RingOA: Fast Oblivious Access for Large-Scale Privacy-Preserving Structured Data Analysis

Tomoki Uchiyama
Waseda University
tomo-uchiyama@moegi.waseda.jp

Kana Shimizu
Waseda University
shimizu.kana@waseda.jp

Abstract

Many privacy-preserving data analysis tasks based on multi-party computation require oblivious retrieval of data elements for downstream use. As database sizes grow, this retrieval step becomes a dominant bottleneck, highlighting the need for more efficient oblivious access (OA) primitives that retrieve a database entry without revealing the accessed position. A common approach uses distributed point functions (DPFs), which reduce communication but still incur local computation that scales linearly with the database size. Early termination (ET) optimization reduces local computation, but applying it yields only Boolean shares that require costly conversion for arithmetic use. This incompatibility between ET and arithmetic outputs makes it difficult for OA to scale efficiently on large databases. We introduce RingOA, the first three-party OA protocol that supports ET while directly producing arithmetic shares. Our method eliminates the need for share conversion and preserves the computational benefits of ET. RingOA achieves a 13.1–15.7× improvement in runtime on databases exceeding one billion entries compared to a state-of-the-art OA protocol. Building on RingOA, we construct an oblivious rank query, a core primitive underlying many structured-data analyses. We develop two practical applications: a fully oblivious full-text search protocol for pattern matching over secret-shared string datasets, and a fully oblivious range-search protocol supporting statistical queries over secret-shared numerical sequences. Experiments on real large-scale genomic datasets show that these applications achieve practical performance, demonstrating the utility of our OA protocol and its applications.

Keywords

multi-party computation, distributed point functions, oblivious access, privacy-preserving search, full-text search, range search

1 Introduction

The increasing use of sensitive information in data-driven systems has prompted a strong demand for efficient privacy-preserving data analysis based on secure multi-party computation (MPC) [4, 11, 26]. Such analyses often require oblivious retrieval of data elements that are subsequently used in downstream computation [13, 16, 24, 36, 37]. In clinical research, for example, analyses often begin by retrieving sensitive information, such as accessing specific genetic variants in a database, and computing relevant statistics [15, 23]. As datasets continue to grow, this retrieval step becomes a dominant bottleneck, motivating more efficient *oblivious access* (OA) primitives that retrieve elements from a private database without revealing the queried index. Recently, MPC protocols based on distributed point functions (DPFs) have been actively studied to accelerate this costly retrieval step [2, 13, 16, 24, 35, 40, 43–45, 47]. These protocols substantially reduce round complexity and communication overhead compared to earlier approaches without DPFs [25, 27, 29, 30, 37, 41, 42], but still incur local computation that grows linearly with the size of the database. To further reduce this local computation, an *early termination* (ET) optimization can be applied [8]. Its benefit increases as the DPF output domain becomes smaller and is maximized when reduced to a single bit. Applying ET to OA in this setting yields Boolean-shared retrieval results. However, OA is typically used as a subroutine in downstream analyses, where retrieved values are ideally represented as arithmetic shares. This necessitates Boolean-to-arithmetic share conversion, introducing additional rounds and communication overhead. Existing OA protocols reflect this trade-off. Protocols that operate entirely in the arithmetic domain evaluate the DPF directly as arithmetic shares, which limits the benefit of ET [24, 44]. In contrast, protocols that support ET rely on Boolean shares, which makes share conversion unavoidable [2]. As a result, it remains an open question whether an OA protocol can simultaneously (i) apply ET to reduce local computation and (ii) produce arithmetic shares without relying on additional share-conversion steps.

In this study, we present *ring-based oblivious access* (RingOA), a three-party OA protocol that fully exploits ET while returning outputs directly as arithmetic shares. Our key idea is a technique enabling single-bit DPF outputs to be interpreted in the arithmetic domain without any share-conversion step. This enables us to obtain the maximum computational benefit of ET and to avoid the overhead associated with Boolean-to-arithmetic conversion, thereby realizing efficient oblivious access directly in the arithmetic domain. To demonstrate the practical utility of RingOA, we further introduce a set of RingOA-based protocols for structured-data analysis. Succinct data structures provide compact indexed representations for various data types such as texts and graphs, where many high-level queries reduce to *rank* operations on underlying bit vectors. Motivated by this foundational role of rank, we first construct an *oblivious rank query* on top of RingOA, and then build two higher-level search protocols upon it. The first is a *fully oblivious full-text search* protocol for pattern matching over secret-shared string datasets. The second is a *fully oblivious range-search* protocol that supports minimum and maximum queries over a secret-shared interval in numerical sequences. Applied to real human genome sequences and genetic-variant datasets, these protocols achieve practical performance in realistic scenarios.

Proceedings on Privacy Enhancing Technologies 2026(4), 1088–1105

https://doi.org/10.56553/popets-2026-0161

1.1 Contribution

In this work, we make the following key contributions:

- **Design of RingOA.** We introduce RingOA, the first three-party oblivious access protocol that fully exploits early termination of single-bit DPFs while outputting results directly in the arithmetic domain. This removes the need for costly share conversion and enables fast access to very large databases. We also discuss a restricted round-optimized variant applicable in single-use or refreshed-preprocessing settings.
- **Applications to structured-data analysis.** Using RingOA, we first construct an efficient oblivious rank query that serves as a core primitive for privacy-preserving analysis. We then extend this primitive to support rank queries on numerical sequences by incorporating the wavelet matrix, a succinct data structure for indexing sequences. Building upon this extended primitive, we design two practical protocols. The first is a fully oblivious full-text search protocol that employs an FM-index implemented with the wavelet matrix, enabling variable-length substring search while hiding both the query pattern and the underlying database. The second is an oblivious range-search protocol for numerical sequences, also implemented with the wavelet matrix, which supports minimum, maximum, and quantile queries over secret-shared sequences and intervals. Both protocols achieve round and communication costs that depend only on the query length and on the domain size, and do not depend on the size of the database, which is often prohibitively large in practical applications.
- **Implementation and evaluation on real genomic datasets.** We implement RingOA and its applications, and conduct extensive experiments. Our results show a 13.1–15.7× speedup over Ji et al. [24] for oblivious access on databases exceeding one billion entries. Furthermore, our fully oblivious full-text search protocol achieves practical query performance for a real human genome sequence, significantly outperforming prior approaches such as Sudo et al. [41] and Nakagawa et al. [30]. Experiments on a large cancer genetic variant database demonstrate that our oblivious range-search protocol efficiently identifies a representative variant allele frequency within an obliviously selected gene. To foster reproducibility and enable further research in privacy-preserving data analysis, we provide a complete and efficient C++ implementation of RingOA and its applications. The source code is publicly available at https://github.com/u-tmk/RingOA.

1.2 Limitations of Existing Approaches

1.2.1 Boolean-to-Arithmetic Conversion Overhead. Early termination reduces the cost of DPF evaluation when the output domain is small, achieving its maximum computational savings when the output consists of a single bit [8]. In this setting, the retrieved database element is naturally obtained as k-bit Boolean shares, where k denotes the bit-length of the domain. However, oblivious access is typically used as a subroutine in arithmetic MPC protocols, where values are expected to be represented as arithmetic shares. Consequently, Boolean outputs must be converted into arithmetic form via a Boolean-to-arithmetic (B2A) conversion [10, 28]. Recent work by Cheng et al. provides constant-round B2A protocols in the three-party semi-honest setting, including one-round and two-round variants [10]. Despite their low round complexity, these protocols incur preprocessing costs quadratic in the bit-length k. Even widely used frameworks such as ABY3 [28] employ a B2A procedure that requires $1 + \log k$ rounds, which can still be expensive. As a result, existing conversion frameworks do not fully eliminate the overhead stemming from Boolean-domain outputs. This overhead becomes particularly costly when OA is invoked repeatedly, as each call adds an additional conversion step that accumulates into substantial extra cost.

1.2.2 Limitations of Existing DPF-based Oblivious Access Protocols. Protocols that operate entirely in the arithmetic domain, such as Shared OT [24] and DuORAM [35, 44], interpret each DPF output as an arithmetic share, thereby limiting the benefit of early termination. In contrast, Bai et al. proposed an OA protocol based on DPF evaluation over Boolean shares [2]. Because the outputs are Boolean-shared, combining their protocol with arithmetic computations requires a B2A conversion. Since their protocol itself requires two rounds, integrating it with constant-round B2A protocols [10] results in three or four rounds in total, together with substantial preprocessing costs. In their design, the queried index is also represented in the Boolean domain. Once the retrieved value is converted to arithmetic form, the index must be transformed back via an arithmetic-to-Boolean (A2B) conversion when performing additional OA operations, further increasing latency. Generic A2B procedures such as those in ABY3 [28] require $1 + \log k$ rounds, making these repeated conversions particularly costly. Moreover, their protocol masks the index in the Boolean domain as $idx \oplus r$. Unlike arithmetic masking (e.g., $idx + r$), which corresponds to a simple rotation and thus preserves sequential access to the database, Boolean masking changes the accessed positions in a bitwise manner. As a result, the access pattern becomes non-contiguous, preventing sequential memory reads and cache-friendly batched evaluation of the DPF. This loss of spatial locality reduces the practical effectiveness of ET.

2 Related Work

Techniques for obliviously retrieving elements from a database have been studied in several settings, including public-database, private-database, and secret-shared settings. In the public-database setting, private information retrieval (PIR) hides the queried index while accessing data stored on a server. DPF-based constructions enable communication-efficient PIR and have been applied in several privacy-preserving applications [40, 43, 45]. These approaches assume that the database is held in the clear by the server. Oblivious RAM (ORAM) considers the setting in which a client encrypts its database, outsources it to an untrusted server, and hides the access pattern of read and write operations. DPF-based ORAM constructions have been proposed, including schemes for time-series data [13] and digital currency systems [47]. Distributed ORAM (DORAM) enables the realization of general RAM programs within secure multi-party computation (MPC) [16, 35, 44, 47]. In this setting, the database is secret-shared among multiple parties, and both read and write accesses are performed obliviously, allowing computation over distributed data without revealing information beyond the prescribed output. While DORAM supports fully general RAM computation, many practical settings involve distinct data holders and query providers, where the querying party does not modify the

Table 1: Comparison of DPF-based oblivious access protocols.

Protocol	Retrieved Share	Arith. Native	ET Support	Conversion-Free	MPC-Free KeyGen
RingOA	$(2,3)$-RSS (or $(3,3)$-ASS)	✓	✓	✓	✓
Shared OT [24]	$(2,3)$-RSS (or $(3,3)$-ASS)	✓	×	✓	✓
DuORAM [35, 44]	$(2,2)$-ASS	✓	×	✓	×
Zyskind et al. [47]	$(2,3)$-SSS	✓	×	✓	×
Floram [16]	Boolean-based $(2,2)$-ASS	×	✓	×	×
Bai et al. [2]	Boolean-based $(2,3)$-RSS	×	✓	×	✓

Arith. Native indicates whether the retrieved value is directly output as an arithmetic share. *Conversion-Free* indicates that no Boolean-to-arithmetic (B2A) conversion is required for downstream arithmetic computation. *MPC-Free KeyGen* indicates that DPF key generation is performed locally without invoking secure multi-party computation. Abbreviations: RSS = replicated secret sharing, ASS = additive secret sharing, SSS = Shamir's secret sharing. Symbols: ✓ = supported, × = not supported.

underlying database and only requires read access. This has motivated protocols specialized to the read-only setting [2, 24]. In this paper, we refer to such read-only oblivious retrieval as oblivious access (OA). To position our construction among prior DPF-based protocols, Table 1 summarizes their key properties, including the type of retrieved shares, support for arithmetic computation, early termination, and the need for share conversion or MPC-based key generation. Floram [16] first employed DPFs for OA as a building block for DORAM. Their construction relies on garbled circuits to generate DPF keys, leading to substantial communication and computational overhead. DuORAM [44] is a server-aided DORAM protocol in the $(2,2)$-additive secret sharing setting. DPF key generation is executed via MPC, introducing additional preprocessing and communication overhead. Moreover, its Boolean-to-arithmetic share conversion mechanism limits the depth to which early termination can be applied. Sasy et al. [35] extend DuORAM with application-specific optimizations for binary search and AVL trees, retaining the same structural characteristics. Zyskind et al. [47] construct an ORAM based on $(2,3)$-Shamir's secret sharing. Because the DPF output has bit-length equal to the security parameter rather than a single bit, early termination is not applicable. In a DORAM setting, DPF key generation requires MPC, adding overhead.

3 Cryptographic Preliminaries

Notations. Let $\lambda \in \mathbb{Z}$ denote the security parameter. We denote the ring of integers modulo 2^k by $\mathbb{Z}_{2^k}$ and the ring of integers modulo 2 by $\mathbb{Z}_2$ (the Boolean domain). Scalars are denoted by lowercase letters (e.g., x), and vectors by boldface letters (e.g., $\mathbf{x}$), where the i-th element of $\mathbf{x}$ is written as $\mathbf{x}[i]$. For a finite set S, the notation $x \xleftarrow{\$} S$ indicates that x is chosen uniformly at random from S. We consider three parties P_i for $i \in \mathbb{Z}_3$, with indices taken modulo 3. Thus, P_{i-1} and P_{i+1} denote the previous and next party relative to P_i. We use the following secret sharing notations over $\mathbb{Z}_{2^k}$:

- $[x]$: 2-out-of-2 additive secret sharing of $x \in \mathbb{Z}_{2^k}$.
- $[\![x]\!]$: 3-out-of-3 additive secret sharing of $x \in \mathbb{Z}_{2^k}$.
- $\langle x \rangle$: 2-out-of-3 replicated secret sharing of $x \in \mathbb{Z}_{2^k}$.

3.1 Secret Sharing Schemes

In this paper, we use two secret sharing schemes over the ring $\mathbb{Z}_{2^k}$: (1) a t-out-of-t additive secret sharing scheme ($t \in \{2,3\}$), and (2) a 2-out-of-3 replicated secret sharing (RSS) scheme [1]. We briefly describe their definitions and basic operations.

3.1.1 Additive Secret Sharing Scheme. In a 2-out-of-2 additive secret sharing scheme, a secret $x \in \mathbb{Z}_{2^k}$ is represented as two shares x_0, x_1 satisfying $x = x_0 + x_1$ over $\mathbb{Z}_{2^k}$. We denote this by $[x] = ([x]_0, [x]_1)$.

- $[x] \leftarrow \mathcal{F}^{[\cdot]}_{\mathsf{Share}}(x)$: Sample $[x]_0 \xleftarrow{\$} \mathbb{Z}_{2^k}$ and set $[x]_1 = x - [x]_0$. Distributed $[x]_i$ to party P_i.
- $x \leftarrow \mathcal{F}^{[\cdot]}_{\mathsf{Reconst}}([x])$: Each party broadcasts its share, and the secret is reconstructed as $x = [x]_0 + [x]_1$.
- $[z] \leftarrow \mathcal{F}^{[\cdot]}_{\mathsf{Add}}([x], [y])$: Local addition of two shares $[x]$ and $[y]$, resulting in a new share $[z]$ such that $z = x + y$.
- $[z] \leftarrow \mathcal{F}^{[\cdot]}_{\mathsf{Mult}}([x], [y])$: Secure multiplication of two additive shares $[x]$ and $[y]$, resulting in a new share $[z]$ such that $z = x \cdot y$. This operation requires one round of interaction with Beaver's triple [3]. Given a triple $([a], [b], [c])$ satisfying $c = a \cdot b$, the parties reconstruct $d = x - a$ and $e = y - b$, and locally compute $[z] = e \cdot [a] + d \cdot [b] + [c] + i \cdot de$ for $i \in \{0, 1\}$.

