathbb{P}[G = k] = 2^{-k-1}$ for integers $k \ge 0$. Set $Q_0 = 0$ and, for every integer $h \ge 0$, define*

$$Q_{h+1} \overset{\text{d}}{=} \left(G + Q_h^{(0)} + Q_h^{(1)} - 4\right)^+,$$

where $x^+ = \max\{x, 0\}$, and $Q_h^{(0)}$ and $Q_h^{(1)}$ are independent copies of Q_h that are also independent of G. Then, for every integer $h \ge 0$,

$$\mathbb{E}\left[(7/5)^{Q_h}\right] \le \frac{23}{20}. \tag{18}$$

Proof of Proposition 4.4. Proposition 4.4 bounds $\mathbb{E}[(7/5)^{Q_h}]$ by $23/20$ uniformly over $h \geq 0$. We prove it using three lemmas. Lemma 4.19 gives the moment recurrence and its monotonicity properties. Lemma 4.20 uses these properties to identify three conditions at a single height that imply the proposition. Finally, Lemma 4.21 verifies all three conditions at height twelve.

Before stating the three lemmas, we introduce some notation. Throughout this subsection, we set $u = 7/5$ and $M_h = \mathbb{E}[u^{Q_h}]$.[3] We also write

$$a = \frac{1}{u^4(2-u)} = \frac{3125}{7203}.$$

To write the moment recurrence compactly, define an auxiliary function c of the total number k of blocks arriving from the two children, and let B_h be its expectation when those child subtrees have height h. With $x^+ = \max\{x, 0\}$, set

$$\begin{aligned} c(k) &= \sum_{g=0}^{\infty} 2^{-g-1}(1-u^{g+k-4})^+, \qquad k \in \mathbb{Z}_{\geq 0}, \\ B_h &= \mathbb{E}\Big[c\Big(Q_h^{(0)} + Q_h^{(1)}\Big)\Big]. \end{aligned} \tag{32}$$

We note that the sum in the definition of $c(k)$ averages $(1-u^{G+k-4})^+$ over G, and the definition of B_h then averages $c(k)$ over the random child sum.

We now state and prove the three lemmas, respectively.

Lemma 4.19 (Moment Recurrence and Monotonicity)**.** *For every integer $h \geq 0$,*

$$M_0 = 1, \qquad M_{h+1} = aM_h^2 + B_h. \tag{33}$$

The moments M_h are nondecreasing in h, whereas the nonnegative quantities B_h are nonincreasing.

Proof. Since $Q_0 = 0$, we have $M_0 = 1$. For each nonnegative integer z,

$$u^{\max\{z-4,0\}} = u^{z-4} + (1-u^{z-4})\mathbf{1}_{\{z\leq 3\}}.$$

Apply this identity to $z = G + Q_h^{(0)} + Q_h^{(1)}$. Independence gives

$$\mathbb{E}[u^{z-4}] = u^{-4}\mathbb{E}[u^G]M_h^2 = aM_h^2.$$

By the overflow recursion, $M_{h+1} = \mathbb{E}[u^{\max\{z-4,0\}}]$. Combining these identities gives

$$\begin{aligned} M_{h+1} - aM_h^2 &= \mathbb{E}\big[u^{\max\{z-4,0\}} - u^{z-4}\big] \\ &= \mathbb{E}\big[(1-u^{z-4})\mathbf{1}_{\{z\leq 3\}}\big]. \end{aligned}$$

Let $K = Q_h^{(0)} + Q_h^{(1)}$ be the total number of blocks arriving from the children, so $z = G + K$. If $K = k \leq 3$, only the loads $G = 0, \ldots, 3-k$ contribute to this difference; if $K \geq 4$, the difference is zero. Since G is independent of K, the probability of $K = k$ and $G = g$ is $\mathbb{P}[K = k]2^{-g-1}$. Summing over all contributing pairs gives

$$M_{h+1} - aM_h^2 = \sum_{k=0}^{3} \mathbb{P}[K=k] \sum_{g=0}^{3-k} 2^{-g-1}(1-u^{g+k-4}).$$

[3]We verify $M_h < \infty$ to ensure that $M_{h+1} - aM_h^2$ below is well defined. For each fixed h, Q_h is bounded by the sum of finitely many independent geometric loads, each with $\mathbb{E}[u^G] = 1/(2-u) < \infty$. Hence $M_h < \infty$.

The inner sum is exactly $c(k)$ from (32), and $c(k) = 0$ for $k \geq 4$. Thus

$$M_{h+1} - aM_h^2 = \sum_{k=0}^{\infty} \mathbb{P}[K = k]c(k) = \mathbb{E}[c(K)] = B_h.$$

This establishes the moment recurrence (33).

Next, we show $M_{h+1} \geq M_h$, i.e., M_h does not decrease with a larger height. Consider adding a bottom level to the previous tree with height h, which leads to a new tree with height $h+1$. These new leaves are also assigned independent geometric loads. Under upward packing, new leaves can pass blocks to existing nodes but cannot receive blocks from them. The added capacity cannot receive blocks from the original tree and therefore cannot reduce the overflow caused by its existing loads. Hence, induction from the bottom gives $Q_{h+1} \geq Q_h$. Since $u > 1$ and $M_h = \mathbb{E}[u^{Q_h}]$, we have $M_{h+1} \geq M_h$.

Finally, we show $B_{h+1} \leq B_h$ using the monotonicity of c. For integers $k' \geq k \geq 0$, the inequality $u > 1$ gives $(1 - u^{g+k'-4})^+ \leq (1 - u^{g+k-4})^+$ for every $g \geq 0$. Multiplying by the nonnegative weights 2^{-g-1} and summing gives, by (32),

$$c(k') = \sum_{g=0}^{\infty} 2^{-g-1}(1 - u^{g+k'-4})^+ \leq \sum_{g=0}^{\infty} 2^{-g-1}(1 - u^{g+k-4})^+ = c(k).$$

Apply the preceding height comparison separately to the two child subtrees, using independent loads for the two subtrees. For each child $i \in \{0, 1\}$, this gives $Q_{h+1}^{(i)} \geq Q_h^{(i)}$ almost surely, and hence

$$Q_{h+1}^{(0)} + Q_{h+1}^{(1)} \geq Q_h^{(0)} + Q_h^{(1)} \quad \text{a.s.}$$

Since c is nonincreasing, applying it to these sums reverses the inequality. Taking expectations and using the definition of B_h, we obtain

$$\begin{aligned} B_{h+1} &= \mathbb{E}\Big[c\Big(Q_{h+1}^{(0)} + Q_{h+1}^{(1)}\Big)\Big] \\ &\leq \mathbb{E}\Big[c\Big(Q_h^{(0)} + Q_h^{(1)}\Big)\Big] = B_h. \end{aligned}$$

□

Lemma 4.20 (Conditions for a Uniform Geometric Moment Bound)**.** *Suppose that an integer $H \geq 0$ and numbers $r, b \geq 0$ satisfy*

$$M_H \leq r, \qquad B_H \leq b, \qquad ar^2 + b \leq r.$$

Then $M_h \leq r$ for every integer $h \geq 0$.

Proof. For $h \leq H$, monotonicity gives $M_h \leq M_H \leq r$. For $h \geq H$, monotonicity gives $B_h \leq B_H \leq b$. Whenever $M_h \leq r$, recurrence (33) therefore yields

$$M_{h+1} \leq ar^2 + b \leq r.$$

Induction proves the bound at every larger height. □

Lemma 4.21 (Verification of the Three Conditions at $h = 12$)**.** *For the recursion*

$$Q_0 = 0, \qquad Q_{h+1} \stackrel{\mathrm{d}}{=} (G + Q_h^{(0)} + Q_h^{(1)} - 4)^+, \qquad h \geq 0,$$

all three conditions of Lemma 4.20 hold with

$$H = 12, \qquad r = \frac{23}{20}, \qquad b = \frac{57621}{100000}.$$

More precisely,

$$M_{12} < \frac{111}{100} < r, \qquad B_{12} < b, \qquad ar^2 + b < r. \tag{34}$$

Proof. We verify the three conditions by computing upper bounds up to height twelve. Starting from $Q_0 = 0$ and $M_0 = 1$, we compute the required probabilities at successive heights and use $M_{h+1} = aM_h^2 + B_h$ to bound the moments. The computation gives $M_{12} < 111/100 < r$ and $B_{12} < b$; direct substitution then verifies $ar^2 + b < r$. The proof has three steps. Step 1 specifies a finite calculation and proves that its outputs $\widehat{M}_{12}$ and $\widehat{B}_{12}$ satisfy $M_{12} \leq \widehat{M}_{12}$ and $B_{12} \leq \widehat{B}_{12}$. Step 2 evaluates these outputs and verifies $\widehat{M}_{12} < 111/100 < r$ and $\widehat{B}_{12} < b$. Together, these two steps establish the required inequalities for M_{12} and B_{12}. Step 3 verifies the remaining inequality $ar^2 + b < r$ by direct substitution, completing all three conditions.

Step 1: Construct a Finite Computation and Prove That Its Outputs Are Upper Bounds. We first derive finite updates, specify their evaluation, and prove that the resulting values are upper bounds.

(a) Derive the Probability Updates. Let $p_h(k) = \mathbb{P}[Q_h = k]$ be the probability that upward packing with capacity four sends exactly k blocks above the root of an independent geometric-load tree of height h. Also define $b_h(j) = \sum_{i=0}^{j} p_h(i)p_h(j-i)$. The latter is the probability that the two independent child overflows sum to j. Q_{h+1} is zero if $G + j \leq 4$ and is $k \geq 1$ if $G + j = k + 4$. Consequently,

$$p_{h+1}(0) = \sum_{j=0}^{4} b_h(j)(1 - 2^{-(5-j)}), \tag{35}$$

$$p_{h+1}(k) = \sum_{j=0}^{k+4} b_h(j)2^{-(k+5-j)} \quad (k \geq 1), \tag{36}$$

$$B_h = \sum_{j=0}^{3} b_h(j)c(j). \tag{37}$$

(b) Identify a Finite Set of Required Probabilities. By (37), B_{12} depends only on $b_{12}(0), \ldots, b_{12}(3)$. For $j \leq 3$, the definition $b_{12}(j) = \sum_{i=0}^{j} p_{12}(i)p_{12}(j-i)$ uses only $p_{12}(0), \ldots, p_{12}(3)$. Thus these four probabilities suffice to compute B_{12}. Let K_h denote the largest count whose probability we retain at height h, and set $K_{12} = 3$. Equation (35) uses child sums $j \leq 4$, while (36) uses $j \leq k + 4$ to compute $p_{h+1}(k)$. Each $b_h(j)$ in turn uses only $p_h(0), \ldots, p_h(j)$ by its definition. Consequently, computing all probabilities through count K_{h+1} requires probabilities only through count $K_{h+1} + 4$ at height h. Working backward from $K_{12} = 3$, we therefore choose

$$K_h = K_{h+1} + 4 = 4(12 - h) + 3 \qquad (0 \leq h < 12).$$

For example, $K_{11} = 7$, $K_{10} = 11$, and $K_0 = 51$. This choice retains every probability that can contribute to the required updates, and $K_h \geq 3$ ensures that B_h can be computed at every height used in the calculation.

The full moment M_h also includes contributions from counts larger than K_h. We account for all counts by advancing the moment through the exact recurrence $M_{h+1} = aM_h^2 + B_h$ from Lemma 4.19. Its only additional input, B_h, uses the retained probabilities by (37). This is why finitely many probability entries suffice to control the entire moment.

