Secure Wavelet Matrix: Alphabet-Friendly Privacy-Preserving String Search for Bioinformatics

Hiroki Sudo, Masanobu Jimbo, Koji Nuida, and Kana Shimizu

Abstract—Biomedical data often includes personal information, and the technology is demanded that enables the searching of such sensitive data while protecting privacy. We consider a case in which a server has a text database and a user searches the database to find substring matches. The user wants to conceal his/her query and the server wants to conceal the database except for the search results. The previous approach for this problem is based on a linear-time algorithm in terms of alphabet size $|\Sigma|$, and it cannot search on the database of large alphabet such as biomedical documents. We present a novel algorithm that can search a string in logarithmic time of $|\Sigma|$. In our algorithm, named secure wavelet matrix (sWM), we use an additively homomorphic encryption to build an efficient data structure called a wavelet matrix. In an experiment using a simulated string of length 10,000 whose alphabet size ranges from 4 to 1024, the run time of the sWM was up to around two orders of magnitude faster than that of the previous method. sWM enables the searching of a private database efficiently and thus it will facilitate utilizing sensitive biomedical information.

Index Terms—String search, privacy, wavelet matrix, FM-index, homomorphic encryption

1 Introduction

Privacy protection is an emerging problem in the field of life science where many data are private, such as clinical documents, health records, personal genomes and protein sequences. Currently, the most common approach for protecting such data is to limit access by segregating the data, which eventually deteriorates their value. To overcome this problem, a technology is eagerly demanded that enables such sensitive data to be searched while maintaining individual privacy. Such a technology is considered as one of privacy-preserving data mining technologies. (See related books and survey papers such as: [1], [2], [3] for more details about privacy-preserving data mining).

- *H. Sudo and M. Jimbo are with the the Department of Computer Science and Communications Engineering, Faculty of Science and Engineering, Waseda University, Tokyo 169-8050, Japan, and the AIST-Waseda University Computational Bio Big-Data Open Innovation Laboratory (CBBD-OIL), Tokyo 169-0072, Japan. E-mail: hsudo108@ruri.waseda.jp, jimwase@asagi.waseda.jp.*
- *K. Nuida is with Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 100-8921, Japan, and the Japan Science and Technology Agency, Kawaguchi-shi 332-0012, Japan. E-mail: k.nuida@aist.go.jp.*
- *K. Shimizu is with the the Department of Computer Science and Communications Engineering, Faculty of Science and Engineering, Waseda University, Tokyo 169-8050, Japan, the AIST-Waseda University Computational Bio Big-Data Open Innovation Laboratory (CBBD-OIL), Tokyo 169-0072, Japan, and the Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST), Tokyo 100-8921, Japan. E-mail: shimizu.kana@waseda.jp.*

Manuscript received 4 Dec. 2017; accepted 3 Mar. 2018. Date of publication 8 Mar. 2018; date of current version 7 Oct. 2019.
(Corresponding author: Kana Shimizu.)
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TCBB.2018.2814039

In this study, we consider a typical scenario where the user wants to search on a text database but does not wish to show his/her query to the server, and the server wants to return only the search result and does not want the user to obtain any other information. In a previous study aiming to achieve such a scenario, Freedman et al. introduced a keyword search [4], in which both the server and the user have a set of keywords and only the user knows common keywords. Bruekers et al. also developed a similar approach and applied it to a genetic test [5]. There is another line of studies aiming to evaluate similarity of a query and each database entry by computing a Jaccard Index of two keyword lists. Applications have been developed for a DNA sequence search [6], [7] and a chemical compound search [8]. In addition to those keyword-based searches, a privacy-preserving substring search is also an important problem in bioinformatics. In a substring search, the server has a long text and the user has a query (a relatively shorter text), and only the user knows the positions where the query matches the server's text. In previous work [9], a DNA sequence search that mainly targets an exact match was demonstrated using garbled circuit [10] which is a relatively time and communication-size consuming approach. Series of recent studies by Hazay et al. [11], Vergnaud et al. [12], Baron et al. [13] and Yasuda et al. [14] presented more efficient methods to find matches that have a small fixed Hamming distance. While those studies aimed to evaluate similarity between an entire query and a server's text, another study [15] aimed for a variable-length prefix/suffix match either on a regular text or a set of aligned texts such as SNP sequences (e.g., it enables the user to find only occurrences of the longest prefix/suffix match), which is also useful for various practical settings. In this study, we also tackle the same pattern matching problem. In principle, the method in this previous study

(called PBWT-sec) can be applied for any type of text data. However, it does not perform sufficiently for practical problems when it is used for searching on a text with a large alphabet size $|\Sigma|$, such as a protein sequence ($|\Sigma| = 21$) and clinical documents (e.g., $|\Sigma| = 36$ for Roman alphabet and Arabic numerals, $|\Sigma| = 83$ for Japanese alphabet), because the algorithm was originally designed for haploid genome sequences ($|\Sigma| = 2$) and its time complexity is linear to $|\Sigma|$. To overcome this critical drawback, here we present an efficient algorithm named the secure Wavelet Matrix (sWM) that searches on an efficient data structure called a wavelet matrix [16], [17], [18] by using additively homomorphic encryption [19]. As we will discuss in Section 3.5, sWM significantly improves asymptotic performance compared to PBWT-sec (see Table 1). As a result, the sWM is up to several orders of magnitude faster than the PBWT-sec in experimental results with a large alphabet size as shown in Section 4, with slight increase of communication rounds and, thus it can practically deal with various types of text data even over networks with high latency. Moreover, our new method even improves communication complexity despite the previous approach requires only sublinear communication size in terms of database length. We will also describe a detailed algorithm to use sWM for a full-text search based on FM-Index [20].

The rest of the paper is organized as follows. In Section 2, we describe the main idea of our approach, which is followed by Section 3 where the detailed algorithms of sWM and its application to full-text search are provided. We also describe the complexity of the algorithms and compare it with that of previous methods. In Section 4, we evaluate the performance of our method both on a simulated dataset and two real datasets (protein sequences and clinical texts) and compare our method with a previous approach and a baseline approach. Finally, we conclude our study in Section 5.

2 Approach

2.1 Problem Setting

In this study, we assume a case in which a user has a query text $\boldsymbol{q}$ and a database holder has a long text S, and the user wants to know only the longest prefix of $\boldsymbol{q}$ that matches a substring of S. As explained in Section S1 in the supplementary material, the proposed method is easily modified to find the longest prefix that matches more than ϵ different positions in S given the threshold ϵ, which may fit more practical applications. For ease of explanation, we describe a series of algorithms specialized for searching on a single long text, but the same approach is able to search on other types of text data such as a set of aligned texts by modifying it slightly.

2.2 Overview

To explain the central idea of our approach, we will first briefly describe PBWT-sec [15]. PBWT-sec is designed to search on an iteratively queriable data structure such as FM-Index, in which a database text is efficiently indexed. During the search using such a data structure, a match between a query and a database text is reported as a left-closed and right-open interval $[f, g)$ on the auxiliary data structure, that is, BWT of the original text as shown in Fig. 1.

We do not provide a detailed mechanism of the data structure. However, to understand the key idea, two points

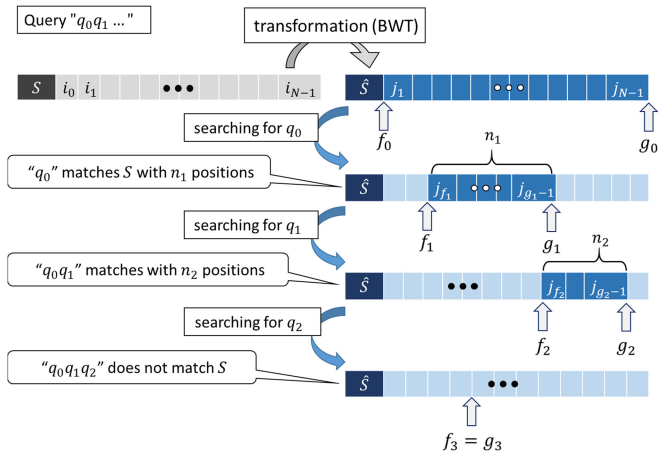

Fig. 1. Schematic view of search on recursive data structure such as FM-Index. Each match is represented by interval $[f, g)$, and interval is updated by previously computed interval.

are sufficient to know. (1) The lower bound f and upper bound g of the interval are computed by a function called RankCF. (2) An interval $[f_{k+1}, g_{k+1})$ for a prefix match of length $k+1$ between a query text $\boldsymbol{q}$ and a database text S is computed in the form of:

$$\begin{aligned} f_{k+1} &= \mathsf{RankCF}(\hat{S}, f_k, q[k+1]), \\ g_{k+1} &= \mathsf{RankCF}(\hat{S}, g_k, q[k+1]), \end{aligned} \tag{1}$$

where $\hat{S}$ is an auxiliary text such as BWT of S, and $\boldsymbol{q}[k+1]$ is the $(k+1)$th letter of $\boldsymbol{q}$.