3.1.2 Replicated Secret Sharing Scheme. In a 2-out-of-3 replicated secret sharing scheme over $\mathbb{Z}_{2^k}$, a secret x is shared among three parties P_0, P_1, P_2 such that each P_i holds a pair $([\![x]\!]_i, [\![x]\!]_{i-1})$ where $x = [\![x]\!]_0 + [\![x]\!]_1 + [\![x]\!]_2$. In this paper, we use the notation $\langle x \rangle_i = ([\![x]\!]_i, [\![x]\!]_{i-1})$ to denote the share of P_i.

- $\langle x \rangle \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Share}}(x)$: Sample $[\![x]\!]_0, [\![x]\!]_1 \xleftarrow{\$} \mathbb{Z}_{2^k}$ and set $[\![x]\!]_2 = x - [\![x]\!]_0 - [\![x]\!]_1$. Distribute $\langle x \rangle_i = ([\![x]\!]_i, [\![x]\!]_{i-1})$ to party P_i.
- $x \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Reconst}}(\langle x \rangle)$: Party P_i sends $[\![x]\!]_{i-1}$ to P_{i+1}. Then, P_i reconstructs the secret as $x = [\![x]\!]_0 + [\![x]\!]_1 + [\![x]\!]_2$ for $i \in \{0, 1, 2\}$.
- $x \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Reveal}}(\langle x \rangle, i)$: Only party P_i learns the secret by receiving $[\![x]\!]_{i+1}$ from P_{i+1}, reconstructing $x = [\![x]\!]_0 + [\![x]\!]_1 + [\![x]\!]_2$.
- $\langle z \rangle \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Add}}(\langle x \rangle, \langle y \rangle)$: P_i locally computes $[\![z]\!]_i = [\![x]\!]_i + [\![y]\!]_i$ and $[\![z]\!]_{i-1} = [\![x]\!]_{i-1} + [\![y]\!]_{i-1}$, resulting in new share $\langle z \rangle_i = ([\![z]\!]_i, [\![z]\!]_{i-1})$ such that $z = x + y$ for $i \in \{0, 1, 2\}$.
- $\langle z \rangle \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Mult}}(\langle x \rangle, \langle y \rangle)$: For $i \in \{0, 1, 2\}$, P_i locally computes $[\![z]\!]_i = [\![x]\!]_i [\![y]\!]_i + [\![x]\!]_{i-1} [\![y]\!]_i + [\![x]\!]_i [\![y]\!]_{i-1} + [\![r]\!]_i$, where $[\![r]\!]_0 + [\![r]\!]_1 + [\![r]\!]_2 = 0$. Each pair of parties (P_i, P_{i-1}) shares a PRF key k_i^{prf}, and each P_i computes $[\![r]\!]_i = F(k_i^{\mathsf{prf}}, \mathsf{sid}) - F(k_{i-1}^{\mathsf{prf}}, \mathsf{sid})$. Then P_i sends $[\![z]\!]_i$ to P_{i+1} and receives $[\![z]\!]_{i-1}$ from P_{i-1}, and sets $\langle z \rangle_i = ([\![z]\!]_i, [\![z]\!]_{i-1})$.
- $\langle z \rangle \leftarrow \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Select}}(\langle x \rangle, \langle y \rangle, \langle b \rangle)$: Compute $\langle z \rangle = \langle x \rangle + \mathcal{F}^{\langle \cdot \rangle}_{\mathsf{Mult}}(\langle y \rangle - \langle x \rangle, \langle b \rangle)$, which results in $z = x$ if $b = 0$ and $z = y$ if $b = 1$.

3.1.3 Security Model and Setting. We consider a three-party computation setting in which at most one party may be corrupted by a

static semi-honest adversary, ensuring an honest majority [1, 28]. We assume an outsourced setting in which a data owner distributes the database [24]. The data owner is also semi-honest and does not collude. Parties follow the prescribed protocol but may attempt to infer additional information from their received messages. Our construction follows the standard preprocessing model, where input independent correlated randomness is generated before the online phase. The security of our scheme is defined in the standard simulation-based framework for three-party computation; a formal definition is provided in Appendix A.

3.2 Distributed Point Functions

A distributed point function (DPF) [7, 8, 22] enables two parties to hold compact keys that define a point function without revealing the target index. Formally, for an input domain $\{0,1\}^k$ and an output domain $\mathbb{G}$, a point function $f_{\alpha,\beta} : \{0,1\}^k \to \mathbb{G}$ is defined as

$$f_{\alpha,\beta}(x) = \begin{cases} \beta & \text{if } x = \alpha, \\ 0 & \text{otherwise,} \end{cases}$$

where $\alpha \in \{0,1\}^k$ is the target index and $\beta \in \mathbb{G}$ is the designated value. A DPF consists of two algorithms (Gen, Eval) such that:

- $\mathsf{Gen}(\alpha, \beta) \to (k_0, k_1)$: on input (α, β), outputs a pair of keys for the two parties P_0 and P_1.
- $\mathsf{Eval}(b, k_b, x) \to y_b$: on input the party identifier $b \in \{0,1\}$, the corresponding key k_b, and an input $x \in \{0,1\}^k$, outputs a share $y_b \in \mathbb{G}$ such that $y_0 + y_1 = f_{\alpha,\beta}(x)$ for all x.

Let $N = 2^k$ denote the size of the input domain. In DPF-based DORAM, oblivious access is realized by evaluating the DPF over the entire domain to obtain the corresponding unit vector, and then computing its inner product with the database to extract the desired element. If the database size is smaller than N, it suffices to evaluate the DPF only over the indices corresponding to the database. For simplicity, we assume that the database size equals N.

3.2.1 Full-Domain Evaluation and Early Termination. To obtain a secret-shared unit vector $(\mathbf{u}[0], \ldots, \mathbf{u}[N-1])$ with $\mathbf{u}[\alpha] = 1$ and $\mathbf{u}[i] = 0$ for $i \neq \alpha$, each party evaluates the DPF on all inputs. While a naive approach takes $O(kN)$ time, the EvalAll procedure computes all outputs in $O(N)$ time by expanding the evaluation tree level by level [8]. Early termination (ET) reduces the depth of DPF evaluation when the output domain $\mathbb{G}$ is small [8]. Each node in the evaluation tree carries a λ-bit seed, from which the final output mask is derived. If the output requires only $\log_2 |\mathbb{G}|$ bits and $\log_2 |\mathbb{G}| < \lambda$, then a single λ-bit seed can generate $\lambda / \log_2 |\mathbb{G}|$ independent output values. Hence, the evaluator need not expand the tree to its full depth and terminate the evaluation early. More concretely, the evaluator can skip the final $\log_2(\lambda / \log_2 |\mathbb{G}|)$ levels of the tree, thereby reducing the number of PRF invocations. For example, when $\lambda = 128$ and $\mathbb{G} = \mathbb{Z}_2$, a single seed generates 128 output bits, so 7 levels of the tree can be omitted. The computational reduction is maximized when $\mathbb{G} = \mathbb{Z}_2$.

4 Oblivious Access via Single-Bit DPFs

Our goal is to realize a three-party oblivious access protocol that retains the early termination advantages of single-bit DPFs while directly producing arithmetic shares. In oblivious access, the queried position is encoded as a unit vector that has a single 1 at the target index and 0 elsewhere. We employ the subtraction-based reconstruction of single-bit DPFs, previously used in the two-party setting for public database [45]. This reconstruction yields shares of a unit vector that becomes a signed unit vector taking value +1 or −1 at the queried position. Using this signed vector preserves the magnitude of the selected entry but may flip its sign, which must therefore be corrected within the share space. Such correction was efficient in prior work because the database was public.

We construct RingOA by adapting this reconstruction to the three-party RSS setting with a private database. In this setting, the sign of the retrieved value remains unresolved, requiring one communication round for correction. For single-use or refreshed-preprocessing settings, we introduce a restricted optimization that allows each party to resolve the sign locally by employing a modified database-sharing scheme, which we call *free sign correction* (FSC). In our setting, both the database $\mathbf{D}$ and the index idx are shared in $(2,3)$-RSS over $\mathbb{Z}_{2^k}$. The ideal functionality realized by RingOA is presented in Functionality 1.

Functionality 1. ($\mathcal{F}_{\mathrm{OA}}$ Oblivious Access).
Upon receiving a share of database $\langle \mathbf{D} \rangle$ and an index $\langle idx \rangle$, $\mathcal{F}_{\mathrm{OA}}$ reconstructs $\mathbf{D}$ and idx, computes $d \leftarrow \mathbf{D}[idx]$, generates shares $\langle d \rangle$, and sends $\langle d \rangle_i$ to P_i.

4.1 Protocol Overview and Workflow

An overview of the workflow is illustrated in Figure 1. The RingOA protocol consists of the following three main steps:

(1) **Local Inner Product:** Each party evaluates its received single-bit DPF key to obtain a Boolean share of the unit vector. After applying subtractive reconstruction, the parties obtain a signed unit vector in $\mathbb{Z}_{2^k}$. Each party then takes the local inner product between this vector and the replicated database share it holds, resulting in either a positive or negative share of the target element.
(2) **Sign Correction:** The values obtained in Step 1 may carry a ± 1 sign due to the subtractive reconstruction. Each party uses the correction value precomputed during DPF key generation to remove this ambiguity and recovers a share of the database element.
(3) **Resharing:** After sign correction, each party holds two shares of the accessed element. These two values are added to form a $(3,3)$-ASS, which is then converted into a $(2,3)$-RSS via a resharing procedure, yielding the final RSS share usable in subsequent ring-based computations.

Step 1: Local Inner Product. For party identifier $i \in \{0,1,2\}$ (mod 3), parties P_i and P_{i+1} hold a common share $[\![\mathbf{D}]\!]_i$. In Figure 1, the same color indicates the same replicated share. To access the element at a secret index idx, the party P_{i+2} generates a pair of single-bit DPF keys $(k_0, k_1) \leftarrow \mathsf{Gen}(idx, 1)$ and sends k_0 to P_i and k_1 to P_{i+1}. [1] Each party locally evaluates its received key to obtain a Boolean share $[\mathbf{u}]_b$ of the unit vector, where $b \in \{0,1\}$ corresponds to the next and previous party, respectively. To apply the subtractive

[1] To hide the actual index, P_{i+2} uses a masked index with a random offset; see Section 4.2.

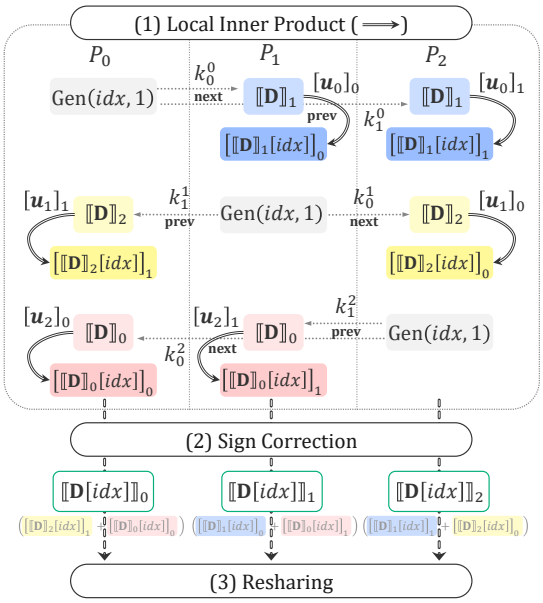

Figure 1: Overview and Workflow of the RingOA.

reconstruction described earlier, P_{i+1} simply negates its Boolean share, replacing $[\mathbf{u}]_1$ with $-[\mathbf{u}]_1$. This yields a signed unit vector in $\mathbb{Z}_{2^k}$ that takes value ± 1 at the queried index and 0 elsewhere. Then, P_i computes the inner product between $[\![\mathbf{D}]\!]_i$ and $[\mathbf{u}]_0$, while P_{i+1} computes the inner product between $[\![\mathbf{D}]\!]_i$ and $-[\mathbf{u}]_1$. As shown by the double arrows in Figure 1, this yields for each $b \in \{0,1\}$ a value $[[\![\mathbf{D}]\!]_i[idx]]_b$ (or $[-[\![\mathbf{D}]\!]_i[idx]]$) held by P_{i+b}.

Step 2: Sign Correction. After Step 1, each party holds either $[[\![\mathbf{D}]\!]_i[idx]]$ or $[-[\![\mathbf{D}]\!]_i[idx]]$, depending on whether the subtractive reconstruction flipped its sign. This ambiguity can be resolved using information available at DPF key generation. While prior work notes that the final control bit determines this sign [45], the exact correction procedure is not described. We provide a concrete derivation that uses both the final-layer seeds and final control bits. Let $s_b \in \{0,1\}^\lambda$ and $t_b \in \{0,1\}$ denote the final seed and control bit held by party P_b. For an input α, let $\hat{\alpha}$ be its lower v bits, where v is the early termination depth. Define $\gamma \in \{0,1\}^\lambda$ as the bit string that has a 1 at position $\hat{\alpha}$ and 0 elsewhere. The output of the DPF evaluation at party P_b can be written as

$$y_b = (s_b \oplus t_b \cdot CW)[\hat{\alpha}], \quad \text{with } CW = \gamma \oplus s_0 \oplus s_1.$$

The expressions above for y_0 and y_1 specify, for each combination of final seeds and control bits, whether the sign is $+1$ or -1. This allows us to define the *sign correction value* $w \in \{\pm 1\}$ explicitly, as shown in Table 2. Multiplying the value obtained in Step 1 by this share of w yields a correctly signed output, completing the sign correction.

Step 3: Resharing. After sign correction, each party P_i holds two shares of the accessed element: $[[\![\mathbf{D}]\!]_i[idx]]$ from the computation with P_{i+1} and $[[\![\mathbf{D}]\!]_{i+2}[idx]]$ from the computation with P_{i+2}. (For instance, when $i = 0$, the two values obtained from P_1 and P_2 correspond to the red and yellow arrows in Figure 1.) Adding these two values yields a $(3,3)$-ASS $[\![\mathbf{D}[idx]]\!]_i$ of the target element. To convert this into the standard $(2,3)$-RSS form over $\mathbb{Z}_{2^k}$, the parties execute a PRF-based resharing procedure [1]. The resulting RSS shares $\langle \mathbf{D}[idx] \rangle$ are immediately usable in subsequent arithmetic computations over $\mathbb{Z}_{2^k}$, completing the oblivious access protocol.

Table 2: Sign correction table.

	$t_0 = 1$ $(t_1 = 0)$		$t_0 = 0$ $(t_1 = 1)$	
	$s_1[\hat{\alpha}] = 0$	$s_1[\hat{\alpha}] = 1$	$s_0[\hat{\alpha}] = 0$	$s_0[\hat{\alpha}] = 1$
(y_0, y_1)	$(1,0)$	$(0,1)$	$(0,1)$	$(1,0)$
w	$+1$	-1	-1	$+1$

4.2 Ring-Based Oblivious Access

The RingOA protocol, described in Algorithm 4.1, proceeds in two phases: the preprocessing phase and the online phase.

4.2.1 Preprocessing Phase. Each party P_i samples a random mask $r_i \xleftarrow{\$} \mathbb{Z}_{2^k}$, which will later be used to hide the true query index. Using this mask, P_i generates a pair of single-bit DPF keys $(k_0^i, k_1^i) \leftarrow \mathrm{Gen}(r_i, 1)$ encoding a unit vector with a 1 at position r_i. During key generation, P_i also computes a sign correction value $w_i \in \{\pm 1\}$, which resolves the ± 1 ambiguity introduced by the subtractive reconstruction method. As summarized in Table 2, w_i is given by

$$w_i = (-1)^{s_{(t_1 \oplus 1)}[\hat{r}_i] \oplus t_1}, \tag{1}$$

where $\hat{r}_i$ denotes the lower v bits of r_i. Finally, both r_i and w_i are secret-shared using $(2,2)$-ASS via $\mathcal{F}_{\mathrm{Share}}^{[\cdot]}$, yielding shares $[r_i]$ and $[w_i]$ distributed to P_{i+1} and P_{i+2}.

