Dynamic Range-Optimized DUORAM

UGP REPORT

Srishti Chandra Roll No. 221088

Mentor: Prof. Adithya Vadapalli

Abstract :

We proposed a privacy-aware and computationally efficient extension of DUORAM, inspired by differential privacy techniques applied in Root ORAM. To reduce the overhead of Distributed Point Function (DPF) key generation, we limited each query to a dynamically chosen subrange of size t(t < n), rather than accessing the entire database. To further obscure the true access pattern, we explored the possibility of introducing fake queries to help mask the actual range being queried. Our design organized database indices in a binary tree structure, where queries corresponded to nodes along the path from a leaf (target index) to the root. The size of the queried range directly impacted both the computational cost and the level of privacy leakage. We analyzed this trade-off and provided bounds on the adversary's ability to infer the queried index. To balance privacy and efficiency, we developed a strategy for dynamically selecting range sizes while ensuring that the total computation remained within a given time budget.

Index Terms: Differential Privacy,Distributed Point Function (DPF), hiding the access pattern.

1 Introduction

As more data is stored and accessed in the cloud, keeping that data private is more important than ever. While encryption protects the content of the data, it does not hide *how or when* the data is accessed. This can lead to *access pattern leaks*, where an attacker learns which locations are being accessed, even without seeing the actual data.

To solve this problem, **Oblivious RAM (ORAM)** was introduced. ORAM makes all memory accesses look the same, so the server cannot learn anything about which data is being used. One efficient version of ORAM is called **Root ORAM**, which improves speed by organizing the memory into smaller parts (subtrees). However, it still hides everything about the access pattern, which leads to high computation and communication costs.

To reduce this cost, researchers introduced **Differential Privacy (DP)** into ORAM. Instead of making all access paths completely hidden, DP allows a small and controlled amount of information to leak. In *DP-based Root ORAM*, the block being accessed is remapped in a biased way — closer options are chosen more often. This makes the system faster while still giving formal privacy guarantees.

Inspired by this, my work focuses on applying a similar idea to **DUORAM**, which is a version of ORAM that uses *two non-colluding servers* and a technique called **Distributed Point Functions (DPFs)**. In DUORAM, the client usually generates DPFs over the full database, which is slow. I explore how to make this more efficient by generating DPFs over smaller *ranges*. By allowing the server to know the range — but not the exact index — we can make DUORAM faster, while still limiting the privacy leak using differential privacy.

1.1 Contribution

This paper has the following contributions:

- We propose an approach to improve the efficiency of DUORAM by generating DPFs over smaller query ranges instead of the full database.
- We analyze how reducing the range size impacts computation time and privacy, and formalize the trade-off using a tree-based range selection model.

1.2 Organization of the Paper

The rest of this report is structured as follows:

- **Section 2 (Background).** We review classical ORAM, Distributed Point Functions and secure MPC, and then survey statistical-privacy approaches—including DP-ORAM and DP-PIR—citing Wagh *et al.* Wagh et al., 2018 and Albab *et al.*Albab et al., 2022.
- **Section 3 (Root ORAM Scheme).** We describe the original Root ORAM construction of Wagh *et al.*, its data structure and access/update protocols.
- **Section 4 (Differential Privacy in ORAM).** We show how DP can be applied to ORAM access patterns, formalize (ϵ, δ)-DP for ORAM, and discuss trade-offs.

- **Section 5 (Distributed ORAM with DPFs & MPC).** We review DUORAM's use of DPFs and MPC to achieve oblivious reads/writes with sublinear online cost.
- **Section 6 (Range-Optimized DUORAM).** We introduce our new range-restricted DPF variant, analyze its computation–privacy trade-off, and give the full protocol.
- **Section 7 (Design Rationale).** We compare alternatives (dummy padding, full-domain DPFs, fixed vs. adaptive ranges) and justify our choices.
- **Section 8 (Future Work).** We outline directions for improving DP budgets, noise-verification, and MPC optimizations.
- **Section 9 (Conclusion).** We summarize our findings.