As we explain in detail later in this section, RankCF is pre-computable and stored in a long lookup table $\boldsymbol{v}$, and thus $[f, g)$ can be obtained by referring to $\boldsymbol{v}$:

$$f_{k+1} = \boldsymbol{v}[f_k + o_k], \quad g_{k+1} = \boldsymbol{v}[g_k + o_k],$$

where o_k is proper offsets necessary for storing all pre-computed values. Therefore, the interval for a prefix match of length $k+1$ can be obtained by referring to the same table $\boldsymbol{v}$ recursively as follows.

$$\begin{aligned} f_{k+1} &= \boldsymbol{v}[\boldsymbol{v}[\ldots \boldsymbol{v}[f_0 + o_0] \ldots] + o_{k-1}] + o_k], \\ g_{k+1} &= \boldsymbol{v}[\boldsymbol{v}[\ldots \boldsymbol{v}[g_0 + o_0] \ldots] + o_{k-1}] + o_k]. \end{aligned}$$

To enable the user to refer the lookup table held by the server, a cryptographic technique called a *Recursive Oblivious Transfer* (ROT) protocol [21] is used, by which $\boldsymbol{v}[\boldsymbol{v}[\ldots \boldsymbol{v}[i] \ldots]]$ is returned to the user when the user inputs an index i and the server inputs a vector $\boldsymbol{v}$ to the protocol. In the process of the ROT protocol, the user and the server repeat communication, and in each round, the user obtains $(\boldsymbol{v}[x] + r)_{\mathrm{mod}\ N}$ where x is the user's input, r is the server's input (a random factor) and N is the length of $\boldsymbol{v}$. All the user's inputs are encrypted by the user's secret key before it is sent to the server, and the server computes return values on the basis of the user's encrypted values without decryption. Since the server does not have the secret key, the user's inputs are not leaked to the server. On the other hand, the user decrypts the returned values, but cannot see $\boldsymbol{v}[x]$ because it is randomized by the server's private value r. The key feature of ROT is that when the user inputs previously returned value $(\boldsymbol{v}[x] + r)_{\mathrm{mod}\ N}$, the server

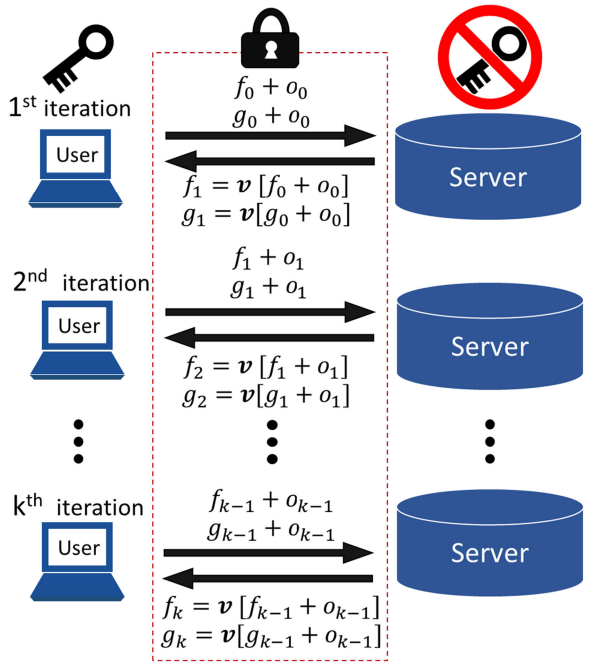

Fig. 2. Schematic view of communication between user and server for computation of interval $[f, g)$. Lower bound f_k and upper bound g_k are obtained by $f_{k+1} = \boldsymbol{v}[f_k + o_k]$ and $g_{k+1} = \boldsymbol{v}[g_k + o_k]$, respectively. All intermediates $f_0, \ldots, f_k$ and $g_0, \ldots, g_k$ are concealed by using ROT protocol.

properly removes r from the user's input and returns $(\boldsymbol{v}[\boldsymbol{v}[x]] + r')_{\mathsf{mod}\ N}$ where r' is another random factor. At the end of the protocol, the user can obtain a non-randomized result if the server does not input a random factor. In this way, ROT enables the user to refer to $\boldsymbol{v}$ without leaking any intermediate information. (See Section 3.2 for more details about the algorithm of ROT.) By using this property and an additional cryptographic technique, the PBWT-sec can safely compute f and g until it finds the longest prefix (i.e., $g - f$ becomes 0). Fig. 2 illustrates how the user and the server communicate when computing the interval.

The ROT task is a computational bottleneck of the PBWT-sec, and the time complexity is linear to the length of $\boldsymbol{v}$. The goal of this study is to improve the time complexity of the ROT task. Let us give more details about $\boldsymbol{v}$ and RankCF, which are key building blocks of ROT. Given an index p, a character c and a sorted text $\hat{S}$ (such as BWT) of length N, $\mathsf{RankCF}(\hat{S}, p, c)$ is defined as follows.

$$\mathsf{RankCF}(\hat{S}, p, c) = |\{\, i \mid \hat{S}[i] = c, 0 \le i < p \,\}| + |\{\, i \mid \hat{S}[i] < c, 0 \le i < N \,\}|.$$

We denote $\hat{S}[i] < c$, when a letter $\hat{S}[i]$ is lexicographically smaller than a letter c. For example, if $\hat{S}$ = "MDGGIPQAGG", $p = 5$ and $c =$ 'G', $\mathsf{RankCF}(\hat{S}, 5, \text{'G'})$ becomes 4 because the two leftmost 'G's are within the first five letters, and 'D' and 'A' are lexicographically smaller than 'G'.

The straightforward method to store all outputs of RankCF for the $\hat{S}$ of length N, $0 \le p \le N$, and $c \in \Sigma$ is to create a lookup table of length $(N + 1) \times |\Sigma|$. In fact, PBWT-sec designs $\boldsymbol{v}$ in this simple way, and thus its computational complexity is linear to $|\Sigma|$.

To reduce the total cost for the ROT task, we propose a novel approach using *wavelet matrix* [16] to design $\boldsymbol{v}$ very efficiently.

2.3 Wavelet Matrix

A wavelet matrix (WM) is an efficient data structure that supports a wide range of query operations such as RankCF. It achieves logarithmic-time complexity in terms of alphabet size $|\Sigma|$ while keeping space complexity close to information-theoretic lower bound. Given a text T_0, the key feature of WM is to encode each letter in a binary form and to compute RankCF bit by bit from the least to the most significant bit in order to compute final RankCF for T_0. Let us describe this process in more detail. A WM algorithm creates a bit array $B_0 = b^0(T_0[0]), \ldots, b^0(T_0[N-1])$, where $b^i(c)$ denotes ith bit of the binary encoding of c. The algorithm obtains new text T_1 by stably sorting characters in T_0 by the most significant bit, and another bit array $B_1 = b^1(T_1[0]), \ldots, b^1(T_1[N-1])$ is created in the next step. B_i is created from B_{i-1} in a similar manner until i reaches $\lambda - 1$ when a bit length $\lambda = \lceil \log_2 |\Sigma| \rceil$. There is the recursion relationship between an index p_i for the ith round and an index p_{i+1} for the $i + 1$th round: $p_{i+1} = \mathsf{RankCF}(B_i, p_i, b^i(c))$ which leads to $\mathsf{RankCF}(T_0, p_0, c) = \mathsf{RankCF}(B_{\lambda-1}, p_{\lambda-1}, b^{\lambda-1}(c))$. Fig. 3 illustrates an example of a search on WM.

