Efficient Privacy-Preserving Variable-Length Substring Match for Genome Sequence

Yoshiki Nakagawa
Department of Computer Science and Engineering, Waseda University, Tokyo, Japan

Satsuya Ohata
Digital Garage, Inc., Tokyo, Japan

Kana Shimizu[1]
Department of Computer Science and Engineering, Waseda University, Tokyo, Japan
National Institute of Advanced Industrial Science and Technology, Tokyo, Japan

Abstract

Finding a similar substring that commonly appears in query and database sequences is an essential task for genome data analysis. This study proposes a secure two-party variable-length string search protocol based on secret sharing. The unique feature of our protocol is that time, communication, and round complexities are not dependent on the database length N, after the query input. This property brings dramatic performance improvements in search time, since N is usually quite large in an actual genome database, and the same database is repeatedly used for many queries. Our concept hinges on a technique that efficiently applies the compressed full-text index (FOCS 2000) for a secret-sharing scheme. We conducted an experiment using a human genomic sequence with the length of 10 million as the database and a query with the length of 100 and found that the query response time of our protocol was at least three orders of magnitude faster than a well-designed baseline protocol under the realistic computation/network environment.

2012 ACM Subject Classification Theory of computation → Pattern matching; Security and privacy → Privacy-preserving protocols; Theory of computation → Cryptographic protocols; Applied computing → Genomics

Keywords and phrases Private Genome Sequence Search, Secure Multiparty Computation, Secret Sharing, FM-index, Suffix Tree, Maximal Exact Match

Digital Object Identifier 10.4230/LIPIcs.WABI.2021.2

Supplementary Material *Software (Source Code)*: `https://waseda.box.com/v/wabi2021-suppl-ppgs-src`

Funding This work is partially supported by JST CREST grant number JPMJCR19F6.
Kana Shimizu: Supported part in MEXT/JSPS KAKENHI Grant Number 19K12209 and 21H04871.

1 Introduction

The dramatic reduction in the cost of genome sequencing has prompted increased interest in personal genome sequencing over the last 15 years. Extensive collections of personal genome sequences have been accumulated both in academic and industrial organizations, and there is now a global demand for sharing the data to accelerate scientific research [13, 24]. As discussed in previous studies, disclosing personal genome information has a high privacy risk [10], so it is crucial to ensure that individuals' privacy is protected upon data sharing. At present, the most popular approach for this is to formulate and enforce a privacy policy, but it is a time-consuming process to reach an agreement, especially among stakeholders with different legal backgrounds, which slows down the pace of research. Therefore, there is a

[1] Corresponding author

strong demand for privacy-preserving technologies that can potentially compensate for or even replace the traditional policy-based approach [3, 21]. One important application that needs a privacy-preserving technology is private genome sequence search, where different stakeholders respectively hold a query sequence and a database sequence and the goal is to let the query holder know the result while simultaneously keeping the query and the database private. Many studies have addressed the problem of how to compute exact or approximate edit distance or the longest common substring (LCS) through techniques based on homomorphic encryption [17, 8, 22] and secure multi-party computation (MPC) [15, 31, 33, 7, 2, 26, 23], or how to compute sequence similarity based on private set intersection [4]. While these studies can evaluate global sequence similarity for two sequences of similar length, other studies address the problem of finding a substring between a query and a long genome sequence or a set of long genome sequences, with the aim of evaluating local sequence similarity [28, 16, 30, 29, 18, 6, 25]. [28] proposed an approach to combine an additive homomorphic encryption and index structures such as FM-index [11] and the positional Burrows-Wheeler transform [9] to find the longest prefix of a query that matches a database (LPM) and a set-maximal match for a collection of haplotypes. [30] used a similar approach and improved the time and communication complexities for LPM on a protein sequence by using a wavelet matrix. [16] improved the round complexity of a set-maximal match, though the search time was more than one order of magnitude slower than [28] due to the heavy computational cost caused by the fully homomorphic encryption. [29] used the Goldreich-Micali-Wigderson protocol to build a suffix tree for a set-maximal match. According to experiments by [18], the search time of [29] is one order of magnitude slower than [28] and [18]. [18] used a garbled circuit to build a suffix tree for substring match and a set-maximal match under a different security assumption such that the tree-traversal pattern is leaked to the cloud server. [6] and [25] found fixed-length substring matches using a one-way hash function or homomorphic encryption on a public cloud under a security assumption such that the database is a public sequence and a query is leaked to a private cloud server.

In this study, we aim to improve privacy-preserving substring match under the security assumption such that both the query and the database sequence are strictly protected. We first propose a more efficient method for finding LPM, and then extend it to find the longest maximal exact match (LMEM), which is more practically important in bioinformatics. We designed the protocol for LMEM for ease of explanation, and the protocol can be applied to similar problems such as finding all maximal exact matches (MEMs) with a small modification. To our knowledge, this is the first study to address the problem of securely finding MEMs.

Our Contribution

The time complexity of the previous studies [28, 30] include the factor of N, and thus they do not scale well to a large database. For a similar reason, using secure matching protocols (e.g., [32]) for the shares (or tags in searchable encryption) of all substrings in a query and database is even worse in terms of time complexity. To achieve a real-time search on an actual genome database, we propose novel secret-sharing-based protocols that do not include the factor of N in the time, communication, and round complexities for the search time (i.e., the time after the input of a query until the end of the search).

The basic idea of the protocols is to represent the database string by a compressed index [11, 12] and store the index as a lookup table. LPM and MEMs are found by at most ℓ and 2ℓ table lookups respectively, where ℓ is the length of the query. More specifically, the table V is referenced in a recursive manner; i.e., one needs to obtain $V[j]$, where $j = V[i]$, given i. To ensure security, we need to compute $V[j]$ without seeing any element of V.

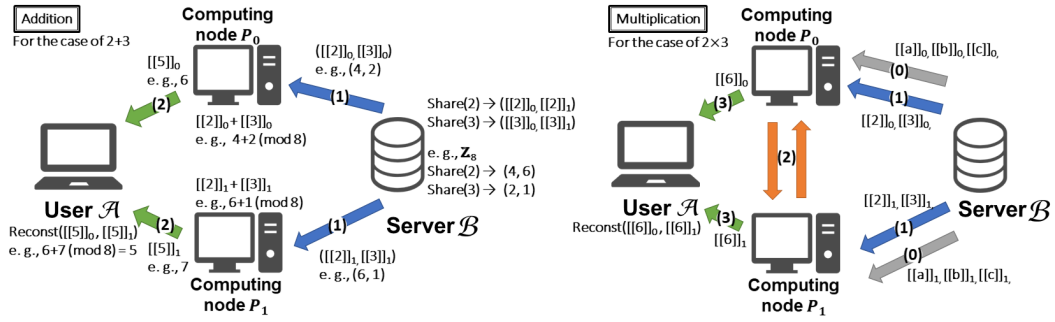

Figure 1 Arithmetic addition and multiplication over secret sharing.

The key technical contribution of this study is an efficient protocol that achieves this type of recursive reference. We named the protocol secret-shared recursive oblivious transfer (ss-ROT). While the previous studies requires $O(N)$ time complexity to ensure security, the time, communication, and round complexities of ss-ROT are all $O(\ell)$ for ℓ recursive table lookups, except for the preparation of the table and generation of shares before the query input. Since the entire protocols mainly consist of ℓ table lookups for LPM, and 2ℓ table lookups and 2ℓ inner product computations for LMEM, the search times for LPM and LMEM do not depend on the database size.

We implemented the proposed protocol and tested it on substrings of a human genome sequence 10^3 to 10^7 in length and confirmed that the actual CPU time and data transfer overhead were in good agreement with the theoretical complexities. We also found that the search time of our protocol was three orders of magnitude faster than that of the previous method [28, 30]. For conducting further performance analysis, we designed and implemented baseline protocols using major techniques of secret-sharing-based protocols. The results showed that the search times of our protocols were at least two orders of magnitude faster than those of the baseline protocols.

2 Preliminaries

2.1 Secure Computation based on Secret Sharing

Here, we explain the 2-out-of-2 additive secret sharing ($(2,2)$-SS) scheme and how to securely compute arithmetic/Boolean gates (Figure 1).

Secret Sharing and Secure Computation

In t-out-of-n secret sharing (e.g., [27]), we split the secret value x into n pieces, and can reconstruct x by combining more or an equal number of t pieces. We call the split pieces "share". The basic security notion for secret sharing is that we cannot obtain any information about x even if we gather less than or equal to $(t-1)$ shares. In this paper, we consider a case with $(t,n) = (2,2)$. A 2-out-of-2 secret sharing ($(2,2)$-SS) scheme over $\mathbb{Z}_{2^n}$ consists of two algorithms: Share and Reconst. Share takes as input $x \in \mathbb{Z}_{2^n}$ and outputs $(\llbracket x \rrbracket_0, \llbracket x \rrbracket_1) \in \mathbb{Z}_{2^n}^2$, where the bracket notation $\llbracket x \rrbracket_i$ denotes the arithmetic share of the i-th party (for $i \in \{0,1\}$). We denote $\llbracket x \rrbracket = (\llbracket x \rrbracket_0, \llbracket x \rrbracket_1)$ as their shorthand. Reconst takes as inputs $\llbracket x \rrbracket_0$ and $\llbracket x \rrbracket_1$ and outputs x. For arithmetic sharing $\llbracket x \rrbracket_i$ and Boolean sharing $\llbracket x \rrbracket_i^B$, we consider power-of-two integers n (e.g., $n = 16$) and $n = 1$, respectively.

Depending on the secret sharing scheme, we can compute arithmetic/Boolean gates over shares; that is, we can execute some kind of processing related to x without x. This means it is possible to perform some computation without violating the privacy of the secret data,

Table 1 Secure subprotocols used in this paper.

	Input	Output
Equality	$[\![x]\!]$, $[\![y]\!]$	$[\![z]\!]^B$ s.t. $z = 1$ if $x = y$ otherwise $z = 0$
Comp	$[\![x]\!]$, $[\![y]\!]$	$[\![z]\!]^B$ s.t. $z = 1$ if $x < y$ otherwise $z = 0$
CastUp	$[\![x]\!] \in \mathbb{Z}_{2^n}$, n'	$[\![x]\!] \in \mathbb{Z}_{2^{n'}}$ $(n < n')$
B2A	$[\![x]\!]^B$	$[\![x]\!]$
Choose	$[\![x]\!]$, $[\![y]\!]$, $[\![e \in \{0,1\}]\!]$	$[\![z]\!]$ s.t. $z = x$ if $e = 1$, otherwise $(e = 0)$ $z = y$

and is called secure (multi-party) computation. It is known that we can execute arbitrary computation by combining basic arithmetic/Boolean gates. In the following paragraphs, we show how to concretely compute these gates over shares.

Semi-Honest Secure Two-Party Computation Based on $(2, 2)$-Additive SS

We use a standard $(2, 2)$-additive SS scheme, defined by

- $\mathsf{Share}(x)$: randomly choose $r \in \mathbb{Z}_{2^n}$ and let $[\![x]\!]_0 = r$ and $[\![x]\!]_1 = x - r$.
- $\mathsf{Reconst}([\![x]\!]_0, [\![x]\!]_1)$: output $[\![x]\!]_0 + [\![x]\!]_1$.