2 Background

Oblivious RAM (ORAM) enables a client to outsource data storage to an untrusted server while hiding its memory access pattern so that any two access sequences of the same length remain indistinguishable to the server Wagh et al., 2018.

Differentially Private ORAM (DP-ORAM) relaxes this strict indistinguishability by allowing a bounded statistical leak, formalized via the (ε, δ) definition from differential privacy. The Root ORAM family provides a tunable trade-off between bandwidth, client storage, and privacy guarantees Wagh et al., 2018.

DUORAM extends ORAM to two- and three-party computation settings using $(2, 2)$-distributed point functions (DPFs) for private reads and writes. By offloading most work to a preprocessing phase and introducing a one-round dot-product evaluation, DUORAM achieves $O(m \log n)$ total communication and $O(m)$ words of online bandwidth for m accesses on an n-element database Vadapalli et al., 2023.

2.1 Literature Review

Wagh *et al.* Wagh et al., 2018 introduce the notion of *Differentially Private ORAM* (DP-ORAM), which extends the classical ORAM security definition by permitting a small, quantifiable leakage in exchange for performance gains. They formalize (ε, δ)-DP for ORAM access patterns and present the *Root ORAM* family, a tunable, tree-based construction that trades off bandwidth and client storage against statistical privacy. Their analysis shows up to 2× reduction in local storage or up to 10× bandwidth savings under practical privacy budgets, with rigorous DP guarantees.

Vadapalli *et al.* Vadapalli et al., 2023 build on DPF-based ORAMs to create *DUORAM*, a two- and three-party protocol optimized for secure multiparty computation. By leveraging (2,2)-Distributed Point Functions, a preprocessing phase, and a novel one-round dot-product evaluation, DUORAM achieves $O(m \log n)$ total communication and only $O(m)$ words of online bandwidth per sequence of m accesses on an n-element database—significantly improving over previous DPF-based schemes.

3 Root ORAM Structure and Operation

Root ORAM is a tree-based ORAM variant that allows a tunable trade-off between bandwidth, client storage, and statistical privacy. Its design builds on Path ORAM but generalizes the tree into multiple "sub-trees" and employs a non-uniform remapping distribution to improve performance.

3.1 Server Storage

The server stores the outsourced data as 2^k independent binary sub-trees, each of depth $L - k$ (where $N = 2^L$ is the total number of blocks). Each node is a *bucket* holding up to Z blocks (real or dummy). Buckets are identified by level i ($k \le i \le L$) and leaf index $x \in \{0, \dots, 2^L - 1\}$, and the path from leaf x to its sub-tree root is denoted $P(x)$

3.2 Client State

- **Position map:** Recursively stored via smaller ORAMs, it maps each data block to a leaf index in $\{0, \dots, 2^L - 1\}$.
- **Stash:** A small local buffer for overflow blocks that cannot fit into buckets during eviction

3.3 Access Procedure

To read or write block a:

1. Lookup its current leaf $x \leftarrow \text{position}[a]$.
2. Fetch all buckets along $P(x)$ into the stash.
3. Perform the read or update in the stash.
4. *Re-map* a to a new leaf x' by sampling from a non-uniform distribution that favors leaves in the same sub-tree (see below).
5. *Evict:* For each level $i = L, \dots, k$, pick up to Z blocks from the stash whose current leaf maps through the bucket at level i of $P(x)$, and write them back, pushing them as far down toward x as possible.
6. Return the read data (if any) to the client.

This single "read-and-evict" step suffices; no separate reshuffle or stash-flush is needed.

3.4 Remapping Distribution

After each access, block a's new leaf is chosen by

$$\Pr[x' = z \mid x] = \begin{cases} p_{\max}, & \text{if } z \text{ is in the same sub-tree as } x, \\ p_{\min}, & \text{otherwise,} \end{cases}$$

with

$$p_{\max} = \frac{1 + (2^k - 1)p}{2^L}, \quad p_{\min} = \frac{1 - (1 - \delta_{k0})p}{2^L},$$

where $p \in [0, 1]$ is a privacy–performance parameter and δ_{k0} is the Kronecker delta. This bias reduces stash usage and overall bandwidth compared to uniform remapping.

