Two-Cloud Private Read Alignment to a Public Reference Genome

Sindhuja Madabushi
sindhujamadabushi19@gmail.com
University of Wisconsin-Madison
USA

Parameswaran Ramanathan
parmesh.ramanathan@wisc.edu
University of Wisconsin-Madison
USA

ABSTRACT

The human genome is the ultimate identifier of an individual even though most of it is identical across human beings. Biological differences between two individuals are encoded in a set of base pair variations called Single Nucleotide Polymorphisms (SNPs), which may be indicative of an individual's personal information such as skin color and susceptibility to diseases. The large-scale nature of human genome necessitates outsourcing of genomic computations to public clouds. However, this raises some serious privacy concerns. The fact that the human reference template is public poses additional challenges. In this paper, we propose a two-cloud private read alignment algorithm using the Burrows-Wheeler Transform and the FM-Index. Our algorithm runs in the same order of complexity as the core FM-Index alignment algorithm without privacy. Our proposed scheme is able to achieve accuracy comparable to modern alignment algorithms such as Bowtie with complete privacy.

KEYWORDS

Read Alignment, Privacy, FM-Index, Burrows-Wheeler Transform, Human Genome

1 INTRODUCTION

The field of genomics has witnessed several major advances in the last century. An important milestone was the design of the Sanger sequencing technology in the 1970s [66], which was the primary sequencing technology for the next four decades. The human genome project [25] and several applications in molecular medicine, energy, and agriculture have, among others, led to the emergence of next generation high-throughput sequencing technologies. Rapid advances in genome sequencing have led to the direct-to-consumer genetic testing market being valued at over 1 billion USD. Genetic tests have also become much more affordable, with the price of sequencing an entire human genome reducing from over 500 million USD in 2003 to nearly 1500 USD[1] in 2018 [14]. Today, genetic tests can cost anywhere between 100 USD and 2000 USD and can be used to diagnose, predict susceptibility to, and guide treatment of diseases for individuals, and identify gene changes that may be passed to children.

[1]https://www.genome.gov/about-genomics/fact-sheets/Sequencing-Human-Genome-cost

Proceedings on Privacy Enhancing Technologies 2023(2), 449–463
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0062

There are several computational techniques from bioinformatics [63] that are capable of making inferences from next generation sequencing genomics data. At the core of these computations lies full genome sequence alignment, which involves mapping a set of genome sequences called reads to a publicly known reference genome template. Human genome alignment typically involves aligning tens of millions of reads of 20-1500 base pairs (bps) to a reference human genome template that is approximately 3 billion bps long. The challenges in handling such large human genome data necessitates the use of modern computing paradigms, such as cloud computing, for their storage and computation. This has led to the advent of several "Genomics as a Service" platforms, such as Microsoft Genomics platform on Azure Clouds and the Seven Bridges Genomics platform on Amazon Web Services [62].

The human genome is the ultimate identifier of an individual even though 99.5% of the genome is the same for all human beings. This apparent paradox can be explained via key variations in the genome known as Single Nucleotide Polymorphisms (SNPs). Human DNA is a sequence comprising four nucleotide bases: Adenine (A), Cytosine (C), Guanine (G) and Thymine (T). It encodes all of the genetic information pertaining to an individual. SNPs are certain key differences in the base pairs in specific regions of the genome that uniquely identify a particular human being. These variations act as biological markers linking them to numerous health conditions or unique phenotypic traits. In particular, SNPs are responsible for biological differences between individuals and can be indicative of features such as the color of hair/eyes, height, and even an increase in susceptibility to terminal diseases [50, 51, 59]. Consequently, genome alignment algorithms on cloud platforms need to ensure that an individual's SNP information is kept private.

Dealing with human genome data using cloud computing raises ethical, legal, and social concerns [56, 68]. As discussed above, knowledge of whether an individual's genome contains a SNP at a particular location can reveal sensitive information about that individual. The privacy concern therefore boils down to ensuring that an adversary is unable to gain any information about an individual's unique set of SNPs. Ad hoc approaches, such as anonymization of the reads, may be insufficient to ensure privacy because SNPs are still exposed to the sequencing server and an adversary may be able to uniquely identify an individual with some additional information [26]. In some other situations, details of the SNPs are part of the intellectual property of the researcher, which may be violated when sequencing is done by a cloud-based computing entity. Thus, given that a sequencing task routinely evaluates the alignment for millions of reads, designing scalable privacy preserving techniques for alignment of genomic data is an important open problem in computational biology [9, 28]. This is a challenge because we wish to protect an individual's personal identifiable information despite the fact that the reference human genome template, which is nearly identical to the individual's genome, is public knowledge. In this work, we design a two-cloud solution for private human genome alignment that does not reveal SNP information in an individual's genome.