4.2.2 Online Phase. Given a secret-shared index $\langle idx \rangle$, each party P_i first uses the random masks r_{i+1} and r_{i+2} to compute the masked shift values $\hat{p}_{i+1} = idx - r_{i+1}$ and $\hat{p}_{i+2} = idx - r_{i+2}$ in replicated form (Lines 1–2). These values are then reconstructed using $\mathcal{F}_{\mathrm{Reveal}}^{\langle\cdot\rangle}$: $\hat{p}_{i+1}$ is reconstructed by P_i and P_{i+2} but hidden from P_{i+1}, and symmetrically for $\hat{p}_{i+2}$ (Line 3). Next, each party evaluates its received DPF key over the full domain to obtain a Boolean share of a unit vector with the 1 located at the random mask positions r_{i+1} or r_{i+2} (Lines 5–6). Before computing the local inner product (Lines 8–9), the database shares are cyclically shifted by the reconstructed shift amounts $\hat{p}_{i+1}$ and $\hat{p}_{i+2}$, ensuring correct alignment with the queried index. In Line 8, each party computes the negated inner product between its Boolean share $[\mathbf{u}_{i+1}]$ and the appropriately shifted database share, thereby realizing the subtractive reconstruction method. This produces either $[[\![\mathbf{D}]\!]_{i+2}[idx]]$ or $[-[\![\mathbf{D}]\!]_{i+2}[idx]]$. Each party then applies its sign correction value w via the secure multiplication functionality $\mathcal{F}_{\mathrm{Mult}}^{[\cdot]}$ to remove the ± 1 ambiguity. After sign correction, each party holds two additive shares of the accessed element. These shares are combined and converted into the final RSS format using the resharing step (Step 3), completing the oblivious access protocol.

Theorem 4.1. *The RingOA protocol Π_{RingOA} securely computes $\mathcal{F}_{\mathrm{OA}}$ in the $\left(\mathcal{F}_{\mathrm{Share}}^{[\cdot]}, \mathcal{F}_{\mathrm{Reveal}}^{\langle\cdot\rangle}, \mathcal{F}_{\mathrm{Mult}}^{[\cdot]}\right)$-hybrid model in the presence of*

Algorithm 4.1 Ring-Based Oblivious Access Protocol Π_{RingOA}

Parameter: An RSS share of database $\langle \mathbf{D} \rangle$ where each element $\mathbf{D}[i] \in \mathbb{Z}_{2^k}$ for $i \in [N]$, N denotes the size of the database. An RSS share of index $\langle idx \rangle$ for $idx \in [N]$. (P_i, P_{i+1}) hold a common key $k_i^{\text{prf}} \leftarrow \{0,1\}^\lambda$. A common session identifier sid.

Preprocessing: $P_i, i \in \{0,1,2\}$ does:

1: Sample randomness: $r_i \xleftarrow{\$} \mathbb{Z}_{2^k}$.
2: Generate DPF keys: $(k_0^i, k_1^i) \leftarrow \text{Gen}(r_i, 1)$.
3: Compute sign correction: $w_i \leftarrow (-1)^{s_{(t_1 \oplus 1)}[\hat{r}_i] \oplus t_1}$, where $\hat{r}_i = r_i[v+1], \ldots, r_i[k]$ and s_b, t_b are the final-layer seed and control bit ($b \in \{0,1\}$).
4: Share randomness: $([r_i]_0, [r_i]_1) \leftarrow \mathcal{F}_{\text{Share}}^{[\cdot]}(r_i)$.
5: Share sign correction: $([w_i]_0, [w_i]_1) \leftarrow \mathcal{F}_{\text{Share}}^{[\cdot]}(w_i)$.
6: Send $(k_0^i, [r_i]_0, [w_i]_0)$ to P_{i+1} and $(k_1^i, [r_i]_1, [w_i]_1)$ to P_{i+2}.
7: Receive $(k_1^{i+1}, [r_{i+1}]_1, [w_{i+1}]_1)$ from P_{i+1}.
8: Receive $(k_0^{i+2}, [r_{i+2}]_0, [w_{i+2}]_0)$ from P_{i+2}.

Online: $P_i, i \in \{0,1,2\}$ does:

1: Set $\langle r_{i+1} \rangle_i := (0, [r_{i+1}]_1)$ and $\langle r_{i+2} \rangle_i := ([r_{i+2}]_0, 0)$.
2: Compute $\langle \hat{p}_{i+1} \rangle_i := \langle idx \rangle_i - \langle r_{i+1} \rangle_i, \langle \hat{p}_{i+2} \rangle_i := \langle idx \rangle_i - \langle r_{i+2} \rangle_i$.
3: Reconstruct $\hat{p}_{i+1}$ with P_{i+2} and $\hat{p}_{i+2}$ with P_{i+1} using $\mathcal{F}_{\text{Reveal}}^{\langle \cdot \rangle}$.
4: */* Full-domain evaluation of DPFs. */*
5: Set $[\mathbf{u}_{i+1}]_1 \leftarrow \text{EvalAll}(1, k_1^{i+1})$.
6: Set $[\mathbf{u}_{i+2}]_0 \leftarrow \text{EvalAll}(0, k_0^{i+2})$.
7: */* Compute local inner product. */*
8: Set $[\pm[\![\mathbf{D}]\!]_{i+2}[idx]]_1 \leftarrow -\sum_{j=0}^{N-1} [\![\mathbf{D}]\!]_{i+2}[(j + \hat{p}_{i+1})] \cdot [\mathbf{u}_{i+1}]_1[j]$.
9: Set $[\pm[\![\mathbf{D}]\!]_i[idx]]_0 \leftarrow \sum_{j=0}^{N-1} [\![\mathbf{D}]\!]_i[(j + \hat{p}_{i+2})] \cdot [\mathbf{u}_{i+2}]_0[j]$.
10: */* Apply sign correction. */*
11: Set $[[\![\mathbf{D}]\!]_{i+2}[idx]] \leftarrow \mathcal{F}_{\text{Mult}}^{[\cdot]}([\pm[\![\mathbf{D}]\!]_{i+2}[idx]], [w_{i+1}])$.
12: Set $[[\![\mathbf{D}]\!]_i[idx]] \leftarrow \mathcal{F}_{\text{Mult}}^{[\cdot]}([\pm[\![\mathbf{D}]\!]_i[idx]], [w_{i+2}])$.
13: */* Resharing. */*
14: Set $[\![\mathbf{D}[idx]]\!]_i \leftarrow [[\![\mathbf{D}]\!]_{i+2}[idx]]_1 + [[\![\mathbf{D}]\!]_i[idx]]_0$.
15: Set $[\![d]\!]_i \leftarrow [\![\mathbf{D}[idx]]\!]_i + F(k_i^{\text{prf}}, \text{sid}) - F(k_{i+2}^{\text{prf}}, \text{sid})$.
16: Send $[\![d]\!]_i$ to P_{i+1} and receive $[\![d]\!]_{i+2}$ from P_{i+2}.
17: **return** $\langle d \rangle_i = ([\![d]\!]_i, [\![d]\!]_{i+2})$.

a semi-honest adversary in the three-party honest-majority setting, assuming F is a secure PRF.

Correctness. We show that Π_{RingOA} correctly realizes $\mathcal{F}_{\text{OA}}$. Let $[\mathbf{u}_{i+1}]$ and $[\mathbf{u}_{i+2}]$ denote the Boolean shares of the unit vectors generated by the DPF evaluation, where $\mathbf{u}_{i+1}[r_{i+1}] = 1$ and $\mathbf{u}_{i+2}[r_{i+2}] = 1$, and all other positions are zero. Since the masked shift values satisfy $\hat{p}_{i+1} = idx - r_{i+1}$, cyclically shifting the database share by $\hat{p}_{i+1}$ aligns the 1 in $\mathbf{u}_{i+1}$ with the true query index. Formally, the local inner product computes

$$[\![\mathbf{D}]\!]_{i+2}[\, j + \hat{p}_{i+1}\,] \cdot [\mathbf{u}_{i+1}][j] = [\![\mathbf{D}]\!]_{i+2}[\, j + (idx - r_{i+1})\,] \cdot [\mathbf{u}_{i+1}][j],$$

which evaluates to $[\![\mathbf{D}]\!]_{i+2}[idx]$ exactly when $j = r_{i+1}$, the unique position at which $\mathbf{u}_{i+1}[j] = 1$. An identical argument applies to $\hat{p}_{i+2}$ and $\mathbf{u}_{i+2}$. As described in the protocol, the subtractive reconstruction produces values of the form $[\pm[\![\mathbf{D}]\!]_{i+2}[idx]]$ from the computation with P_{i+1} and $[\pm[\![\mathbf{D}]\!]_i[idx]]$ from the computation with P_{i+2}. Multiplying by the sign correction value w removes the ± 1 ambiguity, leaving correctly signed additive shares of the accessed element. Summing these values yields a $(3,3)$-ASS of $[\![\mathbf{D}[idx]]\!]$, which is then re-randomized and converted into a $(2,3)$-RSS through the resharing step. Thus, the protocol outputs $\langle \mathbf{D}[idx] \rangle$, exactly as required by $\mathcal{F}_{\text{OA}}$.

Security Proof Sketch. We sketch the security of Π_{RingOA} in the $(\mathcal{F}_{\text{Share}}^{[\cdot]}, \mathcal{F}_{\text{Reveal}}^{\langle \cdot \rangle}, \mathcal{F}_{\text{Mult}}^{[\cdot]})$-hybrid model against a semi-honest adversary corrupting one party. The adversary's view consists of its input and output shares, together with all messages it receives during the execution of the protocol. The query index idx is always held in $(2,3)$-RSS form and is never revealed. When masked shift values $\hat{p} = idx - r_i$ are reconstructed, the independently sampled masks r_i ensure that each $\hat{p}$ is uniformly distributed and reveals nothing about idx. By DPF security, the keys (k_0^i, k_1^i) and their evaluation outputs are computationally indistinguishable from uniform. The sign correction values w_i, derived only from final DPF seeds and control bits and shared additively, are also independent of the query. The PRF-generated values used in the resharing step ensure that the final $(2,3)$-RSS shares $\langle d \rangle$ reveal only $\mathbf{D}[idx]$. Thus, a simulator can sample uniform values consistent with $\mathcal{F}_{\text{OA}}$ and generate a view indistinguishable from that of any corrupted party, proving that Π_{RingOA} securely realizes $\mathcal{F}_{\text{OA}}$ in the semi-honest setting.

4.3 Restricted Free Sign Correction

In the standard RingOA protocol, correcting the sign of the retrieved value requires a secure multiplication via $\mathcal{F}_{\text{Mult}}^{[\cdot]}$, which incurs an additional communication round. Free sign correction (FSC) is a restricted optimization for settings in which the preprocessed flipped database sharing is used only once or refreshed for each invocation. Under this restriction, FSC eliminates this extra round by masking the sign correction with random sign flips embedded in the database sharing.

We give the construction and its correctness argument in Appendix B. Since the applications considered in this paper invoke OA repeatedly, the remainder of the paper uses the standard RingOA protocol.

4.4 Complexity Analysis

RingOA requires $O(N)$ local computation for full-domain DPF evaluation and $O(1)$ communication in the online phase. These online costs are summarized in Table 3. An ET depth of 7 reduces the number of PRF evaluations by a factor of 2^7, which substantially lowers the dominant local computation cost. For comparison, we report round complexity assuming that all schemes output RSS shares, although Shared OT [24] is not originally defined with an RSS output. We also include the construction of Bai et al. [2] combined with a B2A conversion, which requires at least three rounds. Although DuORAM [35, 44] describes its online phase as a single round, its protocol involves two sequentially dependent message steps. Hence, it incurs two sequential communication delays in practice. The preprocessing phase corresponds to DPF key generation. All protocols considered here require $O(\log N)$ computation and $O(\log N)$ communication. Moreover, except for DuORAM, key

Table 3: Online complexity of oblivious access protocols.

Protocol	Comp.	Comm.	Rounds	ET Depth
RingOA	$O(N)$	$O(1)$	3	7
Shared OT [24]	$O(N)$	$O(1)$	2	0*
DuORAM [35, 44]	$O(N)$	$O(1)$	1†	0*
Bai et al. [2] + B2A [10]	$O(N)$	$O(1)$	≥ 3	7

* These protocols do not support the ET in their original form. For certain parameters (e.g., $k = 32$), ET depth is at most 2.

† Although counted as one round in the original paper [44], the protocol consists of two dependent message flows, $P_0, P_1 \to P_2$ and $P_2 \to P_0, P_1$, resulting in latency of at least $2q$ for one-way delay q.

generation can be completed in a single round. DuORAM instead performs key generation via MPC, resulting in $O(\log N)$ rounds.

5 Applications

Various high-level data types can be represented using so-called succinct data structures [32], which provide compact representations of data while still supporting constant- or logarithmic-time queries. Such structures have been developed for a wide range of data, including texts [18], arrays [12], trees [14], and graphs [5, 19]. These representations are built primarily from collections of bit vectors, and queries on the original data are carried out by applying simple operations to these bit vectors. Among these operations, the *rank query* is a fundamental operation that counts the occurrences of a bit in a prefix of a bit vector. In this work, we use the *wavelet matrix* [12] to support rank operations on numerical sequences by representing them with multiple levels of bit vectors. It transforms the input into a hierarchy of bit vectors, and its queries are executed by traversing these levels and repeatedly computing rank operations. A detailed description of the wavelet matrix and its query algorithms is provided in Appendix C. Since rank is a fundamental operation underlying many succinct data structures, an efficient oblivious rank primitive enables secure evaluation of a broad range of queries. Motivated by this, we first develop an efficient *oblivious rank* protocol for secret-shared bit vectors using RingOA. Combining this primitive with the wavelet matrix yields two applications: (i) fully oblivious full-text search and (ii) fully oblivious range search.

5.1 Oblivious Rank

Given a bit vector $B \in \{0,1\}^N$ and a position $p \in [N]$, the rank operation $\mathsf{Rank}_b(B, p)$ returns the number of occurrences of b in the prefix $B[0, p-1]$. Our goal is to compute this value when both B and p are secret-shared in the three-party RSS setting. We focus on the case $b = 0$ and design the *oblivious rank0* (OR0) protocol, which securely outputs the number of zeros in the prefix $B[0, p-1]$ given RSS shares of B and p. The ideal functionality for the OR0 is defined in Functionality 2.

Functionality 2. ($\mathcal{F}_{\mathsf{Rank0}}$ — Oblivious Rank0).
Upon receiving shares of a bit vector $\langle B \rangle$ and a position $\langle p \rangle$, the functionality $\mathcal{F}_{\mathsf{Rank0}}$ reconstructs B and p, computes $z \leftarrow \mathsf{Rank}_0(B, p)$, generates shares $\langle z \rangle$, and sends $\langle z \rangle_i$ to P_i.

Algorithm 5.1 Oblivious Rank0 Protocol Π_{Rank0}

Input: RSS shares of the bit vector $\langle B \rangle$ and query position $\langle p \rangle$.
Output: RSS share of the $\langle rank0 \rangle$ where $rank0 = \mathsf{Rank}_0(B, p)$.