3.5 *Comparison to Path ORAM*

When $k = 0$, there is a single tree of depth L, $p_{\max} = p_{\min} = 1/N$, and Root ORAM exactly coincides with Path ORAM. For $k > 0$, Root ORAM caches the top k levels (reducing tree height to $L - k$) and allows non-uniform remapping, yielding a flexible trade-off between communication, client storage, and statistical privacy .

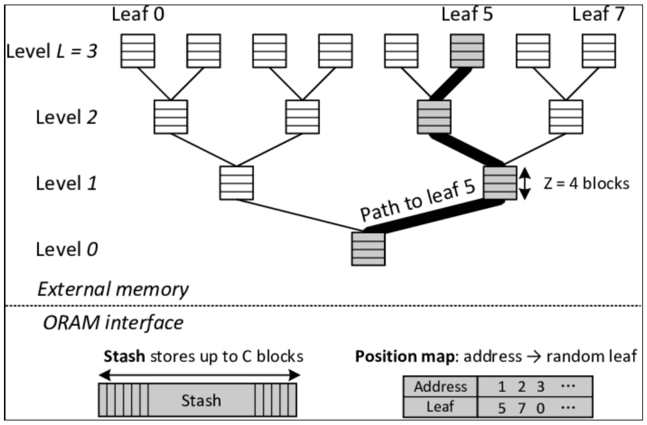

Figure 1. Server-side storage in Root ORAM: the data is partitioned into 2^k subtrees of depth $L - k$. Each bucket (node) holds up to Z blocks (real or dummy), and any accessed block is located along the path from its assigned leaf to the subtree root or in the client stash.

4 Differential Privacy for Access Pattern Protection

Differential Privacy (DP) provides mathematically rigorous guarantees about how much information a system leaks—even against adversaries with arbitrary background knowledge. In ORAM, DP ensures that observing server access patterns reveals minimal information about which data items were accessed.

Recent work has also applied differential privacy to Private Information Retrieval (PIR). In particular, Albab *et al.* Albab et al., 2022 introduce a batched DP-PIR protocol achieving constant amortized cost when the total query batch is large. We build on the same (ϵ, δ)-DP framework, but focus on a two-party ORAM setting with DPFs and MPC.

4.1 *Core Concepts*

Definition 1 (Neighboring Access Sequences) *Two access sequences*

$$a = (a_1, \ldots, a_m), \quad a' = (a'_1, \ldots, a'_m)$$

are neighboring *if they differ in exactly one operation. Equivalently, there exists a unique index i such that*

$$a_j = a'_j \quad (j \neq i), \quad a_i \neq a'_i.$$

Example:

$$a: \texttt{read(5), read(3), read(5)}$$

$$a': \texttt{read(5), read(7), read(5).}$$

Definition 2 (Access–Pattern Histogram and Sensitivity) *An ORAM access sequence a induces a* histogram $H(a) \in \mathbb{Z}^k$ *that counts how many times each bucket (or tree-path) was touched. Its* sensitivity Δ *is*

$$\Delta = \max_{a,a' \text{ neighboring}} \|H(a) - H(a')\|_1 = 1,$$

since changing one access can affect at most one bucket count by one.