1.1 Existing Alignment Algorithms

Sequencing technologies produce fragments of DNA, RNA, or proteins as reads. These reads need to be assembled in order to obtain the complete genome sequence. Read alignment is usually the core step in the genome assembly pipeline [45]. It is the process by which a read is compared to a much larger reference genome to determine where it occurs within the reference. Modern alignment algorithms [39, 46, 47, 53] are designed to handle high-throughput sequencing technologies such as Roche/454, SOLiD, and Illumina that produce giga base-pairs in a machine day [55]. They also take advantage of some characteristics specific to sequencing technologies, such as short read lengths from Illumina and low substitution error rate from Helicos [45].

Most state-of-the-art alignment algorithms construct auxiliary data structures for fast processing and can be broadly classified as either local or global alignment algorithms. Global alignment algorithms aim at aligning two related sequences end-to-end, whereas local alignment algorithms align specific local regions of both sequences with high similarity. Needleman-Wunsch [57] and Smith-Waterman [69] are popular global and local alignment algorithms, respectively, and are among the earliest to use dynamic programming for sequence alignment. Depending on the characteristics of their auxiliary data structures, alignment algorithms can be grouped into the following two categories [45]:

(1) **Hash Table-Based Algorithms:** This category of algorithms follows the seed-and-extend strategy. In this strategy, the algorithm generates seeds from reads, maps the seeds to the reference template, and extends the seeds using standard alignment algorithms that use dynamic programming. BLAST [3, 4] is one of the widely used seed-and-extend algorithms for read alignment. SOAP [47], SeqMap [29], MAQ [46], and SHRiMP [64] are some other modern alignment algorithms that use hash tables.
(2) **Suffix and Prefix Tree-Based Algorithms:** These alignment algorithms build suffix trees, suffix arrays, or full-text minute-space (FM) indices from tries in order to find exact matches. They leverage the fact that the alignment of read patterns in multiple locations of the template need to be done only once. Aligners like MUMmer [35] and OASIS [54] depend on suffix trees, Vmatch [1] and Segemehl [27] on suffix arrays, and Bowtie [40], BWA [43], BWT-SW [36], and SOAP2 [48] on the FM-index.

Long reads usually have a higher likelihood of containing sequencing errors. Therefore, it is imperative that read aligners allow gaps and partial alignments. BLAT [30], SSAHA [58], and BWA-SW [44] are a few modern read aligners available for long reads.

1.2 Related Work

Several works in the past few decades address (or can be adapted to address) the problem of private genome alignment. Blatt et al. [12], Kim and Lauter [31], and Lauter et al. [41] use homomorphic encryption to perform genomic computations on encrypted genomic data in a genome-wide association studies setting. Atallah et al. [7], Baron et al. [10], and De Cristofaro et al. [19] propose private string/pattern matching algorithms using homomorphic encryption that can be adapted to carry out private genome alignment. An important drawback of homomorphic encryption is that it can be computationally cumbersome because of the use of large keys to achieve privacy. Atallah and Li [8] performs string matching using Yao's garbled circuits [74] and a secure two-party protocol. Atallah et al. [7] and Jha et al. [28] are some other works that propose secure multi-party protocols to perform private genomic computations such as read alignment. Al Aziz et al. [2], Jha et al. [28], Szajda et al. [71], and Wang et al. [72] propose privacy-preserving edit distance protocols based on set intersection, garbled circuits, and oblivious transfer that can be used as a building block for private genome alignment. However, these approaches do not scale to the human genome. Recently, Kim et al. [32] introduces SHiMMer, a secure hidden Markov model evaluation method that uses homomorphic encryption for read alignment. Cogo et al. [16], Loka et al. [49], Widanage et al. [73] although for read alignment, use privacy in the context of genome data to perform operations like privacy-preserving real time filtering. Chen et al. [15] uses a hybrid public-private cloud setting to design a secure seed-and-extend alignment algorithm. Popic and Batzoglou [60] also proposes a private read alignment algorithm in a hybrid public-private cloud setting using the MinHash technique. Zhao et al. [75] proposes a hybrid cloud alignment algorithm that delegates both the seeding and extension subtasks to the public cloud, while only performing encryption/decryption tasks on the private cloud. Lambert et al. [37] leverage Intels SGX powered cloud enclave for secure alignment of filtered reads. Fernandes et al. [22] also use hybrid cloud environments to securely align reads. Although private clouds are more secure, they pose major challenges in terms of hardware expenses, visibility and adaptability to communicate with public clouds. Some of the recent secure Private Information Retrieval protocols implemented by Angel et al. [5], Angel and Setty [6], Dai et al. [18], Dong and Chen [21] achieve sub-linear communication with $k \geq 2$ copies of databases stored separately. In our two-cloud algorithm, both clouds communicate with the client less than four times. Bhuyan et al. [11] and Külekci [34] use the Burrows-Wheeler transform for secure compression and transmission respectively.