2.4 Efficient Design Principle of Lookup Table $\boldsymbol{v}$

Here, we explain how to design the lookup table $\boldsymbol{v}$ efficiently. As described in Section 2.3, $\mathsf{RankCF}(T_0, p_0, c)$ is computed by repeating RankCF on auxiliary bit-arrays $B_0, \ldots, B_{\lambda-1}$. In our approach, we create a set of sub-lookup tables $\boldsymbol{v}^i$ for $i = 0, \ldots, \lambda - 1$, each of which corresponds to a bit-array B_i as follows, and use ROT to refer to $\boldsymbol{v}^i$.

$$\boldsymbol{v}^i[p + x \times (N+1)] = \mathsf{RankCF}(B_i, p, x) \quad (x \in \{0, 1\}).$$

The outline of our method sWM is as follows. Note that the goal of sWM is to compute $\mathsf{RankCF}(T_0, p_0, c)$.

Step 1 The server creates $\boldsymbol{v}^0, \ldots, \boldsymbol{v}^{\lambda-1}$ from $B_0, \ldots, B_{\lambda-1}$.
Step 2 The user's initial input to ROT is an encrypted $p_0 + o_0$. The offset is $o_0 = 0$ when 0th bit of user's character c is 0, otherwise $o_0 = N + 1$.
Step 3 The server's initial inputs to ROT are an encrypted $\boldsymbol{v}^0$ and a random factor r_0.
Step 4 The user obtains $\hat{p}_1 = (\boldsymbol{v}^0[p_0 + o_0] + r_0)_{\mathsf{mod}\ N+1}$ from ROT.
Step 5 *for* $i = 1, \ldots, \lambda - 1$
- The user inputs encrypted $\hat{p}_i + o_i$ to ROT.
- The server inputs r_{i-1} and a new random factor r_i and $\boldsymbol{v}^i$ to compute $\hat{p}_{i+1} = (\boldsymbol{v}^i[p_i + o_i] + r_i)_{\mathsf{mod}\ N+1}$ in an encrypted form.
- The user obtains $\hat{p}_{i+1}$.

By the above protocol, only the user can safely obtains $(\mathsf{RankCF}(T_0, p_0, c) + r_{\lambda-1})_{\mathsf{mod}\ N+1}$. Note that the user can obtain $\mathsf{RankCF}(T_0, p_0, c)$ if the server sets $r_{\lambda-1} = 0$. Since the length of each sub-lookup table is $2(N + 1)$ and sWM repeats ROT for λ times, the total time complexity becomes $O(N \log |\Sigma|)$, which is up to several orders of magnitude better than the previous approach's time complexity $O(N|\Sigma|)$. Fig. 4 illustrates the design principle of $\boldsymbol{v}$ for sWM and that for PBWT-sec. We will describe the sWM algorithm in more detail in Section 3.3.

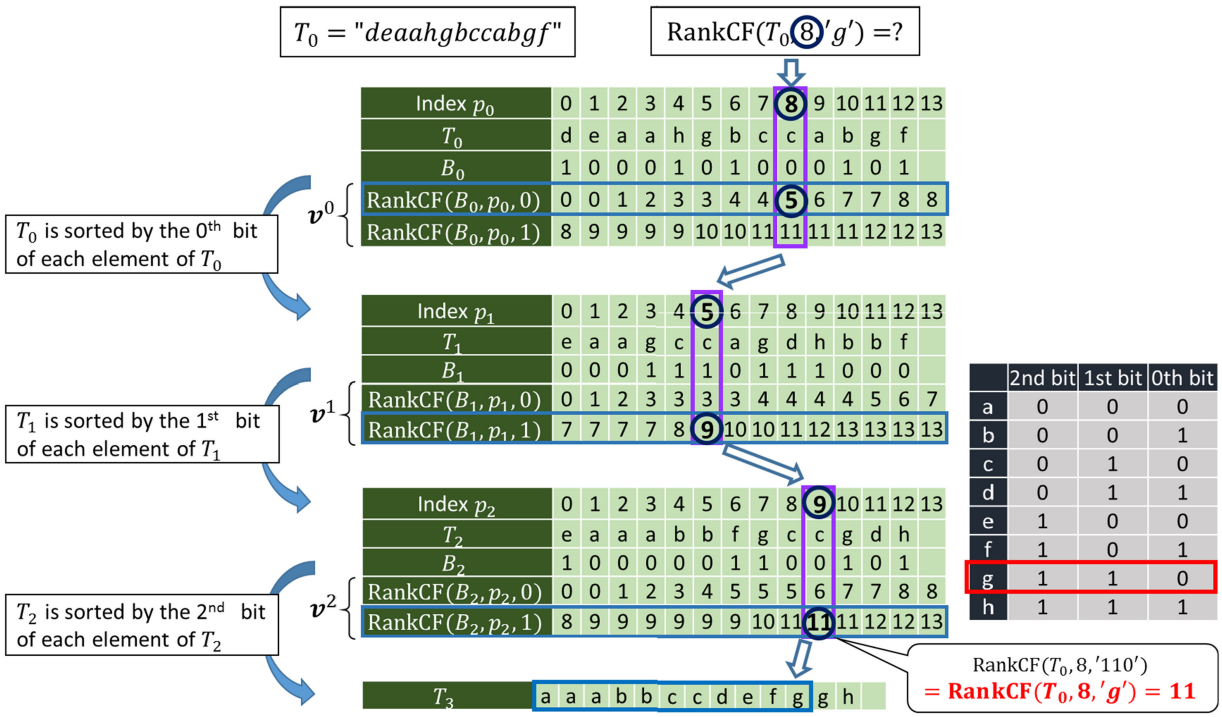

Fig. 3. Example of search on wavelet matrix. RankCF$(T_0, 8, \text{'}g\text{'})$ is obtained by RankCF$(B_0, 8, b^0(\text{'}g\text{'}))$, which returns 5; RankCF$(B_1, 5, b^1(\text{'}g\text{'}))$, which returns 9; and RankCF$(B_2, 9, b^2(\text{'}g\text{'}))$, which returns 11.

Note that we intentionally employ WM instead of the wavelet tree [22] to avoid information leakage from the server's database. If the lookup table is constructed based on the wavelet tree in a straight forward manner, we need to create 2^i sub-lookup tables for the ith significant bit of a character, and the length of each sub-lookup table becomes the number of characters in the database that corresponds to top i bits. Since the length of each sub-lookup table should be common information between the user and the server when conducting ROT, the server cannot protect such information.

3 Method

As described in Section 2, the sWM is designed on the basis of the ROT, which we implemented by using additively homomorphic encryption. In this section, we describe those cryptographic building blocks and provide more details about the sWM algorithm.