Definition 3 ((ε, δ)-Differential Privacy) *An ORAM mechanism $\mathcal{M}$ that releases $\widetilde{H}(a) = H(a) + \eta$ is (ε, δ)-differentially private if for every pair of neighboring sequences a, a' and every measurable set $S \subseteq \mathbb{Z}^k$*

$$\Pr\big[\mathcal{M}(a) \in S\big] \leq e^{\varepsilon} \Pr\big[\mathcal{M}(a') \in S\big] + \delta.$$

Here

- ε *(the* privacy loss*) controls the multiplicative distinguishability; smaller ε gives stronger privacy.*
- δ *(the* failure probability*) is typically negligible in n, bounding the chance that this guarantee is broken.*

Key Properties

- **Composition:** Releasing k DP-ORAM histograms on the same data degrades privacy to $(k\varepsilon, k\delta)$.
- **Group privacy:** If two sequences differ in up to g accesses, the guarantee becomes $(g\varepsilon, e^{g\varepsilon}\delta)$.
- **Plausible deniability:** Even knowing all but one access, an adversary cannot distinguish whether that access was at position 5 or at position 7 beyond the ε, δ bounds.

4.2 *DP Mechanisms in Root ORAM*

Root ORAM limits how much information each access can leak by combining three complementary differential-privacy techniques. First, it biases the remapping of blocks to nearby leaves, so that most accesses stay within a small subtree. Second, it can inject dummy block evictions to further blur the real pattern, though in many implementations (including ours) we omit this step because (i) it multiplies bandwidth by the number of dummies, (ii) it complicates stash management, and (iii) its marginal privacy gain is small once remapping is already DP. Finally, it carefully controls how privacy parameters compose across multiple accesses, preventing unbounded leakage over long sequences.

4.2.1 Biased Remapping

After each access to leaf x, the block is remapped to a new leaf x' according to

$$\Pr[x' \mid x] \;\propto\; \exp\bigl(-\varepsilon\, d(x,x')\bigr),$$

where $d(x,x')$ measures the distance between subtrees. This concentrates remappings nearby (reducing communication) while guaranteeing $(\varepsilon,0)$-DP for each individual access.

4.2.2 Dummy-Eviction (Optional)

To add another layer of obfuscation, one can evict up to D dummy blocks along each accessed path. These extra writes make it harder for an adversary to tell which bucket contains the real block. In practice, however, we often omit dummy-eviction because:

- It increases bandwidth and round-trip cost by up to a factor of D.
- It demands careful stash sizing to avoid overflows.
- The incremental privacy gain beyond biased remapping is modest, especially for small ε.

4.2.3 Composition Control

Over a sequence of k accesses, naive composition would yield

$$(\varepsilon_{\text{total}},\ \delta_{\text{total}}) = (k\varepsilon,\ k\delta)\,.$$

Root ORAM therefore sets $\varepsilon = O(1/\sqrt{k})$ and $\delta \ll 1/k$, so that

$$\varepsilon_{\text{total}} = O(\sqrt{k}), \qquad \delta_{\text{total}} \le k\,\delta = o(1)\,,$$

thus avoiding a linear loss of privacy as the number of accesses grows.

5 Distributed Point Functions, MPC, and Client–Server Interaction

Before introducing any differential-privacy enhancements, DUORAM relies on two core cryptographic tools—Distributed Point Functions (DPFs) and secure Multi-Party Computation (MPC)—to achieve oblivious reads and writes with sublinear communication.

5.1 Distributed Point Function

Distributed Point Functions (DPFs) are the cornerstone of DUORAM's compact address encoding. A DPF allows two non-colluding servers to hold *secret shares* of a point function

$$f_{i^*,M}(i) \;=\; \begin{cases} M, & i = i^*,\\ 0, & i \neq i^*, \end{cases}$$

without either server learning i^* or M on its own Vadapalli et al., 2023.

The DPF scheme consists of two efficient algorithms:

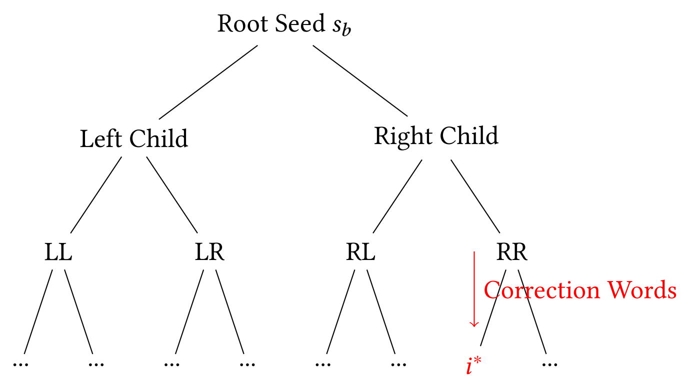

Figure 2. DPF key generation tree structure. Red path shows correction words applied to target index i^*.