Threat model: Our algorithm performs secure read alignment across the whole human genome using two semi-honest and non-colluding clouds. A semi-honest adversary follows the protocol as specified honestly but they may try to learn as much information as possible from the messages they receive from other parties.

1.3 Outline of the Paper

Our goal in this work is to devise an algorithm to align a set of reads from an individual's genome to a public reference human genome (template) without revealing SNPs. We begin with an overview of existing alignment algorithms in Section 1.1 and a survey of related work in Section 1.2. We outline the conventional Burrows-Wheeler Transform alignment algorithm in Section 2 and define the problem of private genome alignment in Section 3. Section 4 outlines the different components of our alignment algorithm in detail. Section 5 discusses the security and computational complexity of our algorithm, whereas Section 6 provides formal correctness and privacy guarantees. We provide empirical evidence of the effectiveness of our proposed algorithm in Section 7. Finally, we conclude in Section 8 with some avenues for future work.

2 THE BURROWS-WHEELER TRANSFORM (BWT) ALGORITHM

The Burrows-Wheeler Transform (BWT) is a reversible permutation of a string [13]. The BWT algorithm begins by creating all possible rotations of a string and sorting them lexicographically to create a Burrows-Wheeler matrix of characters. The first column of this BW matrix comprises lexicographical runs of all the characters present in the string. The last column of the BW matrix is the Burrows-Wheeler Transform. The BWT algorithm has applications in several diverse fields, including sequence alignment [40, 43, 48], image and data compression [17, 20, 52, 61]. A useful concept in the derivation of the BWT is the so-called T-ranking.

Definition 1. [T-ranking] The T-ranking assigns ranks to each character in a string X such that the rank of its ith symbol $X[i]$ is defined as

$$\mathtt{rank}(X[i]) := |\{j \in \{1, \ldots, i\} : X[j] = X[i]\}| - 1.$$

The BWT algorithm considers strings to be cyclic and transforms its characters by their right context. If a string t follows a symbol s in the text, then t is called a right context of s. The BW matrix will always have a row beginning with t and ending with s. Since the rows are sorted lexicographically, all symbols with the right context t appear consecutively in the BWT, which makes it easy for compression. In addition, the sorting order of the rotations of the string is exactly the same as the sorting order of suffixes. Therefore, the algorithm can be implemented in linear time using suffix arrays [13]. This concept is fundamental to the FM-index, a data structure used for the compression and search over large bodies of text [23]. The BWT can be easily reversed because the BW matrix satisfies an important property known as LF mapping. The LF mapping property notes that the ith occurrence of a character in the first and last columns of the BW matrix correspond to the same character and location in the original string. The LF mapping helps us locate the occurrence of any particular symbol in the first column of the BW matrix.

Reads are aligned using the FM-index. The FM-index comprises the BWT and the first column of the BW matrix. Consider a string w and let y be a substring of w. The locations of y within w correspond to an interval in the suffix array of w. Therefore, aligning a read using the FM-index is the same as finding the right intervals corresponding to the read in the suffix array. Figure 1 shows alignment of a read "*CCT*" to a template "*ACCTCTA*". In order to map this read to the template, the algorithm first locates all the occurrences of the smallest suffix of the read in the template. This is equivalent to checking for locations of the base T in the first column. Next, the algorithm looks for the base C in the corresponding rows in the last column. This is same as searching for all occurrences of the substring "*CT*" in the input template string. This corresponds to "C_2T_1" and "C_1T_0" in our original template string. Then, the corresponding C's in the first column are located. In our example, these are C_1 and C_2 in the first column. The algorithm continues to look for the next largest suffix. The algorithm stops once all the characters in the read are processed. The suffix array interval is extracted using the rows of the first column after the algorithm terminates. For inexact alignment, algorithms such as the Bowtie and BWA employ a backtracking approach to approximate string matching [38, 43]. There are several possible optimizations to ensure efficiency of the BWT alignment algorithm [24, 33, 65]. The FM-Index can be used in cloud setting by passing first column of Burrows-Wheeler Matrix and the BWT to different clouds by allowing communication between clouds and finally retrieve the alignments using suffix array. Our algorithm uses the FM-index with a few key modifications. Instead of lexicographically sorting the rotations, our algorithm independently shuffles the first and last columns of the rotation matrix directly for privacy considerations. Additionally, the first row of the rotation matrix is simply the concatenation of the reference template string and the \$ symbol, whereas its last row is the concatenation of the \$ symbol and the reference template. Therefore, our algorithm does not require explicit construction of all of the rotations of the reference template string.