Note that one of the shares of x ($[\![x]\!]_0$ or $[\![x]\!]_1$) does not reveal any information about x. In Figure 1, the secret value $x = 2$ is split into $[\![x]\!]_0 = 4$ and $[\![x]\!]_1 = 6$. These are valid $(2, 2)$-additive shares because $4 + 6 \equiv 2 \pmod 8$ holds. Even if we can see $[\![x]\!]_0 = 4$, we cannot decide the value of x since we execute a split of x uniformly at random. This means, in Figure 1, computing nodes P_0 and P_1 cannot obtain any information about x as long as these two nodes do not collude. On the other hand, we can compute arithmetic ADD/MULT gates over shares as follows:

- $[\![z]\!] \leftarrow \mathsf{ADD}([\![x]\!], [\![y]\!])$ can be done locally by just adding each party's share on x and on y. In Figure 1 (left), we show an example of secure addition. P_0/P_1 obtain shares 6/7 by adding their two shares. In this process, P_0/P_1 cannot find they are computing $2 + 3$.
- Multiplication is more complex than addition. There are various methods for multiplication over shares, most of which require communication between computing nodes. In this paper, we use the standard method for $[\![w]\!] \leftarrow \mathsf{MULT}([\![x]\!], [\![y]\!])$ based on Beaver triples (BT) [5]. Such a triple consists of $\mathsf{bt}_0 = (a_0, b_0, c_0)$ and $\mathsf{bt}_1 = (a_1, b_1, c_1)$ such that $(a_0 + a_1)(b_0 + b_1) = (c_0 + c_1)$. Hereafter, a, b, and c denote $a_0 + a_1$, $b_0 + b_1$, and $c_0 + c_1$, respectively. We use these BTs as auxiliary inputs for computing MULT. Note that we can compute them in advance (or in offline phase) since they are independent of inputs $[\![x]\!]$ and $[\![y]\!]$. We adopt a trusted initializer setting (e.g., [19, 20]); that is, BTs are generated by the party other than two computing nodes and then distributed. In the online phase of MULT, each i-th party P_i $(i \in \{0, 1\})$ can compute the multiplication share $[\![z]\!] = [\![xy]\!]$ as follows:
  1. P_i first computes $([\![x]\!]_i - a_i)$ and $([\![y]\!]_i - b_i)$, and sends them to P_{1-i}.
  2. P_i reconstructs $x' = x - a$ and $y' = y - b$.
  3. P_0 computes $[\![z]\!]_0 = x'y' + x'b_0 + y'a_0 + c_0$, and P_1 computes $[\![z]\!]_1 = x'b_1 + y'a_1 + c_1$. Here, $[\![z]\!]_0$ and $[\![z]\!]_1$ calculated with the above procedures are valid shares of xy; that is, $\mathsf{Reconst}([\![z]\!]_0, [\![z]\!]_1) = xy$. We shorten the notations and write the ADD and MULT protocols simply as $[\![x]\!] + [\![y]\!]$ and $[\![x]\!] \cdot [\![y]\!]$, respectively.

We also write $\mathsf{ADD}(\mathsf{ADD}([\![x_\mathrm{A}]\!], [\![x_\mathrm{B}]\!]), [\![x_\mathrm{C}]\!])$ as $\Sigma_{c=\{\mathrm{A},\mathrm{B},\mathrm{C}\}}[\![x_c]\!]$. Note that, similarly to the ADD protocol, we can also locally compute multiplication by constant c, denoted by $c \cdot [\![x]\!]$. We can easily extend the above protocols to Boolean gates. By converting $+$ and $-$ into

$\oplus$ in the arithmetic ADD and MULT protocols, we can obtain the XOR and AND protocols, respectively. We can construct NOT and OR protocols from the properties of these gates. When we compute $\mathsf{NOT}([\![x]\!]_0^B, [\![x]\!]_1^B)$, P_0 and P_1 output $\neg[\![x]\!]_0^B$ and $[\![x]\!]_1^B$, respectively. When we compute $\mathsf{OR}([\![x]\!]^B, [\![y]\!]^B)$, we compute $\neg\mathsf{AND}(\neg[\![x]\!]^B, \neg[\![y]\!]^B)$. We shorten the notations and write XOR, AND, NOT, and OR simply as $[\![x]\!] \oplus [\![y]\!]$, $[\![x]\!] \wedge [\![y]\!]$, $\neg[\![x]\!]$, and $[\![x]\!] \vee [\![y]\!]$, respectively. By combining the above gates, we can securely compute higher-level protocols. The functionality of the secure subprotocols [23] used in this paper are shown in Table 1. Due to space limits, we omit the details of their construction. Note that we can compute Choose by $[\![z]\!] = [\![y]\!] + [\![e]\!] \cdot ([\![x]\!] - [\![y]\!])$. In this paper, we consider the standard simulation-based security notion in the presence of semi-honest adversaries (for 2PC), as in [14]. We show the definition in Appendix B. Roughly speaking, this security notion guarantees the privacy of the secret under the condition that computing nodes do not deviate from the protocol; that is, although computing nodes are allowed to execute arbitrary attacks in their local, they do not (maliciously) manipulate transmission data to other parties. The building blocks we adopt in this paper satisfy this security notion. Moreover, as described in [14], the composition theorem for the semi-honest model holds; that is, any protocol is privately computed as long as its subroutines are privately computed.

2.2 Index Structure for String Search

Notation and Definition

Σ denotes a set of ordered symbols. A string consists of symbols in Σ. We denote a lexicographical order of two strings S and S' by $S \leq S'$ (i.e., A < C < G < T and AAA < AAC). We denote the i-th letter of a string S by $S[i]$ and a substring starting from the i-th letter to the j-th letter by $S[i, j]$. The index starts with 0. The length of S is denoted by $|S|$. A reverse string of S (i.e., $S[|S|-1], \ldots, S[0]$) is denoted by $\hat{S}$. We consider a direction from the i-th position to the j-th position as rightward if $i < j$ and leftward otherwise.

Given a query w and a database S, we define the longest prefix that matches a database string (LPM) by $\max_{(0,j)}\{j | w[0, \ldots, j] = S[k, \ldots, l]\}$, where $0 \leq j < \ell$ and $0 \leq k \leq l < N$, and the longest maximal exact match (LMEM) by $\max_{(i,j)}\{j - i | w[i, \ldots, j] = S[k, \ldots, l]\}$, where $0 \leq i \leq j < \ell$ and $0 \leq k \leq l < N$.

FM-Index and related data structures

FM-Index [11] and related data structures [12] are widely used for genome sequence search. Given a query string w of length ℓ and a database string S of length N, [11] enables LPM to be found in $O(\ell)$ time regardless of N, and it also enables LMEM to be found in $O(\ell)$ if auxiliary data structures are used [12]. Given all the suffixes of a string S: $S[0, \ldots, |S|-1]$, $S[1, \ldots, |S|-1], \ldots, S[|S|-1]$, a suffix array is an array of positions $(p_0, \ldots, p_{|S|-1})$ such that $S[p_0, \ldots, |S|-1] \leq S[p_1, \ldots, |S|-1] \leq S[p_2, \ldots, |S|-1], \ldots, \leq S[p_{|S|-1}, \ldots, |S|-1]$. We denote the suffix array of S by SA and denote its i-th element by $SA[i]$. A Burrows-Wheeler transform (BWT) is a permutation of the sequence S such that its i-th letter becomes $S[SA[i]-1]$. We denote a BWT of S by L and denote its i-th letter by $L[i]$. Let us define a rank of S for a letter $c \in \Sigma$ at position t by $\mathsf{Rank}_c(t, S) = |\{j | S[j] = c, 0 \leq j < t\}|$ and a count of occurrences of letters that are lexicographically smaller than c in S by $\mathsf{CF}_c(S) = \sum_{r<c} \mathsf{Rank}_r(|S|, S)$, and the operation $\mathsf{LF}_c(i, S) = \mathsf{CF}_c(L) + \mathsf{Rank}_c(i, L)$. The match between w and S is reported as a form of left-closed and right-open interval on SA, and the lower and upper bounds of the interval are respectively computed by LF. Given a

letter c and an interval $[f, g)$ that corresponds to suffixes that share the prefix x (i.e., $[f, g)$ reports the locations of the substring x in S), we can find a new interval that corresponds to all suffixes that share the prefix cx (i.e., locations of the substring cx) by

$$[f', g') = [\mathsf{LF}_c(f, S), \mathsf{LF}_c(g, S)). \tag{1}$$

The leftward extension of the match is called a backward search, which is the main functionality of FM-Index. By starting the search with the initial interval $[0, N)$ and conducting the backward searches for $w[\ell-1], w[\ell-2], \ldots$, the longest suffix match is detected when $f = g$. Rank and CF are precomputed and stored in an efficient form that can be searched in constant time. Therefore, the longest suffix match can be computed in $O(\ell)$ time. LPM is found if the search is conducted on $\hat{S}$ and match is extended by $w[0], w[1], \ldots, w[\ell-1]$.

Searching LMEM by repeating LPM for $w[0, \ldots, \ell-1], w[1, \ldots, \ell-1], w[2, \ldots, \ell-1], \ldots, w[\ell-1]$ takes $O(\ell^2)$ time. We can improve it to $O(\ell)$ time by using the longest common prefix (LCP) array and related data structures [12]. The LCP array, denoted by LCP, is an array that stores the length of the longest prefix of $S[\mathsf{SA}[i-1], |S|-1]$ and $S[\mathsf{SA}[i], |S|-1]$ in $\mathsf{LCP}[i]$ for $0 < i \leq N$. The lcp-interval $[i, j)$ of lcp-value d is an interval such that it satisfies $\mathsf{LCP}[i] < d$, $\mathsf{LCP}[j] < d$, $\mathsf{LCP}[k] > d$ for all $k \in \{i+1, \ldots, j-1\}$, and $\mathsf{LCP}[k] = d$ for at least one $k \in \{i+1, \ldots, j-1\}$, and is denoted by $d-[i, j)$. $d-[i, j)$ corresponds to all the suffixes that share the prefix $S[SA[i], \ldots, SA[i]+d-1]$. The parent interval of $d-[i, j)$ is the lcp-interval $h-[m, n)$ such that $h < d$ and $0 \leq m \leq i < j \leq n < N$, and there is no other lcp-interval $t-[r, s)$ such that $h < t < d$ and $0 \leq m \leq r \leq i < j \leq s \leq n < N$. The parent of the lcp-interval $[f, g)$ can be found by

$$[f', g') = \begin{cases} [\mathsf{PSV}[f_i], \mathsf{NSV}[f_i]) & \mathsf{LCP}[g_i] \leq \mathsf{LCP}[f_i] \\ [\mathsf{PSV}[g_i], \mathsf{NSV}[g_i]) & (otherwise)\ , \end{cases} \tag{2}$$

where $\mathsf{PSV}[i] = \max\{j | 0 \leq j < i \wedge \mathsf{LCP}[j] < \mathsf{LCP}[i]\}$ and $\mathsf{NSV}[i] = \min\{j | i \leq j < N \wedge \mathsf{LCP}[j] < \mathsf{LCP}[i]\}$. By finding a parent interval using PSV and NSV whenever it fails to extend the match, we can avoid useless backward searches, and thus LMEM is found at most 2ℓ backward searches. LCP, PSV and NSV are precomputed and stored in an efficient form that can be searched in constant time, so we can find LMEM in $O(\ell)$ time. See section 5.2 of [12] for more details of the data structures. Examples of the search by FM-Index, LCP, PSV, and NSV are provided in Appendix A.

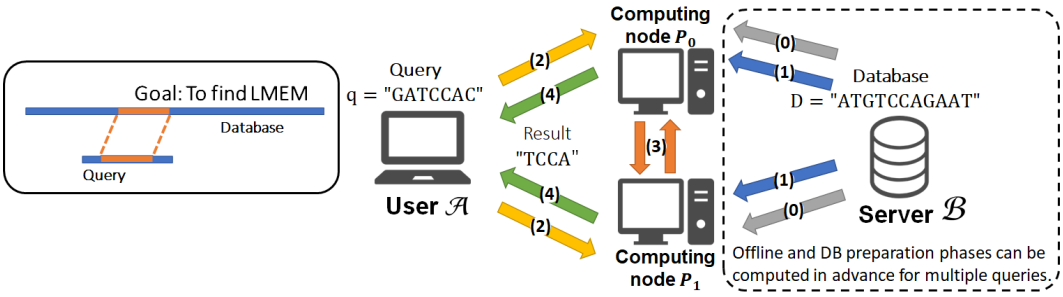

Figure 2 Schematic view of our goal and model. (0) Server (DB holder) distributes Beaver triples. (A reliable third party can serve as the trusted initializer instead.) (1) Server distributes shares of the database. (2) User (query holder) distributes shares of the query. (3) The computing nodes jointly calculate shares of the result. (4) The results are sent to User. The offline phase is (0), DB preparation phase is (1), and Search phase consists of (2)–(4).