Key Generation (GEN)	Evaluation (EVAL)
1. Initialize root seeds $s^{(0)}, s^{(1)}$ 2. Expand seeds with PRG at each level 3. Apply correction words for target path 4. Final correction for M at leaf i^*	1. Follow path for queried index i 2. XOR correction words at divergence points 3. Compute final share $v_b(i)$ 4. Amortized $O(1)$ cost with EVAL-FULL

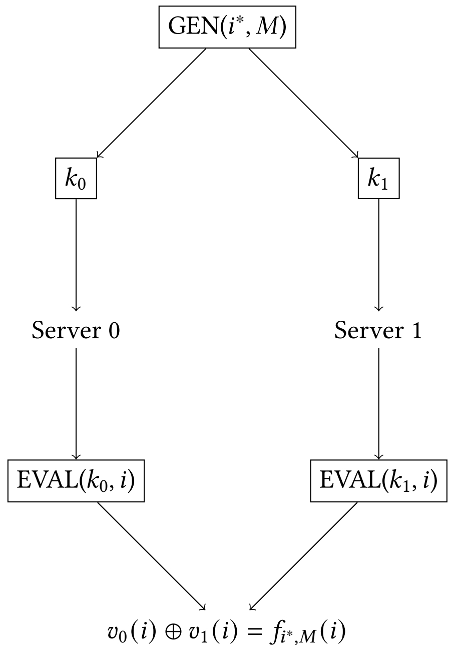

Figure 3. DPF workflow: Key generation and distributed evaluation

Security and Efficiency Properties

- **Local Evaluation**: No server communication during EVAL
- **Information Theoretic Security**: Single key reveals no information about i^* or M

5.2 Secure Multi-Party Computation (MPC)

MPC enables n parties $P_1, \dots, P_n$ to jointly compute a function $f(x_1, \dots, x_n)$ over their private inputs x_i, while preserving input privacy. Formally, for inputs $x_i \in \mathcal{X}$ and output $y \in \mathcal{Y}$:

$$\forall i \in [n],\ y = f(x_1, \dots, x_n) \text{ with } \mathcal{V}_i \not\ni x_j\ (j \neq i)$$

where $\mathcal{V}_i$ denotes party P_i's view of the protocol. This is achieved through secret sharing: each party splits its input x_i into shares $[x_i]_1, \dots, [x_i]_n$ such that:

$$x_i = \bigoplus_{j=1}^{n} [x_i]_j$$

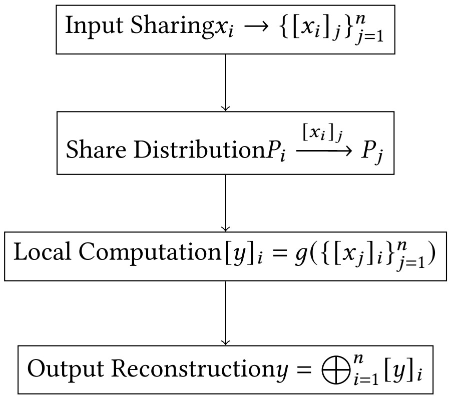
Input Sharing$x_i \rightarrow \{[x_i]_j\}_{j=1}^n$
Share Distribution$P_i \xrightarrow{[x_i]_j} P_j$
Local Computation$[y]_i = g(\{[x_j]_i\}_{j=1}^n)$
Output Reconstruction$y = \bigoplus_{i=1}^n [y]_i$

Figure 4. MPC workflow using additive secret sharing. Arrows show information flow between parties $P_1, \dots, P_n$.