Set $\langle \mathbf{D}_{\mathsf{zero}}[0] \rangle \leftarrow \langle 0 \rangle$.
for $i = 0$ to $N - 1$ **do**
  $\langle \mathbf{D}_{\mathsf{zero}}[i+1] \rangle \leftarrow \langle \mathbf{D}_{\mathsf{zero}}[i] \rangle + (1 - \langle B[i] \rangle)$
$\langle rank0 \rangle \leftarrow \mathcal{F}_{\mathsf{OA}}(\langle \mathbf{D}_{\mathsf{zero}} \rangle, \langle p \rangle)$
return $\langle rank0 \rangle$

Protocol Description. Algorithm 5.1 presents our oblivious rank0 protocol (OR0), which computes $\mathsf{rank0} = \mathsf{Rank}_0(B, p)$ from RSS shares of a bit vector B and a query position p. The parties first construct a *zero table* $\mathbf{D}_{\mathsf{zero}}$ of length $N + 1$ by computing the prefix sum $\mathbf{D}_{\mathsf{zero}}[0] = 0$ and $\mathbf{D}_{\mathsf{zero}}[i+1] = \mathbf{D}_{\mathsf{zero}}[i] + (1 - B[i])$ in a single pass over the shared bits. Once the table is constructed, the parties use RingOA to obliviously retrieve the value $\mathbf{D}_{\mathsf{zero}}[p]$, obtaining an RSS share of rank0. Because all prefix computations are local and the protocol performs only one oblivious-access call, the overall cost of OR0 essentially matches that of a single RingOA invocation.

Correctness is immediate from the definition of $\mathbf{D}_{\mathsf{zero}}[p]$, and all remaining steps are local RSS operations. Thus, Π_{Rank0} is secure in the $\mathcal{F}_{\mathsf{OA}}$-hybrid model.

Extensions. The value *rank1* can be computed directly as $rank1 = p - rank0$. Therefore, the general Rank_b protocol for any $b \in \{0,1\}$ follows immediately from OR0, and only the zero table $\mathbf{D}_{\mathsf{zero}}$ needs to be stored. A full description of the general Rank_b construction is provided in Appendix D. The rank operation is also closely related to the *select* query, which returns the position of the $(j+1)$-th occurrence of a bit. Although select is not the focus of this work, it can be supported by constructing a similar lookup table.

5.2 Fully Oblivious Full-Text Search

Full-text search locates all occurrences of a given query string in a large text and is widely used in domains such as natural language processing, genome sequence analysis, and large-scale log processing. In many real-world scenarios, both the query and the database must remain confidential; for example, in a clinical study, a researcher may wish to query a patient's genome sequence against a biomedical database maintained by another stakeholder, where ideally both inputs remain private [17, 33]. This dual privacy requirement motivates a *fully oblivious full-text search*, in which neither the query nor the database is revealed during the search, and only the final match results are disclosed. The FM-index [18] is a succinct data structure of text that supports pattern matching in $O(\ell)$ time for a query of length ℓ, and it underlies many practical systems such as large-scale genome sequence search in bioinformatics [39]. Several privacy-preserving variants have been proposed [30, 37, 41], but existing approaches incur substantial communication and computation overhead, especially for large databases. To address these limitations, we propose an *oblivious FM-index* (OFMI) protocol based on our OR0 primitive. It achieves $O(\sigma\ell)$ online communication and round complexity, independent of the database size N. To describe the OFMI protocol, we first give a brief overview of the FM-index.

5.2.1 Preliminaries: FM-Index.

Notations. Let Σ be a finite alphabet and let $T \in \Sigma^N$ denote a string of length N over Σ. The i-th character of T is written as $T[i]$ (0-indexed), and a substring from position i to j (inclusive) is denoted by $T[i, j]$. We write $\sigma = \lceil \log |\Sigma| \rceil$ for the number of bits required to represent each symbol of Σ. Given a query string $q \in \Sigma^\ell$ of length ℓ, the *longest prefix match* (LPM) is defined as the maximum $m \le \ell$ such that the prefix $q[0, m-1]$ appears as a substring of T.

Text Indexing. The FM-index [18] is built on the suffix array (SA) of the text T, which is the array $\mathsf{SA} = (p_0, \ldots, p_{N-1})$ of text positions such that the suffixes $T[p_i, N-1]$ appear in lexicographic order. The Burrows-Wheeler Transform (BWT), denoted L, is defined by $L[i] = T[(\mathsf{SA}[i] - 1) \bmod N]$, that is, the character preceding the suffix beginning at $\mathsf{SA}[i]$. We assume that the text is terminated by a special symbol \$ smaller than any alphabet symbol.

LF-Mapping and Backward Search. For a symbol $c \in \Sigma$ and a position p, let $\mathsf{Rank}_c(L, p)$ be the number of occurrences of c in $L[0, p-1]$. Let $\mathsf{CF}_c(L)$ denote the number of symbols in L that are lexicographically smaller than c.

$$\mathsf{LF}(L, p, c) = \mathsf{CF}_c(L) + \mathsf{Rank}_c(L, p), \tag{2}$$

which maps a position in the BWT L to the corresponding position in the lexicographically sorted suffix array array during backward search. A match between a pattern q and the text T corresponds to a suffix-array interval $[f, g)$ in which all suffixes begin with q. Given the interval $[f, g)$ for q, the interval for the extended pattern cq is obtained by applying the LF-mapping to both bounds:

$$[f', g') = [\ \mathsf{LF}(L, f, c),\ \mathsf{LF}(L, g, c)\ ).$$

This operation, which extends the pattern by one symbol and updates the corresponding interval, is the *backward search* and forms the core of FM-index matching. Starting from the initial interval $[0, N)$ and processing the pattern from right to left yields the maximal suffix of the pattern that appears in T. If the index is built on the reversed text $\hat{T}$, scanning the pattern from left to right instead gives the longest prefix match. An example of the FM-index backward search process is given in Appendix E.

5.2.2 Oblivious LF-Mapping Protocol.

The LF-mapping can be evaluated using the bit vectors constructed by the wavelet matrix of the BWT L. Since these bit vectors record the symbols of L according to their binary encoding, $\mathsf{Rank}_c(L, p)$ can be computed by following the bits of the query character c. Consequently, LF-mapping reduces to a sequence of rank operations on these bit vectors. A detailed explanation is given in Appendix C.2. The oblivious LF-mapping (OLFM) protocol securely evaluates $\mathsf{LF}(L, p, c)$ over these bit vectors. Let $B^{(\sigma)} = (B^{(0)}, \ldots, B^{(\sigma-1)})$ denote the bit vectors, and let $c^{(\sigma)} = (c^{(0)}, \ldots, c^{(\sigma-1)})$ be the bitwise representation of the query character. The ideal functionality for OLFM is given in Functionality 3.

Algorithm 5.2 Oblivious LF-Mapping Π_{OLFM}

Input: RSS shares of the bit vector $B^{(\sigma)}$ for a wavelet matrix, query character $\langle c^{(\sigma)} \rangle$ (the σ-th bit of c), and query position $\langle p \rangle$.
Output: RSS share of the $\langle m \rangle$ where $m = \mathsf{LF}(p, c)$.

1: Set $\langle m \rangle \leftarrow \langle p \rangle$.
2: **for** $j = 0$ to $\sigma - 1$ **do**
3: $\quad \langle rank0 \rangle \leftarrow \mathcal{F}_{\mathsf{Rank0}}(\langle B^{(j)} \rangle, \langle m \rangle)$.
4: $\quad \langle offset \rangle = \sum_{i=0}^{N-1} (1 - \langle B^{(j)}[i] \rangle)$.
5: $\quad \langle rank1 \rangle \leftarrow \langle m \rangle - \langle rank0 \rangle + \langle offset \rangle$.
6: $\quad \langle m \rangle \leftarrow \mathcal{F}_{\mathsf{Select}}^{\langle \cdot, \cdot \rangle}(\langle rank0 \rangle, \langle rank1 \rangle, \langle c^{(j)} \rangle)$.
7: **return** $\langle m \rangle$.

Functionality 3. ($\mathcal{F}_{\mathsf{OLFM}}$ — Oblivious LF-Mapping).
Upon receiving shares of the bit vectors $\langle B^{(\sigma)} \rangle$, the query character $\langle c^{(\sigma)} \rangle$, and the query position $\langle p \rangle$, the functionality $\mathcal{F}_{\mathsf{OLFM}}$ reconstructs $B^{(\sigma)}$, $c^{(\sigma)}$, and p, computes $m \leftarrow \mathsf{LF}(L, p, c)$, generates shares $\langle m \rangle$, and sends $\langle m \rangle_i$ to P_i.

Protocol Description. Algorithm 5.2 securely realizes LF-mapping on RSS shares of the bit vectors encoding the BWT. At level j, the parties use OR0 protocol to obtain $\langle \mathsf{Rank}_0(B^{(j)}, p) \rangle$. They compute the number of zeros in $B^{(j)}$ to determine the starting index of symbols whose j-th bit is 1, and derive $\langle \mathsf{Rank}_1(B^{(j)}, p) \rangle$ from the difference between $\langle p \rangle$ and $\langle rank0 \rangle$. Using the bit $c^{(j)}$ of the query character, the updated position is chosen between the two candidates via secure multiplication. After processing all σ levels, the parties obtain $\langle p \rangle$, which equals $\mathsf{LF}(p, c)$ in the plaintext setting. Correctness follows from the equivalence to the plaintext LF-mapping with all values maintained in RSS form. Security follows from the use of ideal functionalities, so Π_{OLFM} is secure in the hybrid model.

Complexity Analysis. The OLFM protocol processes σ bit levels sequentially. At each level, it performs one OR0 invocation followed by a secure selection step. Since oblivious access requires three rounds with RingOA, and the selection step incurs one additional round, each level requires four rounds. Therefore, the total round complexity is 4σ, and the total communication is $O(\sigma)$.

5.2.3 Oblivious FM-Index Protocol.

The oblivious FM-index (OFMI) protocol is the privacy-preserving variant of the FM-index, obtained by replacing each LF-mapping step with the OLFM protocol.

Let $B^{(\sigma)}$ denote the bit vectors encoding the BWT of a text $T \in \Sigma^N$, and let $q^{(\sigma)} = (q_0^{(\sigma)}, \ldots, q_{\ell-1}^{(\sigma)})$ denote the bitwise representations of the characters of a query $q \in \Sigma^\ell$. The ideal functionality for OFMI is given in Functionality 4.

Functionality 4. ($\mathcal{F}_{\mathsf{OFMI}}$ — Oblivious FM-Index).
Upon receiving shares of $\langle B^{(\sigma)} \rangle$ and $\langle q^{(\sigma)} \rangle$, the functionality $\mathcal{F}_{\mathsf{OFMI}}$ reconstructs $B^{(\sigma)}$ and $q^{(\sigma)}$, computes the longest prefix match (LPM) for each prefix of q, generates shares $\langle z_0 \rangle, \ldots, \langle z_{\ell-1} \rangle$, where $z_i = 1$ if the prefix $q[0, i]$ appears in T, and sends $\langle z_i \rangle$ to P_i.

Protocol Description. Algorithm 5.3 securely realizes FM-index using the OLFM protocol. The parties hold RSS shares of the bit

Algorithm 5.3 Oblivious FM-Index Protocol Π_{OFMI}

Input: RSS shares of the bit vector $\langle B^{(\sigma)} \rangle$ for a wavelet matrix of BWT from text $T \in \Sigma^N$, query string $\langle q \rangle = (\langle q_0^{(\sigma)} \rangle, \ldots, \langle q_{\ell-1}^{(\sigma)} \rangle)$ of length ℓ.
Output: RSS shares of match results $\langle z_0 \rangle, \ldots, \langle z_{\ell-1} \rangle$ where $z_i = 1$ iff the prefix $q_0 \ldots q_i$ appears in T.

1: Initialize: $\langle f_0 \rangle \leftarrow 0$, $\langle g_0 \rangle \leftarrow N - 1$.
2: */* Update the interval $[f_j, g_j)$ for each query character q_j. */*
3: **for** $j = 0$ to $\ell - 1$ **do**
4: $\quad \langle f_{j+1} \rangle \leftarrow \mathcal{F}_{\mathsf{OLFM}}(\langle B^{(\sigma)} \rangle, \langle q_j^{(\sigma)} \rangle, \langle f_j \rangle)$.
5: $\quad \langle g_{j+1} \rangle \leftarrow \mathcal{F}_{\mathsf{OLFM}}(\langle B^{(\sigma)} \rangle, \langle q_j^{(\sigma)} \rangle, \langle g_j \rangle)$.
6: $\quad \langle d_j \rangle \leftarrow \langle g_{j+1} \rangle - \langle f_{j+1} \rangle$.
7: */* Calculate the length of the match. */*
8: **for** $j = 0$ to $\ell - 1$ **do** ▷ *This part can be parallelized.*
9: $\quad P_{i+1}$ and P_{i+2} do: $[d_i] \leftarrow \mathcal{F}_{\mathsf{Conversion}}(\langle d_i \rangle)$.
10: $\quad P_{i+1}$ and P_{i+2} do: $[z_i] \leftarrow \mathcal{F}_{\mathsf{ZeroTest}}([d_i])$.
11: $\quad$ Three parties do: $\langle z_i \rangle \leftarrow \mathcal{F}_{\mathsf{Conversion}^{-1}}([z_i])$.
12: **return** $(\langle z_0 \rangle, \ldots, \langle z_{\ell-1} \rangle)$.

vectors $B^{(\sigma)}$ encoding the BWT of the text $T \in \Sigma^N$ and of the bit-wise representations $q_j^{(\sigma)}$ of the query characters $q = (q_0, \ldots, q_{\ell-1})$. The search starts from the initial interval $[f_0, g_0) = [0, N)$ and proceeds character by character. At step j, the parties invoke OLFM to update both interval bounds (Lines 4–5). The interval length $g_{j+1} - f_{j+1}$ equals the number of occurrences of the prefix $q[0, j]$ in T. We employ the DPF-based zero-test protocol [6] to check whether $d_j = g_{j+1} - f_{j+1}$ is zero. Party P_i generates the zero-test keys, and two designated parties P_{i+1} and P_{i+2} perform the evaluation. Since the DPF-based protocol is defined for two-party additive shares, the RSS shares of d_j are first converted from $(2, 3)$-RSS to $(2, 2)$-ASS using $\mathcal{F}_{\mathsf{Conversion}}$ [31]. The designated parties then invoke $\mathcal{F}_{\mathsf{ZeroTest}}$ on the ASS shares to obtain an additive share of the bit $z_j \in \{0, 1\}$ indicating whether $d_j = 0$. Since the tests for different prefixes are independent, they can be executed in parallel. Finally, the outputs are converted back to RSS form using $\mathcal{F}_{\mathsf{Conversion}^{-1}}$. The share conversion and zero-test subprotocols are described in Appendix F. The protocol outputs the result vector $(z_0, \ldots, z_{\ell-1})$, where $z_i = 1$ if and only if the prefix $q[0, i]$ occurs in T. Correctness follows from the equivalence to the plaintext FM-index search, with each LF-mapping realized via OLFM and each zero-test performed on secret shares. Security follows from the use of ideal functionalities, so Π_{OFMI} is secure in the hybrid model.

Complexity Analysis. For a query of length ℓ, OFMI invokes OLFM twice per character to update the left and right interval bounds. These two invocations can be executed in parallel, and therefore do not increase the round complexity. Since one OLFM invocation requires 4σ rounds, processing ℓ characters results in $4\sigma\ell$ rounds. In addition, OFMI performs one zero-test and one share conversion at the end of the protocol, each incurring one additional round. Hence, the total round complexity is $4\sigma\ell + 2$. Table 4 summarizes the asymptotic complexities of OFMI and related privacy-preserving FM-index constructions [30, 41]. In the preprocessing phase, DPF key generation incurs $O(\sigma\ell \log N)$ computation and communication costs.