(c) Compute Upper Bounds with Upward Rounding. We write $P_h(k)$ for the computed bound on $p_h(k)$, and $\widehat{M}_h$ and $\widehat{B}_h$ for the computed bounds on M_h and B_h. The upper-bound property of these quantities will be proved in part (d) below. To specify the computation using integers, store $P_h(k)$ and $\widehat{M}_h$ as multiples of $1/D$, where $D = 2^{80}$. Define upward rounding by

$$\mathrm{up}(x) = \frac{\lceil Dx \rceil}{D}, \qquad x \le \mathrm{up}(x) < x + \frac{1}{D}.$$

Thus $\mathrm{up}(x)$ is the smallest multiple of $1/D$ that is at least x; rounding in this direction preserves an upper bound. The strict inequalities checked below establish that this precision is sufficient. Since $Q_0 = 0$, the initial probabilities and moment are known exactly:

$$P_0(0) = 1, \qquad P_0(k) = 0 \quad (1 \le k \le K_0), \qquad \widehat{M}_0 = 1.$$

At each height $h = 0, \ldots, 12$, form

$$\widehat{b}_h(j) = \sum_{i=0}^{j} P_h(i) P_h(j-i) \quad (0 \le j \le K_h), \qquad \widehat{B}_h = \sum_{j=0}^{3} \widehat{b}_h(j) c(j).$$

These expressions follow from the formulas for $b_h(j)$ and B_h by substituting the computed bounds for the true probabilities. Both quantities are evaluated exactly as rational numbers. In particular, $c(0), \ldots, c(3)$ are rational by (32), because $u = 7/5$ and only terms with $g + j \le 3$ can be nonzero. For $h < 12$, apply (35)–(36) and the moment recurrence to compute the next height:

$$\begin{aligned}
P_{h+1}(0) &= \mathrm{up}\left(\sum_{j=0}^{4} \widehat{b}_h(j)(1 - 2^{-(5-j)}) \right), \\
P_{h+1}(k) &= \mathrm{up}\left(\sum_{j=0}^{k+4} \widehat{b}_h(j) 2^{-(k+5-j)} \right) \quad (1 \le k \le K_{h+1}), \\
\widehat{M}_{h+1} &= \mathrm{up}(a\widehat{M}_h^2 + \widehat{B}_h).
\end{aligned}$$

All required entries of $\widehat{b}_h$ are available because $K_{h+1} + 4 = K_h$. The last moment update uses $\widehat{B}_{11}$ to obtain $\widehat{M}_{12}$; at height twelve we separately compute $\widehat{B}_{12}$ from $P_{12}(0), \ldots, P_{12}(3)$.

(d) Prove That Every Update Preserves Upper Bounds. We prove by induction on h that

$$P_h(k) \ge p_h(k) \quad (0 \le k \le K_h), \qquad \widehat{M}_h \ge M_h, \qquad \widehat{B}_h \ge B_h.$$

The initial probability and moment bounds are equalities because $Q_0 = 0$. If the probability bounds hold at height h, nonnegativity of the entries gives

$$\widehat{b}_h(j) = \sum_{i=0}^{j} P_h(i) P_h(j-i) \ge \sum_{i=0}^{j} p_h(i) p_h(j-i) = b_h(j).$$

Since $c(j) \ge 0$ by (32), multiplying these inequalities by $c(j)$ and summing for $j = 0, \ldots, 3$ gives $\widehat{B}_h \ge B_h$ by (37). The weights in the probability updates are also nonnegative: $1 - 2^{-(5-j)} \ge 0$ for

$0 \le j \le 4$, and $2^{-(k+5-j)} > 0$. Therefore, replacing b_h by $\widehat{b}_h$ in (35)–(36) and rounding upward gives $P_{h+1}(k) \ge p_{h+1}(k)$ for every retained count. For the moment, upward rounding and $a > 0$ give

$$\widehat{M}_{h+1} \ge a\widehat{M}_h^2 + \widehat{B}_h \ge aM_h^2 + B_h = M_{h+1},$$

where the last equality is (33). This completes the induction. The entries $P_h(k)$ are individual upper bounds, so their sum need not be one; we keep them as computed without rescaling.

Step 2: Verify $M_{12} < 111/100 < r$ and $B_{12} < b$. We first bound M_{12} using the probability bounds $P_h(k)$ and the moment updates from Step 1(c). Starting from $\widehat{M}_0 = 1$, we compute

$$\begin{aligned}\widehat{B}_h &= \sum_{j=0}^{3} \left(\sum_{i=0}^{j} P_h(i) P_h(j-i) \right) c(j), \\ \widehat{M}_{h+1} &= \mathrm{up}(a\widehat{M}_h^2 + \widehat{B}_h), \qquad h = 0, \ldots, 11,\end{aligned}$$

where $\mathrm{up}(x) = \lceil Dx \rceil / D$ and $D = 2^{80}$. We evaluate these expressions using exact rational arithmetic. At height eleven, the computation gives

$$\begin{aligned}\widehat{M}_{11} &= \frac{666283210292802720039869}{604462909807314587353088}, \\ \widehat{B}_{11} &= \frac{161791715968231228158497452161530781937571615184757 23}{280725234498519832528563782668143642415511522854830 08}.\end{aligned}$$

Substituting these values into the final moment update gives

$$\begin{aligned}\widehat{M}_{12} &= \mathrm{up}(a\widehat{M}_{11}^2 + \widehat{B}_{11}) \\ &= \frac{13340030971236116272667 01}{12089258196146291747061 76} < \frac{111}{100}.\end{aligned} \tag{38}$$

By Step 1(d), this proves $M_{12} \le \widehat{M}_{12} < 111/100 < r$.

We next bound B_{12} using the four terminal probability bounds. With the notation $P_h(k)$ defined in Step 1(c), we write $d_j = P_{12}(j) \ge \mathbb{P}[Q_{12} = j]$, $j = 0, 1, 2, 3$. Since we store each d_j as an integer multiple of $1/D$, we specify these four bounds by the integers Dd_j. The probability updates for $h = 0, \ldots, 11$ give

$$\begin{aligned} Dd_0 &= 1145132569100065099651614, & Dd_1 &= 27984400913325093562021, \\ Dd_2 &= 15636199646813528063096, & Dd_3 &= 8768468384720531179657. \end{aligned}$$

From these four probability bounds, we compute $\widehat{b}_{12}(j) = \sum_{i=0}^{j} d_i d_{j-i}$ for $j = 0, \ldots, 3$. We then substitute the resulting child-sum bounds into the formula for $\widehat{B}_h$ in Step 1(c), obtaining

$$\begin{aligned}\widehat{B}_{12} &= \sum_{j=0}^{3} \left(\sum_{i=0}^{j} d_i d_{j-i} \right) c(j) \\ &= \frac{10109712877157324344889325653889852625164259505088 27}{17545327156157489533035236416758977650969470178426 88} \\ &< \frac{57621}{100000}.\end{aligned} \tag{39}$$

Combining Step 1(d) with (38)–(39) proves the first two conditions:[4]

$$M_{12} \le \widehat{M}_{12} < \frac{111}{100} < r, \qquad B_{12} \le \widehat{B}_{12} < b.$$

Step 3: Verify $ar^2 + b < r$ by Direct Substitution. Substitute $a = 3125/7203$, $r = 23/20$, and $b = 57621/100000$ to obtain

$$r - ar^2 - b = \frac{19687}{720300000} > 0,$$

which is equivalent to $ar^2 + b < r$. Thus all three inequalities in (34) hold. □

Lemma 4.21 verifies all three conditions of Lemma 4.20 with $r = 23/20$. Hence $M_h \le 23/20$ for every integer $h \ge 0$, which completes this proof. □

5 Failure of a Uniform Exponential Stash Tail Bound for $Z = 3$

We consider the binary Path ORAM design $\mathcal{P}_{3,N}$ of Section 2.1, with $N = 2^L$ and tree depth L. We show that capacity three does not admit the uniform exponential stash tail bound of Definition 2.1. To disprove this guarantee, it suffices to exhibit one family of allowed instances whose stash sizes violate every proposed pair of constants. We use the full scan $\alpha = (1, \dots, N)$, starting from the empty tree and stash, and write $\pi = \mathcal{P}_{3,N}(N, \alpha)$ for its random execution. All initial and newly assigned labels are independent and uniform, and $S(\pi; N)$ is measured after the final refill.

Theorem 1.3 (Insufficiency of Capacity Three, Restated)**.** *For every integer $L \ge 49$, let $N = 2^L$, let $\alpha = (1, \dots, N)$, and let $\pi = \mathcal{P}_{3,N}(N, \alpha)$. Then*

$$\mathbb{E}[S(\pi; N)] > \frac{N}{2^{18}} + 3, \qquad \mathbb{P}[S(\pi; N) > N/2^{19}] > \frac{1}{2^{19} - 1}. \tag{4}$$

Consequently, the family $\{\mathcal{P}_{3,N} : N \ge 1\}$ does not admit a uniform exponential stash tail bound.

Theorem 1.3 gives a fixed positive probability of exceeding a stash budget proportional to the database size. A uniform exponential bound would instead make that probability tend to zero as N grows. The constants are chosen to leave explicit margins in the proof, rather than to estimate failure probabilities for small databases.

5.1 Proof Organization

Section 5.2 reduces Theorem 1.3 to a lower bound on the expected overflow from each region, where a region is a subtree rooted at a chosen depth. It defines the local quantities, states the bound in Proposition 5.3, and derives the theorem by subtracting the storage available above these regions.

Section 5.3 proves the local bound by transferring a surplus from the independent geometric-load tree to an actual scan region. Lemma 5.4 states the transfer with two explicit losses, one for a finite

[4]For independent verification, the certificate package (v1.0) contains `verify_certificates.py`, `certificates.json`, and a README with reproduction instructions. The script recomputes the rational updates in Algorithm 1 and checks the supplied table without overwriting it. The `z4` section of the JSON file records the retained values $P_h(k)$, together with $\widehat{M}_h$ and $\widehat{B}_h$, for all thirteen heights from zero to twelve. Both strict comparisons follow by integer cross multiplication because the denominators are positive. The upper bounds follow from the specified updates and the induction in Step 1(d).

observation window and one for discrete scan rounds. The section proves both estimates and checks that the chosen parameters leave the required surplus.

Section 5.4 supplies the finite geometric-tree surplus used in this transfer. Its integer calculation maintains a lower bound on the capped expected overflow and verifies Lemma 5.5, completing the local premise. The common packing and label facts used throughout are proved for general capacities in Appendix A.

5.2 Formal Proof of Theorem 1.3

We first specify the target representation at capacity three and define the local overflow quantities used in the proof. A deterministic storage inequality will combine these quantities across regions. We then state the local expectation bound and show that it implies Theorem 1.3. The later subsections establish this bound, including its finite calculation and both losses in transferring it to a real scan.

5.2.1 Target Loads and Upward Packing

The target definition (19) depends on the access sequence and leaf labels, but not on the bucket capacity. For an execution at capacity Z, let $A_v(t)$ count the blocks with target v after t accesses, and write $A(t) = (A_v(t))_v$. As in the preceding chapter, nodes are binary strings, ε is the root, and $|v|$ is the depth of node v. For any nonnegative node-load vector A, define upward packing with capacity Z by

$$D_v^{(Z)}(A) = \left(A_v + D_{v0}^{(Z)}(A) + D_{v1}^{(Z)}(A) - Z\right)^+, \qquad D^{(Z)}(A) = D_\varepsilon^{(Z)}(A), \tag{40}$$

where $x^+ = \max\{x, 0\}$ and missing children contribute zero. The superscript Z records the capacity. In particular, $D^{(3)}(A)$ is the number of blocks passed above the root when every bucket retains at most three blocks.

We use the following general-capacity facts from Appendix A.