3.1 Additively Homomorphic Encryption

Additively homomorphic encryption is a kind of public-key encryption scheme that enables addition of *encrypted*

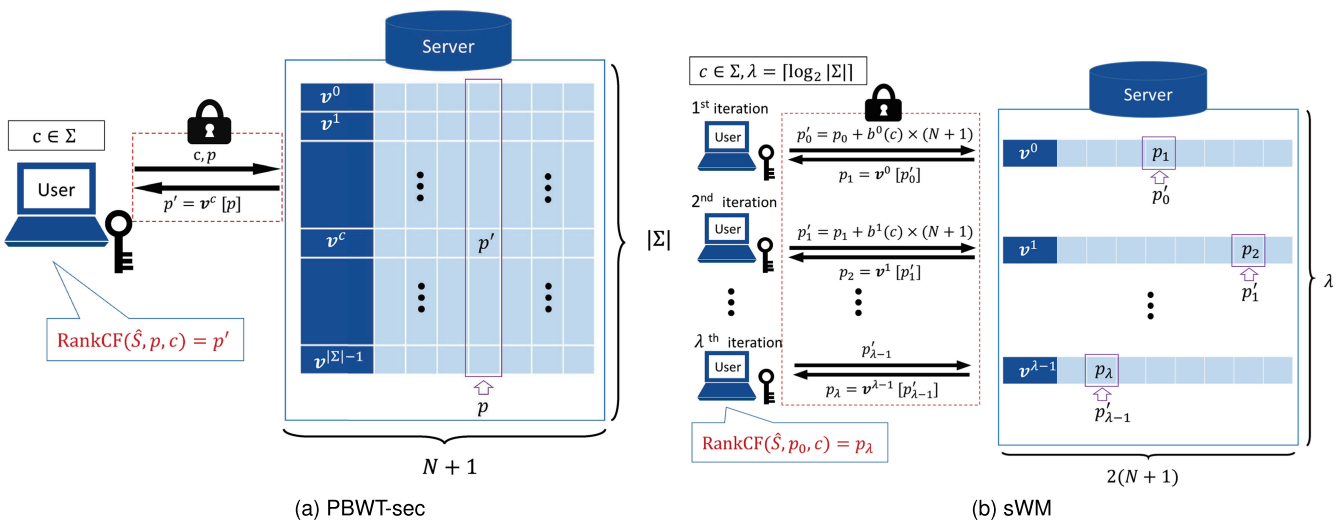

Fig. 4. Schematic view of updating each bound of interval in (a) PBWT-sec (previous method) and (b) secure wavelet matrix (proposed method). In PBWT-sec, user sends c, p in such a way that only single communication occurs. Server returns $v^c[p]$ in encrypted form by scanning lookup table of length $N|\Sigma|$. Therefore, the time complexity of this task becomes $O(N|\Sigma|)$. In sWM, $\lceil \log_2 |\Sigma| \rceil$ communications occur to obtain the same information. For each communication, server returns $p_{k+1} = v^k[p'_k]$ by scanning sub-lookup table of length $2(N+1)$. Therefore, time complexity of total communications becomes $O(N \log |\Sigma|)$.

values to be computed. A public-key encryption scheme consists of three algorithms: the key generation algorithm KeyGen generates a public key pk and a secret key sk; the encryption algorithm Enc generates a ciphertext $\mathsf{Enc}(m)$ of message m under the given pk; and the decryption algorithm Dec computes the decryption result of a ciphertext under the given sk. An additively homomorphic encryption scheme also has the following additively homomorphic functionalities:

- An operation $\mathsf{Enc}(m_1) \oplus \mathsf{Enc}(m_2)$ to generate $\mathsf{Enc}(m_1 + m_2)$ from two given ciphertexts $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$ of integer messages m_1 and m_2, without knowing m_1, m_2, or the secret key.
- An operation $e \otimes \mathsf{Enc}(m)$ to generate $\mathsf{Enc}(e \cdot m)$ from a given ciphertext $\mathsf{Enc}(m)$ and an integer e, without knowing m or the secret key (in particular, $\mathsf{Enc}(-m)$ can be computed by the operation).

We suppose that the scheme used in this study is semantically secure; that is, no information in the original message can be learned from a ciphertext [23]. A more precise description of the security property for encryption schemes is given in Section S2 in the supplementary material. Examples of other such schemes are the Paillier cryptosystem [24] and the "lifted" version of the ElGamal cryptosystem [25], where the second operation $\otimes$ can be achieved by iteration of the first operation $\oplus$.

3.2 Recursive Oblivious Transfer

The ROT protocol consists of three sub-modules.

- $\mathsf{PrepQuery}(\hat{t})$ is a user-side sub-module. It takes an index $\hat{t}$ and returns an encrypted query. (We note that the privacy for the user's secret input is protected by the semantic security of the underlying encryption scheme assumed in this paper; while the privacy for the server's secret input is protected without such cryptographic assumptions on the encryption scheme. See Section S2 in the supplementary material, for a more precise description of the security property for our protocol and a proof of the security.) In fact, the query is in the form of a vector of ciphertexts for the further process of ROT. However, we do not go into detail about the specifications of ROT and we abuse $\vec{\mathsf{Enc}}(\hat{t})$ to denote an encrypted query.
- $\mathsf{RecQuery}(\vec{\mathsf{Enc}}(\hat{t}), r')$ is a server-side sub-module. It takes a user's input $\vec{\mathsf{Enc}}(\hat{t})$ and a random factor r' that was used to compute $\hat{t} = (t + r')_{\mathsf{mod}\ N}$ in the previous round and removes r' to return $\vec{\mathsf{Enc}}(t)$.
- $\mathsf{RanOT}(\vec{\mathsf{Enc}}(t), \boldsymbol{v}, r)$ is a server-side sub-module. It is a computationally dominant part and computes an encrypted and randomized result $\mathsf{Enc}((\boldsymbol{v}[t] + r)_{\mathsf{mod}\ N})$ from $\vec{\mathsf{Enc}}(t)$, the server's lookup table $\boldsymbol{v}$ and a random factor r.

Those sub-modules are executed in the following order, and all steps are repeated until a condition predefined by the main algorithm calling ROT is satisfied.

Step 1 The user conducts $\mathsf{PrepQuery}(\hat{t})$ and sends $\vec{\mathsf{Enc}}(\hat{t})$ to the server.

Step 2 The server conducts $\mathsf{RecQuery}(\vec{\mathsf{Enc}}(\hat{t}), r')$ to obtain an encryption of a correct index $\vec{\mathsf{Enc}}(t)$.

Step 3 The server conducts $\mathsf{RanOT}(\vec{\mathsf{Enc}}(t), \boldsymbol{v}, r)$ to compute an encrypted and randomized result $\mathsf{Enc}((\boldsymbol{v}[t] + r)_{\mathsf{mod}\ N})$.

Step 4 The user obtains $\hat{t} = (\boldsymbol{v}[t] + r)_{\mathsf{mod}\ N}$ by $\mathsf{Dec}(\mathsf{Enc}(\boldsymbol{v}[t] + r)_{\mathsf{mod}\ N})$, and goes back to *Step 1*.

ROT returns randomized results as long as the server uses $r \neq 0$ in *Step 3*. For a full-text search, we use a trick for returning a proper flag to the user only when the longest prefix match is found, which is described in Section 3.4. See our previous work [15] for more details about the ROT algorithm. The similar protocol is also introduced as a primitive for searching with a finite automata [26], [27].

3.3 Secure Wavelet Matrix

Here, we describe the algorithm of sWM in detail. A pseudocode of sWM is written in Algorithm 1. The protocol starts with the initialization task, in which the server prepares sub-lookup tables $\boldsymbol{v}^0, \ldots, \boldsymbol{v}^{\lambda-1}$ (Step 1), and the user sets an initial index p_0 (Step 2). p denotes $true$ RankCF, which should be held by the user at the end of the protocol, and $\hat{p}$ is the corresponding $randomized$ RankCF, to which the server adds a random factor r. Analogously, we denote $\hat{t} = \hat{p} + o_k$ and $t = p + o_k$, where o_k is an offset for searching for $\boldsymbol{v}^k$. In the recursive search task (Step 3), the user and the server collaboratively compute $\hat{p}_k = (\boldsymbol{v}^k[t] + r)_{\mathsf{mod}\ N+1}$. On the server side, the user's input $\vec{\mathsf{Enc}}(\hat{t})$ is *not* decrypted, and the result is computed by those ciphertexts. On the other hand, the user obtains a plaintext $(\boldsymbol{v}[t] + r)_{\mathsf{mod}\ N+1}$. However, this is randomized by r, and the user cannot see $\boldsymbol{v}[t]$ that includes the server's private information. The random factor r is added by the ROT task in the third item of Step 3b. Therefore, the protocol is secure for both the user and the server sides. Though Algorithm 1 returns randomized results for all communication rounds, the user can obtain a true RankCF at the end of the protocol if the server uses $r = 0$ only at the final round of the protocol. In Step 4, the user sends an additional query to let the server hold an encrypted result $\mathsf{Enc}(p_\lambda)$. This is an unnecessary task for computing $\mathsf{RankCF}(T_0, p_0, c)$, however, it is often convenient to hold an encrypted result $\mathsf{Enc}(p_\lambda)$ by the server when the sWM is used as a building block of other search algorithm. We will introduce such a search algorithm in Section 3.4, and will show how the encrypted result held by the server is used in the algorithm.