Table 2 Summary of complexities for our protocols and related protocols. BTime and Bsize are generation time and size of BTs. Dtime and Dsize are generation time for the shares of the database and size of the shares. Stime is the time for Search phase. Comm. is the size of data exchanged between computing nodes. Round is the number of data exchanges.

	Btime	Bsize	Dtime	Dsize	Stime	Comm.	Round
ss-ROT (ours)	0	0	ℓN	ℓN	ℓ	ℓ	ℓ
Secure LPM (ours)	ℓ	ℓ	ℓN	ℓN	ℓ	ℓ	ℓ
[30, 28] (LPM by AHE)	–	–	–	–	ℓN	$\ell\sqrt{N}$	ℓ
Baseline (LPM)	$\ell^2 N$	$\ell^2 N$	N	N	$\ell^2 N$	$\ell^2 N$	$\log \ell + \log N$
Secure LMEM (ours)	ℓ^2	ℓ^2	ℓN	ℓN	ℓ^2	ℓ^2	ℓ
Baseline (LMEM)	$\ell^3 N$	$\ell^3 N$	N	N	$\ell^3 N$	$\ell^3 N$	$\log \ell + \log N$

3 Proposed protocols

Problem Setting and Outline of Our Protocols

We assume that a query holder $\mathcal{A}$, a database holder $\mathcal{B}$, and two computing nodes P_0 and P_1 participate the protocol. $\mathcal{A}$ holds a query string w of length ℓ and $\mathcal{B}$ holds a database string T of length N. After the protocol is run, only $\mathcal{A}$ knows LPM or LMEM between w and T. P_0 and P_1 do not obtain any information of w and T, except for ℓ and N.

Our protocol consists of offline, DB preparation, and Search phases. In the offline phase, $\mathcal{B}$ generates BTs (correlated randomness used for multiplication) and sends them to P_0 and P_1. In the DB preparation phase, $\mathcal{B}$ creates a lookup table and distributes its shares to P_0 and P_1. In the Search phase, $\mathcal{A}$ generates shares of the query and sends them to P_0 and P_1, and P_0 and P_1 jointly compute the result without obtaining any information of the lookup table. Finally, $\mathcal{A}$ obtains the results. Figure 2 shows the schematic view of our goal and model. Note that the Offline and DB preparation phases do not depend on a query string, so they can be computed in advance for multiple queries.

In Section 3.1, we propose the important building block ss-ROT that enables recursive reference to a lookup table. In Section 3.2, we describe how to design the lookup table based on FM-Index, and propose an efficient protocol for LPM by using the lookup table and ss-ROT. In Section 3.3, we describe the additional table design for auxiliary data structures, and propose the complete protocol for LMEM. Table 2 summarizes the theoretical complexities of the three protocols. For comparison, the complexities of the baseline protocols and a previous method for LPM based on an additive homomorphic encryption [28, 30] are shown. As we mentioned in Section 1, the baseline protocols are designed using major techniques of secret-sharing-based protocols. The detailed algorithms are described in Appendix C.

3.1 Secret-shared Recursive Oblivious Transfer

We define a problem called a secret-shared recursive oblivious transfer (ss-ROT) as follows.

▶ **Definition 1.** *We assume a database holder $\mathcal{B}$ and two computing nodes P_0 and P_1 participate the protocol. $\mathcal{B}$ holds a vector V of length N and $0 \le V[i] < N$. Given the initial position p_0 and the depth of recursion ℓ $(2 \le \ell)$, the secret-shared recursive oblivious transfer protocol outputs shares of*

$$\underbrace{V[V[\cdots V}_{\ell}[p_0]\cdots]] \tag{3}$$

without leaking V to P_0 and P_1.

Protocol 1 Secret-shared Recursive Oblivious Transfer (ss-ROT).

Input: Public input: p_0
Input: Private input of server: $R^j[i]$ $(i = 0, \ldots, N-1,\ j = 0, \ldots, \ell - 1)$
1: (Preparation by $\mathcal{B}$) $\mathcal{B}$ generates and distributes $[\![R^j[i]]\!]_0$ and $[\![R^j[i]]\!]_1$ to P_0 and P_1
2: **for** $0 \le j \le \ell - 2$ **do** ▷ **Step 1**
3: P_0 and P_1 obtain a position p_{j+1} by $p_{j+1} = \mathsf{Reconst}([\![R^j[p_j]]\!]_0, [\![R^j[p_j]]\!]_1)$.
4: **end for**
5: P_0 and P_1 output $[\![R^{\ell-1}[p_{\ell-1}]]\!]_0$ and $[\![R^{\ell-1}[p_{\ell-1}]]\!]_1$. ▷ **Step 2**

For simplicity, we denote the recursion of Eq. 3 by $V^{(\ell)}[p_0]$ (e.g., $V[V[p_0]]$ is denoted by $V^{(2)}[p_0]$). In our protocol, all the random values are uniformly generated from $\mathbb{Z}_{2^n}$.

DB Preparation Phase. $\mathcal{B}$ generates $\ell - 1$ random values $r^0, \ldots, r^{\ell-2}$ and computes the following vectors $R^0, \ldots, R^{\ell-1}$. Each vector R^j has N elements.

$$R^j[i] = \begin{cases} (V[i] + r^j) \bmod N & (j = 0) \\ (V[(i - r^{j-1}) \bmod N] + r^j) \bmod N & (1 \le j \le \ell - 2) \\ (V[(i - r^{j-1}) \bmod N]) \bmod N & (j = \ell - 1) \end{cases} \tag{4}$$

$\mathcal{B}$ computes $\mathsf{Share}(R^j[i])$ and sends $[\![R^j[i]]\!]_0$ and $[\![R^j[i]]\!]_1$ to P_0 and P_1, for $i = 0, \ldots, N-1$ and $j = 0, \ldots, \ell - 1$.

Search Phase. The Search phase consists of two steps and is described in Lines 2–5 of Protocol 1. The input is the initial position p_0 and shares of R. The output is $[\![V^{(\ell)}[p_0]]\!]$. An example of a search is illustrated in Figure 3.

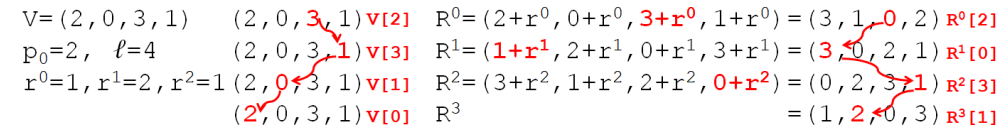

Figure 3 Example of a search when $V = (2, 0, 3, 1)$, $p_0 = 2$, and $\ell = 4$. The goal is to compute $[\![V^{(4)}[2]]\!] = [\![2]\!]$. Here we assume $\mathcal{B}$ generates $r^0 = 1, r^1 = 2, r^2 = 1$. In Step 1 of Search phase, P_0 and P_1 jointly compute $\mathsf{Reconst}([\![R^0[2]]\!]_0, [\![R^0[2]]\!]_1)$ to obtain $R^0[2] = 0$. ($R^0[2]$ is randomized by r^0, so any element of V is leaked.) In a similar way, P_0 and P_1 compute $R^1[0] = 3$ and $R^2[3] = 1$. In Step 2, P_0 and P_1 output $[\![R^3[1]]\!]_0$ and $[\![R^3[1]]\!]_1$ respectively. Since $R^0[2] = V[2] + r^0$, $R^1[V[2] + r^0] = V[V[2] + r^0 - r^0] + r^1$, $R^2[V[V[2]] + r^1] = V[V[V[2]] + r^1 - r^1] + r^2$, and $R^3[V[V[V[2]]] + r^2] = V[V[V[V[2]]] + r^2 - r^2]$, ss-ROT successfully computes $[\![V^{(4)}[2]]\!]$.

3.1.1 Security and Complexities

▶ **Theorem 2.** *ss-ROT is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix D.

In the DB preparation phase, $\mathcal{B}$ generates shares of V of length N for ℓ times. Therefore, time and communication complexities are $O(\ell N)$. For the Search phase, $\mathsf{Reconst}$ is computed ℓ times in Step 1. Since the time, communication, and round complexities of $\mathsf{Reconst}$ are $O(1)$, those of the Search phase become $O(\ell)$.

3.2 Secure LPM

Construction of Lookup Table. The goal is to find LPM securely. To apply FM-Index for a prefix search, the reverse string of T (i.e., $\hat{T}$) is used. The backward search of FM-Index is formulated by Eq. 1. If we precompute $\mathsf{LF}_c(i, \hat{T})$ for $i = 0, \ldots, N$ and $c \in \{\mathrm{A},\mathrm{T},\mathrm{G},\mathrm{C}\}$, and store them in a lookup table that consists of four vectors: V_A, V_C, V_G, and V_T such that $V_c[i] = \mathsf{LF}_c(i, \hat{T})$, Eq. 1 is replaced by the following table lookup

$$f_{k+1} = V_{w[k]}[f_k], \qquad g_{k+1} = V_{w[k]}[g_k]. \tag{5}$$

I.e., starting with the initial interval $[f_0 = 0, g_0 = N)$, we can compute the match by recursively referring to the lookup table while $f < g$.

Protocol Overview. The key idea of Secure LPM is to refer to V by ss-ROT, i.e., P_0 and P_1 jointly refer to V ℓ times in a recursive manner. To achieve backward search, P_0 and P_1 need to select $V_x[\cdot]$ for each reference, where x is a query letter to be searched with. This is achieved by expressing the query letter by unary code (Eq. 7) and computing the inner product of Eq. 7 and $(V_\mathrm{A}[\cdot], V_\mathrm{C}[\cdot], V_\mathrm{G}[\cdot], V_\mathrm{T}[\cdot])$. To find LPM, P_0 and P_1 need to check $f = g$ for each reference. We use the subprotocol Equality to check it securely. Since V is randomized with different numbers for searching f and g, the difference of the random numbers is precomputed and removed securely upon the equality check. $\mathcal{A}$ receives only the result of each equality check to know LPM. For examples, LPM is the prefix of length $i-1$ when $f = g$ for the i-th reference. If $f \neq g$ for all references, LPM is the entire query.

DB Preparation Phase. $\mathcal{B}$ creates a lookup table and generates the following 4ℓ vectors in a similar manner to ss-ROT. For simplicity, we denote the length of V_c by $N' = N + 1$.

$$R^j_{c,f}[i] = \begin{cases} (V_c[i] + r^j_f) \bmod N' & (j = 0) \\ (V_c[(i - r^{j-1}_f) \bmod N'] + r^j_f) \bmod N' & (1 \le j < \ell) \end{cases} \tag{6}$$

$R^j_{c,f}[i]$ is used for computing the lower bound f of the interval $[f, g)$. We also generate $R^j_{c,g}[i]$ for the upper bound g. R consists of 8ℓ vectors, each of length N'. Since the longest match is found when $f = g$, $\mathcal{B}$ also generates a vector $r'[j] = r^j_f - r^j_g$ that is used for equality check of f and g. Then, $\mathcal{B}$ sends shares of $R^j_{c,f}[i]$, $R^j_{c,g}[i]$, and $r'[j]$ to P_0 and P_1.

Search Phase. Protocol 2 describes the algorithm in detail. $\mathcal{A}$ generates four vectors q_A, q_C, q_G, q_T, each of length ℓ, as follows.

$$q_c[j] = \begin{cases} 1 & (c = w[j]) \\ 0 & (c \neq w[j]) \end{cases} \tag{7}$$

For each j, $(q_\mathrm{A}[j], q_\mathrm{C}[j], q_\mathrm{G}[j], q_\mathrm{T}[j])$ encodes $w[j]$ (e.g., $(q_\mathrm{A}[j], q_\mathrm{C}[j], q_\mathrm{G}[j], q_\mathrm{T}[j]) = (1, 0, 0, 0)$ if $w[j] = \mathrm{A}$). The aim of the encode is to compute $[\![R_x[j]]\!] = [\![\sum_{c\in\Sigma} q_c[j] \cdot R_c[j]]\!]$ when $w[j] = x$. Figure 4 illustrates an example of the table lookup.