Lemma 5.1 (Packing Identity and Monotonicity)**.** *For every positive integer Z, every instance (s, α) of $\mathcal{P}_{Z,N}$, and every realization of its execution π,*

$$S(\pi; t) = D^{(Z)}(A(t)) \qquad (0 \le t \le s).$$

For fixed target loads, removing targets or increasing bucket capacity cannot increase root overflow.

Proof. The identity is Lemma A.1 in Appendix A.2. The monotonicity statements follow from Lemma A.4 in the same appendix. □

For the full scan, let $X_1, \ldots, X_N$ be its old path labels in reverse chronological order, and let $Y_1, \ldots, Y_N$ be its final block labels in reverse access order. Lemma A.5 in Appendix A.3 gives joint independence and uniformity of these $2N$ labels. By the exact scan representation in Lemma A.6, the target of the ith block in reverse access order is

$$\delta_i = \text{the deepest node among } \{\mathrm{lcp}(Y_i, X_j) : 1 \le j \le i\}. \tag{41}$$

Here lcp denotes the longest common prefix, as in Section 4.3.1. The representation includes X_i because a block's own access assigns its final label before refilling the old path. These label and target statements hold at capacity three without using the capacity-four probability bound.

5.2.2 Aggregating Overflow from Actual Scan Regions

A region is the full physical subtree rooted at a chosen node. Blocks passed above several disjoint regions may still occupy their ancestor buckets. The next lemma subtracts all of that available space before counting any contribution to the stash.

Lemma 5.2 (Aggregating Local Overflow)**.** *Fix an instance (s, α), the execution $\pi = \mathcal{P}_{Z,N}(s, \alpha)$, and a depth d in its tree. For each node v at depth d, retain any subset of targets in its subtree, and let W_v be the root overflow of this retained field. For any quantities $0 \le F_v \le W_v$,*

$$S(\pi; s) \ge \sum_{|v|=d} F_v - Z(2^d - 1). \tag{42}$$

The inequality holds for each realization, and hence also for random choices of the retained fields and the F_v.

Proof. Discard all targets not retained. By Lemma 5.1, removing targets cannot increase overflow. The same lemma identifies the original field's overflow with $S(\pi; s)$. The disjoint subtrees rooted at depth d send at least $\sum_{|v|=d} F_v$ blocks toward their ancestors. Above those roots are exactly $2^d - 1$ buckets, each holding at most Z blocks. Equivalently, applying $(x - Z)^+ \ge x - Z$ at each ancestor subtracts their capacity once and gives (42). If the right-hand side is negative, the inequality still holds because the stash count is nonnegative. No independence between the subtrees is used. □

For the full scan $\pi = \mathcal{P}_{3,N}(N, (1, \dots, N))$, we now retain only a specified part of each region's target field. The depth d chooses the region roots, H is the number of bucket levels retained below each root, and n is the number of reverse rounds observed. We cap the local overflow at K to control the loss caused by comparison errors: two quantities in $[0, K]$ that agree except with probability δ differ in expectation by at most $K\delta$. This analytical cap does not change the ORAM execution.

Take integer parameters satisfying

$$H \ge 1, \qquad K \ge 1, \qquad 0 \le d \le L - H, \qquad 1 \le n \le N. \tag{43}$$

For each node v at depth d, retain the targets of the first n blocks in reverse scan order only when their relative depths below v are $0, \dots, H - 1$. Write A_v^{disc} for this vector of target counts, and set

$$F_v = \min\{K, D^{(3)}(A_v^{\mathrm{disc}})\}. \tag{44}$$

The overflow here is evaluated on the retained H-level local tree. Its bottom buckets need not be leaves of the physical tree. After all other targets are removed, the nodes below these H levels are empty, so packing the local tree gives the same overflow as packing the whole region. Removing targets and capping can only reduce overflow, by Lemma 5.1. Thus Lemma 5.2 applies with this F_v.

The targets retained in A_v^{disc} are actual final scan targets. Equation (41) shows that a block among the first n reverse rounds uses only old paths from those rounds. No earlier path is needed to determine such a block's target.

The following proposition provides more local expected overflow than the ancestors can absorb on average. We prove it in Section 5.3.

Proposition 5.3 (Surplus in a Real Scan Region)**.** *Let $N = 2^L$ and $\pi = \mathcal{P}_{3,N}(N, (1, \dots, N))$. Choose $H = 14$, $K = 32$, and integers d, n satisfying*

$$d \ge 30, \qquad d + 19 \le L, \qquad n = 2^{d+19}. \tag{45}$$

For every node v at depth d, the quantity in (44) satisfies $\mathbb{E}[F_v] > 5$.

The proposition applies to each region separately. Their actual loads may be dependent, and the local constructions proving their bounds need not all agree with the scan on the same event. The theorem uses only their expectations.

Proof of Theorem 1.3. Apply Lemma 5.2 and take expectations. Proposition 5.3 gives

$$\mathbb{E}[S(\pi; N)] \geq \sum_{|v|=d} \mathbb{E}[F_v] - 3(2^d - 1) > 2^{d+1} + 3. \tag{46}$$

Choose $L \geq 49$ and $d = L - 19$. Then $d \geq 30$ and $n = 2^{d+19} = N$, so every condition in (45) holds. The number of regions is a fixed fraction of N, and (46) becomes

$$\mathbb{E}[S(\pi; N)] > \frac{N}{2^{18}} + 3.$$

A linear expectation alone might come from rare, extremely large values. Here $S(\pi; N) \leq N$: the stash cannot contain more blocks than the database. Put $R = 2^d = N/2^{19}$. On the event $S(\pi; N) \leq R$, bound the stash by R; on its complement, bound it by N. Averaging these bounds gives

$$\mathbb{E}[S(\pi; N)] \leq R + (N - R)\mathbb{P}[S(\pi; N) > R].$$

Combining the last two inequalities yields

$$\mathbb{P}[S(\pi; N) > N/2^{19}] > \frac{N/2^{19} + 3}{N - N/2^{19}} > \frac{1}{2^{19} - 1}. \tag{47}$$

This probability lower bound is a positive constant independent of N.

The scan family contradicts every proposed uniform exponential bound at capacity three. For any fixed $C, c > 0$, such a tail at $R = N/2^{19}$ would be $Ce^{-cN/2^{19}}$, which tends to zero as $N = 2^L$ grows. It eventually falls below (47). Thus no such pair of uniform constants exists. □

5.3 Proof of Proposition 5.3

Proposition 5.3 (Surplus in a Real Scan Region)**.** *Let $N = 2^L$ and $\pi = \mathcal{P}_{3,N}(N, (1, \ldots, N))$. Choose $H = 14$, $K = 32$, and integers d, n satisfying*

$$d \geq 30, \qquad d + 19 \leq L, \qquad n = 2^{d+19}.$$

For every node v at depth d, the quantity in (44) *satisfies $\mathbb{E}[F_v] > 5$.*

We prove the proposition by transferring a finite geometric-tree surplus through a continuous local model. The transfer has two explicit errors: the loss from observing only a finite time window and the loss from replacing continuous arrivals by discrete scan rounds. We first state the transfer and the finite surplus, then check the parameters that yield the proposition.

5.3.1 A Geometric Surplus and the Losses in Transferring It

We use the independent geometric-load tree of Definition 4.2, now packed with capacity three. At every node place an independent load G with $\mathbb{P}[G = k] = 2^{-k-1}$ for integers $k \geq 0$. Let $Q_h^{(3)}$ be the number of blocks passed above the root of this tree with h bucket levels, and set $Q_0^{(3)} = 0$ for

the empty tree. The superscript records the capacity. If $Q_{h,0}^{(3)}$ and $Q_{h,1}^{(3)}$ denote independent copies corresponding to the two child subtrees, then

$$Q_{h+1}^{(3)} \stackrel{\mathrm{d}}{=} \left(G + Q_{h,0}^{(3)} + Q_{h,1}^{(3)} - 3\right)^{+}, \tag{48}$$

where G, $Q_{h,0}^{(3)}$, and $Q_{h,1}^{(3)}$ are independent.

For a real region rooted at depth d, put $q = 2^{-d}$. Both an old path and a final label enter the region with probability q per scan round. Assigning duration q to each round gives the continuous observation time $T = nq$. The next lemma quantifies how much geometric surplus remains after the two transfers to this real region.

Lemma 5.4 (Local Comparison with Two Explicit Losses)**.** *For the full scan and parameters in* (43)*, every node v at depth d satisfies*

$$\mathbb{E}[F_v] \geq \mathbb{E}[\min\{K, Q_H^{(3)}\}] - K2^H e^{-T/2^H} - 3KTq, \qquad q = 2^{-d}, \quad T = nq. \tag{49}$$

The first term is the geometric surplus. The second pays for observing it only until time T, and the third pays for the discrete scan rounds. A longer window reduces the time loss but increases the discrete-round loss. A deeper region makes the latter loss smaller, but its local tree and the required number of rounds must still fit in the actual scan. We will keep all three requirements explicit.

An exact finite calculation supplies the surplus with enough room for both losses.

Lemma 5.5 (Finite Local Surplus)**.** *For a capacity-three geometric tree with fourteen bucket levels,*

$$\mathbb{E}[\min\{32, Q_{14}^{(3)}\}] > \frac{11}{2}. \tag{50}$$

Lemma 5.4 is proved in the next three subsubsections. Section 5.4 proves Lemma 5.5 by an integer calculation that underestimates the capped expectation. Together they imply the proposition.

Proof of Proposition 5.3. The following table collects the choices and the reason for each one. This is also the check that the local comparison fits inside the real tree and scan.

Table 1: Parameter choices for the local comparison.

Quantity	Choice	Proof obligation
Local levels	$H = 14$	The finite certificate gives surplus on these bucket levels.
Overflow cap	$K = 32$	Limits expectation loss when the fields disagree.
Time window	$T = 2^{19}$	Makes the probability of unseen mark categories small.
Root depth	$d \geq 30$	Makes local visits rare enough for a small coupling loss.
Available depth	$L - d \geq 19$	Allows n rounds and at least H bits below each root.
Scan window	$n = T/q = 2^{d+19}$	Gives $n \leq N$ and the required continuous duration.

First, the time window makes the continuous-time loss small. Each of the 2^H finest categories has mean $T/2^H = 32$ old-path arrivals by time T. Using $e > 2$ gives

$$32\, 2^{14} e^{-32} < 2^{-13} = \frac{1}{8192}. \tag{51}$$

Second, the discrete-round loss in Lemma 5.4 is $3KTq = 96Tq$, and $d \geq 30$ gives

$$96Tq = 96\,2^{19-d} \leq \frac{3}{64}. \tag{52}$$

The geometric conditions make both comparisons legitimate for the real scan. The inequality $L - d \geq 19$ gives $n = 2^{d+19} \leq 2^L = N$. It also gives $L - d \geq H = 14$, so all the local mark categories exist. The choice $d = L - 19$ used in the theorem satisfies every condition when $L \geq 49$, uses $n = N$ rounds, and provides $2^d = N/2^{19}$ regions. Thus the same choice supplies both a valid local window and a linear number of regions for aggregation.

Apply Lemma 5.4 with the parameters just verified. Lemma 5.5 supplies a geometric mean greater than $11/2$. Equations (51) and (52) bound the two losses, giving

$$\mathbb{E}[F_v] > \frac{11}{2} - \frac{1}{8192} - \frac{3}{64} = \frac{44671}{8192} > 5. \tag{53}$$

This proves the required expectation bound in every real region. □

5.3.2 Realizing the Geometric Field by Continuous Arrivals

We construct a local process whose entire retained count field, over unlimited time, has the independent geometric law at all H retained bucket levels. This will let us apply the finite surplus lemma to that field. The next subsubsection bounds the loss from a finite observation window.