3.4 Secure FM-Index

In this study, we apply the sWM to the problem of full-text search by using the FM-Index algorithm, in which BWT of an original text is created for indexing a database string [20]. As we mentioned in Section 2.2, a substring match is reported as an interval $[f, g)$ on the BWT, and there is a recursion relationship described in Equation (1). (For the case of FM-Index, $\hat{S}$ in Equation (1) is BWT.) The main idea of our algorithm is to use sWM to compute RankCF. It repeats sWM until the longest match is found (e.g., $g = f$). Algorithm 2 is the detailed algorithm of the secure FM-Index (sFMI). Note that some of the steps in Algorithm 1 are slightly modified to fit it to the FM-Index algorithm. The key part of the sFMI is in Step 3b, in which the server computes an encrypted flag by using the

function isLongest. This flag becomes an encrypted 0 only when the match is longest, and becomes an encryption of a random value otherwise. Therefore, only the user knows whether or not each substring match is the longest by checking the flag in Step 3c. The algorithm is scalable for different search options if the function isLongest is replaced by another function holding a different end condition. For example, it enables searching for a longest substring match whose occurrence is at least ϵ, just by computing encrypted flags $x_i \leftarrow (\mathsf{Enc}(g) \oplus \mathsf{Enc}(-f) \oplus \mathsf{Enc}(\epsilon - i)) \otimes r$ for $i = 0, \ldots, \epsilon - 1$ and checking whether or not it includes an encrypted 0. (The algorithm is detailed in Section S1 in the supplementary material) Since the algorithm repeats sWM ℓ times for the query of length ℓ, total time complexity becomes $O(\ell N \log|\Sigma|)$. We will discuss the complexity in more detail in Section 3.5.

Algorithm 1. Detailed Description of Secure Wavelet Matrix

- Public input: Problem size N; alphabet Σ; public key pk
- Private input of user: An index p_0, a query character $c \in \Sigma$, private key sk
- Private input of server: A database text T_0

1) (*Server initialization*)
   a) Create auxiliary bit-arrays $B_0, \ldots, B_{\lambda-1}$ from T_0
      for $(k = 0, \ldots, \lambda - 1)$ **do** $\triangleright$ $b^k(c)$: kth bit of c
      - $B_k = b^k(T_k[0]), \ldots, b^k(T_k[N-1])$
      - Stably sort characters of T_k by kth significant bit

      end for
   b) Create sub-lookup tables $\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots, \boldsymbol{v}^{\lambda-1}$
      $\boldsymbol{v}^k[i + x \times (N+1)] = \mathsf{RankCF}(B_k, i, x)$
      $(0 \le k < \lambda,\ 0 \le i \le N,\ x \in \{0, 1\})$
2) (*User initialization*)
   Set initial index: $\hat{p}_0 \leftarrow p_0$
3) (*Recursive operation*) Set initial bit position: $k = 0$
   while $(k < \lambda)$ **do** $\triangleright$ Computing from 0th to $(\lambda-1)$th bit
   a) (*Query entry*) The user performs the following steps:
      - Set an offset o_k:
        if $(b^k(c) = 0)$ $o_k \leftarrow 0$
        else $o_k \leftarrow N + 1$
      - Calculate next index:
        $\hat{t} = \hat{p}_k + o_k$ $\triangleright$ $\hat{p}_k = p_k + r$ $(k \neq 0)$
      - Prepare next query:
        $\vec{\mathsf{Enc}}(\hat{t}) \leftarrow \mathsf{PrepQuery}(\hat{t})$
      - Send $\vec{\mathsf{Enc}}(\hat{t})$ to the server.
   b) (*Operate*) The server performs the following steps:
      - Generate a random value r
      - Set $r' \leftarrow 0$ iff. $k = 0$
      - Compute the next index:
        $\mathsf{Enc}(t) \leftarrow \mathsf{RecQuery}(\vec{\mathsf{Enc}}(\hat{t}), r')$ $\triangleright$ $t = p_k + o_k$
        $\mathsf{Enc}((\boldsymbol{v}^k[t] + r)_{\mathsf{mod}\ N+1})$
        $\leftarrow \mathsf{RanOT}(\vec{\mathsf{Enc}}(t), \boldsymbol{v}^k, r)$
      - Store a random value $r' \leftarrow r$
      - Send $\mathsf{Enc}((\boldsymbol{v}^k[t] + r)_{\mathsf{mod}\ N+1})$ to the user
   c) (*Receive randomized index*) The user obtains:
      $\hat{p}_{k+1} \leftarrow \mathsf{Dec}(\mathsf{Enc}(\boldsymbol{v}^k[t] + r)_{\mathsf{mod}\ N+1})$
      $\triangleright$ $\hat{p}_{k+1} = (\boldsymbol{v}^k[p_k + o_k] + r)_{\mathsf{mod}\ N+1}$
   $k \leftarrow k + 1$
   end while
4) The user sends the last query:
   $\vec{\mathsf{Enc}}(\hat{p}_\lambda) \leftarrow \mathsf{PrepQuery}(\hat{p}_\lambda)$
   Send $\vec{\mathsf{Enc}}(\hat{p}_\lambda)$ to the server.
5) The server obtains encrypted p_λ:
   $\mathsf{Enc}(p_\lambda) \leftarrow \mathsf{RecQuery}(\vec{\mathsf{Enc}}(\hat{p}_\lambda), r)$

At the end of the protocol:

- The user holds: $\hat{p}_\lambda = (\mathsf{RankCF}(T_0, p_0, c) + r)_{\mathsf{mod}\ N+1}$
- The server holds: $\mathsf{Enc}(p_\lambda) = \mathsf{Enc}(\mathsf{RankCF}(T_0, p_0, c))$ and r

Our method also enables to report a count of matches at the end of the protocol (or even in each iteration) with small modification. This apparently gives additional information about the database to the user, however, it may be useful for some applications if revealing the additional information is acceptable.

The original FM-Index algorithm provides a backward search where a search direction starts from the tail to the head of a query. This can be easily converted into a forward search by querying in the reverse direction. For simplicity, we described the algorithm in such a way that the query is searched in a forward direction.

3.5 Complexity

In this section, we compare the theoretical complexity of sFMI and PBWT-sec. The computational obstacle of both algorithms is a ROT task. Given a lookup-table of length N, the time complexity of the ROT is $O(N)$ and the communication complexity is $O(\sqrt{N})$. It might not be intuitive that the communication complexity is not linear to the size of the lookup-table. However, there is an efficient algorithm for reducing query length to $\sqrt{N}$. The key idea is to describe a lookup table as a matrix $M^{\sqrt{N} \times \sqrt{N}}$, and user tries to receive the element of d_1th column of d_2th row. To this end, the user creates an encrypted vector of length $\sqrt{N}$ in which only d_1th element is $\mathsf{Enc}(1)$ and others are $\mathsf{Enc}(0)$, and the server computes inner products of the query and rows of M to send back $\sqrt{N}$ ciphertexts. We do not describe details, however, there is an efficient method to return only d_2th element and hide rest of all, and to enable to hide intermediate results during the recursive search. See our previous work [15] for more detailed information. The similar approach is also taken for standard OT [28], and we will describe more detailed information in Section S3 in the supplementary material.