$\mathcal{A}$ generates shares of q_A, q_C, q_G, q_T and distributes them to P_0 and P_1. P_0 and P_1 compute $\mathsf{LF}_{w[j]}(f', \hat{T}) + r^j_f$ and $\mathsf{LF}_{w[j]}(g', \hat{T}) + r^j_g$ in Lines 5–8 without leaking f' and g', where $[f', g')$ corresponds to the match of $w[0, j]$ and $\hat{T}$. In Lines 10–12, the equality of f' and g' is examined for all rounds. Note that f_j and g_j are randomized by different values r^{j-1}_f and r^{j-1}_g in order to conceal f' and g', so our protocol computes the equality of $f_j - g_j - r'[j-1]$ and 0. In Lines 15–16, $\mathcal{A}$ receives all the results of equality checks (i.e., $[\![o[1]]\!]^B, \ldots, [\![o[\ell]]\!]^B$) from P_0 and P_1, and knows LPM by reconstructing them. For example, if w =GCT and $o = (0, 0, 1)$, $\mathcal{A}$ knows that LPM is GC.

Protocol 2 Secure LPM.

Input: Public input: N, $N' = N + 1$, ℓ, $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$, $f_0 = 0$, $g_0 = N$
Input: Private input of user: query $q_c \in \{0,1\}^\ell$ $(c \in \Sigma)$
Input: Private input of server: $R^j_{c,f}, R^j_{c,g} \in \mathbb{Z}_{N'}^{N'}$ $(c \in \Sigma, 0 \le j < \ell)$, $r' \in \mathbb{Z}^\ell_{N'}$

1: (Preparation by $\mathcal{B}$) $\mathcal{B}$ distributes $[\![R^j_{c,f}]\!]$, $[\![R^j_{c,g}]\!] (c \in \Sigma, 0 \le j < \ell)$, $[\![r']\!]$ to P_0 and P_1
2: (Preparation by $\mathcal{A}$) $\mathcal{A}$ distributes $[\![q_c]\!]$ $(c \in \Sigma)$ to P_0 and P_1.
3: (Computation by P_0 and P_1)
4: **for** $j = 0, \cdots, \ell - 1$ **do**
5: $\quad [\![f_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,f}[f_j]]\!], [\![q_c[j]]\!])$ ▷ Select $[\![R^j_{w[j],f}[f_j]]\!]$
6: $\quad [\![g_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,g}[g_j]]\!], [\![q_c[j]]\!])$ ▷ Select $[\![R^j_{w[j],g}[g_j]]\!]$
7: $\quad f_{j+1} \leftarrow \mathsf{Reconst}([\![f_j]\!]_0, [\![f_j]\!]_1)$ ▷ Update randomized lower bound
8: $\quad g_{j+1} \leftarrow \mathsf{Reconst}([\![g_j]\!]_0, [\![g_j]\!]_1)$ ▷ Update randomized upper bound
9: **end for**
10: **for** $j = 1, \cdots, \ell$ **parallelly do**
11: $\quad [\![o[j]]\!]^B \leftarrow \mathsf{Equality}([\![f_j]\!] - [\![g_j]\!] - [\![r'[j-1]]\!], [\![0]\!])$ ▷ Equality check of upper and lower bounds.
12: **end for**
13: P_0 and P_1 send $[\![o]\!]^B_0, [\![o]\!]^B_1$ to $\mathcal{A}$
14: (Verification by $\mathcal{A}$)
15: **for** $j = 1, \cdots, \ell$ **do**
16: $\quad o[j] \leftarrow \mathsf{Reconst}([\![o[j]]\!]^B_0, [\![o[j]]\!]^B_1)$
17: **end for**

Output: $\mathcal{A}$ outputs $o[1], \ldots, o[\ell]$ to determine LPM.

3.2.1 Security and Complexities

▶ **Theorem 3.** *Protocol 2 is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix E. $\mathcal{A}$ may reveal T by making many queries. Such a problem is called output privacy. Although output privacy is outside of the scope of this paper, we should mention here that $\mathcal{A}$ needs to make an unrealistically large number of queries for obtaining T by such a brute-force attack, considering that N is very long.

The DB preparation phase generates shares of $R^j_{c,f}$ and $R^j_{c,g}$ ($c \in \Sigma$ and $0 \le j < \ell$); i.e., $8 \times \ell$ vectors of length N'. Therefore, the time and communication complexities are $O(\ell N)$. For the Search phase, MULT and Reconst are computed twice in Lines 4–9 for ℓ rounds and Equality is computed once in Lines 10–12 for ℓ rounds. Each time, the communication and round complexities of these subprotocols are $O(1)$, so those of the Search phase become $O(\ell)$.

3.3 Secure LMEM

Construction of Lookup Table. As described in Section 2.2, we can find a parent interval by a reference to LCP, PSV, and NSV. Therefore, in addition to V_c defined in Section 3.2, we prepare lookup tables that simply store all the outputs of them; i.e., $V_{lcp}[i] = \mathsf{LCP}[i]$, $V_{psv}[i] = \mathsf{PSV}[i]$, and $V_{nsv}[i] = \mathsf{NSV}[i]$.

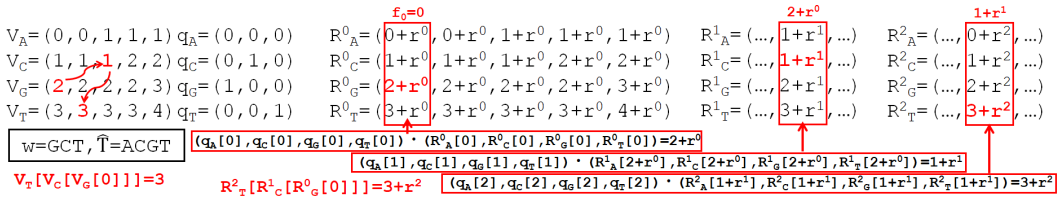

Figure 4 Example of a secure table lookup when w =GCT and $\hat{T}$=ACGT. Only the lookup for a lower bound is shown. For simplicity, $R^j_{c,f}$ and r^j_f are denoted by R^j_c and r^j. $\mathsf{LF}_{w[i]}(f_i, \hat{T})$ $(i = 0, 1, 2)$ is computed by $V_{\mathrm{G}}[0], V_{\mathrm{C}}[2]$, and $V_{\mathrm{T}}[1]$. V is referenced securely by using R. $R^0_{\mathrm{G}}[0]$ is computed by $\sum_{c\in\Sigma} q_c[0] \cdot R_c[0]$. $R^1_{\mathrm{C}}[2 + r^0]$ is computed by $\sum_{c\in\Sigma} q_c[1] \cdot R_c[2 + r^0]$. $R^2_{\mathrm{T}}[1 + r^1]$ is computed by $\sum_{c\in\Sigma} q_c[2] \cdot R_c[1 + r^1]$.

DB Preparation Phase. $\mathcal{B}$ generates randomized vectors $R_{c,f}$, $R_{c,g}$ and $r'[j] = r^j_f - r^j_g$ using the same algorithm in Section 3.2 for length 2ℓ. As shown in Eq. 2, V_{lcp} is referred by the upper and lower bounds of $[f, g)$. Therefore, $\mathcal{B}$ generates following circular permutations of V_{lcp} such that $W_{l,f}$ and $R_{c,f}$, and $W_{l,g}$ and $R_{c,g}$, are permutated by the same random values, respectively. I.e.,

$$W^j_{l,x}[i] = \begin{cases} V_{lcp}[i] & (j = 0) \\ V_{lcp}[(i - r^{j-1}_x) \bmod N] & (1 \le j < 2\ell) \end{cases},$$

where x is either f or g. V_{psv} is referred by both f and g, and is plugged in to f. Therefore, $\mathcal{B}$ generates $W^j_{p,f}$ and $W^j_{p,g}$ such that both of them are randomized by r^j_f, and $W^j_{p,f}$ is permutated by r^{j-1}_f and $W^j_{p,g}$ is permutated by r^{j-1}_g as follows.

$$W^j_{p,f}[i] = \begin{cases} (V_{psv}[i] + r^j_f) \bmod N & (j = 0) \\ (V_{psv}[(i - r^{j-1}_f) \bmod N] + r^j_f) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$$W^j_{p,g}[i] = \begin{cases} (V_{psv}[i] + r^j_g) \bmod N & (j = 0) \\ (V_{psv}[(i - r^{j-1}_g) \bmod N] + r^j_f) \bmod N & (1 \le j < 2\ell) \end{cases}$$

Similarly, V_{nsv} is referred by both f and g, and is plugged in to g. Therefore, $\mathcal{B}$ generates $W^j_{n,f}[i]$ and $W^j_{n,g}[i]$ as follows.

$$W^j_{n,f}[i] = \begin{cases} (V_{nsv}[i] + r^j_f) \bmod N & (j = 0) \\ (V_{nsv}[(i - r^{j-1}_f) \bmod N] + r^j_g) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$$W^j_{n,g}[i] = \begin{cases} (V_{nsv}[i] + r^j_g) \bmod N & (j = 0) \\ (V_{nsv}[(i - r^{j-1}_g) \bmod N] + r^j_g) \bmod N & (1 \le j < 2\ell) \end{cases}$$

$\mathcal{B}$ distributes shares of $R_{c,f}$, $R_{c,g}$, r', $W_{l,f}$, $W_{l,g}$, $W_{p,f}$, $W_{p,g}$, $W_{n,f}$, and $W_{n,g}$ to P_0 and P_1.

Search Phase. Protocol 3 describes the algorithm in detail. $\mathcal{A}$ generates query vectors q_{A}, q_{C}, q_{G}, q_{T} by Eq. 7 and distributes shares of the vectors to P_0 and P_1. In Line 6 of Protocol 3, $[\hat{f}, \hat{g})$ is computed by the reference to R (i.e., a search based on a backward search) similarly to Lines 5–6 of Protocol 2. In Line 9, $[f_{ex}, g_{ex})$ is computed by the reference to W (i.e., a search based on LCP, PSV and NSV). In Line 11, the interval is updated by either $[\hat{f}, \hat{g})$ or $[f_{ex}, g_{ex})$ based on the result of $f' = g'$ in Line 7, where $[f', g')$ corresponds to the interval that corresponds to a substring match.

In each round, we need to know a query letter to be searched with, so we need to maintain the right end position of the match in the query. The position moves toward the right while the match is extended, but remains the same when the interval is updated based on PSV and NSV. To memorize the position, we prepare shares of a unit bit vector u of length ℓ, in which the position t is memorized as $u[t] = 1$ and $u[i \neq t] = 0$. In Lines 14–18, u remains the same if the interval is updated based on PSV and NSV, and $u = (u, [\ell - 1], u[0], u[1], \ldots, u[\ell - 2])$ otherwise. In Lines 19–21, the inner product of q_c ($c \in \Sigma$) and u becomes the encode of $w[t]$ that is used for the next round.

We also maintain the left end position of the match. While the match is extended, the position remains the same and it moves toward the right when the interval is updated by $[f_{ex}, g_{ex})$. The new left end position can be computed by $p + m - c$ where p is the current position, m is the length of the current match, and c is the lcp-value of $[f_{ex}, g_{ex})$ (i.e., the longest common prefix length of suffixes contained in $[f_{ex}, g_{ex})$). The position is computed in Line 23. The match length is incremented by 1 for each extension. When the interval is updated by $[f_{ex}, g_{ex})$, the match length is reduced to the lcp-value of $[f_{ex}, g_{ex})$, which is computed by $\max(\mathsf{LCP}[f], \mathsf{LCP}[g])$. The match length is computed in Line 22. In Line 25, the longest match length and the corresponding left end position are updated. After all the positions in the query have been examined, LMEM and its left end position are sent to $\mathcal{A}$ in Line 27.