The construction separates old paths from final labels into two streams of random arrivals. Fix the number H of retained bucket levels from Lemma 5.4. Use two independent rate-one Poisson processes, called X and Y. A rate-one process has a Poisson number of arrivals with mean t in an interval of length t, and counts in disjoint intervals are independent. Each arrival also carries an independent uniform H-bit string, called its mark, specifying one of the 2^H local leaf categories. The X marks represent old paths; the Y marks represent final labels. Their order will play the role of reverse scan order.

We assign a target to a Y arrival by comparing its mark with the X marks already seen. Its target is their deepest common-prefix match, as in the scan representation. If no X has arrived, there is no local target. If an X mark agrees in all H bits, the deepest match has length at least H, and we discard that target as well. Thus only nodes at relative depths $0, \ldots, H-1$ receive retained targets. The H-bit categories at relative depth H serve only to determine first arrivals; the retained buckets are their ancestors at depths $0, \ldots, H-1$.

A node receives targets during a specific waiting interval. For the root, imagine that an X mark has entered the left child but none has entered the right child. A Y mark in the right child then has its deepest match at the root. A Y mark in the left child matches farther down; after the first X mark in the right child, every Y mark matches farther down too. The same reasoning applies at any retained node w. Let a_w and b_w be the earlier and later first-arrival times in its two children, and put $p_w = 2^{-|w|-1}$, the probability of a mark falling in either specified child. The contributing Y points are exactly those in $[a_w, b_w)$ whose marks lie in the later-hit child. Before this interval neither child has an X match; after it both do. Endpoint conventions do not affect the law, since simultaneous arrivals have probability zero.

Lemma 5.6 (Joint Geometric Law)**.** *Over infinite time, the entire retained count field has independent coordinates with law $\mathbb{P}[G = k] = 2^{-k-1}$ for integers $k \geq 0$.*

Proof. First fix the whole X process, so that every node's contributing region of times and marks is known. These regions are disjoint: each Y point has at most one target. They depend only on X, which is independent of Y. Consequently their counts are conditionally independent Poisson variables with means

$$\Lambda_w = p_w(b_w - a_w).$$

The mean is the child arrival rate p_w multiplied by its waiting time $b_w - a_w$.

The conditional independence concerns regions of both time and mark, not necessarily disjoint intervals of time. To see it directly, split the mark space into its 2^H categories and the time axis at the finitely many child first-arrival times. Poisson marking and independent increments give independent counts in the resulting disjoint rectangles [LP18, Chapter 5]. Combining different rectangles into disjoint node regions preserves independence. This proves the conditional count description for the whole field at once.

We next identify the joint distribution of the random means. The first X arrivals in the 2^H finest categories are independent rate-2^{-H} exponentials. Apply Lemma 4.16 to the complete local tree with $\ell = H$ and leaf rate $\rho = 2^{-H}$. At a retained node w, its child rate is $\rho 2^{H-|w|-1} = p_w$, so the lemma's scaled gap is exactly Λ_w. The lemma proves that all these means are jointly independent Exp(1) variables. The full induction is needed here: nested nodes share X arrivals, and the individual exponential laws of their means do not imply independence.

The conditional count law given X depends on X only through this mean vector, so the same product law holds when we condition on the mean vector alone. These are the two hypotheses of Lemma 4.18. Given the mean vector, the node counts are conditionally independent Poisson variables; the means themselves are jointly independent Exp(1) variables. Their mixture therefore gives independent geometric counts with probabilities 2^{-k-1}. Both independence statements matter: the correct one-node distributions alone would not identify the joint overflow law. □

5.3.3 Retaining the Surplus in a Finite Window

The exact geometric law concerns unlimited time, whereas a real scan contains finitely many rounds. We therefore need a separate bound on how much overflow remains unseen when the continuous process stops at time T. The cap K makes this possible using only the probability that the retained field has not yet stopped changing.

Let $A^{\mathrm{cont}}(T)$ count retained targets from Y points arriving by time T, and define

$$F^{\mathrm{cont}}(T) = \min\{K, D^{(3)}(A^{\mathrm{cont}}(T))\},$$

with overflow evaluated on the local H-level tree. The preceding lemma identifies the unlimited-time field with the independent geometric tree, so its capped overflow has the same law as $\min\{K, Q_H^{(3)}\}$. The finite-time field need not itself have independent geometric counts; we instead bound how much it can still be missing.

Lemma 5.7 (Finite-Time Loss)**.** *For every $T \geq 0$,*

$$\mathbb{E}[F^{\mathrm{cont}}(T)] \geq \mathbb{E}[F^{\mathrm{cont}}(\infty)] - K2^H e^{-T/2^H}. \tag{54}$$

Proof. The retained field stops changing once every finest X category has arrived. After that time, every future Y mark has an exact H-bit match and its target is discarded. Thus, if all categories have appeared by T, the capped overflow at T already equals its unlimited-time value. Each category has probability $e^{-T/2^H}$ of still being absent. A union bound over the 2^H categories bounds the

chance that any is missing, and the possible loss on that event is at most K. Taking expectations gives (54). All variables lie in $[0, K]$, so no uncapped expectation bound is needed. □

5.3.4 Transferring the Finite Window to Discrete Scan Rounds

We next compare the finite continuous window with the real local field in (44). A coupling constructs these two fields on one probability space while preserving each one's prescribed distribution. We will bound the chance that they differ and then use the cap to turn that bound into a loss in expectation.

A uniform old path enters the subtree of a depth-d node with probability $q = 2^{-d}$ in each scan round. The final block label enters with the same probability. Assign duration q to each round, so that a rate-one continuous process has the same expected number of arrivals in that interval. The n rounds in the local definition then occupy a window of duration $T = nq$. This time scaling matches the rates; it does not yet match event counts or their order.

Only local entry events, marks, and their order determine this retained field. By (41), reverse block i targets the deepest common prefix of Y_i with $X_1, \ldots, X_i$, including the current X_i. In each round, entry of X_i and entry of Y_i into the subtree are independent Bernoulli events of probability q: each either occurs once or does not occur. These events are also independent across rounds. Conditional on entry, the next H bits give uniform independent marks. An old path outside v cannot match a label inside v at or below v, and a label outside v cannot target a node inside it. Hence ignoring such outside events loses no information needed for the retained local field.

Lemma 5.8 (Finite-Window Coupling)**.** *The fields A_v^{disc} and $A^{\mathrm{cont}}(T)$ admit a coupling such that*

$$\mathbb{P}[A_v^{\mathrm{disc}} \neq A^{\mathrm{cont}}(T)] \leq 3Tq. \tag{55}$$

For any common function f of these fields taking values in $[0, K]$,

$$\left|\mathbb{E}[f(A_v^{\mathrm{disc}})] - \mathbb{E}[f(A^{\mathrm{cont}}(T))]\right| \leq 3KTq.$$

Proof. We first match the numbers of local events in a single round. Divide $[0, T]$ into $n = T/q$ intervals of length q. Each continuous process has independent Poisson(q) counts in these intervals, whereas each discrete entry count is Bernoulli(q). Their distributions are close because a short Poisson interval rarely contains two or more points. The *total variation distance* between two laws is the largest difference between the probabilities they assign to the same event. For $0 \leq q \leq 1$, the distance here is

$$d_{\mathrm{TV}}(\mathrm{Bernoulli}(q), \mathrm{Poisson}(q)) = q(1 - e^{-q}) \leq q^2. \tag{56}$$

Indeed, 1 is the only value with greater Bernoulli mass than Poisson mass, and the difference there is $q - qe^{-q}$. At zero the Bernoulli mass $1 - q$ is at most e^{-q}; at all values above one it is zero.

The total variation bound gives a concrete way to match the counts. For each possible count, place as much common probability mass as the two distributions allow on equal outcomes, then assign the remaining mass to their unmatched outcomes. This is a maximal coupling: its mismatch probability equals the total variation distance, here at most q^2. Use independent copies of this construction for every round and for the X and Y processes. When the paired counts both equal one, give the two points the same uniform mark. Sample all other counts and marks according to their prescribed conditional laws, and give continuous points independent uniform times within their intervals. Independent Poisson counts with these conditional locations reconstruct the two independent continuous Poisson processes with their correct laws [LP18, Chapter 3]. The discrete entry events retain their required laws too.

Matching counts alone does not always match event order. In one round, a mismatch in either count has probability at most $2q^2$. Also declare the round bad if both discrete memberships equal one, which has probability q^2. In that case the discrete target rule uses X_i before placing Y_i, but the continuous points could arrive in the opposite order. Declaring the whole case bad avoids claiming agreement when a newly arrived Y would see different preceding X marks.

If no round is bad, both constructions have exactly the same retained field. Every interval then has at most one local event in total; matching counts and marks therefore match the full order of all X and Y events. The target of each Y point is determined by that information. A union bound over the rounds gives $3nq^2 = 3Tq$, proving (55). This compares the two unconditional marginal distributions. We do not condition on coupling success, which could change the geometric law.

On agreement the two evaluations of f coincide; on disagreement they can differ by at most K. Thus the expected absolute difference is at most $3KTq$, proving the second assertion. □

The two losses prove the parameterized comparison.

Proof of Lemma 5.4. The conditions $H \le L - d$ and $n \le N$ ensure that the local marks and rounds used by the construction exist in the actual scan. Lemma 5.6 gives $\mathbb{E}[F^{\mathrm{cont}}(\infty)] = \mathbb{E}[\min\{K, Q_H^{(3)}\}]$. Lemma 5.7 bounds the loss on stopping at time T by $K2^H e^{-T/2^H}$. Applying Lemma 5.8 to the function $f(A) = \min\{K, D^{(3)}(A)\}$ bounds the further loss on passing to the real discrete field by $3KTq$. Combining the two lower inequalities gives (49). The construction preserves the unconditional law on each side; no conditioning on the coupling's success is used. □

5.4 Exact Finite Verification of the Local Surplus

Lemma 5.5 (Finite Local Surplus)**.** *For a capacity-three geometric tree with fourteen bucket levels,*

$$\mathbb{E}[\min\{32, Q_{14}^{(3)}\}] > \frac{11}{2}.$$

Proof. The proof has three steps. We construct finite distributions that underestimate the capped overflow, compute their updates with integer arithmetic, and verify that the resulting mean at height fourteen exceeds 11/2. This lower bound then proves the lemma.

Step 1: Construct Finite Distributions That Underestimate Overflow. A numerical estimate would not suffice: we need a calculation that is guaranteed to underestimate the capped geometric overflow. We therefore construct finite distributions that are smaller in the following sense: for every integer threshold $k \ge 1$, their probability of being at least k is no greater than the true probability. This stochastic lower bound also makes their expectations lower bounds.

Let μ_h be the computed distribution on $\{0, \dots, 32\}$ at height h, starting with $\mu_0 = \delta_0$, the distribution concentrated at zero. To update it, draw two independent children with law μ_h and use the clipped input $\min\{G, 35\}$. Apply the capacity-three overflow recursion and cap the output at 32. The input cutoff 35 is sufficient because an input of 35 already leaves at least 32 after subtracting three; a larger input cannot change the capped output. Thus every update can be represented by finitely many probabilities.

We also keep the stored probabilities on a finite rational grid without turning a lower bound into an upper bound. Set $D = 2^{48}$, round each positive-value probability down to a multiple of $1/D$, and place all lost mass at zero. For example, any mass removed from value seven is reassigned to zero; no event of the form "the value is at least a positive threshold" can become more likely. This preserves total probability one while decreasing every positive tail.