The sWM used in sFMI repeats ROT for all sub-lookup tables, and thus the time and communication complexities become $O(N \log|\Sigma|)$ and $O(\sqrt{N} \log|\Sigma|)$, respectively. On the other hand, PBWT-sec performs a single ROT task for a lookup-table of length $N|\Sigma|$. Therefore, its time complexity is $O(N|\Sigma|)$ and the communication complexity is $O(\sqrt{N|\Sigma|})$. Apparently, time complexity is up to several orders of magnitude better for sFMI, while the communication complexity is comparable as long as the Σ is not very large. Since the communication complexity is already sub-linear to the size

TABLE 1
Summary of Time and Communication Complexities of sFMI (Proposed Method), PBWT-Sec (Previous Method), and Naïve Approach

	Time	Total communication size	Communication round
sFMI (User)	$O(\ell\sqrt{N}\log\lvert\Sigma\rvert)$	$O(\ell\sqrt{N}\log\lvert\Sigma\rvert)$	$\ell\lceil\log_2\lvert\Sigma\rvert\rceil$
sFMI (Server)	$O(\ell N\log\lvert\Sigma\rvert)$		
PBWT-sec (User)	$O\left(\ell\sqrt{N\lvert\Sigma\rvert}\right)$	$O\left(\ell\sqrt{N\lvert\Sigma\rvert}\right)$	ℓ
PBWT-sec (Server)	$O(\ell N\lvert\Sigma\rvert)$		
Naïve (User)	$O\left(\sqrt{\lvert\Sigma\rvert^\ell}\right)$	$O\left(\sqrt{\lvert\Sigma\rvert^\ell}\right)$	1
Naïve (Server)	$O\left(\lvert\Sigma\rvert^\ell\right)$		

Note that the communication complexities of both user-to-server and server-to-user is the same.

of N, the improvement in time complexity is much more important for practical problems. Regarding round complexity, sWM requires $\lceil\log_2|\Sigma|\rceil$ dependent communications while PBWT-sec requires a single communication. Considering the improvement in time complexity, we expect that sWM still has a large advantage in total run time when N is not too small and network latency is not too high. We will discuss this point in more details in Section 4.

Table 1 summarizes time and communication complexities of both sFMI and PBWT-sec. Note that both time and communication complexities are multiplied by the query length ℓ. In addition to those two algorithms using a clever data structure, for comparison, we also describe complexity of a Naïve method, in which all occurrences of substrings appearing in a database text are stored in a lookup table and the user uses ROT to search for a substring match.

Algorithm 2. Detailed Description of Secure FM-Index

protocol sWM($\hat{p}$, pk, sk, c, V, φ) ▷ $\hat{p}$, pk, sk and c are the user's input, V and φ are the server's input
Perform Algorithm 1 with the following modifications
- Use pk and sk for a public key and a private key
- Omit Step1 and use input V as sub-lookup tables
- Initialize $\hat{p}_0 \leftarrow \hat{p}$ in Step2
- Initialize $r' \leftarrow \varphi$ in the second item of Step 3b

return (only to user) $(\mathsf{RankCF}(\hat{T}, p, c) + r)_{\mathsf{mod}\ N+1}$
▷ $\hat{T}$ is a text from which V is created, $\hat{p} = (p+\varphi)_{\mathsf{mod}\ N+1}$.
return (only to server) $\mathsf{Enc}(\mathsf{RankCF}(\hat{T}, p, c))$ and r
▷ r is the last random value generated in Algorithm 1
end protocol

function isLongest($\mathsf{Enc}(f), \mathsf{Enc}(g)$)
Generate random value r
$x \leftarrow (\mathsf{Enc}(g) \oplus \mathsf{Enc}(-f)) \otimes r$
return x ▷ $x = \mathsf{Enc}(0)$ iff. match is the longest
end function

- Public input: Problem size N; alphabet Σ
- Private input of user: A query sequence $\boldsymbol{q}$ of length ℓ
- Private input of server: A database text T

0. (*Key setup of cryptosystem*) The user generates key pair (pk, sk) by using key generation algorithm KeyGen for additive-homomorphic cryptosystem and sends public key pk to server
1) (*Server initialization*)
   - The server creates BWT of T and stores it as $\hat{T}$
   - The server creates a set of sub-lookup tables for $\hat{T}$:
     $V = \{\boldsymbol{v}^0, \boldsymbol{v}^1, \ldots, \boldsymbol{v}^{\lambda-1}\}$, by using the same process described in Step 1 of Algorithm 1
2) (*User initialization*) Set initial interval $[\hat{f}_0 = 0,\ \hat{g}_0 = N)$
3) (*Recursive search*)
   Initialize an index: $i \leftarrow 0$ and random factors: $r_f \leftarrow 0$, $r_g \leftarrow 0$
   while $(i < \ell)$ **do**
   a) (*Update interval*)
      - The user and server execute:
        $\hat{f}_{i+1}, \mathsf{Enc}(f_{i+1}), r_f \leftarrow \mathsf{sWM}(\hat{f}_i, \mathsf{pk}, \mathsf{sk}, q[i], V, r_f)$
        $\hat{g}_{i+1}, \mathsf{Enc}(g_{i+1}), r_g \leftarrow \mathsf{sWM}(\hat{g}_i, \mathsf{pk}, \mathsf{sk}, q[i], V, r_g)$
        to obtain:
        $\hat{f}_{i+1}$, $\hat{g}_{i+1}$ for the user
        $\mathsf{Enc}(f_{i+1})$, $\mathsf{Enc}(g_{i+1})$, r_f, r_g for the server
   b) (*Operate*) The server performs the following steps:
      - Compute an encrypted flag showing if the match is longest
        $x \leftarrow \mathsf{isLongest}(\mathsf{Enc}(f_{i+1}), \mathsf{Enc}(g_{i+1}))$
      - Send x to the user
   c) (*Decryption of the encrypted flag*) The user performs the following steps:
      $d \leftarrow \mathsf{Dec}(x)$
      if $d = 0$
        if $i = 0$ Report that no prefix matches T
        else Report that $q[0, \ldots, i-1]$ is the longest match
        Send decoy queries to server until $i = \ell - 1$
      $i \leftarrow i + 1$
   end while
   The user reports that $q[0, \ldots, \ell-1]$ is the longest match, if $d \neq 0$ for $i = 0, \ldots, \ell-1$.

3.6 Security Notion

In our method, all the user's inputs are sent to the server by using the recursive oblivious transfer protocol (ROT). Therefore, the security of the proposed method depends on that of ROT. ROT assumes the security model called Semi-honest model where both parties follow the protocol, but an adversarial one is allowed to infer additional information about the other party's secret input from the legally obtained information. Due to the semantic security of the encryption scheme used in ROT (i.e., additively homomorphic encryption described in Section 3.1), the server cannot infer any information about the user's query during the protocol. Also, the user cannot infer any information about the database except for the result. In fact, the ROT is easily

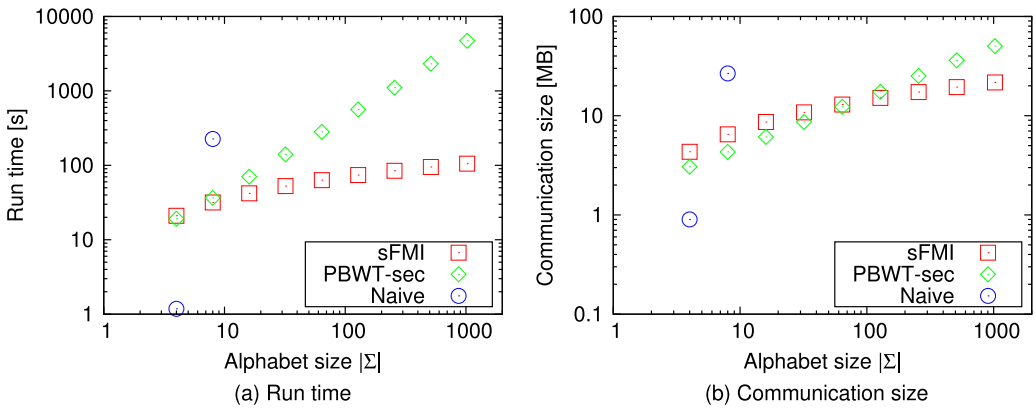

Fig. 5. (a) Run time and (b) Communication size of sFMI, PBWT-sec, and Naive method for random string of 10,000 with alphabet size $|\Sigma| = 4, 8, 16, \ldots, 1024$, when query length is 10. Both user and server used a single thread and the program was run on the network with almost no latency. Run time includes communication overhead.

modified to stronger security model called Malicious model where an adversarial party is allowed to input maliciously chosen invalid values in order to illegally obtain additional information about the secret. In the ROT designed for the malicious model, a cryptographic method called Non-Interactive Zero Knowledge Proofs is used to detect an illegal query. See the previous work [8] for more detailed information. The entire protocol is also modified to Malicious model by replacing ROT by the ROT using the Zero Knowledge Proofs with a slight additional computing cost.