5.3 Client–Server Interaction and Key Distribution

Once the client has chosen its query index i^* (and, in the range-optimized variant, a subtree $S \ni i^*$ of size s), it locally generates a pair of Distributed Point Function (DPF) keys (k_0, k_1). Each key encodes the same "1-hot" vector over the selected range S, secret-shared between the two servers. Concretely, the client runs

$$(k_0,\ k_1) \ = \ \mathsf{DPF.Gen}(S,\ i^*)$$

and then transmits

$$(k_0,\ S) \ \longrightarrow\ \text{Server } 0, \qquad (k_1,\ S) \ \longrightarrow\ \text{Server } 1$$

over secure, authenticated channels. This single one-round exchange fully hides i^* (up to the known leakage of S) and does not reveal which key goes to which server.

Upon receipt, each server $b \in \{0, 1\}$ invokes the local evaluation algorithm

$$v_b \ = \ \mathsf{DPF.Eval}(k_b,\ S)$$

to obtain its share of the indicator vector for the client's access. Server b then uses v_b to select and mask the appropriate database word(s) along the path or range, and—after a brief MPC-style exchange of compact inner-product shares—returns its response share to the client. The client finally reconstructs the value by XORing (or adding) the two server replies. Thus, in exactly two messages (keys down, shares up), the client reads or writes at i^* without ever revealing the precise index to either server."'

6 Range-Optimized DUORAM

6.1 Mathematical Model of Range Trade-offs

Let s be the selected range size and k the number of queries. We establish formal bounds:

$$\begin{aligned}
\text{Computation Cost:} \quad & C(s) = O(k \cdot s) \\
\text{Privacy Leakage:} \quad & P_{\text{adv}}(s) = \frac{1}{s} \\
\text{Union Bound (k queries):} \quad & \delta \leq \frac{k}{s} \\
\text{Differential Privacy:} \quad & \epsilon \leq \ln\left(\frac{\alpha}{1-\alpha}\right) \quad \text{where } \alpha = \frac{1}{s}
\end{aligned}$$

6.2 Binary Tree Optimization

For database size $n = 2^L$, the binary tree structure enables efficient range selection:

$$\begin{aligned}
\text{Depth } d \text{ node covers:} \quad & s_d = \frac{n}{2^d} \text{ indices} \\
\text{Path length for } i^* : \quad & L - d \text{ levels} \\
\text{Total computation:} \quad & \sum_{q=1}^{k} s_{d_q} \leq t_{\max}
\end{aligned}$$

6.3 Range Announcement and Its Privacy Implications

In our range-optimized DUORAM, the client transmits not only the two DPF keys (k_0, k_1) but also the chosen index range S (of size s) to each server:

$$(k_0,\ S) \ \longrightarrow\ \text{Server } 0, \qquad (k_1,\ S) \ \longrightarrow\ \text{Server } 1.$$

This "range announcement" is crucial for correctness, since each server must know which subset of leaves to evaluate the DPF over. However, revealing S also leaks that the true access i^* lies somewhere in S, reducing the adversary's uncertainty from n possible indices down to s. Concretely, the best-case posterior guess probability becomes