These operations maintain the stochastic lower bound by induction. Suppose μ_h is smaller than $\min\{32, Q_h^{(3)}\}$ in the stated sense. The two variables can be sampled from the same uniform number using their inverse cumulative distributions; the smaller distribution then always gives the smaller value. Use independent uniform numbers for the two children, so the child copies remain independent. Clipping the geometric input decreases it, and the overflow recursion and output cap are increasing functions of their inputs. The unrounded next output is therefore no larger, in this joint construction, than $\min\{32, Q_{h+1}^{(3)}\}$. Finally the downward rounding and reassignment to zero can only reduce its tails further. Starting from the equal zero distributions proves the required bound at every computed height.

Step 2: Compute the Updates with Integer Arithmetic. We now specify each update entirely with integers. Let p_i be the stored mass numerator at value i, so its probability is p_i/D and $\sum_{i=0}^{32} p_i = D$. For a threshold $k \in \{1, \dots, 32\}$ and child values i, j, the capped output is at least k precisely when $G \geq \max\{0, k+3-i-j\}$. This threshold is at most 35, so clipping G at 35 does not change the event. The geometric survival probability at a nonnegative threshold m is 2^{-m}. Multiplying by the child probabilities and using the common denominator $D^2 2^{35}$ gives the tail numerator

$$T_k = \sum_{i,j=0}^{32} p_i p_j \, 2^{35-\max\{0,k+3-i-j\}}, \qquad T_{33} = 0. \tag{57}$$

The separate convention $T_{33} = 0$ expresses the output cap: an output of at least 33 is impossible.

Differences of these tails recover the individual positive masses. Before rounding, the mass at k is $(T_k - T_{k+1})/(D^2 2^{35})$. Multiplying by D and taking the integer floor rounds that mass down to the $1/D$ grid. Assigning all remaining mass to zero gives

$$p'_k = \left\lfloor \frac{T_k - T_{k+1}}{D 2^{35}} \right\rfloor \quad (1 \leq k \leq 32), \qquad p'_0 = D - \sum_{k=1}^{32} p'_k. \tag{58}$$

These two formulas, together with the initial point mass, specify the whole finite certificate.

Step 3: Certify the Surplus at Height Fourteen. Apply (57)–(58) fourteen times to $(D, 0, \dots, 0)$. The resulting lower distribution has mean

$$\sum_{k=0}^{32} k \mu_{14}(k) = \frac{775231701030647}{140737488355328} > \frac{11}{2}.$$

The comparison above bounds this mean by $\mathbb{E}[\min\{32, Q_{14}^{(3)}\}]$. This proves the lemma with the required direction. All quantities in the calculation are integers or exact rational numbers; the strict inequality is not based on sampling or floating-point output.

The certificate package (v1.0) provides the verifier, exact tables, and reproduction instructions for this calculation, as described in Appendix B. Algorithm 2 lists the updates. The accompanying verifier computes them from both the tail recurrence and a direct probability-mass convolution, and checks that all 15 rows, including the initial row, agree. Agreement checks the implementation of the finite update; the stochastic comparison above is what proves that its result is a lower bound. □

6 Related Work

The closest prior results differ in the process they analyze, the states covered by their guarantee, or the loads supplied to their probability model. We organize the comparison around these three

differences. First we discuss the capacity question for ordinary Path ORAM and the scope of its experimental evidence. We then compare guarantees for modified eviction rules. Finally, we distinguish the inherited representations and probability tools from the connections established here.

Table 2 places the closest capacity results side by side. The operation or state being analyzed is part of each result; the integer capacity alone does not determine which theorem applies.

Table 2: Closest capacity guarantees and the processes they cover.

Result	Capacity	Operation or state covered
Path ORAM [SvDS+18]	$Z = 5$	Uniform exponential stash tail bound after ordinary accesses, with the theorem's tree-depth choice $L = \lceil \log_2 N \rceil$.
This paper	$Z = 4$; $Z = 3$	Uniform exponential bound at four for the design $\mathcal{P}_{Z,N}$ at depth $L = \lceil \log_2 N \rceil$; a first-scan linear obstruction at three with $N = 2^L$.
Circuit ORAM [WCS15]	$Z = 4$	Exponential tail for the proof variant with two scheduled eviction paths in bit-reversal order and an additional partial eviction on the read path.

6.1 The Ordinary Process and Its Capacity Question

Path ORAM combines a simple tree structure with a rigorous stash analysis. Stefanov, van Dijk, Shi, Chan, Fletcher, Ren, Yu, and Devadas [SvDS+18] establish the uniform exponential stash tail bound that is the direct theoretical starting point of our work. Path ORAM appeared at CCS 2013 [SvDS+13]; its journal analysis proves $\mathbb{P}[S(\mathcal{P}_{5,N}(s,\alpha);s) > R] \leq 14(0.6)^R$ for every fixed instance (s,α) at capacity five [SvDS+18, Theorem 5.1]. Shi's Path Oblivious Heap [Shi20] extends the tree-based approach to an oblivious priority queue, combining asymptotically optimal bandwidth with strong practical performance. This paper also identifies tighter bucket constants as a continuing challenge [Shi20, Section 3.1, Remark 4]. Our main theorem settles capacities three and four for the process defined in Section 2. To our knowledge, earlier work has neither established a uniform exponential bound at capacity four for this process nor proved a linear expected stash at capacity three after a fixed scan.

The experiments give evidence about particular executions and sizes. Path ORAM reports small stashes at four, while early secure-processor experiments find substantially poorer behavior at three [SvDS+18, Section 7],[RYF+13, Section 2.5.1]. Neither observation determines whether the same uniform exponential bound holds for every allowed size and fixed address sequence.

The ratio between blocks and leaves is also part of the comparison. The journal theorem uses $L = \lceil \log_2 N \rceil$, whereas its experiments use $L = \lceil \log_2 N \rceil - 1$ [SvDS+18, Sections 3.2 and 7]. For power-of-two N, those experiments therefore place twice as many logical blocks as leaves. Our upper theorem uses $L = \lceil \log_2 N \rceil$, so $N/2^L \leq 1$. For example, at $N = 8$ the experimental choice gives four leaves, whereas our theorem requires at least eight. Our result settles the stated capacity question in the theorem's load range; it does not prove the same guarantee in that denser experimental setting. The lower result uses $N = 2^L$ and conservative asymptotic constants, so it does not estimate failure at the experimental sizes.

6.2 Different Eviction Rules

A small-capacity theorem must be read together with its placement rule. An *eviction* moves blocks toward available tree buckets; changing its schedule changes the opportunities that blocks receive. Wang, Chan, and Shi's Circuit ORAM [WCS15] achieves nearly optimal circuit size, making it particularly useful for secure computation. For its execution π^{circ} of s operations, Circuit ORAM proves $\mathbb{P}[S(\pi^{\mathrm{circ}}; s) > R] \leq 14e^{-R}$ at capacity four for the proof variant that uses two deterministically selected eviction paths per operation, in bit-reversal order, and an additional partial eviction that fills empty slots on the read path [WCS15, Section 3.4, Theorem 2, and Appendices C.1 and D.2]. That schedule improves the boundary-load estimate enough for a sum over overloaded subtrees to converge. Our capacity-four theorem instead concerns the single old-path refill already performed by ordinary Path ORAM. Thus the two results have different operation rules even though they use the same integer bucket capacity.

Other tree ORAM designs also change the process that determines stash occupancy. Ring ORAM [RFK+15] separates online reads from scheduled evictions and provides extra dummy slots. Recursive-tree ORAM [MBN15] changes the tree construction and analyzes scheduled eviction. Background eviction gives blocks further placement opportunities beyond those supplied by ordinary requests [RYF+13]. These changes distinguish their guarantees from the process fixed in Section 2.

More recent systems combine such changes with other storage or operation choices. Radix Path [ASB19] enlarges the root bucket and uses two-way background eviction. AB-ORAM [RYTZ23] adjusts bucket storage within Ring ORAM. SONIC [TD26] builds concurrent ORAM and data structures around Ring ORAM, while the Onyx preprint [RWZ+26, Theorem 3 and Appendix E] analyzes stash overflow for a d-ary Ring ORAM, where d is the number of children of an internal node. Their schedules, capacities, or branching factors differ from ours. The capacity-three lower bound therefore does not rule out smaller buckets in these modified designs.

6.3 Inherited Representations and the New Comparisons

The infinite-bucket representation is an established foundation of Path ORAM analysis [SvDS+18, Section 5]. It first removes bucket-capacity limits and then restores them by greedy postprocessing. The same analysis characterizes the stash through an overloaded rooted subtree. Ring ORAM [RFK+15, Section 4.2 and Appendix B] and Circuit ORAM [WCS15, Appendix C] also prove equivalence between finite buckets and postprocessed infinite buckets. Our targets express this established representation directly in terms of each block's access history. The exact-count proof makes the common basis for our two comparisons explicit.

The original proof also reduces a fixed rooted subtree's usage to a scan of distinct addresses [SvDS+18, Lemma 5.6 and Remark 5.7]. We retain that worst-case-scan principle. Lemma 4.12 gives a construction using shared random labels that preserves the prescribed distributions and bounds the actual root overflow by one comparison field for every outcome. This root overflow is the quantity used by the later moment comparison. Moving from a fixed witness to a common whole-field comparison is the relevant distinction, not the idea of using a scan.

Several probability tools used after this representation are also established. Directional convexity is used to compare multivariate randomness and random measures [BY09]. For tree parking, Bahl, Barnet, and Junge [BBJ21, Theorem 1.7] show that increasing-convex order on i.i.d. input arrivals on a Galton–Watson tree is preserved by the total number of arrivals at the root. The relevant difference here is that many target counts depend on the same old paths. We prove the needed properties of the overflow cost and use conditional Poisson comparisons and random arrival times

to control that dependent field as a whole. The contribution is this comparison for the Path ORAM process, rather than a new general principle of convex ordering or Poissonization.

The independent-load model itself is a form of parking on a tree: each node keeps up to Z arrivals and passes the excess to its parent. Aldous, Contat, Curien, and Hénard [ACCH23] locate the phase transition for i.i.d. arrivals on the infinite binary tree with one slot per node. Chen and Contat [CC25] study one-slot parking on trees with geometric offspring. In that model the number of children is geometric; in our auxiliary model the tree is fixed and the number of arrivals at a node is geometric. The random object and node capacity must therefore be matched before using a parking theorem for this problem.

These comparisons identify three distinct roles in our proof. The target representation records an established deterministic packing principle. The global upper comparison supplies independent loads that bound a particular cost of the real process. The local lower comparison transfers a certified surplus back to actual scan regions. The last connection is necessary because a large overflow in an upper comparison model gives no lower bound for the real stash. Together, the two connections let the auxiliary tree analysis settle the ordinary process's capacity question.

Beyond stash reliability, ORAM research has addressed complementary questions about locality, parallelism, and bandwidth. Asharov, Chan, Nayak, Pass, Ren, and Shi [ACN+22] achieve polylogarithmic bandwidth and locality overhead while allowing leakage of the lengths of accessed contiguous regions. Boyle, Chung, and Pass [BCP16] extend oblivious computation to parallel RAM with polylogarithmic overhead in both total work and parallel running time. Hubáček, Koucký, Král, and Slívová [HKKS19] strengthen the logarithmic bandwidth lower bound for online ORAM to hold even when the server does not know the boundaries between logical operations.

7 Conclusion and Discussions

We have determined the minimum bucket capacity for a uniform exponential stash tail bound in the Path ORAM model of Section 2. Capacity four suffices for every database size and every fixed access sequence. Capacity three fails even for a full scan: the stash exceeds a fixed positive fraction of the database size with probability bounded away from zero as the database grows. Together, these results establish four as the optimal bucket capacity for this reliability requirement. For a finite execution at capacity four, the stash budget grows only logarithmically with the number of accesses and the inverse of the target failure probability, regardless of database size.