Table 4: Online complexities of oblivious full-text search.

Protocols	Computation	Comm.	Rounds
OFMI	$O(\sigma\ell N)$	$O(\sigma\ell)$	$4\sigma\ell + 2$
SecureLPM [30]	$O(\sigma\ell N)^{\dagger} + O(\sigma\ell)$	$O(\sigma\ell N)^{\dagger} + O(\sigma\ell)$	$2\sigma\ell + 2$
sFMI [41]	$O(\sigma\ell N)$	$O(\sigma\ell\sqrt{N})$	$\sigma\ell$

† $O(\sigma\ell N)$ denotes the cost for preparing query-independent correlated randomness on the database holder's side [30].

Algorithm 5.4 Oblivious Range Query Protocol Π_{ORQ}

Input: RSS shares of the bit vector $\langle B^{(\sigma)} \rangle$ for a wavelet matrix of sequential data $S \in \Sigma^N$, left and right indices $\langle l \rangle, \langle r \rangle$, and quantile order $\langle k \rangle$.
Output: RSS shares of the $\langle c \rangle$ where $c = \mathsf{Range}(S, l, r, k)$.

1: Initialize: $\langle c \rangle \leftarrow 0$.
2: **for** $j = \sigma - 1$ to 0 **do** ▷ *process bits from MSB to LSB*
3: $\quad \langle z_L \rangle \leftarrow \mathcal{F}_{\mathsf{Rank0}}(\langle B^{(j)} \rangle, \langle l \rangle), \langle z_R \rangle \leftarrow \mathcal{F}_{\mathsf{Rank0}}(\langle B^{(j)} \rangle, \langle r \rangle)$.
4: $\quad \langle z \rangle \leftarrow \langle z_R \rangle - \langle z_L \rangle$.
5: $\quad P_{i+1}$ and P_{i+2} do: $[z] \leftarrow \mathcal{F}_{\mathsf{Conversion}}(\langle z \rangle)$.
6: $\quad P_{i+1}$ and P_{i+2} do: $[k] \leftarrow \mathcal{F}_{\mathsf{Conversion}}(\langle k \rangle)$.
7: $\quad P_{i+1}$ and P_{i+2} do: $[cond] \leftarrow \mathcal{F}_{\mathsf{Comparison}}([k], [z])$.
8: $\quad$ Three parties do: $\langle cond \rangle \leftarrow \mathcal{F}_{\mathsf{Conversion}^{-1}}([cond])$.
9: $\quad \langle offset \rangle \leftarrow \sum_{i=0}^{N-1} (1 - \langle B^{(j)}[i] \rangle)$.
10: $\quad$ */* Update $[l, r)$ and k for the next level. */*
11: $\quad \langle l \rangle \leftarrow \mathcal{F}_{\mathsf{Select}}(\langle z_L \rangle, \langle offset \rangle + (\langle l \rangle - \langle z_L \rangle), \langle cond \rangle)$.
12: $\quad \langle r \rangle \leftarrow \mathcal{F}_{\mathsf{Select}}(\langle z_R \rangle, \langle offset \rangle + (\langle r \rangle - \langle z_R \rangle), \langle cond \rangle)$.
13: $\quad \langle k \rangle \leftarrow \mathcal{F}_{\mathsf{Select}}(\langle k \rangle, \langle k \rangle - \langle z \rangle, \langle cond \rangle)$.
14: $\quad \langle c \rangle \leftarrow \langle c \rangle + 2^j \cdot \langle cond \rangle$.
15: **return** $(\langle c \rangle)$.

5.3 Fully Oblivious Range Search

Range search retrieves statistical information from a numerical sequence over a specified interval and is widely used in applications such as time-series analysis and genetic variant analysis. In many settings, the interval itself may be sensitive, motivating the need for a *fully oblivious range search* that hides both the query range and the numerical database. A range query takes as input an interval $[l, r)$ of a sequence $S \in \Sigma^N$ and an order k, and returns the k-th smallest value in $S[l, r)$, denoted $\mathsf{Range}(S, l, r, k)$. Since the order parameter k is also secret, the protocol hides not only the queried interval but also whether the query corresponds to a minimum or maximum. When S is represented as a wavelet matrix, this query can be answered efficiently by descending through the σ bit levels [2] while updating the interval and the order parameter according to rank computations on the bit vectors. A detailed explanation of this procedure is given in Appendix C.3.

5.3.1 Oblivious Range Query Protocol. The oblivious range query (ORQ) protocol is the privacy-preserving variant of the plaintext range query. By combining OR0 with secure comparison and conditional selection, the protocol allows the parties to compute the

[2] Range queries examine the bits of the output value from the MSB to the LSB, in contrast to LF-mapping (Algorithm C.2), which processes the bits of the query character from the LSB to the MSB.

k-th smallest value in a secret interval $S[l, r)$ without revealing the interval boundaries and the order k.

Functionality 5. ($\mathcal{F}_{\text{ORQ}}$ — Oblivious Range Query).
Upon receiving shares of the bit vectors $\langle B^{(\sigma)} \rangle$, and shares of the interval $\langle l \rangle, \langle r \rangle$ and the order $\langle k \rangle$, the functionality reconstructs $(B^{(\sigma)}, l, r, k)$, computes the value $c \leftarrow \text{Range}(S, l, r, k)$, generates RSS shares $\langle c \rangle$, and sends $\langle c \rangle_i$ to P_i.

Protocol Description. Algorithm 5.4 presents the ORQ protocol, which securely evaluates the range query. The number of zeros in $[l, r)$ is obtained from the difference of two prefix counts computed using $\mathcal{F}_{\text{Rank0}}$. To compare k with this value, the protocol employs a DCF-based secure comparison [6]. Since this comparison operates on two-party additive shares, the RSS shares of k and the zero count are temporarily converted using $\mathcal{F}_{\text{Conversion}}$, and the result is converted back to RSS form afterward. The interval $[l, r)$ and the order parameter k are then updated using the oblivious selection functionality $\mathcal{F}_{\text{Select}}$, which enables conditional updates while keeping the comparison result hidden. The comparison bit $cond \in \{0, 1\}$ determines the j-th bit of the output value, which is accumulated by adding $cond \cdot 2^j$. Correctness follows from the equivalence to the plaintext range query evaluated level by level. Security follows from the use of ideal functionalities, so Π_{ORQ} is secure in the hybrid model.

Complexity Analysis. For each bit level, the protocol performs two OR0, one secure comparison, one inverse conversion, and three secure selections. The two OR0 and the three selections can be executed in parallel, and the RSS-to-ASS conversions are local. Each level therefore requires six rounds with RingOA. Over σ levels, the total round complexity is 6σ, and the total online communication is $O(\sigma)$. In the preprocessing phase, DPF key generation incurs $O(\sigma \log N)$ computation and communication costs.

6 Evaluation

6.1 Implementation

We implemented RingOA and its applications, OFMI and ORQ, in C++.[3] Network communication and AES primitives are provided by `cryptoTools`.[4] All AES computations use AES-128 with hardware acceleration via AES-NI. We set the security parameter to $\lambda = 128$ and instantiate the PRG using two AES-128 calls per expansion. The correlated randomness required by the protocols is generated in the preprocessing phase and stored as binary files.

Baselines. For oblivious access, we evaluate our protocol against Shared OT [24], DuORAM [35, 44] and Bai et al. [2]. We implemented Shared OT and the protocol of Bai et al. based on the descriptions in their papers. For DuORAM, we used the publicly available implementation.[5] For oblivious full-text search, we consider the following baselines: SecureLPM [30] and sFMI [41], which support full-text search under comparable functionality. For sFMI, we used the publicly available implementation.[6] Since SecureLPM does not provide a complete communication layer, we estimated its bandwidth and latency costs following the evaluation methodology described in [30].

[3] https://github.com/u-tmk/RingOA
[4] https://github.com/ladnir/cryptoTools
[5] https://git-crysp.uwaterloo.ca/iang/prac
[6] https://github.com/cBioLab/sWM

6.2 Experimental Setup and Datasets

All experiments were conducted on a Linux machine equipped with an AMD Ryzen Threadripper 3970X CPU (3.7 GHz) and 256 GB of RAM. Each protocol was executed with a single thread per party. We measured the elapsed real time of each protocol. To evaluate performance under different network conditions, we considered three environments: LAN (0.2 ms / 10 Gbps), MAN (5 ms / 400 Mbps), and WAN (100 ms / 10 Mbps). These network conditions were emulated using the `tc` command on Linux. Each execution was aborted if the total running time exceeded 10^7 ms.

Datasets for oblivious full-text search. We used a real genomic sequence based on the human reference genome GRCh38 [34]. The alphabet size is $|\Sigma| = 6$, consisting of {A, C, G, T, N, $}, where $ denotes the terminal symbol. To evaluate scalability across different database sizes, we extracted prefixes of length 2^{10} through 2^{30} from chromosomes 1 through 6, and these prefixes served as the input texts. For each database size, queries were generated by selecting a random position within the corresponding text and extracting a substring of length $\ell = 2^4 = 16$.

Datasets for oblivious range search. We used somatic mutation data in mutation annotation format (MAF) from the International Cancer Genome Consortium (ICGC) [46]. Each record includes the genomic position of a detected mutation and the read counts supporting the reference and alternate alleles. From these counts, we computed the variant allele frequency (VAF) as $\text{VAF} = \text{alt}/(\text{alt} + \text{ref})$, representing the proportion of sequencing reads that contain the mutation. We discretized each VAF into an integer in $[0, 100]$ and used these values as the entries of our secret-shared database. This results in a value domain of size 101, requiring $\sigma = 7$ bits to represent each value. The dataset contains approximately 2.3×10^7 mutation records, which fits within a domain size less than 2^{25}.

6.3 Oblivious Access Performance

6.3.1 Overall Performance. Figure 2 shows the online running time per oblivious access under LAN, MAN, and WAN settings. Each value is averaged over 50 executions. For medium to large databases, RingOA achieves faster online performance than the evaluated baselines. Although its online round introduces higher latency for small databases, especially under WAN settings, this overhead becomes less significant as N increases. From around $N \geq 2^{26}$, RingOA outperforms the other protocols even in the WAN setting. Note that for the protocol of Bai et al. [2], we measure only the OA phase without share conversion. In practice, additional conversion steps (e.g., B2A) would be required, introducing extra rounds that are not counted here. Table 5 reports the preprocessing running time, online running time under LAN and WAN, and communication cost per oblivious access for $N = 2^{20}, 2^{24}, 2^{30}$. For the online phase, we compare the protocols at the largest database size. At $N = 2^{30}$, RingOA attains a 13.1–15.7× speedup over Shared OT [24], a 25.3–30.2× speedup over DuORAM [35, 44], and a 2.0–2.3× speedup over Bai et al. [2], completing a single access in 1.26–1.51 seconds

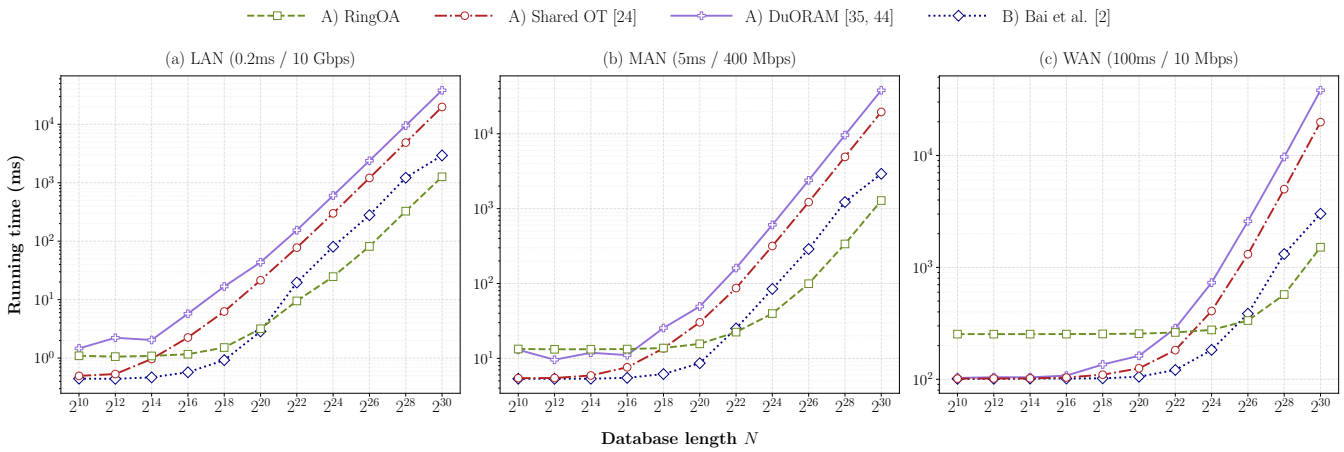

Figure 2: Online running time per oblivious access for varying database sizes N under different network settings. Labels A) and B) denote the retrieved share type: A) arithmetic shares and B) Boolean shares that would require share conversion.

Table 5: Preprocessing running time (ms), online running time (ms), and communication cost (KB) per oblivious access under LAN and WAN network settings.

Protocol	N	Time (LAN)		Time (WAN)		Communication	
		Prep.	Online	Prep.	Online	Prep.	Online
RingOA	2^{20}	0.41	3.17	101.37	255.41	0.69	0.07
	2^{24}	0.41	24.76	101.29	276.31	0.83	0.07
	2^{30}	0.42	1264.47	101.00	1512.40	1.04	0.07
Shared OT [24]	2^{20}	0.43	21.45	101.62	125.26	0.86	0.04
	2^{24}	0.42	299.73	101.57	407.50	1.00	0.04
	2^{30}	0.42	19795.66	101.44	19860.82	1.21	0.04
DuORAM [35, 44]	2^{20}	449.27	43.62	4438.69	162.11	4.47	0.02
	2^{24}	1715.41	604.54	6009.38	730.12	5.37	0.02
	2^{30}	67910.92	38179.94	72891.20	38205.42	6.72	0.02
Bai et al. [2]	2^{20}	0.41	2.86	101.58	105.41	0.61	0.04
	2^{24}	0.41	80.41	101.58	183.11	0.75	0.04
	2^{30}	0.42	2933.29	101.55	3013.83	0.96	0.04

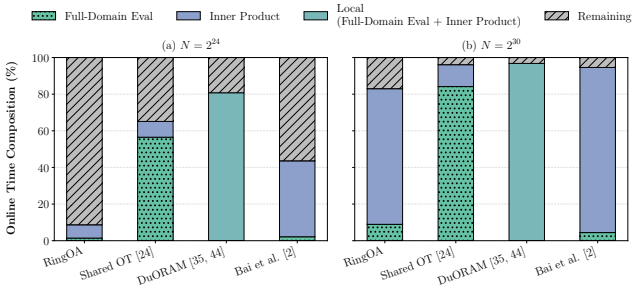

Figure 3: Breakdown of the per-access online running time under WAN settings for database sizes $N = 2^{24}$ and $N = 2^{30}$. Full-Domain Eval denotes DPF key expansion and evaluation over the full domain. Inner Product denotes the database inner product with the resulting unit vector. Remaining includes communication and auxiliary operations. For DuORAM, full-domain evaluation and inner product are reported jointly as Local.

under LAN and WAN. The online communication cost remains small across all protocols. Regarding preprocessing, all protocols except DuORAM exhibit comparable costs. In contrast, DuORAM incurs substantially higher preprocessing time and communication overhead, as it relies on MPC-based generation of DPF keys.