$$\Pr[\text{guess } i^*] = \frac{1}{s} \ = \ P_{\text{adv}}(s),$$

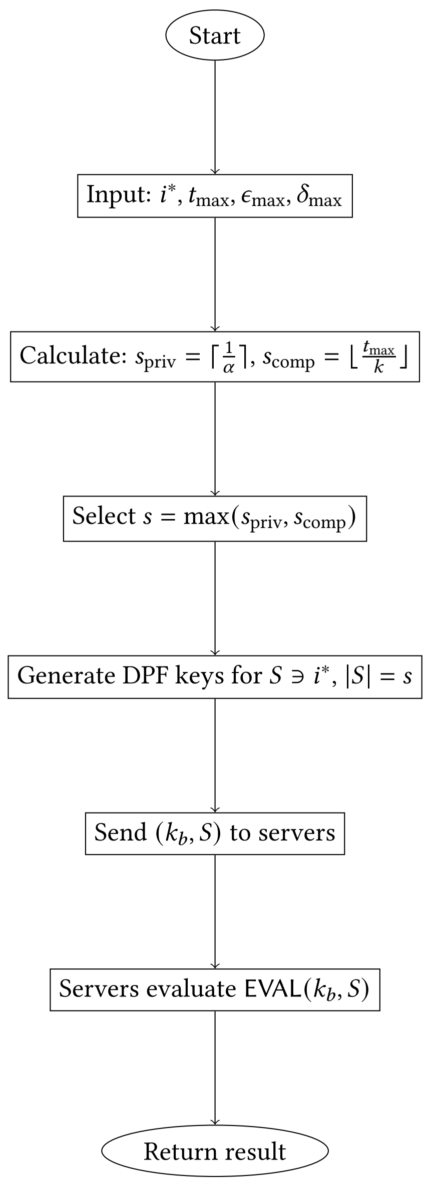
Start
Input: $i^*, t_{\max}, \epsilon_{\max}, \delta_{\max}$
Calculate: $s_{\text{priv}} = \lceil \frac{1}{\alpha} \rceil$, $s_{\text{comp}} = \lfloor \frac{t_{\max}}{k} \rfloor$
Select $s = \max(s_{\text{priv}}, s_{\text{comp}})$
Generate DPF keys for $S \ni i^*$, $|S| = s$
Send (k_b, S) to servers
Servers evaluate $\text{EVAL}(k_b, S)$
Return result

Figure 5. Range-optimized DPF workflow with adaptive size selection

matching the bound in our mathematical model. Thus, while announcing S enables an $O(s)$-time offline DPF generation instead of $O(n)$, it necessarily trades off a controlled $1/s$ privacy leakage, as captured by our differential-privacy parameter $\alpha = 1/s$.

6.4 Formal Privacy Guarantees

For (ϵ, δ)-DP, we enforce:

$$\frac{1}{s} \leq \alpha \quad \text{and} \quad \frac{k}{s} \leq \delta$$

Solving for s gives:

$$s \geq \max\left(\frac{1}{\alpha}, \frac{k}{\delta}\right)$$

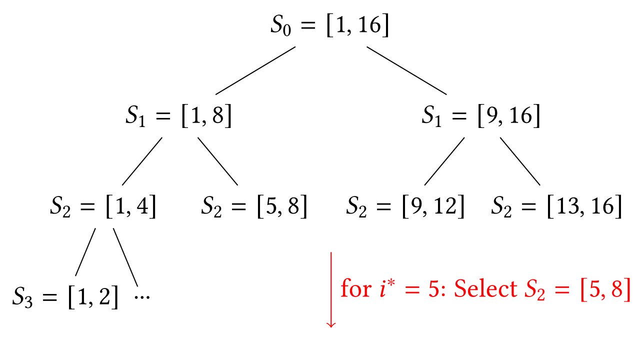

Figure 6. Binary tree range selection example ($n = 16$). Red arrow shows optimal range choice balancing privacy and efficiency.

This leads to the fundamental efficiency-privacy equation:

$$t_{\min} = k \cdot \max\left(\frac{1}{\alpha}, \frac{k}{\delta}\right)$$

7 Design Rationale and Alternatives

When we moved to range-optimized DPFs, we considered adding dummy queries (fake range accesses) to hide the true range more. However, padding every query with s dummy points would multiply both computation and communication by s, undoing the efficiency gains of smaller DPFs. Instead, we keep a simple, direct range and rely on its size to control leakage.