3.3.1 Security and Complexities

▶ **Theorem 4.** *Protocol 3 is correct and secure in the semi-honest model.*

Due to space limits, the proof is shown in Appendix F.

The DB preparation phase generates shares of $R^j_{c,f}$ and $R^j_{c,g}$ ($c \in \Sigma$, $0 \leq j < \ell$) and $W^j_{x,f}$ and $W^j_{x,g}$ ($x \in \{l, p, n\}$ and $0 \leq j < \ell$); $14 \times \ell$ vectors of length $N + 1$. Therefore, the time and communication complexities are $O(\ell N)$. For the Search phase, MULT is computed ℓ times in parallel in Lines 15–16. (These are not dependent on each other.) In Line 20, MULT is computed ℓ times in parallel, and Line 20 is computed in parallel four times in Lines 19–21. Lines 15–16 and Lines 19–21 are repeated for $2\ell - 1$ rounds. Other subprotocols are also computed for $2\ell - 1$ rounds. The time, communication, and round complexities are $O(1)$ for MULT, and independent computation of MULT for ℓ times does not increase the round complexity. The time, communication and round complexities are $O(1)$ for the other subprotocols used in Protocol 3. Therefore, the complexities of the Search phase are $O(\ell^2)$ for time and communication, and $O(\ell)$ for the number of rounds.

4 Experiment

We implemented Protocol 2 (Secure LPM) and Protocol 3 (Secure LMEM). For comparison, we also implemented baseline protocols (Baseline (LPM) and Baseline (LMEM)). Details of the baseline protocols are provided in Appendix C. All protocols were implemented by Python 3.5.2. The dataset was created from Chromosome 1 of the human genome. We extracted substrings of length $N = 10^3$, 10^4, 10^5, 10^6, and 10^7 for databases, and $\ell = 10$, 25, 50, 75, and 100 for queries. Share was run with $n = 16$ and $n = 32$ for $N < 10^5$ and $10^5 \leq N$ in the proposed protocols, and $n = 1$ for a Boolean share and $n = 8$ for an arithmetic share in the baseline protocols. We did not implement a data transfer module, and each protocol is implemented as a single program. Therefore, the search time of the protocols was measured by the time consumed by either one of P_0 and P_1. To assess the influence of communication

Protocol 3 Secure LMEM.

Input: Public input: N, $N' = N+1$, ℓ, $\Sigma = \{\mathrm{A}, \mathrm{T}, \mathrm{G}, \mathrm{C}\}$, $f_0 = 0$, $g_0 = N$
Input: Private input of user: query $q_c \in \{0,1\}^\ell$ $(c \in \Sigma)$
Input: Private input of server: $R^j_{c,f}, R^j_{c,g}, W^j_{l,f}, W^j_{l,g}, W^j_{p,f}, W^j_{p,g}, W^j_{n,f}, W^j_{n,g} \in \mathbb{Z}_{N'}^{N'}$ $(c \in \Sigma, 0 \le j < \ell)$, $r' \in \mathbb{Z}^\ell_{N'}$

(Preparation by $\mathcal{B}$) $\mathcal{B}$ generates shares of input vectors. $\mathcal{B}$ also generates shares of variables: $[\![u]\!] = [\![(1,0,\ldots,0)]\!]$, $[\![p_{max}]\!] = [\![0]\!]$, $[\![p]\!] = [\![0]\!]$, $[\![m_{max}]\!] = [\![0]\!]$, $[\![m]\!] = [\![0]\!]$. All shares are sent to P_0 and P_1.
(Preparation by $\mathcal{A}$) $\mathcal{A}$ generates and distributes $[\![q_c]\!]$ $(c \in \Sigma)$ to P_0 and P_1.
(Computation by P_0 and P_1)
Set shares of the initial letter $[\![z_c]\!] = [\![q_c[0]]\!]$ $(c \in \Sigma)$.
for $j = 0, \cdots, 2\ell - 1$ **do**
  $[\![\hat{f}_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,f}[f_j]]\!], [\![z[c]]\!])$, $[\![\hat{g}_j]\!] \leftarrow \sum_{c\in\Sigma} \mathsf{MULT}([\![R^j_{c,g}[g_j]]\!], [\![z[c]]\!])$
  $[\![e1]\!]^B \leftarrow \mathsf{Equality}([\![\hat{f}_j]\!] - [\![\hat{g}_j]\!] - [\![r'[j-1]]\!], [\![0]\!])$, $[\![e1]\!] \leftarrow \mathsf{B2A}([\![e1]\!]^B)$ ▷ If $f = g$.
  $[\![e2]\!]^B \leftarrow \mathsf{Comp}([\![W^j_{l,f}[f_j]]\!], [\![W^j_{l,g}[g_j]]\!])$, $[\![e2]\!] \leftarrow \mathsf{B2A}([\![e2]\!]^B)$ ▷ If $\mathsf{LCP}[f] < \mathsf{LCP}[g]$
  $[\![f_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{p,g}[g_j]]\!], [\![W^j_{p,f}[f_j]]\!], [\![e2]\!])$, $[\![g_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{n,g}[g_j]]\!], [\![W^j_{n,f}[f_j]]\!], [\![e2]\!])$
  $[\![l_{ex}]\!] \leftarrow \mathsf{Choose}([\![W^j_{l,g}[g_j]]\!], [\![W^j_{l,f}[f_j]]\!], [\![e2]\!])$
  $[\![f_{j+1}]\!] \leftarrow \mathsf{Choose}([\![f_{ex}]\!], [\![\hat{f}_j]\!], [\![e1]\!])$, $[\![g_{j+1}]\!] \leftarrow \mathsf{Choose}([\![g_{ex}]\!], [\![\hat{g}_j]\!], [\![e1]\!])$
  $f_{j+1} \leftarrow \mathsf{Reconst}([\![f_{j+1}]\!]_0, [\![f_{j+1}]\!]_1)$, $g_{j+1} \leftarrow \mathsf{Reconst}([\![g_{j+1}]\!]_0, [\![g_{j+1}]\!]_1)$ ▷ Update f, g
  $[\![e']\!] \leftarrow \mathsf{B2A}(\mathsf{ADD}([\![e1]\!]^B, [\![1]\!]^B))$
  for $i = 0, \cdots, \ell - 1$ **parallelly do** ▷ Maintain right end of the match.
    $[\![u1[i]]\!] \leftarrow \mathsf{MULT}([\![u[i]]\!], [\![e1]\!])$ ▷ $u1 = u$ and $u2 = (0,\ldots,0)$ if $\hat{f} - \hat{g} - r' = 0$,
    $[\![u2[i]]\!] \leftarrow \mathsf{MULT}([\![u[i]]\!], [\![e']\!])$ $u1 = (0,\ldots,0)$ and $u2 = u$ otherwise.
    $[\![u[i]]\!] \leftarrow \mathsf{ADD}([\![u2[(i-1) \bmod \ell]]\!], [\![u1[i]]\!])$ ▷ u is incremented iff. $\hat{f} - \hat{g} - r' \neq 0$.
  end for
  for $c \in \Sigma$ **parallelly do**
    $[\![z_c]\!] \leftarrow \sum_{i\in\{0,\ldots,\ell-1\}} \mathsf{MULT}([\![q_c[i]]\!], [\![u[i]]\!])$ ▷ Select next letter to be searched with.
  end for
  $[\![m']\!] \leftarrow \mathsf{Choose}([\![l_{ex}]\!], \mathsf{ADD}([\![1]\!], [\![m]\!]), [\![e1]\!])$ ▷ Calculate match length
  $[\![p]\!] \leftarrow \mathsf{Choose}(\mathsf{ADD}(\mathsf{ADD}([\![p]\!], [\![m]\!]), [\![-l_{ex}]\!]), [\![p]\!], [\![e1]\!])$ ▷ Update left end position of match
  $[\![e3]\!] \leftarrow \mathsf{B2A}(\mathsf{Comp}([\![m']\!], [\![m_{max}]\!]))$, $[\![m]\!] \leftarrow [\![m']\!]$
  $[\![m_{max}]\!] \leftarrow \mathsf{Choose}([\![m_{max}]\!], [\![m]\!], [\![e3]\!])$, $[\![p_{max}]\!] \leftarrow \mathsf{Choose}([\![p_{max}]\!], [\![p]\!], [\![e3]\!])$ ▷ Update max
end for
P_0 and P_1 send $[\![m_{max}]\!]_0, [\![m_{max}]\!]_1, [\![p_{max}]\!]_0$, and $[\![p_{max}]\!]_1$ to $\mathcal{A}$
(Verification by $\mathcal{A}$) max $\leftarrow \mathsf{Reconst}([\![m_{max}]\!]_0, [\![m_{max}]\!]_1)$ and $p_{max} \leftarrow \mathsf{Reconst}([\![p_{max}]\!]_0, [\![p_{max}]\!]_1)$.
Output: $\mathcal{A}$ outputs m_{max} and p_{max} to report LMEM.

on a realistic environment, we theoretically estimated delays caused by network bandwidth and latency. We assume three environments: LAN (0.2 ms/10 Gbps), WAN_1 (10 ms/100 Mbps), and WAN_2 (50 ms/10 Mbps). During the run of Search phase, we stored all the data that were transferred from P_0 to P_1 in a file and measured the file size as an actual communication size. Note that the communication is symmetric and data transfer size from P_0 to P_1 is equal to that from P_1 to P_0. Based on the data transfer size D byte, we estimate the communication delay by $D/k + 1000 \times eT$, where k is bandwidth, e is latency and T is a round of communication. All the protocols were run with a single thread on the same machine equipped with Intel Xeon 2.2 GHz CPU and 256 GB memory. We also tested the

Table 3 Offline time (Time), offline size (Size), DB preparation time (Time), DB preparation size (Size), Search time on a local machine (Time), Search communication size (Size), estimated Search time for three environments: LAN (0.2 ms/10 Gbps), WAN_1 (10 ms/100 Mbps), and WAN_2 (50 ms/10 Mbps), for $N = 10^4$ (only for Baseline (LMEM)), $10^5, 10^6, 10^7$, and $\ell = 100$. The size unit is MB and the time unit is sec except for the cell describing "20 h<".

		Offline		DB preparation		Search		Estimated time on network		
	N	Time	Size	Time	Size	Time	Size	LAN	WAN_1	WAN_2
Secure	10^5	0.166	0.013	123	305	0.141	0.010	0.181	2.162	10.249
LPM	10^6	0.141	0.013	1248	3051	0.113	0.010	0.153	2.134	10.221
(ours)	10^7	0.150	0.013	12628	30517	0.126	0.010	0.167	2.147	10.234
	10^5	-	-	-	-	691	163	691	707	838
[30]	10^6	-	-	-	-	7817	517	7818	7863	8261
	10^7	-	-	-	-	20 h<	-	-	-	-
	10^5	3995	184	0.146	0.095	13	122	13	24	118
Baseline (LPM)	10^6	38767	1841	1.522	0.954	164	1227	165	268	1196
	10^7	20 h<	-	-	-	-	-	-	-	-
Secure	10^5	7.619	1.704	435	1068	4.817	0.999	5.577	42.900	195.654
LMEM	10^6	7.882	1.704	4467	10681	4.926	0.999	5.686	43.009	195.763
(ours)	10^7	8.457	1.704	46384	106811	5.740	0.999	6.501	43.824	196.578
Baseline	10^4	12747	611	0.015	0.010	46	407	46	80	389
(LMEM)	10^5	20 h<	-	-	-	-	-	-	-	-

C++ implementation of [30], which is based on AHE. The algorithm for LPM in [28] for the string with $|\Sigma| \leq 4$ (e.g., genome sequence) is the same as [30]. [30] is implemented as a server-client software, and the client and the server were run with individual single threads on the same machine. Therefore, the results of [30] do not include delays caused by bandwidth limitation and latency, so we also estimated delays based on the data transfer size and round of communication in the same manner. Each run of the program was terminated if the total runtime of all phases exceeded 20 hours.