6.3.2 Breakdown of Online Running Time. Figure 3 presents the breakdown of the per-access online running time under WAN settings for $N = 2^{24}$ and $N = 2^{30}$. For DuORAM, full-domain evaluation and inner product are not reported separately, as the implementation expands the DPF output progressively and immediately computes the inner product with the database. The figure highlights the impact of early termination. In Shared OT, where early termination is not applied, full-domain evaluation occupies a large fraction of the online time. Similarly, in DuORAM, the *Local* component accounts for a dominant portion of the running time. In contrast, for RingOA, the fraction attributed to full-domain evaluation is significantly reduced. Comparing Figures 3(a) and (b), the dominant cost shifts with the database size. For smaller databases, the remaining cost dominates, indicating that communication latency has a stronger impact. For larger databases, local computation becomes dominant. In particular, for RingOA, the dominant component in Figure 3(b) is the inner product. The main overhead stems from extracting individual bits from the SIMD-packed unit vector via bit-shift operations prior to computing the inner product with the database. The inner-product component in Bai et al. accounts for a larger fraction than in RingOA, mainly due to the additional overhead introduced by Boolean masking.

6.4 Oblivious Full-Text Search Performance

We evaluated OFMI with the query length fixed at $\ell = 16$ and compared it against the following baselines. SotFMI instantiates the $\mathcal{F}_{\mathsf{OA}}$ component inside OFMI using Shared OT [24]. sFMI performs full-text search using additive homomorphic encryption combined

Table 6: Preprocessing running time (ms), online running time (s), and communication cost (KB) per query of oblivious full-text search protocols under LAN and WAN settings with fixed query length $\ell = 16$.

Protocol	N	Time (LAN)		Time (WAN)		Communication	
		Prep. (ms)	Online (s)	Prep. (ms)	Online (s)	Prep. (KB)	Online (KB)
OFMI	2^{20}	1.49	0.26	318.58	17.36	141.45	7.38
	2^{24}	1.50	2.14	385.55	19.27	168.45	7.38
	2^{30}	1.68	130.84	484.81	151.75	208.95	7.38
SotFMI [24]	2^{20}	0.94	2.00	209.88	9.44	97.20	4.01
	2^{24}	1.03	28.91	242.93	36.61	110.70	4.01
	2^{30}	1.13	1868.96	292.95	1897.55	130.95	4.01
SecureLPM [30]	2^{16}	15.85	8.67	120.91	45.69	6.42	40963.16
	2^{20}	15.40	139.71	120.45	679.54	6.42	655363.16
	2^{24}	15.34	2233.69	120.40	10818.53	6.42	10485763.16
sWM [41]	2^{16}	–	87.11	–	117.36	–	16534.96
	2^{20}	–	1563.05	–	1592.25	–	66184.37
	2^{22}	–	6658.86	–	6777.21	–	132527.73

with a wavelet-matrix-based FM-index [41] and does not require preprocessing phase. SecureLPM adopts an additive secret-sharing approach that directly computes rank values on the BWT using precomputed lookup tables [30]. Due to their substantially higher running times, SecureLPM and sFMI were evaluated only up to $N = 2^{24}$ and $N = 2^{22}$, respectively. Figure 4 shows the average online running time per query over 10 executions under LAN, MAN, and WAN settings. For medium-sized databases, OFMI already outperforms the evaluated baselines in online running time, and this advantage becomes more pronounced as N increases. In the WAN setting for small database sizes, SotFMI is slightly faster due to its lower round complexity. As N increases, SotFMI incurs a growing local computation overhead since it does not employ early termination, whereas OFMI reduces local computation using the underlying RingOA primitive. This difference becomes more pronounced for large databases. A detailed breakdown of the online phase is provided in Appendix G.1. Table 6 reports the preprocessing running time, online running time under LAN and WAN, and communication cost per query for selected database sizes. For the largest database ($N = 2^{30}$), OFMI achieves a 12.5× speedup over SotFMI in WAN and a 14.3× speedup in LAN. At $N = 2^{24}$, it outperforms SecureLPM by 561.4× in WAN and 1042.9× in LAN. Even at $N = 2^{22}$, OFMI is 381.1× faster than sFMI in WAN and 7919.8× faster in LAN. OFMI has a communication cost comparable to SotFMI, and both remain constant across database sizes, whereas the communication of SecureLPM and sFMI increases rapidly with N. These results show that the efficiency of the underlying oblivious access protocol has a direct impact on the overall performance of full-text search.

6.5 Oblivious Range Search Performance

We evaluated ORQ using the genetic variant datasets. The dataset contains approximately 2.3×10^7 mutation records, and we set the value domain for our range queries to $N = 2^{25}$. A query interval corresponds to a biologically meaningful genomic region such as a gene. Table 7 reports the average running time and the online communication cost over 50 trials. For comparison, we additionally implemented a baseline that realizes the same range query functionality by replacing the OA component with the Shared OT [24]. Queries complete in less than one second under LAN conditions,

Table 7: Online running time (ms) and communication cost (KB) for oblivious range search.

Protocol	LAN	MAN	WAN	Comm.
ORQ	599	752	3411	1.31
[24]	8380	8474	10055	0.82

and even under high-latency WAN settings they finish within a few seconds. Compared with the construction based on [24], ORQ achieves approximately 14.0×, 11.3×, and 2.9× faster running time under LAN, MAN, and WAN settings, respectively. The online communication remains below a few kilobytes for all protocols. These communication costs are constant and independent of the database size and interval length, enabling efficient execution even in bandwidth-limited MAN and WAN settings. The detailed breakdown of preprocessing costs and the local computation time in the online phase is provided in Appendix G.2. We additionally compare with a prior approach reported in [38]. According to the reported results, under a network latency of 10 ms and using the same CPU model, their protocol implemented with Python requires 143.73 seconds to evaluate a range minimum query on 10^5 elements. In contrast, our protocol completes queries within one second even for a database of 2.3×10^7 records.

6.6 Discussion

The experimental results show that both OFMI and ORQ achieve practical performance on large-scale genomic datasets. OFMI outperforms prior FM-index-based protocols across database sizes and network conditions with size-independent communication costs. ORQ scales to tens of millions of genetic variants and completes queries within a few seconds even in WAN settings. Overall, our approach enables practical oblivious access, full-text search, and range search over real genomic data. From a performance perspective, further speedups would be achieved through parallelization and hardware acceleration, since the protocol is largely dominated by local computations that are naturally parallelizable. Although our protocol is designed for the semi-honest model, it can be extended to the malicious setting by adopting techniques from Bai et al. [2]. Specifically, integrating verifiable distributed point functions (VDPFs) to guarantee the security of both DPF key generation and evaluation, together with SPDZ-style message authentication codes (MACs) for integrity verification, would provide malicious security with modest overhead in practice. We focus on the three-party setting, as oblivious access is primarily intended to integrate with existing efficient three-party MPC frameworks. Extending RingOA to more parties is non-trivial, since our design leverages the efficiency of three-party replicated secret sharing and a two-party DPF construction, both of which would require substantial redesign in a larger-party setting.

7 Conclusion

We presented RingOA, a three-party oblivious access protocol that achieves early termination through single-bit DPF evaluation while directly producing arithmetic-domain outputs. Our evaluation shows that RingOA achieves an order of magnitude speedup

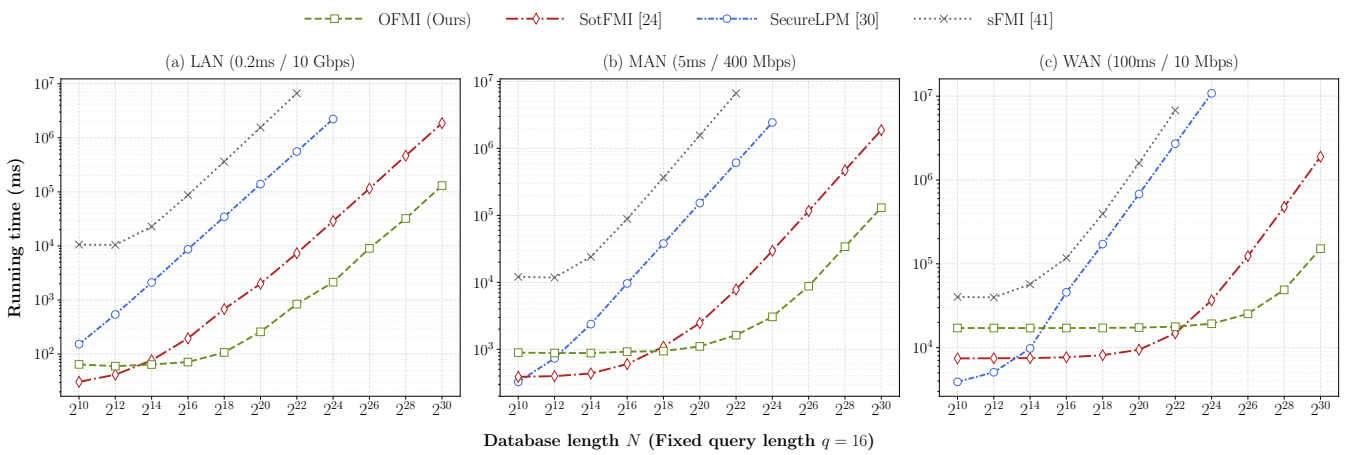

Figure 4: Online running time per query of oblivious full-text search protocols under different network settings, for varying database sizes N with fixed query length $\ell = 16$.

over existing protocols on billion-entry databases and maintains low communication overhead suitable for high latency network settings. Building on this primitive, we developed an efficient oblivious rank query and extended it to numerical sequences using the wavelet matrix. These rank-based primitives enabled practical protocols for full-text search and range search: our OFMI protocol supports privacy-preserving FM-index based search on real genomic data, and our ORQ protocols efficiently handle minimum, maximum, and quantile queries over tens of millions of genetic variants. Overall, our results show that fast and communication efficient oblivious access can serve as a practical foundation for large-scale privacy-preserving genomic analysis. Future directions include extending our protocols to the malicious setting and exploring broader applications such as privacy-preserving machine learning and large-scale graph analytics.

Acknowledgments

This work was supported by JST BOOST, Japan Grant Number JPMJBS2429 (to T.U.), and MEXT/JSPS KAKENHI Grant Numbers JP23K18515 and JP21H05052 (to K.S.). The authors used ChatGPT to revise the text, improve flow, and correct typos, grammatical errors, and awkward phrasing.

References

[1] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 805–817.

[2] Jianli Bai, Xiangfu Song, Xiaowu Zhang, Qifan Wang, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2023. Mostree: Malicious Secure Private Decision Tree Evaluation with Sublinear Communication. In *Proceedings of the 39th Annual Computer Security Applications Conference*. 799–813.

[3] Donald Beaver. 1991. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology - CRYPTO '91*. Springer, 420–432.

[4] Dan Bogdanov, Marko Jõemets, Sander Siim, and Meril Vaht. 2015. How the estonian tax and customs board evaluated a tax fraud detection system based on secure multi-party computation. In *International conference on financial cryptography and data security*. Springer, 227–234.

[5] Alexander Bowe, Taku Onodera, Kunihiko Sadakane, and Tetsuo Shibuya. 2012. Succinct de Bruijn Graphs. In *Algorithms in Bioinformatics*. Springer, 225–235.

[6] Elette Boyle, Nishanth Chandran, Niv Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. 2021. Function Secret Sharing for Mixed-Mode and Fixed-Point Secure Computation. In *Advances in Cryptology – EUROCRYPT 2021*. Springer, 871–900.

[7] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2015. Function Secret Sharing. In *Advances in Cryptology – EUROCRYPT 2015*. Springer, 337–367.

[8] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2016. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1292–1303.

[9] Elette Boyle, Niv Gilboa, and Yuval Ishai. 2019. Secure Computation with Preprocessing via Function Secret Sharing. In *Theory of Cryptography*. Springer, 341–371.

[10] Nan Cheng, Feng Zhang, and Aikaterini Mitrokotsa. 2023. Efficient Three-party Boolean-to-Arithmetic Share Conversion. In *2023 20th Annual International Conference on Privacy, Security and Trust (PST)*. 1–6.

[11] Hyunghoon Cho, David Froelicher, Natnatee Dokmai, Anupama Nandi, Shuvom Sadhuka, Matthew M Hong, and Bonnie Berger. 2024. Privacy-enhancing technologies in biomedical data science. *Annual review of biomedical data science* 7, 1 (2024), 317–343.

[12] Francisco Claude and Gonzalo Navarro. 2012. The Wavelet Matrix. In *String Processing and Information Retrieval*. Springer, 167–179.

[13] Emma Dauterman, Mayank Rathee, Raluca Ada Popa, and Ion Stoica. 2022. Waldo: A Private Time-Series Database from Function Secret Sharing. In *2022 IEEE Symposium on Security and Privacy (SP)*. 2450–2468.

[14] O'Neil Delpratt, Naila Rahman, and Rajeev Raman. 2006. Engineering the LOUDS Succinct Tree Representation. In *Experimental Algorithms*. Springer, 134–145.

[15] Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler. 2017. Privacy-preserving whole-genome variant queries. In *International Conference on Cryptology and Network Security*. Springer, 71–92.

[16] Jack Doerner and abhi shelat. 2017. Scaling ORAM for Secure Computation. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 523–535.

[17] Yaniv Erlich and Arvind Narayanan. 2014. Routes for breaching and protecting genetic privacy. *Nature Reviews Genetics* 15, 6 (2014), 409–421.

[18] P. Ferragina and G. Manzini. 2000. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*. 390–398.

[19] Johannes Fischer and Daniel Peters. 2016. GLOUDS: Representing tree-like graphs. *Journal of Discrete Algorithms* 36 (2016), 39–49. WALCOM 2015.

[20] Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J. Puglisi. 2013. Colored range queries and document retrieval. *Theoretical Computer Science* 483 (2013), 36–50. Special Issue Combinatorial Pattern Matching 2011.

[21] Travis Gagie, Simon J. Puglisi, and Andrew Turpin. 2009. Range Quantile Queries: Another Virtue of Wavelet Trees. In *Proceedings of the 16th International Symposium on String Processing and Information Retrieval* (Saariselkä, Finland) *(SPIRE '09)*. 1–6.

[22] Niv Gilboa and Yuval Ishai. 2014. Distributed Point Functions and Their Applications. In *Advances in Cryptology – EUROCRYPT 2014*. Springer, 640–658.

[23] Karthik A Jagadeesh, David J Wu, Johannes A Birgmeier, Dan Boneh, and Gill Bejerano. 2017. Deriving genomic diagnoses without revealing patient genomes.