4 Experiments

To evaluate the efficiency of the proposed method (sFMI), we performed experiments on both a simulated dataset and two different real datasets. For the simulated dataset, we simply generated random strings each of which has a length of 10,000 with $|\Sigma| = 4, 8, 16, \ldots, 1024$. For the real datasets, we used all the protein sequences included in Ribosomal_S4Pg Family of Pfam [29] and all clinical study titles (879 titles, in Japanese, 53,560 characters in total excluding punctuation) stored in JAPIC Clinical Trials Information [30]. For each real dataset, all the sequences are concatenated into a long single sequence with a delimiter symbol. The Japanese text is usually written in a combination of a Japanese alphabet (Hiragana, $|\Sigma| = 83$ including sound marks) and Chinese ideographs ($|\Sigma| > 10{,}000$). However, Chinese ideographs can be spelled out into Hiragana, though this is generally very unnatural. Therefore, we prepared two datasets for the clinical text: one is the converted text written in Hiragana (Clinical DB1), and the other is the original text (Clinical DB2). Those texts also include words written in Arabic numerals ($|\Sigma| = 10$) and Roman (case-sensitive) and Greek (case-insensitive) alphabets ($|\Sigma| = 26$, 50 respectively), because numbers and technical terms including Roman/Greek letters are usually written in their original form. Clinical DB1 consists of the alphabet of $|\Sigma| = 170$, and Clinical DB2 consists of the alphabet of $|\Sigma| = 21{,}207$ including the delimiter symbol. (Since one Chinese ideograph is usually spelled out by more than one Hiragana character, Clinical DB1 and DB2 have exactly the same meaning but different lengths.) See Section S4 in the supplementary material for more details about the character sets.

We implemented the proposed algorithm in C++ based on an open source C++ library of *elliptic curve ElGamal encryption* [31]. We used the same implementation of ROT for all the three methods, which is provided by another C++ library [21]. For the security parameters, we used a standard configuration called *secp192k1* (SECG curve over a 192-bit prime field) in accordance with the recommendation by The Standards for Efficient Cryptography Group. We used a laptop computer equipped with Corei7 3.00 GHz (2 physical cores) for the user, and a standard desktop PC equipped with Xeon 3.40 GHz (12 physical cores) for the server for all of the experiments.

4.1 Results on the Simulated Dataset

For the simulated dataset, the program was run with a single thread both for the user and the server. Fig. 5 shows total run time and communication size for the simulated dataset when query length is 10. To show the performance on an ideal situation, the program was run on the same network with almost no latency. As shown in panel (a), an observed run time (including data-transfer time) of sFMI is up to orders of magnitude faster than PBWT-sec, which is concordant with the theoretical complexity. Communication size of the proposed method is also better than that of PBWT-sec when $|\Sigma|$ is sufficiently large.

4.2 Results on the Real Datasets

Influence of Network Latency. To investigate the performance of sFMI on a practical situation, we also conducted an experiment on the networks with different latencies ranging from 10 ms to 300 ms, taking into account that the average latency in domestic network in Japan is below 10 ms and that in Trans-Pacific network is below 120 ms [32]. The latency was controlled by tc command of Linux OS. We used the three real datasets as summarized in Table 2 and the program was run with a single thread both for the user and the server. The query length was 10 for all of the

TABLE 2
Summary of the Three Real Datasets: Protein Sequence Database (Protein DB), Clinical Study Results Database Described in Reduced Set of Hiragana Alphabet (Clinical DB1) and That in Original Form (Clinical DB2)

	Protein DB	Clinical DB1	Clinical DB2
DB Length	9,826	77,712	53,560
Alphabet Size	21	170	21,207

TABLE 3
Run Time of sFMI (Proposed Method) and PBWT-Sec (Previous Method) for Searching on the Three Real Datasets on the Networks with Different Latencies

Network latency (ms)		10	30	45	90	150	300
Protein DB	sFMI (s)	52.51	57.92	61.85	73.09	88.42	126.79
	PBWT-sec (s)	92.31	93.79	94.90	98.28	102.72	113.84
Clinical DB1	sFMI (s)	563.49	574.59	583.54	609.43	641.99	727.07
	PBWT-sec (s)	6170.02	6162.70	6167.92	6184.81	6215.71	6263.55
Clinical DB2	sFMI (s)	747.23	766.38	781.53	829.89	892.37	1051.24
	PBWT-sec (s)	-	-	-	-	-	-

experiments. As shown in Table 3, sFMI was faster for all of the configurations except for the case when searching on a text with relatively small alphabet size (the protein sequence database) with the latency of 300 ms. As discussed in Section 3.5, the time complexity of sFMI is $O(|\Sigma|/\log|\Sigma|)$ times better than that of PBWT-sec while round complexity is $\lceil \log_2|\Sigma| \rceil$ times greater. Since the computation required for each round is heavy, we consider that sFMI algorithm, which improves CPU time, outweighs previous algorithms even it has a slightly larger round complexity.

Performance Using Parallelization. We also investigated the performance of sFMI when the program was parallelized. As discussed in Section 3.5, ROT task on the server side is a computational bottleneck both for sFMI and PBWT-sec, in which $\mathsf{Enc}(v[i])$ for $i = 0, \ldots, N$ is computed and only the target element $\mathsf{Enc}(v[t])$ is safely selected. Since each element $\mathsf{Enc}(v[i])$ can be calculated independently, the server can compute them in parallel. e.g., If the server has m CPU cores, the ideal CPU time for this task is reduced to $1/m$. For the user side, encryption of each element of a query vector is parallelized. In our implementation, we used openMP to parallelize those tasks.

Table 4 shows run time when the server used 8 or 16 threads and the user used 2 or 4 threads. The query length was 10 for all of the experiments and the program was run on the network with 10 ms latency. For the search on Clinical DB1 described by the Hiragana alphabet, run time of the proposed method was 7 to 9 times faster than that of the previous method. For the search on Clinical DB2 with 16 CPU cores for the server, run time of the proposed method was only 103.34 although $|\Sigma|$ is huge, which is more than 470 times faster than the previous approach. The results show that our method is practical even for real datasets.

TABLE 4
Run Time of sFMI (Proposed Method) and PBWT-Sec (Previous Method) for Searching on Protein Sequence Database (Protein DB) and Clinical Study Results Database Described in Reduced Set of Hiragana Alphabet (Clinical DB1) and That in Original Form (Clinical DB2) with the Latency of 10 ms

	Protein DB		Clinical DB1		Clinical DB2	
Server run time (s):						
Num. of threads	8	16	8	16	8	16
sFMI	7.47	6.77	76.65	59.71	101.33	80.11
PBWT-sec	12.39	9.74	873.58	553.13	77589.3	48876.9
User run time (s):						
Num. of threads	2	4	2	4	2	4
sFMI	2.37	1.89	8.14	8.22	12.80	12.89
PBWT-sec	1.82	1.96	10.75	9.90	71.43	71.89
Total time (s):						
sFMI	12.80	10.98	90.54	73.93	125.75	103.34
PBWT-sec	15.07	12.53	886.90	565.69	77661.7	48950.3

5 Conclusion

We have developed an efficient algorithm for a secure string search on the basis of a novel technique combining wavelet matrix and homomorphic encryption. It can search any type of string while still protecting privacy as strongly as the previous approach. We implemented the proposed method and tested it on both a simulated dataset and real datasets. The results show that the proposed method is up to orders of magnitude more efficient than the previous approach in terms of alphabet size and that its computational cost is acceptable for practical use. As the proposed method potentially scales to various types of data such as two-dimensional data and tree data, it is expected to be used for an even wider range of life science data and contribute to secure data sharing.

A supplementary material for this work is available from: https://github.com/cBioLab/sWM_doc

Acknowledgments

This study was supported by the Japan-Finland Cooperative Scientific Research Program of AMED (to K.S.) and JST PRESTO grant number JPMJPR14E8 (to K.N.). A part of this work is also supported by JST CREST grant number JPMJCR1688. Hiroki Sudo and Masanobu Jimbo contributed equally to this work.