Our proofs use the upward-packing representation of stash overflow in two different ways. At capacity four, we introduce additional randomness to obtain independent geometric node loads without decreasing the root-overflow moment. Independence lets us bound this moment recursively, avoiding the enumeration of overloaded sets in the classical analysis. At capacity three, we transfer a local overflow surplus from the independent geometric model to disjoint regions of the actual full scan. The total expected overflow from these regions exceeds the storage available above them, forcing a linear expected stash. These arguments give both a uniform upper bound and an obstruction to such a bound within the same packing framework.

7.1 Further Questions

The capacity threshold is now settled under the specified model. The following questions concern sharper storage guarantees and changes to the access or storage rules.

Higher load. The upper comparison uses $N \leq 2^L$ when it adds blocks up to one per physical leaf

in the canonical scan and bounds the physical-leaf means. A denser tree changes these inputs to the comparison. Determining the smallest reliable capacity at larger ratios $N/2^L$, including the ratio used by the original experiments [SvDS+18, Sections 3.2 and 7.1], therefore requires an additional analysis. The capacity-three lower bound at $N = 2^L$ also does not establish the same behavior when substantially more leaves are provided.

Finite-size behavior. The lower theorem starts at $L \geq 49$ and uses conservative constants. Its scale requirement comes from fitting a sufficiently long local observation window inside the scan while making individual region visits rare enough for the coupling error to be small. Sharper finite-time and coupling estimates, or a different local construction, would be needed to turn this asymptotic obstruction into useful lower bounds at much smaller database sizes. The present theorem does not predict those smaller instances' failure probabilities.

Quantitative constants. The upper theorem gives an explicit exponential rate and prefactor, but does not claim that either is optimal. Improving the comparison or the all-height moment bound could give smaller certified budgets for a specified finite execution. Similarly, the lower proof's cutoff and error margins are sufficient choices rather than optimal estimates of the stash fraction. These quantitative questions are distinct from the settled integer threshold under the present assumptions.

Other access and storage rules. The independent label reindexing uses an address list fixed independently of the internal randomness. Address choices that react to the internal state require another argument. Extra eviction paths change the available opportunities and need their own process comparison. In particular, no new capacity theorem for full Ring ORAM [RFK+15] or Circuit ORAM [WCS15] follows from the results here. Path ORAM also supports a position map stored in recursive ORAM instances and a stash shared among those levels [SvDS+18, Sections 4 and 5.7]. Analyzing such a shared stash requires a joint argument beyond the single-instance process studied here. Each question changes an input to the proof, not just its numerical constants.

A General-Capacity Packing Facts

This appendix gives packing and label facts for every positive integer capacity and finite tree depth, including scans with fewer blocks than leaves.

The deterministic representation works for every positive integer Z and every finite tree depth. It describes the established infinite-bucket and greedy-postprocessing view [SvDS+18, RFK+15] directly from the access history. For the deterministic statements, let π be any empty-start unit-block execution of length s using the same access and refill rules on a binary tree of depth L, with an unbounded stash. For the main design, $\pi = \mathcal{P}_{Z,N}(s, \alpha)$ and $L = \lceil \log_2 N \rceil$. The first two subsections concern a particular execution, with all random choices already fixed.

A.1 From Placement Opportunities to Tree Overflow

We record each block's opportunities, then use these records to count overflow.

Recording one block's opportunities. A block's target records how far down the tree the path history has given it an opportunity to go. The block keeps its current leaf label from its most recent access until its next access. During that interval, each path read shares an initial part with the block's assigned path. The deepest node in this shared part is the deepest bucket on that read path in which the block is allowed to be placed. Other blocks may occupy the space, so a placement opportunity need not become an actual placement.

For example, suppose a block's current label is 010 and the paths read since its last access are 001

and 011. The first shares only the node 0 below the root with the assigned path; the second shares the deeper node 01. The deepest opportunity is therefore 01, which we call the target. Depending on competition for space, the block may occupy 01, 0, the root, or the stash.

To name the opportunities, write $P(y)$ for the path from the root to leaf y, and write y_b for block b's current leaf label. At access j, the client reads and refills the path with the old label x_j, while the accessed block receives a fresh label U_j. Only addresses already accessed have stored blocks. Their identifier set after t accesses is $I_t = \{a_1, \ldots, a_t\}$, with $I_0 = \varnothing$. Thus the targets below describe precisely the blocks that exist in the tree or stash at that time.

For a created block b, let $\tau_t(b)$ be its last access time at or before t. Write $\mathrm{lcp}(y, x)$ for the node given by the longest common prefix of two leaf labels: it is the deepest node shared by their paths. Thus $\mathrm{lcp}(010, 001) = 0$ and $\mathrm{lcp}(010, 011) = 01$. The target of b after access t is

$$d_t(b) = \text{the deepest node among } \{\mathrm{lcp}(y_b, x_j) : \tau_t(b) \le j \le t\}. \tag{59}$$

The interval begins with the block's own most recent access, because that access refills the old path after assigning the new label y_b. All candidate nodes lie on $P(y_b)$, so one of them is uniquely deepest. Targets depend on the labels and access history, not on bucket capacity. Let $A_v(t) = |\{b \in I_t : d_t(b) = v\}|$ count the blocks with target v.

Counting overflow from the records. The targets specify placement opportunities, but we still need to turn these records into a number of unplaced blocks. Consider the following auxiliary packing. Place every block at its target without imposing a capacity limit. Then work upward from the leaves: each bucket keeps up to Z blocks and sends any excess to its parent. For example, a bucket of capacity four that receives two local blocks and three excess blocks from its children keeps four and sends one upward. The number sent above the root is the overflow of this packing. This is an auxiliary count, not an additional operation performed by the client.

We name the nodes as in the storage setting. Let $\mathcal{V}_L$ be their set, let the empty string ε name the root, and let $|v|$ be the depth of node v. The leaves are the strings in $\{0,1\}^L$. Writing $v \preceq w$ means that v is a prefix of w, or equivalently that v lies on the path from the root to w. For example, $0 \preceq 01$. The children of v are $v0$ and $v1$ whenever they exist.

For any nonnegative load vector A, the packing is expressed by

$$D_v^{(Z)}(A) = \Big(A_v + D_{v0}^{(Z)}(A) + D_{v1}^{(Z)}(A) - Z\Big)^+, \qquad D^{(Z)}(A) = D_\varepsilon^{(Z)}(A), \tag{60}$$

where $x^+ = \max\{x, 0\}$ and missing children contribute zero. Here $D_v^{(Z)}(A)$ is the excess sent upward from node v. Its incoming load has three parts: the A_v blocks whose target is v and the excess from each of its two children. Subtracting Z accounts for the bucket's capacity; taking the positive part makes the excess zero when everything fits. For the actual process these loads are integers. The same formula also defines D for nonnegative real loads, an extension used only to justify the convexity arguments in the upper comparison.

A.2 Why This Packing Has the Actual Stash Count

The main claim is an equality between two counts: the blocks remaining in the real stash and the blocks sent above the root by auxiliary packing.

Lemma A.1 (Exact Residual Stash)**.** *For every such unit-block execution, every realization of the labels, and every $t \in \{0, \ldots, s\}$,*

$$S(\pi; t) = D^{(Z)}(A(t)). \tag{61}$$

Two properties of real refill will prove the equality. First, no block is stored below its target, so every block entering a bucket's count has a target in that bucket's subtree. Second, if one of those blocks remains above the bucket, the bucket is full. Conservation of blocks then gives the positive-part recursion (60). We prove the properties before deriving this recursion.

To describe the actual locations, let B_v contain the identifiers of real blocks in bucket v, and let $\mathcal{S}$ contain the stash identifiers. Each identifier in I_t occurs in exactly one of these locations. The placement rule is $b \in B_v \Rightarrow v \preceq y_b$, and the completed-access stash count is $S(\pi; t) = |\mathcal{S}|$. We regard the stash as a location strictly above the root. An *ancestor* of a node is a node on its path to the root, including itself; the *subtree rooted at a node* contains it and all its descendants. The following properties are *invariants*: they hold initially and remain true after every access.

Lemma A.2 (Greedy Packing Invariants)**.** *After every access:*

1. *a block in a bucket is stored at an ancestor of its target;*
2. *if a block is stored strictly above a node $v \preceq d_t(b)$, then B_v is full.*

Proof. We prove the properties by induction on the number of completed accesses. They hold before the first access because no real block has been created. Assume they hold just before access t. Only the old path $P(x_t)$ is read and refilled during this access.

Step 1: identify the blocks whose targets can change. A nonaccessed block keeps its label and gains one new path opportunity. Its target therefore either stays fixed or moves down its assigned path. If its old target is not on $P(x_t)$, the new common prefix with x_t cannot be deeper than that target, so the target stays fixed. If its target does move, both its old and new targets lie on $P(x_t)$. For the accessed block, the old label is x_t, so any old target lies on this path. Its new target is $\mathrm{lcp}(U_t, x_t)$, again on the path. If the block is newly created, it has no old target and starts in the stash with this same new target. Thus every changed existing target lies on the read path before and after the update, and every newly created target lies on that path as well.

Step 2: check that no block is placed below its new target. By the first invariant before the update, a block whose old target lies on the read path is stored on that path or in the stash. It is therefore available in the stash when refill begins. The accessed block is available as well, including if it is newly created. For every available block, the new target is at least as deep as its common prefix with x_t, since that prefix is one of its opportunities. Every eligible bucket on the read path is at or above this prefix, and hence is an ancestor of the new target. Refill therefore preserves the first invariant. A block in an unread bucket keeps both its position and its target, so it also preserves the invariant.

Step 3: check full buckets on the refilled path. Take a node v on $P(x_t)$ and suppose that a block ends above v while its target is at or below v. Every bucket above v lies on the read path. A block ending above v was therefore available for refill: its previous location was on that path or in the stash, or it was newly created. It was eligible for bucket v, because v lies on its current assigned path. Since refill works from the leaf toward the root, a block that finally ends above v was still waiting when bucket v was processed. The algorithm leaves it waiting only after filling all Z slots with other eligible blocks. The smallest-identifier rule fills every usable slot, as required.

Step 4: check full buckets outside the refilled path. Now take $v \notin P(x_t)$ and a block that ends above v with target in its subtree. Step 1 shows that this target is unchanged: changed and newly created targets all lie on the read path. If the block's old location was unread, both the block and bucket v keep their previous states, and the prior second invariant applies. Otherwise its old location was on the read path or in the stash. The prior first invariant makes that location an

ancestor of the unchanged target. Because v is outside the read path, this old location must be strictly above v. The prior second invariant therefore says that v was full. Bucket v is unread and remains full. This establishes both invariants after access t and completes the induction. □

Proof of Lemma A.1. Let r_v count blocks whose target is in the subtree of v but whose actual location is strictly above v. The first invariant and conservation of blocks give

$$A_v + r_{v0} + r_{v1} = |B_v| + r_v.$$

The left side counts the blocks with target exactly v, together with blocks that could not stay in either child subtree. Each such block is either stored in bucket v or stored above it, which are the two terms on the right. No block with a target outside this subtree can occupy v, by the first invariant. If $r_v > 0$, the second invariant forces $|B_v| = Z$. If $r_v = 0$, all the incoming blocks fit in v, whose occupancy is at most Z. Thus in either case, $r_v = (A_v + r_{v0} + r_{v1} - Z)^+$. This is exactly the auxiliary packing recursion. Starting at the leaves and moving upward gives $r_v = D_v^{(Z)}(A)$ at every node. At the root, the blocks stored strictly above it are precisely those in the stash, proving (61). □

The equality concerns the number of blocks in the stash, not a unique assignment of identifiers to buckets. Different choices among eligible blocks may give different bucket contents. The proof only needs exhaustive bottom-up refill: whenever an eligible block is waiting, no slot may be left empty. We retain the specified smallest-identifier rule throughout the paper.