Science 357, 6352 (2017), 692–695.
[24] Keyu Ji, Bingsheng Zhang, Tianpei Lu, Lichun Li, and Kui Ren. 2023. UC Secure Private Branching Program and Decision Tree Evaluation. *IEEE Transactions on Dependable and Secure Computing* 20, 4 (2023), 2836–2848.
[25] Marc Joye and Fariborz Salehi. 2018. Private yet Efficient Decision Tree Evaluation. In *Data and Applications Security and Privacy XXXII*. Springer, 243–259.
[26] Andrei Lapets, Frederick Jansen, Kinan Dak Albab, Rawane Issa, Lucy Qin, Mayank Varia, and Azer Bestavros. 2018. Accessible privacy-preserving web-based data analysis for assessing and addressing economic inequalities. In *Proceedings of the 1st ACM SIGCAS Conference on Computing and Sustainable Societies*. 1–5.
[27] Jack P. K. Ma, Raymond K. H. Tai, Yongjun Zhao, and Sherman S. M. Chow. 2021. Let's Stride Blindfolded in a Forest: Sublinear Multi-Client Decision Trees Evaluation. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society.
[28] Payman Mohassel and Peter Rindal. 2018. ABY3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 35–52.
[29] Payman Mohassel and Yupeng Zhang. 2017. SecureML: A System for Scalable Privacy-Preserving Machine Learning. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 19–38.
[30] Yoshiki Nakagawa, Satsuya Ohata, and Kana Shimizu. 2022. Efficient privacy-preserving variable-length substring match for genome sequence. *Algorithms for Molecular Biology* 17, 1 (2022), 9.
[31] Cheng Nan, Gupta Naman, Mitrokotsa Aikaterini, Morita Hiraku, and Tozawa Kazunari. 2024. Constant-Round Private Decision Tree Evaluation for Secret Shared Data. *Proceedings on Privacy Enhancing Technologies* 2024, 1 (2024), 397–412.
[32] Gonzalo Navarro. 2016. *Compact Data Structures: A Practical Approach*. Cambridge University Press.
[33] Muhammad Naveed, Erman Ayday, Ellen W. Clayton, Jacques Fellay, Carl A. Gunter, Jean-Pierre Hubaux, Bradley A. Malin, and Xiaofeng Wang. 2015. Privacy in the Genomic Era. *ACM Comput. Surv.* 48, 1, Article 6 (2015), 44 pages.
[34] NCBI. n.d.. GRCh38 Reference Genome. https://www.ncbi.nlm.nih.gov/datasets/genome/GCF_000001405.26/. Accessed: 2025-11-29.
[35] Sajin Sasy, Adithya Vadapalli, and Ian Goldberg. 2024. PRAC: Round-Efficient 3-Party MPC for Dynamic Data Structures. *Proceedings on Privacy Enhancing Technologies* 2024, 3 (2024), 692–714.
[36] Kana Shimizu, Koji Nuida, Hiromi Arai, Shigeo Mitsunari, Nuttapong Attrapadung, Michiaki Hamada, Koji Tsuda, Takatsugu Hirokawa, Jun Sakuma, Goichiro Hanaoka, et al. 2015. Privacy-preserving search for chemical compound databases. *BMC bioinformatics* 16, Suppl 18 (2015), S6.
[37] Kana Shimizu, Koji Nuida, and Gunnar Rätsch. 2016. Efficient privacy-preserving string search and an application in genomics. *Bioinformatics* 32, 11 (2016), 1652–1661.
[38] Shusuke Shirotake and Kana Shimizu. 2024. Efficient Privacy Preserving Range Query Using Segment Tree. In *58th Annual Conference on Information Sciences and Systems, CISS 2024, Princeton, NJ, USA, March 13-15, 2024*. IEEE, 1–6.
[39] Jared T. Simpson and Richard Durbin. 2010. Efficient construction of an assembly string graph using the FM-index. *Bioinformatics* 26, 12 (2010), i367–i373.
[40] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. 2023. Grotto: Screaming fast (2 + 1)-PC for $\mathbb{Z}_{2^n}$ via (2, 2)-DPFs. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2143–2157.
[41] Hiroki Sudo, Masanobu Jimbo, Koji Nuida, and Kana Shimizu. 2019. Secure Wavelet Matrix: Alphabet-Friendly Privacy-Preserving String Search for Bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics* 16, 5 (2019), 1675–1684.
[42] Anselme Tueno, Florian Kerschbaum, and Stefan Katzenbeisser. 2019. Private Evaluation of Decision Trees using Sublinear Cost. *Proceedings on Privacy Enhancing Technologies* 2019 (2019), 266–286.
[43] Tomoki Uchiyama and Kana Shimizu. 2024. Secure Full-Text Search Using Function Secret Sharing. In *Proceedings of the 23rd Workshop on Privacy in the Electronic Society*. 59–72.
[44] Adithya Vadapalli, Ryan Henry, and Ian Goldberg. 2023. Duoram: A Bandwidth-Efficient Distributed ORAM for 2- and 3-Party Computation. In *32nd USENIX Security Symposium (USENIX Security 23)*. 3907–3924.
[45] Sameer Wagh. 2022. Pika: Secure Computation using Function Secret Sharing over Rings. *Proceedings on Privacy Enhancing Technologies* 2022, 4 (2022), 351–377.
[46] J. Zhang, J. Baran, A. Cros, J. M. Guberman, S. Haider, J. Hsu, Y. Liang, E. Rivkin, J. Wang, B. Whitty, M. Wong-Erasmus, L. Yao, and A. Kasprzyk. 2011. International Cancer Genome Consortium Data Portal a one stop shop for cancer genomics data. *Database* (2011).
[47] Guy Zyskind, Avishay Yanai, and Alex 'Sandy' Pentland. 2024. High-Throughput Three-Party DPFs with Applications to ORAM and Digital Currencies. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 4152–4166.

A Security Assumption

Three-Party Secure Computation with Semi-Honest Adversaries. We recall the standard simulation-based security definition for three-party computation in the presence of semi-honest adversaries.

Definition A.1. Let $f : (\{0,1\}^*)^3 \to (\{0,1\}^*)^3$ be a probabilistic 3-ary functionality. For an input vector $\mathbf{x} = (x_0, x_1, x_2)$, we denote the i-th output of f by $f_i(\mathbf{x})$, where $i \in \{0,1,2\}$. Let Π be a three-party protocol that computes f. The view of party P_i during an execution of Π on input $\mathbf{x}$, denoted by $\textsc{View}_i^{\Pi}(\mathbf{x})$, consists of its input x_i, internal randomness r_i, and all messages received during the protocol. The joint output of all parties in protocol Π on input $\mathbf{x}$ is denoted by $\textsc{Output}^{\Pi}(\mathbf{x}) = (y_0, y_1, y_2)$. We say that Π securely computes f in the presence of static semi-honest adversaries if for every corrupted party $P_i \in \{0,1,2\}$, there exists a probabilistic polynomial-time simulator $\mathcal{S}$ such that:

$$\left\{(\mathcal{S}(1^\lambda, x_i, f_i(\mathbf{x})), f(\mathbf{x}))\right\}_{\mathbf{x}\in(\{0,1\}^*)^3} \equiv_c \left\{(\textsc{View}_i^{\Pi}(\mathbf{x}), \textsc{Output}^{\Pi}(\mathbf{x}))\right\}_{\mathbf{x}\in(\{0,1\}^*)^3}$$

where $\equiv_c$ denotes computational indistinguishability and λ is the security parameter.

In addition, we say that Π securely computes f in the presence of semi-honest adversaries in the $\mathcal{F}$-hybrid model if Π contains ideal calls to a trusted party computing a certain functionality $\mathcal{F}$.

B Restricted Free Sign Correction

Recall that the sign correction value w_i in RingOA is computed by Eq. 1. If w_i could be revealed, sign correction would reduce to a local multiplication. However, revealing w_i is insecure because it discloses whether the DPF outputs $(y_0, y_1) = (1, 0)$ or $(0, 1)$ at the queried position, thereby leaking information about *idx*. To enable the direct distribution of w_i in the clear while masking the sensitive sign, RingOA-FSC applies the same random sign flip to both the replicated database shares and the correction value itself.

In the setup phase, the data owner distributes the secret-shared database to the three parties. During this distribution, the owner samples a random bit $v_i \xleftarrow{\$} \{0,1\}$ for each party P_i and generates the shares such that the database held by P_{i+1} and P_{i+2} is sign-flipped by $(-1)^{v_i}$. The owner sends the bit v_i exclusively to P_i, while the corresponding sign-flipped shares are distributed to P_{i+1} and P_{i+2}. This process incurs no additional communication overhead for the computing parties compared to standard database outsourcing. The sign correction value in preprocessing phase is modified to incorporate v_i:

$$w_i = (-1)^{s_{(t_1 \oplus 1)}[r_i] \oplus t_1 \oplus v_i}, \tag{3}$$

Since v_i is sampled independently of the queried index, incorporating it into the sign-correction term masks the sign determined by the DPF evaluation. In a single invocation, the resulting w_i no longer correlates with the access position and can therefore be distributed in the clear. Given the public w_i, each receiving party corrects the sign of its retrieved value purely via local multiplication:

$$[[\mathbf{D}]]_{i+2}[idx]] \leftarrow w_{i+1} \cdot [\pm(-1)^{v_{i+1}}[[\mathbf{D}]]_{i+2}[idx]],$$
$$[[\mathbf{D}]]_{i}[idx]] \leftarrow w_{i+2} \cdot [\pm(-1)^{v_{i+2}}[[\mathbf{D}]]_{i}[idx]].$$

Thus, both sign corrections are completed without invoking $\mathcal{F}_{\mathsf{Mult}}^{[\cdot]}$.

Algorithm B.1 Convert to Valid RSS Share Protocol $\Pi_{\text{toValidRSS}}$

Input: Sign flip database shares $(-1)^{v_{i+2}}[\![\mathbf{D}]\!]_i[idx]$ and $(-1)^{v_{i+1}}[\![\mathbf{D}]\!]_{i+2}[idx]$ with the public index idx.
Output: RSS shares $\langle d\rangle_i = \langle \mathbf{D}[idx]\rangle_i$.

1: P_i computes correction $m_i = (-1)^{v_i}$.
2: Share correction: $([m_i]_0, [m_i]_1) \leftarrow \mathcal{F}^{[\cdot]}_{\text{Share}}(m_i)$.
3: Send $[m_i]_0$ to P_{i+1} and $[m_i]_1$ to P_{i+2}.
4: Receive $[m_{i+1}]_1$ from P_{i+1} and $[m_{i+2}]_0$ from P_{i+2}.
5: Set $[[\![\mathbf{D}]\!]_{i+2}[idx]] \leftarrow \mathcal{F}^{[\cdot]}_{\text{Mult}}((-1)^{v_{i+1}}[\![\mathbf{D}]\!]_{i+2}[idx], [m_{i+1}])$.
6: Set $[[\![\mathbf{D}]\!]_{i}[idx]] \leftarrow \mathcal{F}^{[\cdot]}_{\text{Mult}}((-1)^{v_{i+2}}[\![\mathbf{D}]\!]_{i}[idx], [m_{i+2}])$.
7: Set $[\![\mathbf{D}[idx]]\!]_i \leftarrow [[\![\mathbf{D}]\!]_{i+2}[idx]]_1 + [[\![\mathbf{D}]\!]_{i}[idx]]_0$.
8: Set $[\![d]\!]_i \leftarrow [\![\mathbf{D}[idx]]\!]_i + F(k_i^{\text{prf}}, \text{sid}) - F(k_{i+2}^{\text{prf}}, \text{sid})$.
9: Send $[\![d]\!]_i$ to P_{i+1} and receive $[\![d]\!]_{i+2}$ from P_{i+2}.
10: **return** $\langle d\rangle_i = ([\![d]\!]_i, [\![d]\!]_{i+2})$.

B.1 Correctness and Security

Correctness. The replicated database shares include a sign determined by v_i. Eq. (3) applies this v_i to the correction bit w_i, ensuring that the sign introduced by v_i is always cancelled. Multiplying the locally retrieved value by w_i therefore restores the correct sign. For completeness, Table 8 enumerates all sign cases induced by (y_0, y_1) and v_i, and confirms that Eq. (3) selects the correct correction sign in every case.

Table 8: Free Sign Correction Table

	$t_0 = 1$ ($t_1 = 0$)		$t_0 = 0$ ($t_1 = 1$)	
	$s_1[\hat{\alpha}] = 0$	$s_1[\hat{\alpha}] = 1$	$s_0[\hat{\alpha}] = 0$	$s_0[\hat{\alpha}] = 1$
(y_0, y_1)	$(1, 0)$	$(0, 1)$	$(0, 1)$	$(1, 0)$
$v = 0$	$+1$	-1	-1	$+1$
$v = 1$	-1	$+1$	$+1$	-1

Security Proof Sketch. The bit v_i is known only to P_i and is sampled randomly. Since w_i is obtained by XORing the DPF seeds and control bit with the random flip v_i (as specified in Eq. (3)), the value of w_i is uniformly random from the perspective of P_{i+1} and P_{i+2}. Consequently, revealing w_i does not leak any information about the DPF seeds, the control bits, or the accessed index.

B.2 Convert to Valid RSS Shares

The database shares after applying RingOA-FSC do not directly form valid RSS shares, since the random sign flips prevent straightforward reconstruction. When an RSS representation of $\mathbf{D}[idx]$ is needed, the parties run the lightweight procedure $\Pi_{\text{toValidRSS}}$ in Algorithm B.1, which removes the sign flips using only local multiplications and a brief resharing step.

C Wavelet Matrix

C.1 Construction

The wavelet matrix is a succinct data structure over numerical sequences and texts [12], which supports efficient rank and select queries. Let Σ be a finite alphabet, and let $T = (t_0, \ldots, t_{N-1}) \in \Sigma^N$ be the input sequence. Each symbol t_i is represented in binary using $\sigma = \lceil \log |\Sigma| \rceil$ bits, so it can be viewed as a binary string of length σ. The wavelet matrix represents T using σ bit vectors $B^{(0)}, \ldots, B^{(\sigma-1)}$, one for each bit position.

Algorithm C.1 Wavelet Matrix Construction

Input: Sequence $T = (t_0, \ldots, t_{N-1})$ over alphabet Σ, where each symbol is encoded in $\sigma = \lceil \log |\Sigma| \rceil$ bits.
Output: Bit vectors $B^{(\sigma)} = (B^{(0)}, \ldots, B^{(\sigma-1)})$ and zero counts $(z_0, \ldots, z_{\sigma-1})$.

$T^{(0)} \leftarrow T$.
for $j = 0$ to $\sigma - 1$ **do** ▷ *process bit from LSB to MSB*
  Initialize bit vector $B^{(j)}$ of length N.
  Initialize empty lists L_0 and L_1. ▷ *stable partition buffers*
  for $i = 0$ to $N - 1$ **do**
    $b \leftarrow j$-th bit of $T^{(j)}[i]$.
    $B^{(j)}[i] \leftarrow b$.
    if $b = 0$ **then**
      append $T^{(j)}[i]$ to L_0.
    else
      append $T^{(j)}[i]$ to L_1.
  $T^{(j+1)} \leftarrow (L_0 \,\|\, L_1)$. ▷ *stable ordering is preserved*
  $z_j \leftarrow |L_0|$. ▷ *number of zeros at level j*
return $B^{(\sigma)}, (z_0, \ldots, z_{\sigma-1})$

For each level $j \in \{0, 1, \ldots, \sigma - 1\}$:

(1) **Bit extraction:** Each symbol in the current sequence $T^{(j)}$ is examined at its j-th bit (from LSB to MSB), and this bit is recorded in the bit vector $B^{(j)}$.
(2) **Stable partition:** The sequence is stably rearranged so that all symbols with bit value 0 precede those with bit value 1, producing the next sequence $T^{(j+1)}$. Because the ordering inside each group is preserved, symbols with the same bit prefix remain in a contiguous block.
(3) **Zero count:** The number of zeros in $B^{(j)}$ is recorded as z_j, which is later used for navigating between levels.

The complete construction is given in Algorithm C.1. The bit vectors $B^{(0)}, \ldots, B^{(\sigma-1)}$ together with the counts $z_0, \ldots, z_{\sigma-1}$ constitute the wavelet matrix representation of T.

C.2 LF-Mapping over the Wavelet Matrix

Recall that the LF-mapping is defined as Eq. (2), where $\mathsf{CF}_c(L)$ counts the symbols in L that are lexicographically smaller than c, and $\mathsf{Rank}_c(L, p)$ counts the occurrences of c in the prefix $L[0, p-1]$. Let $c^{(\sigma)} = (c^{(0)}, \ldots, c^{(\sigma-1)})$ denote the σ-bit binary representation of c (from LSB to MSB). When the BWT is stored as a wavelet matrix, these quantities can be evaluated simultaneously by traversing the bit levels of the matrix.