References

[1] M. Akgün, A. O. Bayrak, B. Ozer, and M. S. Sagiroglu, "Privacy preserving processing of genomic data: A survey," *J. Biomed. Informat.*, vol. 56, pp. 103–111, 2015.

[2] F. Bonchi and E. Ferrari, *Privacy-Aware Knowledge Discovery: Novel Applications and New Techniques*, 1st ed. Boca Raton, FL, USA: CRC Press, 2010.

[3] C. C. Aggarwal and P. S. Yu, *Privacy-Preserving Data Mining: Models and Algorithms*, 1st ed. Berlin, Germany: Springer, 2008.

[4] M. J. Freedman, Y. Ishai, B. Pinkas, and O. Reingold, "Keyword search and oblivious pseudorandom functions," in *Proc. 2nd Annu. Theory Cryptography Conf.*, 2005, pp. 303–324.

[5] F. Bruekers, S. Katzenbeisser, K. Kursawe, and P. Tuyls, "Privacy-preserving matching of DNA profiles," *IACR Cryptology ePrint Archive*, 2008. [Online]. Available: http://eprint.iacr.org/2008/203

[6] P. Baldi, R. Baronio, E. D. Cristofaro, P. Gasti, and G. Tsudik, "Countering GATTACA: Efficient and secure testing of fully-sequenced human genomes," in *Proc. 18th ACM Conf. Comput. Commun. Secur.*, 2011, pp. 691–702.

[7] E. D. Cristofaro, S. Faber, and G. Tsudik, "Secure genomic testing with size- and position-hiding private substring matching," in *Proc. 12th Annu. ACM Workshop Privacy Electron. Soc.*, 2013, pp. 107–118.

[8] K. Shimizu, K. Nuida, H. Arai, S. Mitsunari, N. Attrapadung, M. Hamada, K. Tsuda, T. Hirokawa, J. Sakuma, G. Hanaoka, et al., "Privacy-preserving search for chemical compound databases," *BMC Bioinf.*, vol. 16, no. Suppl 18, 2015, Art. no. S6.

[9] J. Katz and L. Malka, "Secure text processing with applications to private DNA matching," in *Proc. 17th ACM Conf. Comput. Commun. Secur.*, 2010, pp. 485–492.

[10] A. C.-C. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Found. Comput. Sci.*, 1986, pp. 162–167.

[11] C. Hazay and T. Toft, "Computationally secure pattern matching in the presence of malicious adversaries," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Secur.*, vol. 10, pp. 195–212, 2010.

[12] D. Vergnaud, "Efficient and secure generalized pattern matching via fast fourier transform," in *Proc. 4th Int. Conf. Cryptology Africa*, 2011, pp. 41–58.

[13] J. Baron, K. E. Defrawy, K. Minkovich, R. Ostrovsky, and E. Tressler, "5PM: Secure pattern matching," *J. Comput. Secur.*, vol. 21, no. 5, pp. 601–625, 2013.

[14] M. Yasuda, T. Shimoyama, J. Kogure, K. Yokoyama, and T. Koshiba, "Privacy-preserving wildcards pattern matching using symmetric somewhat homomorphic encryption," in *Proc. Australasian Conf. Inf. Secur. Privacy*, 2014, pp. 338–353.

[15] K. Shimizu, K. Nuida, and G. Rätsch, "Efficient privacy-preserving string search and an application in genomics," *Bioinf.*, vol. 32, no. 11, pp. 1652–1661, 2016.

[16] F. Claude and G. Navarro, "The wavelet matrix," in *Proc. 19th Int. Symp. String Process. Inf. Retrieval*, 2012, pp. 167–179.

[17] V. Mäkinen, D. Belazzougui, F. Cunial, and A. I. Tomescu, *Genome-Scale Algorithm Design*. Cambridge, U.K.: Cambridge Univ. Press, 2015.

[18] G. Navarro, *Compact Data Structures: A Practical Approach*. Cambridge, U.K.: Cambridge Univ. Press, 2016.

[19] J. Katz and Y. Lindell, *Introduction to Modern Cryptography*, 2nd ed. Boca Raton, FL, USA: CRC Press, 2014.

[20] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proc. 41st Annu. Symp. Found. Comput. Sci.*, 2000, pp. 390–398.

[21] K. Shimizu, "C++ Library implementing the recursive oblivious transfer [15]," 2016. [Online]. Available: https://github.com/iskana/PBWT-sec, Accessed: Sep. 1, 2016.

[22] R. Grossi, A. Gupta, and J. S. Vitter, "High-order entropy-compressed text indexes," in *Proc. 14th Annu. ACM-SIAM Symp. Discrete Algorithms*, 2003, pp. 841–850.

[23] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.

[24] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. 17th Int. Conf. Theory Appl. Cryptographic Techn.*, 1999, pp. 223–238.

[25] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE Trans. Inf. Theory*, vol. IT-31, no. 4, pp. 469–472, Jul. 1985.

[26] J. R. Troncoso-Pastoriza, S. Katzenbeisser, and M. U. Celik, "Privacy preserving error resilient DNA searching through oblivious automata," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2007, pp. 519–528.

[27] M. Blanton and M. Aliasgari, "Secure outsourcing of DNA searching via finite automata," in *Proc. Data Appl. Secur. Privacy XXIV*, 2010, pp. 49–64.

[28] B. Zhang, H. Lipmaa, C. Wang, and K. Ren, "Practical fully simulatable oblivious transfer with sublinear communication," in *Proc. Int. Conf. Financial Cryptography Data Secur.*, 2013, pp. 78–95.

[29] R. D. Finn, P. Coggill, R. Y. Eberhardt, S. R. Eddy, J. Mistry, A. L. Mitchell, S. C. Potter, M. Punta, M. Qureshi, A. Sangrador-Vegas, G. A. Salazar, J. G. Tate, and A. Bateman, "The Pfam protein families database: Towards a more sustainable future," *Nucleic Acids Res.*, vol. 44, no. Database-Issue, pp. 279–285, 2016.

[30] JAPIC, "Japan Pharmaceutical Information Center (JAPIC) Clinical Trials Information," 2008. [Online]. Available: http://www.clinicaltrials.jp/user/ctrSearch.jsp, Accessed: Sep. 15, 2016.

[31] S. Mitsunari, "C++ Library implementing elliptic curve ElGamal crypto system [25]," 2016. [Online]. Available: https://github.com/herumi/mcl, Accessed: Sep. 1, 2016.

[32] Verizon, "IP Latency Statistics," 2017. [Online]. Available: http://www.verizonenterprise.com/about/network/latency/, Accessed: Apr. 10, 2017.

Hiroki Sudo received the BSc degree in computer science from Waseda University, in 2017. Currently, he is working toward the MSc degree at Waseda University. He is also a research assistant with the AIST-Waseda University Computational Bio Big-Data Open Innovation Laboratory (CBBD-OIL). His research interest includes privacy-preserving data mining and machine learning.

Masanobu Jimbo received the bachelor's degree in computer science from Waseda University, in 2017. He is working toward the master's degree in the Department of Computer Science and Communications Engineering of the Faculty of Science and Engineering, Waseda University. He is also a research assistant with the AIST-Waseda University Computational Bio Big-Data Open Innovation Laboratory (CBBD-OIL). His research interest includes privacy-preserving data mining and machine learning.

Koji Nuida received the PhD degree in mathematical sciences from The University of Tokyo, in 2006. Currently, he is a senior researcher with the Information Technology Research Institute (ITRI), National Institute of Advanced Industrial Science and Technology (AIST), in Japan, and is also supported as PRESTO Researcher by the Japan Science and Technology Agency (JST). His research interest mainly includes the theory and applications of public key cryptography as well as fundamental mathematics.

Kana Shimizu received the PhD degree in computer science from Waseda University, in 2006. Currently, she is an associate professor with the Department of Computer Science and Engineering of the Faculty of Science and Engineering, Waseda University. Her research interest mainly centers on algorithms for biological sequence analysis. Her recent interest also includes privacy-preserving data mining for biological/biomedical data analysis.