- **Tree Ranges vs. Full Database:** A full-database DPF ($s = n$) leaks nothing but costs $O(n)$. Using a tree, each index i^* lies in $\log_2 n + 1$ possible node ranges of size $n/2^d$. Choosing a range of size $s = n/2^d$ cuts offline work to $O(s)$ and leaks only $1/s$.
- **Dynamic vs. Fixed Blocks:** Fixed partitions force every query to use the same s. By picking the unique subtree containing i^*, we adapt s per query—meeting both privacy goals (α, δ) and compute limits $t_{\max}$.
- **Hard Bounds with Max-Rule:** We set
$$s = \max\left(\frac{1}{\alpha}, \frac{k}{\delta}, \frac{t_{\max}}{k}\right)$$
so that privacy (α, δ) and total work ($k \cdot s \leq t_{\max}$) are both guaranteed. A weighted average would blur these clear limits.
- **No Dummy Blow-Up:** Fake accesses would multiply server cost by s. Our approach sends two DPF keys of length $O(\log s)$ and does $O(s)$ local work, while the online phase stays at constant size.
- **Reusing Existing Tools:** DUORAM already builds DPFs and uses MPC dot-products. Shrinking the domain simply reuses those same routines with a smaller parameter s, avoiding extra code or protocols.
- **Future Extensions:** The tree model lets us later add per-level noise for differential privacy or choose different (α, δ) budgets at each depth—without touching the core access steps.

In short, this simple rule picks the smallest range that meets all privacy and performance requirements, giving maximal speed-ups while keeping worst-case leakage tightly bounded.

8 Future Work and Improvements

Although range-optimized DUORAM speeds up key generation and query processing, there are several ways to make it even better:

- **Measure Privacy Leakage Precisely.** Right now we only control privacy by picking a range size. We should analyze exactly how much information leaks for different range choices and give clear privacy bounds.
- **Smarter Range Choices.** Instead of picking a fixed range size, we can build an algorithm that adapts the range on the fly—making ranges bigger or smaller based on past queries, user privacy settings, or time constraints.
- **Multi-Level Ranges.** Rather than picking one range in the tree, we could split a query into parts: first pick a big range, then refine to smaller ones. This could give even more flexibility in balancing cost and privacy.
- **Speeding Up Preprocessing.** The offline step still does a lot of work. We could share some randomness across queries or compress the DPF keys when ranges are small to cut down on setup time.
- **Real-World Integration.** Test these ideas in concrete apps—such as private web search, secure machine learning on sensitive data, or health-care databases—where we know more about typical access patterns and can tune our method accordingly.

9 Conclusion

In this report, we have presented a practical extension to DUORAM that introduces *dynamic range optimization* for Distributed Point Function (DPF) key generation. By restricting each ORAM query to a contiguous subrange of size $s \ll n$ in a binary-tree partitioning of the address space, we reduce offline preprocessing costs from $O(n)$ to $O(s)$ while bounding the adversary's guess probability by $1/s$. Our scheme preserves the single-round, constant-size online protocol of standard DUORAM and shifts the bulk of additional work into a predictable preprocessing phase.

We provided formal bounds and demonstrated a simple algorithm to choose the minimal subrange that meets both a computation budget and a privacy budget. This yields $\log_2 n + 1$ natural points on the performance–privacy spectrum, allowing system designers to tailor DUORAM's cost and leakage precisely.

Future work includes exploring non-uniform or k-ary partitionings to achieve finer trade-offs, integrating dummy-query or noise-amplification techniques from differential privacy to further reduce leakage, and building an end-to-end prototype to evaluate real-world performance under realistic network and parallelism constraints.

References

Albab, Kinan Dak et al. (2022). "Batched Differentially Private Information Retrieval". In: *31st USENIX Security Symposium.*

Vadapalli, Adithya, Ryan Henry, and Ian Goldberg (2023). "DUORAM: A Bandwidth-Efficient Distributed ORAM for 2- and 3-Party Computation". In: *32nd USENIX Security Symposium.*

Wagh, Sameer, Paul Cuff, and Prateek Mittal (2018). "Differentially Private Oblivious RAM". In: *Proceedings on Privacy Enhancing Technologies* 2018.4. DOI: `10.1515/popets-2018-0032`.