Comparison to Baseline Protocols. Table 3 shows the offline time and size, DB preparation time and size, and Search time and communication size for $N = 10^5, 10^6, 10^7$, and $\ell = 100$. It also shows the result of Baseline (LMEM) for $N = 10^4$, as the runs for $N > 10^4$ did not finish within 20 hours. The Search times and communication sizes of Secure LPM and Secure LMEM are several orders of magnitudes faster and smaller than those of Baseline (LPM) and Baseline (LMEM). Since the round and communication complexities of the proposed protocols do not depend on N, their estimated Search time remains small even on WAN environments. The left panel of Figure 5 shows the estimated Search time on WAN_1 for $N = 10^3, 10^4, \ldots, 10^7$ and $\ell = 100$. The times of Secure LPM and Secure LMEM do not increase, while those of the baseline protocols increase linearly to N. The right panel of Figure 5 shows the estimated Search time on WAN_1 for $\ell = 10, 25, \ldots, 100$ for $N = 10^6$. We can not show the results of Baseline (LMEM) because none of its runs were finished within the time limit. As shown in the graph, the time of Secure LPM increases linearly to ℓ and that of Baseline (LPM) increases proportionally to ℓ^2, which are in good agreement with the theoretical complexities in Table 2. According to the graph, the time of Secure LMEM also increases linearly to ℓ though its time and communication complexities are $O(\ell^2)$. This is because the CPU times are much smaller than the delays caused by network latency that are influenced by the round complexity $O(\ell)$.

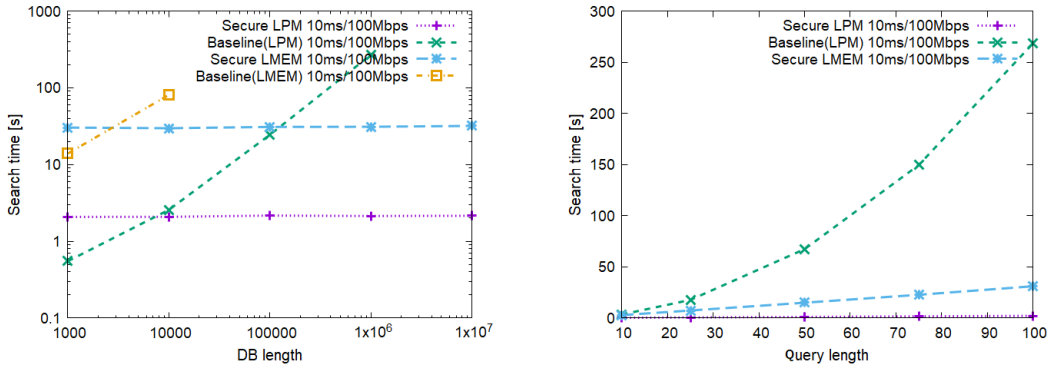

Figure 5 Estimated time (actual search time on a local machine + estimated data transfer time) for various N and ℓ.

We have preliminary results for testing Secure LPM and Baseline (LPM) on the actual network (10 ms/100 Mbps). The results were 40 sec for Secure LPM and 1739 sec for Baseline (LPM) when $N = 10^6$. Though both of the preliminary implementations have room for improvement in the performance of data transfer, the results also indicate that our protocol outperforms the baseline protocol and the previous study.

The time and size of Secure LPM and Secure LMEM are several orders of magnitude better than those of the baseline protocols for the offline phase, and vice versa for the DB preparation phase. The total time of the offline and DB preparation phases of our protocols are more than one order magnitude faster than that of baseline protocols. For the total size of the offline and DB preparation phases, Secure LMEM was better than Baseline (LMEM), but Baseline (LPM) was better than Secure LPM though the complexity is better for Secure LPM. This is because the majority of the shares were Boolean in the baseline protocols, while all of the shares were arithmetic in the proposed protocols.

Comparison to [30]. [30] is a two-party MPC based on AHE. Each homomorphic operation is time consuming and has no offline and DB preparation phases. As shown in Table 3, the Search time of Secure LPM is four orders of magnitude faster than [30] for $N = 10^6$. Since time complexity of [30] includes a factor of N, the difference in Search time becomes greater as N becomes large. Moreover, our protocols have a further advantage in communication for a query response when the network environment is poor, as the round complexity of [30] and our protocols are the same while [30] requires $O(\sqrt{N})$ communication size. The entire runtimes including all the phases are still six times faster for $N = 10^5$ and $N = 10^6$. We can compute LMEM by examining [30] for all the positions in a query string, but this approach consumed 3406 sec and 2.6 GByte of communication for $N = 10^4$.

5 Discussion

As clearly shown by the results, Search time of the proposed protocols are significantly efficient. Considering the importance of query response time for real applications, it is realistic to reduce Search time at the cost of DB preparation time. Since the total times for offline and DB preparation phases of the proposed protocols were significantly better than those of the well-designed baseline protocols, we consider the trade-off between Search and DB preparation times in our approach to be efficient. For further reduction of DB preparation time, parallelizing the share generation is a feasible approach. Regarding the DB preparation

phase, the data transfer between the server and the computing nodes is problematic when the number of queries and the length of the database are large. One potential solution to mitigate the problem is to use an AES-based random number generation that is similar to the technique used in [1]. To explain it briefly, when the server needs to distribute a share of x, (1) the server and P_0 generate the same randomness r using a pre-shared key and a pseudorandom function, and (2) the server computes $x - r$ and sends it to P_1. Although P_0's computation cost increases, we can remove the data transfer from the server to P_0. In our protocols, the generation of shares in the DB preparation phase cannot be outsourced because they are dependent on the database. Designing an efficient algorithm to outsource the share generation is an important open question.

References

1 Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proc. of CCS 2016*, pages 805–817, 2016. doi:10.1145/2976749.2978331.

2 Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. *PoPETs*, 2018(4):104–124, 2018. doi:10.1515/popets-2018-0034.

3 Md Momin Al Aziz, Md. Nazmus Sadat, Dima Alhadidi, Shuang Wang, Xiaoqian Jiang, Cheryl L. Brown, and Noman Mohammed. Privacy-preserving techniques of genomic data - a survey. *Briefings Bioinform.*, 20(3):887–895, 2019.

4 Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *Proc. of CCS 2011*, pages 691–702, 2011. doi:10.1145/2046707.2046785.

5 Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Proc. of CRYPTO 1991*, pages 420–432, 1991. doi:10.1007/3-540-46766-1_34.

6 Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang. Large-scale privacy-preserving mapping of human genomic sequences on hybrid clouds. In *Proc. of NDSS 2012*, 2012. URL: https://www.ndss-symposium.org/ndss2012/large-scale-privacy-preserving-mapping-human-genomic-sequences-hybrid-clouds.

7 Ke Cheng, Yantian Hou, and Liangmin Wang. Secure similar sequence query on outsourced genomic data. In *Proc. of AsiaCCS 2018*, pages 237–251, 2018. doi:10.1145/3196494.3196535.

8 Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. Homomorphic computation of edit distance. In *Proc. of FC 2015*, pages 194–212, 2015. doi:10.1007/978-3-662-48051-9_15.

9 Richard Durbin. Efficient haplotype matching and storage using the positional burrows–wheeler transform (pbwt). *Bioinformatics*, 30(9):1266–1272, 2014.

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

11 Paolo Ferragina and Giovanni Manzini. Opportunistic data structures with applications. In *Proc. of FOCS 2000*, pages 390–398, 2000. doi:10.1109/SFCS.2000.892127.

12 Johannes Fischer, Veli Mäkinen, and Gonzalo Navarro. An(other) entropy-bounded compressed suffix tree. In *Proc. of CPM 2008*, pages 152–165, 2008. doi:10.1007/978-3-540-69068-9_16.

13 Marc Fiume, Miroslav Cupak, Stephen Keenan, Jordi Rambla, Sabela de la Torre, Stephanie OM Dyke, Anthony J Brookes, Knox Carey, David Lloyd, Peter Goodhand, et al. Federated discovery and sharing of genomic data using beacons. *Nature biotechnology*, 37(3):220–224, 2019.

14 Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004. doi:10.1017/CBO9780511721656.

15 Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *Proc. of USENIX 2011*, 2011. URL: http://static.usenix.org/events/sec11/tech/full_papers/Huang.pdf.

16 Y. Ishimaki, H. Imabayashi, K. Shimizu, and H. Yamana. Privacy-preserving string search for genome sequences with fhe bootstrapping optimization. In *Proc. of IEEE Big Data 2016*, pages 3989–3991, 2016.

17 Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *Proc. of IEEE S&P 2000*, pages 216–230, 2008. doi:10.1109/SP.2008.34.

18 Md Safiur Rahman Mahdi, Md Momin Al Aziz, Noman Mohammed, and Xiaoqian Jiang. Privacy-preserving string search on encrypted genomic data using a generalized suffix tree. *Informatics in Medicine Unlocked*, 23:100525, 2021.

19 Payman Mohassel, Ostap Orobets, and Ben Riva. Efficient server-aided 2pc for mobile phones. *PoPETs*, 2016(2):82–99, 2016. doi:10.1515/popets-2016-0006.

20 Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *Proc. of IEEE S&P 2017*, pages 19–38, 2017. doi:10.1109/SP.2017.12.

21 Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 48(1):1–44, 2015.

22 Koji Nuida, Satsuya Ohata, Shigeo Mitsunari, and Nuttapong Attrapadung. Arbitrary univariate function evaluation and re-encryption protocols over lifted-elgamal type ciphertexts. *IACR Cryptology ePrint Archive*, 2019:1233, 2019. URL: https://eprint.iacr.org/2019/1233.

23 Satsuya Ohata and Koji Nuida. Communication-efficient (client-aided) secure two-party protocols and its application. In *proc. of FC 2020*, pages 369–385, 2020.

24 Anthony A Philippakis, Danielle R Azzariti, Sergi Beltran, Anthony J Brookes, Catherine A Brownstein, Michael Brudno, Han G Brunner, Orion J Buske, Knox Carey, Cassie Doll, et al. The matchmaker exchange: a platform for rare disease gene discovery. *Human mutation*, 36(10):915–921, 2015.

25 Victoria Popic and Serafim Batzoglou. A hybrid cloud read aligner based on minhash and kmer voting that preserves privacy. *Nature communications*, 8(1):1–7, 2017.

26 Thomas Schneider and Oleksandr Tkachenko. EPISODE: efficient privacy-preserving similar sequence queries on outsourced genomic databases. In *Proc. of AsiaCCS 2019*, pages 315–327, 2019. doi:10.1145/3321705.3329800.

27 Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

28 Kana Shimizu, Koji Nuida, and Gunnar Rätsch. Efficient privacy-preserving string search and an application in genomics. *Bioinformatics*, 32(11):1652–1661, 2016. doi:10.1093/bioinformatics/btw050.

29 Katerina Sotiraki, Esha Ghosh, and Hao Chen. Privately computing set-maximal matches in genomic data. *BMC Medical Genomics*, 13(7):1–8, 2020.

30 H. Sudo, M. Jimbo, K. Nuida, and K. Shimizu. Secure wavelet matrix: Alphabet-friendly privacy-preserving string search for bioinformatics. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 16(5):1675–1684, 2019.

31 Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proc. of CCS 2015*, pages 492–503, 2015. doi:10.1145/2810103.2813725.

32 Masaya Yasuda, Takeshi Shimoyama, Jun Kogure, Kazuhiro Yokoyama, and Takeshi Koshiba. Secure pattern matching using somewhat homomorphic encryption. In Ari Juels and Bryan Parno, editors, *Proc. of CCSW'13*, pages 65–76, 2013. doi:10.1145/2517488.2517497.

33 R. Zhu and Y. Huang. Efficient and precise secure generalized edit distance and beyond. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2020.