At each level j, the update is determined by the bit $c^{(j)}$:

- **Case $c^{(j)} = 0$:** All symbols whose j-th bit is 0 appear before those with bit 1 in the current sequence $T^{(j)}$. Thus, $\mathsf{Rank}_0(B^{(j)}, p)$ counts how many such symbols occur before position p, which corresponds to the contribution of $\mathsf{CF}_c + \mathsf{Rank}_c$ restricted to this group.
- **Case $c^{(j)} = 1$:** The symbols with j-th bit 1 begin at index z_j. The term $\mathsf{Rank}_1(B^{(j)}, p)$ counts how many of them occur before

Algorithm C.2 LF-Mapping over Wavelet Matrix

Input: Bit vectors $B^{(\sigma)} = (B^{(0)}, \ldots, B^{(\sigma-1)})$ of a wavelet matrix for BWT L, query character $c \in \Sigma$ with bit representation $c^{(\sigma)} = (c^{(0)}, \ldots, c^{(\sigma-1)})$, position $p \in [N]$.
Output: $\mathsf{LF}(L, p, c)$.

$m \leftarrow p$
for $j = 0$ to $\sigma - 1$ **do** ▷ *process bits from LSB to MSB*
  if $c^{(j)} = 0$ **then**
    $m \leftarrow \mathsf{Rank}_0(B^{(j)}, m)$
  else
    $offset \leftarrow$ total number of zeros in $B^{(j)}$
    $m \leftarrow offset + \mathsf{Rank}_1(B^{(j)}, m)$
return m

p, contributing both the remaining part of CF_c and the Rank_c portion. Hence, $z_j + \mathsf{Rank}_1(B^{(j)}, p)$ gives the updated value.

The full procedure implementing this update across all levels is given in Algorithm C.2.

C.3 Range Query over Wavelet Matrix

A range query takes as input an interval $[l, r)$ of a sequence $T \in \Sigma^N$ and an order k, and returns the k-th smallest value in the subarray $T[l, r)$. When T is represented as a wavelet matrix, the query can be answered by descending through the σ bit levels while updating both the interval and the order parameter [20, 21].

At each level j, the bit vector $B^{(j)}$ divides the current interval into elements whose j-th bit is 0 and those whose bit is 1. Let

$$z_L = \mathsf{Rank}_0(B^{(j)}, l), \qquad z_R = \mathsf{Rank}_0(B^{(j)}, r),$$

so that $z_R - z_L$ is the number of zeros in $B^{(j)}[l, r)$. The update rule mirrors the binary search on the alphabet:

- **Case** $k \le z_R - z_L$**:** The k-th smallest element lies among those whose j-th bit is 0. The interval is updated to $l \leftarrow z_L, r \leftarrow z_R$, and the search proceeds to the next bit level.
- **Case** $k > z_R - z_L$**:** The desired element lies among those whose j-th bit is 1. After discarding all zeros in the interval, the interval maps to $l \leftarrow z_j + (l - z_L), r \leftarrow z_j + (r - z_R)$, and k is updated to $k - (z_R - z_L)$.

After processing all σ levels, the accumulated bits reconstruct the value of the k-th smallest element in $T[l, r)$. The full procedure is given in Algorithm C.3.

Algorithm C.3 Range Query

Input: Bit vectors $B^{(\sigma)} = (B^{(0)}, \ldots, B^{(\sigma-1)})$ of a wavelet matrix for $T \in \Sigma^N$, left and right indices $l, r \in [0, N)$, and quantile order $k \in [1, r - l]$.
Output: The k-th smallest element in $T[l, r)$.

$res \leftarrow 0$
for $j = \sigma - 1$ **down to** 0 **do** ▷ *process bits from MSB to LSB*
  $z_L \leftarrow \mathsf{Rank}_0(B^{(j)}, l)$
  $z_R \leftarrow \mathsf{Rank}_0(B^{(j)}, r)$
  $offset \leftarrow z_R - z_L$ ▷ *#zeros in $B^{(j)}[l, r)$*
  if $k \le offset$ **then**
    $l \leftarrow z_L, \quad r \leftarrow z_R$
  else
    $k \leftarrow k - offset$
    $l \leftarrow z_j + (l - z_L), \quad r \leftarrow z_j + (r - z_R)$
    $res \leftarrow res$ **OR** $(1 \ll j)$
return res

D General Oblivious Rank Protocol

The *oblivious rank* protocol Π_{Rank} extends Π_{Rank0} to handle a general rank query $\mathsf{Rank}_b(B, p)$ for $b \in \{0, 1\}$.

Functionality 6. ($\mathcal{F}_{\mathsf{Rank}}$ — Oblivious Rank).
Upon receiving RSS shares of a bit vector $\langle B \rangle$, a query bit $\langle b \rangle$, and a query position $\langle p \rangle$, the functionality $\mathcal{F}_{\mathsf{Rank}}$ reconstructs B, b, p, computes $z \leftarrow \mathsf{Rank}_b(B, p)$, generates shares $\langle z \rangle$, and sends $\langle z \rangle_i$ to each party P_i.

Algorithm D.1 Oblivious Rank Π_{Rank}

Input: RSS shares of the bit vector $\langle B \rangle$, query bit $\langle b \rangle$, and query position $\langle p \rangle$.
Output: RSS share of the rank value $\langle z \rangle$ where $z = \mathsf{Rank}_b(B, p)$.

$\langle rank0 \rangle \leftarrow \mathcal{F}_{\mathsf{Rank0}}(\langle B \rangle, \langle p \rangle)$.
$\langle rank1 \rangle \leftarrow \langle p \rangle - \langle rank0 \rangle$.
$\langle z \rangle \leftarrow \langle rank0 \rangle + \mathcal{F}_{\mathsf{Mult}}^{\langle \cdot \rangle}\big(\langle b \rangle, \langle rank1 \rangle - \langle rank0 \rangle\big)$.
return $\langle z \rangle$.

Theorem D.1. *The Π_{Rank} protocol securely computes $\mathcal{F}_{\mathsf{Rank}}$ in the $(\mathcal{F}_{\mathsf{Rank0}}, \mathcal{F}_{\mathsf{Mult}}^{\langle \cdot \rangle})$-hybrid model in the presence of a semi-honest adversary in the three-party honest-majority setting.*

Correctness and Security. Correctness follows from the fact that Π_{Rank0} correctly computes $\mathsf{Rank}_0(B, p)$, combined with the arithmetic identity $\mathsf{Rank}_1(B, p) = p - \mathsf{Rank}_0(B, p)$ and a secure conditional selection implemented via $\mathcal{F}_{\mathsf{Mult}}^{\langle \cdot \rangle}$. Security is inherited from Π_{Rank0} and the multiplication functionality.

Complexity. In addition to the invocation of Π_{Rank0}, the protocol requires one secure multiplication, which incurs one extra round. Thus, the total number of rounds is 4, with communication dominated by the multiplication.

E FM-Index

Figure 5 illustrates a concrete example of the FM-Index backward search for the query $q = \mathtt{AGT}$ on the text $T = \mathtt{AGATCA\$}$. The procedure begins with the full interval $[f_0, g_0) = [0, 7)$ and iteratively applies the LF-mapping for each query character in reverse:

(1) Processing the first character A yields the interval $[f_1, g_1) = [1, 4)$, which corresponds to suffixes starting with A.
(2) Processing the second character G refines the interval to $[f_2, g_2) = [5, 6)$, corresponding to suffixes beginning with AG.
(3) Processing the third character T produces the interval $[f_3, g_3) = [7, 7)$. Since $f_3 = g_3$, the search terminates with an empty result, meaning that the full query AGT does not occur in the text.

Algorithm F.1 Conversion $\Pi_{\text{Conversion}}$

Input: (2,3)-RSS share $\langle x \rangle$.
Output: (2,2)-ASS share $[x]$.

1: P_i and P_{i+1} sample randomness: $r_i \xleftarrow{\$} \mathbb{Z}_{2^k}$ using $\mathcal{F}_{\text{Rand}}$.
2: P_i computes $[y]_0 = [\![x]\!]_i + [\![x]\!]_{i-1} + r_i$ where $\langle x \rangle_i = ([\![x]\!]_i, [\![x]\!]_{i-1})$.
3: P_{i+1} computes $[y]_1 = [\![x]\!]_{i+1} - r_i$ where $\langle x \rangle_{i+1} = ([\![x]\!]_{i+1}, [\![x]\!]_i)$.
4: P_i returns $[x]_0 = [y]_0$ and P_{i+1} returns $[x]_1 = [y]_1$.

Algorithm F.2 Conversion $\Pi_{\text{Conversion}^{-1}}$

Input: (2,2)-ASS share $[x] = ([x]_i, [x]_j)$, where $[x]_i$ from P_i and $[x]_j$ from P_j.
Output: (2,3)-RSS share $\langle x \rangle$.

1: Three parties generate a random share $\langle r \rangle = (r_0, r_1, r_2)$ using $\mathcal{F}_{\text{Rand}}$.
2: P_i computes $y_i = [x]_i + r_{i-1} + r_i$ and sends it to P_{i+1}.
3: P_j computes $y_j = [x]_j + r_{j-1} - r_j$ and sends it to P_{j+1}.
4: P_k computes $y_k = r_{k-1} - r_k$ and sends it to P_{k+1}.
5: Three parties set $\langle x \rangle = ((y_0, y_2), (y_1, y_0), (y_2, y_1))$.

Thus, the longest prefix of q that appears in T is AG, which is exactly the longest prefix match (LPM) found by the FM-Index backward search procedure.

F Subprotocols Used in Applications

F.1 Share Conversion Protocols

We describe the share conversion protocols used in our applications. These protocols were originally proposed by Cheng et al. [31]. The first protocol converts a (2,3)-RSS share to a (2,2)-ASS share, and the second protocol converts a (2,2)-ASS share back to a (2,3)-RSS share.

F.2 Zero-Test Protocol

The functionality $\mathcal{F}_{\text{ZeroTest}}$ allows the parties to determine whether a secret-shared value $[x]$ over $\mathbb{Z}_{2^k}$ is equal to zero without revealing any further information. The output is a shared bit $[z] \in \{0, 1\}$ satisfying

$$z = \begin{cases} 1 & \text{if } x = 0, \\ 0 & \text{otherwise.} \end{cases}$$

We realize $\mathcal{F}_{\text{ZeroTest}}$ by adapting the DPF-based zero-test protocol of Boyle et al. [9].

F.3 Comparison Protocol

The functionality $\mathcal{F}_{\text{Comparison}}$ allows the parties to determine whether a secret-shared value $[x]$ is greater than another secret-shared value $[y]$ over $\mathbb{Z}_{2^k}$. The output is a shared bit $[z] \in \{0, 1\}$ satisfying

$$z = \begin{cases} 1 & \text{if } x > y, \\ 0 & \text{otherwise.} \end{cases}$$

We realize $\mathcal{F}_{\text{Comparison}}$ by adapting the DCF-based comparison protocol of Boyle et al. [6].

Table 9: Preprocess running time (ms) and communication cost (KB) for oblivious range search under different network settings.

Protocol	LAN	MAN	WAN	Comm.
ORQ	0.59	5.47	101.65	23.59
Ji et al. [24]	0.59	5.47	101.62	26.14

G Detailed Performance Evaluation

G.1 Oblivious Full-Text Search Performance

Figure 6 presents the breakdown of the per-query online running time under WAN settings for $N = 2^{24}$ and $N = 2^{30}$. The total running time is divided into *Local* and *Remaining*. *Local* denotes the time spent on full-domain evaluation and local inner product computation, while *Remaining* includes communication and auxiliary operations. The figure highlights the effect of early termination. In SotFMI [24], where early termination is not applied, local computation accounts for a large fraction of the running time. In contrast, for OFMI, the fraction attributed to local computation is significantly reduced. Comparing $N = 2^{24}$ and $N = 2^{30}$, the impact of local computation becomes more pronounced for SotFMI as the database size increases, whereas the RingOA-based construction maintains a smaller local-computation fraction.

G.2 Oblivious Range Search Performance

Table 9 reports the preprocessing cost of oblivious range search under different network settings. Figure 7 shows the breakdown of the online running time. Under LAN, the *Local* component constitutes a significant fraction of the total cost, indicating that full-domain evaluation and local inner product computation dominate the online phase. In contrast, under WAN, the *Remaining* component becomes dominant, showing that communication latency largely determines the overall performance. These results directly reflect the performance of the underlying oblivious access primitive, since oblivious range search internally invokes OA as its core subroutine. Accordingly, the higher local cost of Ji et al. [24] is consistent with the characteristics of their OA construction.

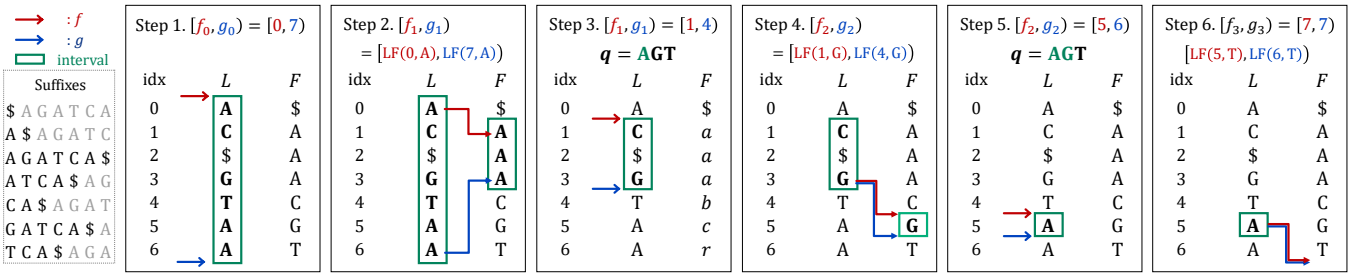

Figure 5: Example of FM-Index search process

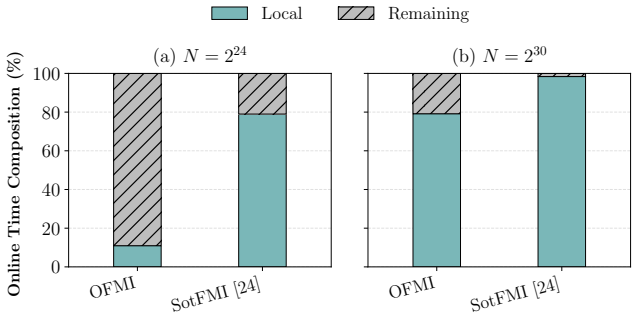

Figure 6: Breakdown of the per-query online running time under WAN settings for database sizes $N = 2^{24}$ and $N = 2^{30}$. Bars show the percentage of local computation and remaining cost for OFMI and SotFMI.

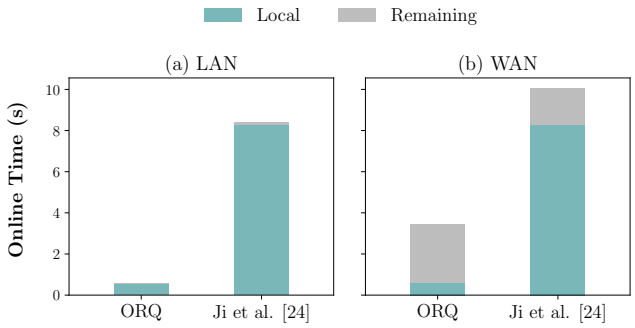

Figure 7: Online running time breakdown for oblivious range search under LAN and WAN settings. Local denotes the time spent on full-domain evaluation and local inner product computation, and Remaining accounts for the rest of the online overhead, including communication.