Reading monotonicity from overloaded node sets. The exact count also tells us which changes can only increase overflow. For this purpose, we expand the recursion into a maximum over node sets. An *ancestor-closed* set contains the full path to the root from each of its nodes. For example, a set containing node 01 must also contain 0 and the root, but need not contain node 1.

Lemma A.3 (Overflow as Maximum Overload)**.** *For every nonnegative real load vector A on the finite tree,*

$$D^{(Z)}(A) = \max\left\{0, \max_{T} \sum_{v \in T} (A_v - Z)\right\}, \tag{62}$$

where T ranges over nonempty ancestor-closed node sets containing the root.

Proof. For a set T, the sum in (62) is its total target load minus its total bucket capacity. To expand the recursion at a node, include that node and decide separately whether to include nodes from each child branch. A branch contributes its largest positive excess, or contributes zero if all its choices have nonpositive excess. By induction, a nonempty choice in that branch is an ancestor-closed set rooted at the child. Combining the chosen branches with their parent therefore gives exactly the ancestor-closed sets rooted at that parent. The positive part also permits choosing no nodes, with value zero. Applying this expansion at the root proves (62). □

Lemma A.4 (Monotonicity of Overflow)**.** *For nonnegative real node loads on a fixed finite tree, the following changes have the indicated effects.*

1. *Adding nonnegative load at any nodes cannot decrease $D^{(Z)}$.*
2. *Moving load from a node to an ancestor cannot decrease $D^{(Z)}$.*
3. *Increasing the bucket capacity cannot increase $D^{(Z)}$. For a fixed address sequence and fixed random leaf inputs, larger capacity cannot increase the stash after any access.*

Proof. Adding load increases, or leaves unchanged, every overload in (62). For an upward move, every ancestor-closed set that contained the old location contains the new one, while some additional sets may now contain that load. No candidate overload decreases. Increasing Z decreases each candidate overload by adding capacity at every included bucket. Finally, targets themselves are unchanged when the same address sequence and random leaf inputs are used at different capacities. Lemma A.1 therefore transfers the capacity comparison to the actual stash. □

These three statements will be used separately. The upper comparison adds targets and moves some upward. The lower comparison discards targets, and therefore can only reduce local overflow. The capacity comparison completes the integer-threshold argument in Theorem 1.1.

A.3 Independent Labels and the Exact Scan Representation

Both proofs use the targets of a real scan, which accesses every address once. We derive their exact form from independent random labels.

Exposing the random inputs. The initial and fresh leaf choices already made by the protocol need names for this argument. Let $U_{0,b}$ be the initial label of address b, and let U_t be the new label drawn at access t. All these choices are mutually independent and uniform over the leaves. Initially $y_b = U_{0,b}$. At access t, save the old label $x_t = y_{a_t}$ before setting $y_{a_t} = U_t$; both the read and the refill use $P(x_t)$. For a fixed address sequence, these initial and new labels are the only random inputs to placement.

Repeated accesses appear to make the path labels dependent, because an old path uses a label selected at an earlier access. For a fixed address sequence, however, each independent random label is used once in exactly one of two roles: an old path label, or a final block label. Relabeling these same random inputs exposes the independence we need.

Lemma A.5 (Independent Label Reindexing)**.** *Fix an address sequence $\alpha = (a_1, \dots, a_s)$ independently of all leaf samples. Let y_b denote the final label of each address $b \in [N]$, including $y_b = U_{0,b}$ for any address never accessed. The old labels $x_1, \dots, x_s$ and final labels $y_1, \dots, y_N$ are jointly independent and uniform over the leaves.*

Proof. The first old leaf of block b is its initial label $U_{0,b}$. At each later access to b, the old leaf is the random label assigned on its preceding access. That label is then replaced, so it is never used as an old leaf again. The final label of a created block is its last relabeling variable, which has not yet been used as an old leaf. An uncreated block retains its unused initial label. Thus the s old leaves and N final labels use each of the $N + s$ independent inputs exactly once. Their ordering is fixed by the address sequence α, rather than by the outcomes of these inputs. They are therefore a fixed permutation of independent uniform labels, which proves the assertion. □

For example, in the sequence $(1, 2, 1)$, the old path labels are $U_{0,1}, U_{0,2}, U_1$, and the final labels of blocks 1 and 2 are U_3, U_2. All five variables are distinct independent inputs. The fixed address sequence determines the input permutation without using the sampled labels.

Reading the scan backward. At the end of a scan, the last block accessed has had one path opportunity, the preceding block has had two, and so on. Reversing the order of the blocks and paths therefore gives the following exact representation.

Lemma A.6 (Exact Scan Representation)**.** *Consider the real scan $(1, \dots, N)$ in a tree of depth L. For $L = \lceil \log_2 N \rceil$, this is $\mathcal{P}_{Z,N}(N, (1, \dots, N))$. The representation below also holds for this scan on*

a tree of any depth. Let $X_1, \dots, X_N$ be the old path labels in reverse chronological order, and let $Y_1, \dots, Y_N$ be the final block labels in reverse access order. These $2N$ labels are jointly independent and uniform. The target of reverse block k is the deepest node among

$$\{\mathrm{lcp}(Y_k, X_j) : 1 \le j \le k\}.$$

Thus these targets give exactly the final count field of the real scan.

Proof. Lemma A.5 gives joint independence of all the old paths and final labels; reversing their order is a fixed permutation. Reverse block k was last accessed exactly k access steps from the end, counting its own access. Its opportunity window is therefore precisely $X_1, \dots, X_k$. In particular, X_k is the path read and refilled when that block was accessed; this opportunity is included even though the block has just received the new label Y_k. Equation (59) gives the claim. □

Section 5 analyzes this real scan directly. At capacity four, Lemma 4.12 compares an arbitrary fixed address history with the same construction using 2^L targets.

B Finite Certificate Specifications

The two calculations in this appendix supply finite premises for the proofs. They have different directions: the upper proof needs certified upper bounds on a moment and its correction, whereas the lower proof needs a certified lower bound on a capped mean. Table 3 gives the input, stored object, output, and consumer of each calculation. The following two subsections specify the algorithms and their terminal checks.

Table 3: The two finite certificates and their proof obligations.

	Capacity four	Capacity three
Inputs	$H = 12$, $u = 7/5$, mesh 2^{-80}	$H = 14$, cap 32, mesh 2^{-48}
Stored object	Upper bounds on low-value probabilities and a separate moment upper bound	A normalized finite distribution below the target in every positive tail
Rounding	Upward; no renormalization	Downward for positive masses; losses moved to zero
Output	Bounds on M_{12} and B_{12}, with a positive closure margin	A capped mean strictly larger than $11/2$
Consumer	Lemma 4.21, then the all-height induction	Lemma 5.5, then the time and coupling losses

The main text proves why these update directions are valid and how the finite outputs enter the proofs for unbounded tree depth. The finite-certificate package (v1.0) is available at `https://github.com/Yijia-Oneplus-Chang/path-oram-certificates/releases/tag/v1.0`. It contains `verify_certificates.py`, the exact tables in `certificates.json`, and reproduction instructions in its README. The same instructions appear in `README-certificates.md` in the manuscript source package. Running `python3 verify_certificates.py` recomputes both tables, checks every entry against the supplied file, and verifies the terminal inequalities without modifying that file. The program uses only Python's standard library and rational or integer arithmetic. Its terminal comparisons are inequalities between exact numbers. The paper proves the mathematical comparisons and the extension from finite premises, independently of the software.

B.1 Capacity-Four Upper Certificate

The first certificate checks the starting bounds for the all-height moment induction. It stores upper bounds on selected small-value probabilities, together with a separate upper bound on the exponential moment. In the algorithm, $P_h(k)$ denotes the stored upper bound on $\mathbb{P}[Q_h = k]$; the hat on $\widehat{M}_h$ or $\widehat{B}_h$ likewise indicates an upper bound. The coefficients in P_h are not a normalized probability distribution. The function c is defined in (32), with $u = 7/5$ throughout this calculation.

The algorithm retains exactly the low-degree information that later updates need. At the terminal height twelve, the correction uses degrees zero through three. Each backward step needs four more degrees, which accounts for the schedule $K_h = 4(H - h) + 3$. Omitted coefficients are not treated as zero: the finite formulas prove that they cannot contribute to the retained outputs. At each height, step 3 forms the child-sum bounds and the correction; step 5 uses them to advance both probabilities and the moment. At the terminal height, step 4 skips that advance and goes directly to the final comparisons.

Algorithm 1: Upper-coefficient certificate for capacity four

1. Set $H = 12$, $D = 2^{80}$, $u = 7/5$, $a = 3125/7203$, $r = 23/20$. Define $\mathrm{up}(x) = \lceil Dx \rceil / D$.
2. Initialize $P_0 = (1, 0, \ldots, 0)$ through index $4H + 3$ and $\widehat{M}_0 = 1$.
3. For $h = 0, \ldots, H$, set $K_h = 4(H - h) + 3$ and compute $\widehat{b}_h(j) = \sum_{i=0}^{j} P_h(i) P_h(j - i)$ for $0 \le j \le K_h$. Set $\widehat{B}_h = \sum_{j=0}^{3} \widehat{b}_h(j) c(j)$ exactly.
4. Check $\widehat{M}_h < 111/100$. If $h = H$, go to step 6.
5. Compute $P_{h+1}(0), \ldots, P_{h+1}(K_h - 4)$ using (35)–(36), rounding each result upward with up. Set $\widehat{M}_{h+1} = \mathrm{up}(a\widehat{M}_h^2 + \widehat{B}_h)$ and return to step 3.
6. Check $\widehat{B}_H < 57621/100000$ and $r - ar^2 - 57621/100000 = 19687/720300000 > 0$.

Upward rounding preserves the required inequalities, but normalization need not do so. The algorithm therefore leaves its upper coefficients unnormalized. Across heights zero through twelve there are 364 retained coefficients, since $\sum_{h=0}^{12}(4(12 - h) + 4) = 364$. This count describes the size of the reproducible table; correctness comes from the nonnegative updates and rounding direction proved in the main text.

The final output supplies the two bounds needed for the induction. The terminal moment upper bound is

$$M_{12} \le \widehat{M}_{12} = \frac{133400309712361162726701}{120892581961462917470617 6} < \frac{111}{100}. \tag{63}$$

The terminal correction bound is

$$\begin{aligned} \widehat{B}_{12} &= \frac{10109712877157324344889325653889852625164259505088 27}{17545327156157489533035236416758977650969470178426 88} \\ &< \frac{57621}{100000}. \end{aligned} \tag{64}$$

Every strict comparison is checked by cross multiplication of exact integers, with positive denominators. The final margin in step 6 then checks that the moment update cannot cross r, as required by the induction in the main text.

B.2 Capacity-Three Lower Certificate

The second certificate must produce a guaranteed lower bound on a capped mean, so its stored object is different. It keeps a normalized distribution on the values zero through 32. Its probabilities need not be individually below the true probabilities; what is below the true distribution is every positive tail. Rounding positive masses down and moving the losses to zero preserves that property and hence gives a lower bound on the mean. The main text proves this comparison inductively.

The algorithm first performs the fourteen finite updates and then checks the resulting mean. It also checks every update by a second calculation: one uses differences of tail probabilities, and the other sums probabilities of individual child and geometric-input outcomes. Both must yield exactly the same integer mass vector after rounding. This is a check of the arithmetic implementation, alongside the mathematical comparison proof.