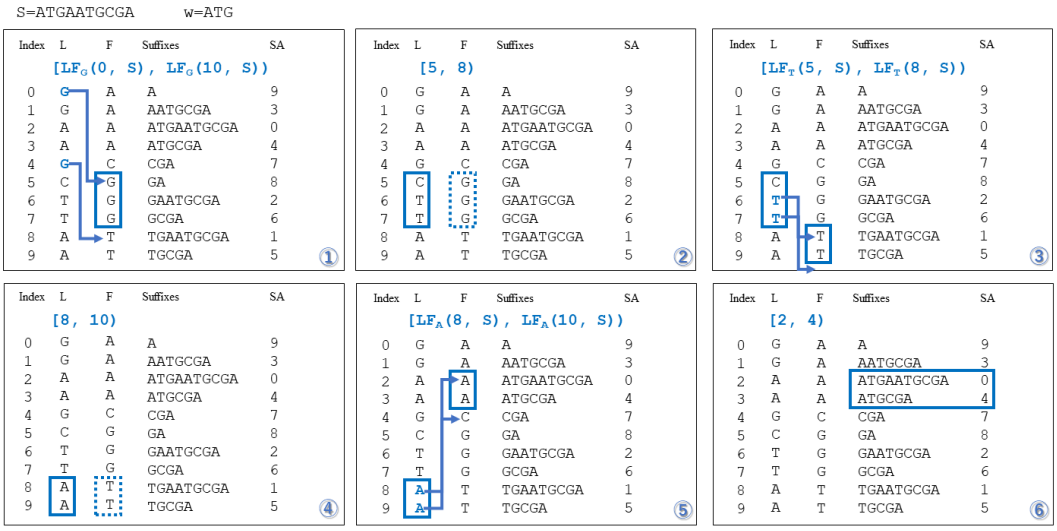

Figure A1 An example of search by FM-Index.

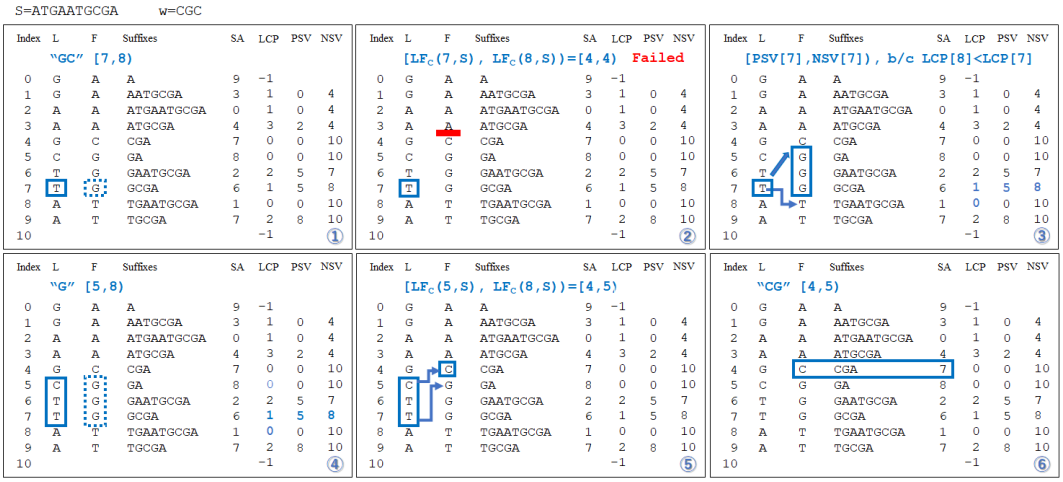

Figure A2 An example of search by FM-ndex, LCP array, PSV and NSV.

A Examples of a Search with FM-Index and Auxiliary Data Structures

Let us show examples of a search with FM-Index, LCP array, PSV and NSV. In addition to the data structures defined in Section 2.2, we also define a string F such that $F[i] = S[SA[i]]$. For the case of S =ATGAATGCGA, the indices become $SA = (9, 3, 0, 4, 7, 8, 2, 6, 1, 5)$, $L =$ GGAAGCTTAA, and $F =$ AAAACGGGTT. Figure A1 illustrates the example of a backward search to find the longest suffix of the query (ATG) that matches the database, and Figure A2 illustrates the search for MEMs with the query (CGC) by using LCP array, PSV, and NSV. As shown in the upper center panel of Figure A2, the search failed when the backward search with 'C' after finding the interval $[7, 8)$ that corresponds to GC. Since $\mathsf{LCP}[8] \leq \mathsf{LCP}[7]$, the parent lcp-interval becomes $[\mathsf{PSV}[7] = 5, \mathsf{NSV}[7] = 8)$, which corresponds to 'G'. The match CG is then searched with the backward search with 'C' from the parent lcp-interval.

B Semi-Honest Security

Here, we recall the simulation-based security notion in the presence of semi-honest adversaries (for two-party computation), as in [14].

▶ **Definition 5.** *Let* $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ *be a probabilistic* 2*-ary functionality and* $f_i(\vec{x})$ *denote the* i*-th element of* $f(\vec{x})$ *for* $\vec{x} = (x_0, x_1) \in (\{0,1\}^*)^2$ *and* $i \in \{0,1\}$*;* $f(\vec{x}) = (f_0(\vec{x}), f_1(\vec{x}))$*. Let* Π *be a* 2*-party protocol to compute the functionality* f*. The view of party* P_i *for* $i \in \{0,1\}$ *during an execution of* Π *on input* $\vec{x} = (x_0, x_1) \in (\{0,1\}^*)$ *where* $|x_0| = |x_1|$*, denoted by* $\mathrm{VIEW}_i^{\Pi}(\vec{x})$*, consists of* $(x_i, r_i, m_{i,1}, \ldots, m_{i,t})$*, where* x_i *represents* P_i*'s input,* r_i *represents its internal random coins, and* $m_{i,j}$ *represents the* j*-th message that* P_i *has received. The output of all parties after an execution of* Π *on input* $\vec{x}$ *is denoted as* $\mathrm{OUTPUT}^{\Pi}(\vec{x})$*. Then, for each party* P_i*, we say that* Π privately computes f *in the presence of semi-honest corrupted party* P_i *if there exists a probabilistic polynomial-time algorithm* $\mathcal{S}$ *such that*

$$\{(\mathcal{S}(i, x_i, f_i(\vec{x})), f(\vec{x}))\} \equiv \{(\mathrm{VIEW}_i^{\Pi}(\vec{x}), \mathrm{OUTPUT}^{\Pi}(\vec{x}))\},$$

where the symbol $\equiv$ *means that the two probability distributions are statistically indistinguishable.*

As described in [14], the composition theorem for the semi-honest model holds; that is, any protocol is privately computed as long as its subroutines are privately computed.

C Our Secure Baseline LPM and LMEM

In this section, we show our secure baseline LCP and LMEM based on secret sharing. We explain how to construct LCP, since we can obtain LMEM by (parallelly) executing LCP for all positions in the query. Note that $\vec{x} = (x_1, x_2, \cdots)$, $\vec{x_i}$ denotes an i-th element of $\vec{x}$, $[\![\vec{t}]\!] = ([\![\vec{t}]\!]_0, [\![\vec{t}]\!]_1)$, and $(|\vec{x}|, |\vec{y}|) = (L, N)$. Here, we assume $N > L$. When $[\![\vec{x}]\!] = ([\![x_1]\!], [\![x_2]\!], \cdots, [\![x_p]\!])$, $[\![\vec{x}]\!] \gg 1$ means $([\![0]\!], [\![x_1]\!], \cdots, [\![x]\!]_{p-1})$. In our protocol, we use two subprotocols as follows:

- All-AND takes a list $[\![\vec{t}]\!]$ (with p Boolean shares) as input and outputs $[\![t_1 \wedge \cdots \wedge t_p]\!]^B$. We can compute this function with $\lceil p \rceil$ communication rounds (by appropriate parallelization) and $O(NL)$-bit data transfer. The total data transfer size for this All-AND is $\{-\frac{1}{3}L^3 + \frac{1}{2}(N+1)L^2 - \frac{1}{6}(3N+1)L\}$-bit since we execute All-AND at most $(N-1)$ times.
- All-OR takes a list $[\![\vec{u}]\!]$ (with p Boolean shares) as input and outputs $[\![u_1 \vee \cdots \vee u_p]\!]^B$. We can compute this function with $\lceil p \rceil$ communication rounds (by appropriate parallelization) and $O(N)$-bit data transfer. The total data transfer size for this All-OR is $\{-\frac{1}{2}L^2 + \frac{1}{2}(2N-1)L\}$-bit since we execute All-OR for L times.

Our protocol is as in Protocol A1.

In the following, we explain the details of our baseline longest common prefix search protocol using an example that strings $\vec{x}$ = "TGA" and $\vec{y}$ = "ATTGC". In this example, $w = 2$ since there exists "TG" in $\vec{y}$, but "TGA" does not. First, we check whether $w = 1$ or not. To achieve this functionality, we check whether the first character of $\vec{x}$ (i.e., "T" in the example) exists in $\vec{y}$ using Equality for N times. If there exists at least one True in this calculation result, it means $w = 1$ (or larger), and we can achieve this functionality using All-OR. Then, we check whether $w = 2$ or not; that is, we check whether there exists "TG" in the subsequence of $\vec{y}$. Here, we only need to consider ("AT", "TT", "TG", "GC") as subsequences in our example. We search for the perfect matching using Equality and

Protocol A1 Baseline Secure LPM.

Functionality: Compute the length of the longest common prefix w
Input: Strings $[\![\vec{x}]\!]$ and $[\![\vec{y}]\!]$, where $(|\vec{x}|, |\vec{y}|) = (L, N)$
Output: $[\![w]\!]$

for $i = 1, \cdots, L$ **do**
  for $j = 1, \cdots, N$ **do**
    $[\![\vec{s}_{i,j}]\!]^B = \mathsf{Equality}([\![\vec{x}_i]\!], [\![\vec{y}_j]\!])$
  end for
end for
for $i = 1, \cdots, L$ **do**
  P_I $(I \in 0, 1)$ locally generates an empty list $[\![\vec{u}]\!]_I$.
  for $j = 1, \cdots, N - i + 1$ **do**
    if $i = 1$ **then**
      P_I locally adds $[\![\vec{s}_{1,j}]\!]_I^B$ to $[\![\vec{u}]\!]_I$.
    else
      P_I locally generates an empty list $[\![\vec{t}]\!]_I$.
      for $k = 1, \cdots, i$ **do**
        P_I locally add $[\![\vec{s}_{k,k+j-1}]\!]_I^B$ to $[\![\vec{t}]\!]_I$.
      end for
      P_I adds $\mathsf{All\text{-}AND}([\![\vec{t}]\!])$ to $[\![\vec{u}]\!]_I$.
    end if
  end for
  $[\![\vec{v}_{L-i+1}]\!]^B = \mathsf{All\text{-}OR}([\![\vec{u}]\!])$
end for
$[\![\vec{v}]\!]^B = [\![\vec{v}]\!]^B \oplus ([\![\vec{v}]\!]^B \gg 1)$
$[\![\vec{v}]\!] = \mathsf{B2A}([\![\vec{v}]\!]^B)$
$[\![w]\!] = \sum_{\ell=1}^{L} [\![\vec{v}_\ell]\!] \cdot \ell$
return $[\![w]\!]$

All-AND in our protocol. The condition $w = 2$ (or larger) holds if there is at least one perfect matching, and we can achieve this functionality using All-OR. We can compute the cases of $w \leq 3$ using almost the same strategy. After that, we extract the number of the longest common prefix. In the above procedure, we can obtain (False, True, True) for $(w = 3, w = 2, w = 1)$, respectively. We can extract the leftmost True using 1-bit right-shift and XOR, which is a common technique for constructing secure protocols. Finally, we can obtain a final output $[\![w]\!]$ using B2A and the inner product (with constant numbers). Note that we can optimize Equality by replacing simple OR. This is because all characters in $\vec{x}$ and $\vec{y}$ are {"A", "T", "G", "C"}, and we can represent them using 2-bit arithmetic sharing. With an appropriate parallelization, we can execute Protocol A1 with $O(\log \lceil L \rceil + \log \lceil N \rceil)$ communication rounds.

D Proof of Theorem 2

Proof. Correctness and security of ss-ROT protocol are proved as follows.

Proof of correctness. We assume the following equation.

$$p_i = (V^{(i)}[p_0] + r^{i-1}) \bmod N \tag{8}$$

In Step 1, for $j = 0$, the protocol computes p_1 by reconstructing $R^0[p_0]$. From the definition of $R^j[i]$ in Eq. 4,

$$p_1 = R^0[p_0] = (V^{(1)}[p_0] + r^0) \bmod N. \tag{9}$$

For $j = k$, the protocol computes p_{k+1} by reconstructing $R^k[p_k]$. From the definition of $R^j[i]$ in Eq. 4 and the assumption of Eq. 8,

$$\begin{aligned} p_{k+1} = R^k[p_k] &= (V[\,(p_k - r^{k-1}) \bmod N\,] + r^k) \bmod N \\ &= (V[\,V^{(k)}[p_0]\,] + r^k) \bmod N \\ &= (V^{(k+1)}[p_0] + r^k) \bmod N. \end{aligned} \tag{10}$$

Eq. 8 holds for $i = 1$ by Eq. 9. It also holds for $i = k + 1$ under the assumption that Eq. 8 holds for $i = k$. Therefore by induction, Eq. 8 holds for $i = 1, \ldots, \ell - 1$.

In Step 2, P_0 and P_1 output $[\![R^{\ell-1}[p_{\ell-1}]]\!]$. Since Eq. 8 holds for $i = \ell - 1$,

$$R^{\ell-1}[p_{\ell-1}] = (V[(p_{\ell-1} - r^{\ell-2}) \bmod N]) \bmod N$$

is transformed into $(V^{(\ell)}[p_0]) \bmod N$ by plugging in $p_{\ell-1} = V^{(\ell-1)}[p_0] + r^{\ell-2}$. Therefore the final output of ss-ROT becomes $[\![(V^{(\ell)}[p_0]) \bmod N]\!]$. The above argument completes the proof of correctness of Theorem 2.

Proof of security. Due to space limits, we only show a sketch of the proof. In the DB preparation phase of ss-ROT, $\mathcal{B}$ does not disclose any private values, and P_0 and P_1 receive the shares. In the Search phase, all the messages exchanged between P_0 and P_1 are shares except for the result of Reconst in Step 1. In the j-th step of the loop in Step 1, $p_{j+1} = R^j[p_j] = (V^{(j+1)}[p_0] + r^j) \bmod N$ is reconstructed. Since the reconstructed value is randomized by r^j, no information is leaked. Note that for each vector R^j, all the elements $R^j[0], \ldots, R^j[N-1]$ are randomized by the same value r^j, but only one of them is reconstructed, and different random numbers $r^0, \ldots, r^{\ell-1}$ are used for $R^0, \ldots, R^{\ell-1}$. In Step 2, P_0 and P_1 output a result, and no information other than the result is leaked. ◀

E Proof of Theorem 3

Proof. Correctness and security of Protocol 2 are proved as follows.

Proof of correctness. The lookup table V simply stores all possible outputs of LF. Therefore, backward search (Eq. 1) is equivalent to Eq. 5. For the case of querying w, $V_{w[k-1]}[\cdots V_{w[0]}[p_0]\cdots]$ becomes lower bound f (for $p_0 = 0$) or upper bound g (for $p_0 = N$) of the interval that corresponds to the prefix match of length k. In Line 5 of Protocol 2, $[\![R^k_{\mathrm{A},f}[f_k] \times q_{\mathrm{A}}[k] + R^k_{\mathrm{C},f}[f_k] \times q_{\mathrm{C}}[k] + R^k_{\mathrm{G},f}[f_k] \times q_{\mathrm{G}}[k] + R^k_{\mathrm{T},f}[f_k] \times q_{\mathrm{T}}[k]]\!]$ is computed. Since $q_{w[j]}[j] = 1$ and $q_c[j] = 0$ $(c \neq w[j])$, it is equivalent to $[\![R^k_{w[k],f}[f_k]]\!]$. Line 6 computes $[\![R^j_{w[k],g}[g_k]]\!]$ in the same manner. Each vector $R^j_{c,f}$ in Eq. 6 is generated in the same manner as R^j in Eq. 4. Since Eq. 6 uses the common random values r^j_f and r^{j-1}_f for $R^j_{\mathrm{A},f}$, $R^j_{\mathrm{C},f}$, $R^j_{\mathrm{G},f}$, $R^j_{\mathrm{T},f}$, we can recursively reference V_c $(c \in \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\})$, which is obvious from the correctness of ss-ROT. Therefore, the recursion by Line 5 and Line 7 can compute $(V_{w[k-1]}[\cdots V_{w[0]}[f_0]\cdots] + r^{k-1}_f) \bmod N'$, and the recursion by Line 6 and Line 8 can also compute $(V_{w[k-1]}[\cdots V_{w[0]}[g_0]\cdots] + r^{k-1}_g) \bmod N'$.

The longest match is found when the interval width becomes 0. Since $f_k = (V_{w[k-1]}[\cdots V_{w[0]}[f_0]\cdots] + r_f^{k-1}) \bmod N'$ and $g_k = (V_{w[k-1]}[\cdots V_{w[0]}[g_0]\cdots] + r_g^{k-1}) \bmod N'$ are randomized, Line 11 computes $f_k - g_k - (r'[k-1] = r_f^{k-1} - r_g^{k-1})$ to obtain the correct interval width. Line 11 also computes the equality of 0 and the interval width for each round. By reconstructing all the results in Lines 15–17, $\mathcal{A}$ knows the round, in which the interval width becomes 0; i.e., he/she knows LPM. The above argument completes the proof of correctness of Theorem 3.

Proof of security. Due to space limitation, we only show a sketch of the proof. For Lines 1–2 of Protocol 2, $\mathcal{A}$ and $\mathcal{B}$ do not disclose any private values, and P_0 and P_1 receive the shares. For Lines 3–13, it is guaranteed by the subprotocols ADD, MULT, and Equality that all the messages exchanged between P_0 and P_1 are shares except for the output of Reconst in Lines 7–8. (See Section 2.1 for details of the subprotocols.) In Lines 7–8, reconstructed values are $R^k_{w[j],f}[f_j]$ and $R^k_{w[j],g}[g_j]$. Since the values are $(V_{w[j]}[f_j]+r_f^j) \bmod N'$ and $(V_{w[j]}[g_j] + r_g^j) \bmod N'$ according to Eq. 6, it is obvious that V is randomized for all rounds $j = 0, \ldots, \ell - 1$, and no information is leaked. For Lines 14–17, only the output of Equality at Line 11 is reconstructed. The reconstructed values are either 1 or 0 according to Equality, and no information other than the result is leaked. ◀

F Proof of theorem 4

Proof. Correctness and security of Protocol 3 are proved as follows.

Proof of correctness. V, R, r' and q are generated by the same algorithm used in Protocol 2. Therefore, Line 6 is equivalent to a backward search, and $e1$ is the result of the equality check of 0 and the width of the obtained interval in Line 7. The lookup tables V_{lcp}, V_{psv}, and V_{nsv} store all the outputs of LCP, PSV and NSV, and W_l, W_p, and W_n are generated based on V_{lcp}, V_{psv}, and V_{nsv}, respectively. Since $W^j_{l,f}$ and $W^j_{l,g}$ are circular permutations of V_{lcp} by the same random values r_f^{j-1} and r_g^{j-1} that are used for generating $R_{c,f}$ and $R_{c,g}$ ($c \in \Sigma$) respectively, Line 8 can compute $\mathsf{LCP}[g_j] \leq \mathsf{LCP}[f_j]$ and $e2$ holds the result. By using Choose and $e2$, either $[W^j_{p,f}[f_j], W^j_{n,f}[f_j])$ or $[W^j_{p,g}[g_j], W^j_{n,g}[g_j])$ is selected. $W^j_{p,f}$ and $W^j_{p,g}$ are permutated by r_f^{j-1} and r_g^{j-1}, but are randomized by the identical random value r_f^j. Similarly, $W^j_{n,f}$ and $W^j_{n,g}$ are permutated by r_f^{j-1} and r_g^{j-1}, but are randomized by r_g^j. Since $W_{p,f}[f_j]$ and $W^j_{n,g}[g_j]$ are generated in the same manner as $R_{c,f}$ and $R_{c,g}$, it is obvious that the reference by them is correct. The reference by $W^j_{n,f}[f_j]$ is transformed into

$$\begin{aligned} X_g^{j+1}[W^j_{n,f}[f_j]] &= V_x[W^j_{n,f}[f_j] - r_g^j] + r_g^{j+1} \\ &= V_x[V_{nsv}[f_j - r_f^{j-1}] + r_g^j - r_g^j] + r_g^{j+1} \\ &= V_x[\, V_{nsv}[f_j - r_f^{j-1}]\,] + r_g^{j+1} \end{aligned} \tag{11}$$

and the reference by $W^j_{p,f}[g_j]$ is transformed into

$$\begin{aligned} X_f^{j+1}[W^j_{p,g}[g_j]] &= V_x[W^j_{p,g}[g_j] - r_f^j] + r_f^{j+1} \\ &= V_x[V_{psv}[g_j - r_g^{j-1}] + r_f^j - r_f^j] + r_f^{j+1} \\ &= V_x[\, V_{psv}[g_j - r_g^{j-1}]\,] + r_f^{j+1} \end{aligned} \tag{12}$$

where X^{j+1} is any one of R_c^{j+1}, W_p^{j+1} and W_n^{j+1}, and V_x is the corresponding lookup table; i.e., either one of V_c, V_{psv} and V_{nsv}. Note that V_x could be a different table for each $j+1$, but we abuse the same notation for simplicity of notation. Since f_j and g_j are described in the form of $V_x^{(j)}[p_0] + r_f^{j-1}$ and $V_x^{(j)}[p_0'] + r_g^{j-1}$ based on Eq. 8, Eq. 11 and Eq. 12 are transformed into $V_x^{(j+2)}[p_0] + r_g^{j+1}$ and $V_x^{(j+2)}[p_0'] + r_f^{j+1}$, which also satisfy the recursion form of Eq. 8. Thus, the intervals $[W_{p,f}^j[f_j], W_{n,f}^j[f_j])$ and $[W_{p,g}^j[g_j], W_{n,g}^j[g_j])$ are correct intervals and Line 9 is equivalent to computing Eq. 2.

Lines 14–18, u remains the same if $e1 = 0$ and is permutated such that $u[i] = u[(i-1) \bmod \ell]$ otherwise. Therefore Lines 19–21 can choose the letter to be searched with. The match length and the start position are obtained based on $e1$ in Lines 22–23, and the longest value and the corresponding position are selected in Lines 24–25. The shares of the length and start position of LMEM are sent to $\mathcal{A}$, and $\mathcal{A}$ reconstructs them. Then, Protocol 3 outputs them. The above argument completes the proof of correctness of Theorem 4.

Proof of security. Due to space limits, we only show a sketch of the proof. For Lines 1–2 of Protocol 3, $\mathcal{A}$ and $\mathcal{B}$ do not disclose any private values, and P_0 and P_1 receive the shares. For Lines 3–27, it is guaranteed by the subprotocols ADD, MULT, Equality, and Choose that all the messages exchanged between P_0 and P_1 are shares except for the output of Reconst in Line 12. (See Section 2.1 for details of the subprotocols.) In Line 12, the reconstructed values are $f_{i+1} = V_x^{(j+1)}[p_0] + r_f^j$ and $g_{j+1} = V_x^{(j+1)}[p_0] + r_g^j$, according to Eq. 8, Eq. 11, and Eq. 12. Since f_{j+1} and g_{j+1} are randomized by r_f^j and r_g^j, respectively, for all rounds $j = 0, \ldots, 2\ell - 1$, no information is leaked. In Line 28, $\mathcal{A}$ reconstructs only the search result (the length and start position of LMEM). ◀