Algorithm 2: Downward mass certificate for capacity three

1. Set $D = 2^{48}$, $K = 32$, and $H = 14$. Initialize $(p_0, \dots, p_{32}) = (D, 0, \dots, 0)$.
2. Repeat 14 times: compute $T_1, \dots, T_{32}$ with (57), set $T_{33} = 0$, and update all masses by (58).
3. At every update, check $p_k \geq 0$ and $\sum_{k=0}^{32} p_k = D$. Recompute the updated masses by convolution of the two child distributions and the clipped geometric distribution. Check every entry against step 2.
4. Check $D^{-1} \sum_{k=0}^{32} k p_k = 775231701030647/140737488355328 > 11/2$.

The second formula uses a finite geometric input without losing its tail mass. The variable $\min\{G, 35\}$ has denominator 2^{35}, with numerator 2^{34-g} for $0 \leq g \leq 34$ and numerator one at $g = 35$. The last entry includes all original outcomes $G \geq 35$, whose combined probability is 2^{-35}; it is not merely the probability of $G = 35$. For children i, j and input g, send their product mass to $\min\{32, (g + i + j - 3)^+\}$. Sum all contributions to each output value before rounding its mass. Finally, round the masses at positive output values down to multiples of $1/D$, and add the discarded probability to the mass at output zero.

These direct mass sums evaluate the same update as the survival probabilities in (57). Agreement must hold for every stored entry, rather than only for the final mean. The complete table contains 495 integer mass entries: fifteen heights, including height zero, times 33 possible values. Step 4 checks that their terminal mean is the exact fraction used in Lemma 5.5 and is strictly larger than $11/2$.

C Arithmetic for the Lower-Bound Parameters

Table 1 in the main proof is the authoritative list of parameter requirements and their roles. Here we check the numerical losses used there. The selected values are

$$H = 14, \qquad T = 2^{19}, \qquad K = 32, \qquad d = L - 19, \qquad L \geq 49.$$

Thus $d \geq 30$, the local observation window has $n = T2^d = 2^L = N$ rounds, and $H \leq L - d = 19$ permits all required local mark categories inside the physical tree. These checks establish that the comparison concerns an actual region and a window contained in the actual scan.

The two approximation losses leave more than the required local surplus. For the time cutoff,

$T/2^H = 32$ and $e > 2$ give

$$K2^H e^{-T/2^H} = 32\,2^{14} e^{-32} < \frac{1}{8192}.$$

For the discrete coupling, $q = 2^{-d}$ and $d \geq 30$ give

$$3KTq = 96\,2^{19-d} \leq \frac{3}{64}.$$

Subtracting both from the finite certificate's lower bound yields

$$\frac{11}{2} - \frac{1}{8192} - \frac{3}{64} = \frac{44671}{8192} = 5 + \frac{3711}{8192} > 5.$$

The strict lower bound on each real local mean therefore survives both transfers.

The same choice of depth also supplies enough regions for the global claim. There are $2^d = N/2^{19}$ disjoint roots at that depth and $2^d - 1$ ancestor buckets above them. Subtracting the ancestors' capacity from more than five expected overflow units per region gives

$$5 \cdot 2^d - 3(2^d - 1) = 2^{d+1} + 3 = \frac{N}{2^{18}} + 3.$$

For the scan execution $\pi = \mathcal{P}_{3,N}(N, (1, \ldots, N))$, the main proof then uses $S(\pi; N) \leq N$ to obtain the stated probability bound. The arithmetic here does not assume independent real regions.

D A Diagnostic Check of the Full Finite Process

The final check exercises the actual finite-state process, including bucket identities and repeated accesses, on two small examples. Its purpose is to detect implementation or bookkeeping errors by comparing actual stash counts with the deterministic target formula. It supplies no premise for either asymptotic theorem.

The source-package program `check_small_cases.py` exhaustively enumerates the initial and new labels for $L = 2, N = 4, Z = 3$. It keeps all labels, bucket identities, stash contents, entered blocks, and targets, merging states only when all these recorded quantities agree. This matters because two configurations with the same stash size can still react differently to the next access. At every transition the program checks bucket capacity, unique placement, label eligibility, and the identity in Lemma A.1, using the prescribed smallest-identifier refill.

The trajectory counts check that the enumeration covers all random choices, rather than a sample of them. There are N initial labels and s new labels, each with 2^L possible values, so the integer trajectory weights sum to $2^{L(N+s)}$ in each case. The table gives a complete scan and then a scan followed by a repeated access, together with the resulting stash probabilities.

Table 4: Exact finite checks with $L = 2, N = 4, Z = 3$. All probability mass was enumerated. In each row, $s = |\alpha|$ and $\pi = \mathcal{P}_{3,4}(s, \alpha)$ for the displayed sequence α. "States" counts terminal augmented states; transitions count evaluations after identical-state merging.

Sequence	Trajectories	States	Transitions	$\mathbb{P}[S(\pi; s) = 1]$
$(1, 2, 3, 4)$	65 536	6 144	25 344	1/128
$(1, 2, 3, 4, 1)$	262 144	6 144	49 920	1/128

In both cases $\mathbb{P}[S(\pi; s) = 0] = 127/128$ and all other probabilities are zero.

These exhaustive small examples are diagnostics, not evidence for an extrapolation to large trees. The accompanying `small-cases.json` records both outputs. To reproduce them, run `python3 check_small_cases.py` in a disposable copy of the manuscript sources, since the program rewrites that file. This diagnostic is separate from the finite-certificate package. These small examples check the implementation against the deterministic representation. The finite certificates and analytical arguments in the main text carry the asymptotic conclusions.

E Proofs of the Stash Corollaries

Proof of Corollary 3.4. For each $1 \le t \le s$, the first t accesses form a fixed instance covered by Theorem 1.2. Thus $\mathbb{P}[S(\pi;t) > R] \le (23/28)(5/7)^R$. Overflow after at least one refill is the union of these s events. The union bound gives (12), without requiring independence between the events. Solving $(23s/28)(5/7)^R \le \delta$ for the nonnegative integer R gives (13). □

Proof of Corollary 3.5. Proposition 4.1, proved in Section 4.2, establishes

$$\mathbb{E}\big[(7/5)^{S(\pi;s)}\big] \le \frac{23}{20}.$$

For every integer $k \ge 0$, $(7/5)^k \ge 1 + (2/5)k$. Taking expectations with $k = S(\pi;s)$ therefore gives

$$1 + \frac{2}{5}\mathbb{E}[S(\pi;s)] \le \mathbb{E}\big[(7/5)^{S(\pi;s)}\big] \le \frac{23}{20},$$

which yields the claimed bound. □

AI Methodology

We used OpenAI's Codex with the GPT-6 Astra model to assist with drafting and revising the exposition, organizing and checking proof arguments, and generating code for the computational certificates and their verification. This assistance affected the exposition throughout the paper, the proof descriptions in Sections 3–5, and the accompanying computational materials. The authors verified the correctness and originality of all content and take full responsibility for the submission.

References

[ACCH23] David Aldous, Alice Contat, Nicolas Curien, and Olivier Hénard. Parking on the infinite binary tree. *Probability Theory and Related Fields*, 187(1–2):481–504, 2023.

[ACN+22] Gilad Asharov, T.-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Locality-preserving oblivious RAM. *Journal of Cryptology*, 35(2), 2022. Article 6.

[ASB19] Kholoud Saad Al-Saleh and Abdelfettah Belghith. Radix Path: A reduced bucket size ORAM for secure cloud storage. *IEEE Access*, 7:84907–84917, 2019.

[BBJ21] Riti Bahl, Philip Barnet, and Matthew Junge. Parking on supercritical Galton–Watson trees. *ALEA, Latin American Journal of Probability and Mathematical Statistics*, 18(2):1801–1815, 2021.

[BCP16] Elette Boyle, Kai-Min Chung, and Rafael Pass. Oblivious parallel RAM and applications. In *Theory of Cryptography – TCC 2016-A, Part II*, volume 9563 of *Lecture Notes in Computer Science*, pages 175–204. Springer, 2016.

[BY09] Bartłomiej Błaszczyszyn and D. Yogeshwaran. Directionally convex ordering of random measures, shot noise fields, and some applications to wireless communications. *Advances in Applied Probability*, 41(3):623–646, 2009.

[CC25] Linxiao Chen and Alice Contat. Parking on supercritical geometric Bienaymé–Galton–Watson trees. *ALEA, Latin American Journal of Probability and Mathematical Statistics*, 22(2):1287–1299, 2025.

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. *Journal of the ACM*, 43(3):431–473, 1996.

[HKKS19] Pavel Hubáček, Michal Koucký, Karel Král, and Veronika Slívová. Stronger lower bounds for online ORAM. In *Theory of Cryptography – TCC 2019, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 264–284. Springer, 2019.

[KKNO16] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1329–1340. ACM, 2016.

[LLG24] Xiang Li, Yunqian Luo, and Mingyu Gao. BULKOR: Enabling bulk loading for Path ORAM. In *IEEE Symposium on Security and Privacy*, pages 4258–4276. IEEE, 2024.

[LP18] Günter Last and Mathew Penrose. *Lectures on the Poisson Process*, volume 7 of *Institute of Mathematical Statistics Textbooks*. Cambridge University Press, 2018.

[MBN15] Tarik Moataz, Erik-Oliver Blass, and Guevara Noubir. Recursive trees for practical ORAM. *Proceedings on Privacy Enhancing Technologies*, 2015(2):115–134, 2015.

[MPC+18] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 279–296. IEEE, 2018.

[RFK+15] Ling Ren, Christopher Fletcher, Albert Kwon, Emil Stefanov, Elaine Shi, Marten van Dijk, and Srinivas Devadas. Constants count: Practical improvements to oblivious RAM. In *24th USENIX Security Symposium*, pages 415–430. USENIX Association, 2015.

[RWZ+26] Deevashwer Rathee, Jean-Luc Watson, Zirui Neil Zhao, G. Edward Suh, and Raluca Ada Popa. Onyx: Cost-efficient disk-oblivious ANN search. arXiv:2604.20401v1, 2026.

[RYF+13] 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 *Proceedings of the 40th Annual International Symposium on Computer Architecture (ISCA)*, pages 571–582. ACM, 2013.

[RYTZ23] Mehrnoosh Raoufi, Jun Yang, Xulong Tang, and Youtao Zhang. AB-ORAM: Constructing adjustable buckets for space reduction in Ring ORAM. In *IEEE International Symposium on High-Performance Computer Architecture*, pages 361–373. IEEE, 2023.

[SGF18] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. ZeroTrace: Oblivious memory primitives from Intel SGX. In *Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2018.

[Shi20] Elaine Shi. Path Oblivious Heap: Optimal and practical oblivious priority queue. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 842–858. IEEE, 2020.

[SvDS+13] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer and Communications Security*, pages 299–310. ACM, 2013.

[SvDS+14] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. arXiv:1202.5150v3, 2014. Version 3, 14 January 2014.

[SvDS+18] Emil Stefanov, Marten van Dijk, Elaine Shi, T.-H. Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. Path ORAM: An extremely simple oblivious RAM protocol. *Journal of the ACM*, 65(4):18:1–18:26, 2018.

[TD26] Nihal Talur and Ioannis Demertzis. SONIC: Concurrent oblivious RAM & data structures for low-latency and high-throughput. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 4561–4580. USENIX Association, 2026.

[WCS15] Xiao Wang, Hubert Chan, and Elaine Shi. Circuit ORAM: On tightness of the Goldreich–Ostrovsky lower bound. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 850–861. ACM, 2015. Full version: `https://eprint.iacr.org/2014/672`, version posted 28 November 2016.