3 PROBLEM SETUP

We are given a publicly available reference human genome template X and a list of reads. Our algorithm performs some processing steps on a secure computer (*Client*) before sending selective information to two non-colluding clouds (*CloudF* and *CloudL*). It then uses the clouds to align the set of reads to the reference template privately without revealing the contents of individual reads or the locations of SNPs to either cloud. We assume that both the clouds are non-colluding semi-honest adversaries. By semi-honest adversaries, we mean that the clouds follow the exact specified protocol and will not make an attempt to change their inputs/outputs however, they may attempt to infer the truth from publicly available information, or by running post-processing steps on any intermediate information they get while executing our algorithm.

3.1 Notation

We write $|e|$ to denote the length of a string/array e and $e[i]$ to denote its ith entry. Throughout, we index arrays and strings starting from one. We assume that the reference human genome template X and the n reads $r_1, r_2, \ldots, r_n$ generated by the sequencing technology are strings over the alphabet $\Sigma := \{A, T, G, C\}$. We assume for simplicity that $|r_i| = l, \forall i \in \{1, \ldots, n\}$. We write $\mathtt{conc}(a_1, a_2, \ldots, a_j)$ to denote the concatenation of strings/arrays $a_1, a_2, \ldots, a_j$ in order. Let $e[i : j]$ denote the substring/subarray of a string/array e starting at position i and ending at position j (both endpoints included). We often omit quotation marks while writing strings. Let $\pi(k)$ denote a random permutation of the array $[1, 2, \ldots, k]$. In our algorithm, we transform the first and last columns of the rotation matrix of the reference human template X. We call the transformed first column array F and the transformed last column array L. We denote the transformed suffix array by SA.

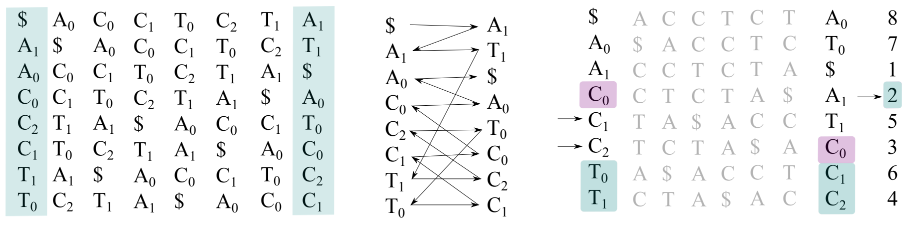
Figure 1: Burrows-Wheeler Transform of the string "ACCTCTA". Left: BWT with T-ranking; Middle: BWT reversibility using LF mapping; Right: FM-index-based alignment of the string "CCT"

4 TWO-CLOUD SHUFFLED BWT ALGORITHM

Our algorithm begins by transforming the reads and the public reference template on the client's secure computer before sending them as inputs to the clouds. The Two-Cloud Shuffled Burrows-Wheeler Transform alignment algorithm is then carried out with communication between the clouds to generate candidate read alignments. These candidate alignments are then returned to the client, which then uses the transformed suffix array (only available to the client) to retrieve the final set of alignments after post-processing.

Algorithm 1 Transforming a Reference Template Chunk

```
1: procedure TransformTemplateChunk($X$)
2:     Shuffle conc($X$, \$) and conc(\$, $X$) independently
       to form $F$ and $L$
3:     Shuffle the suffix array in the same way as
       conc($X$, \$) to form $SA$
4:     Establish links between $F$ and $L$
5:         $F'[i] \leftarrow$ location in $L$ of the ranked base to
                   the left of $F[i]$ in $X$
6:         $L'[i] \leftarrow$ location in $F$ of the ranked base $L[i]$
7: end procedure
8: Store $L'$ in CloudF, $F'$, $L$ in CloudL, $SA$ with Client
```

Transforming the Reference Template: We begin by transforming the reference template on *Client*. We first divide the reference template into large chunks and scramble them. Without loss of generality, we also denote scrambled template chunks by X. We then transform each scrambled chunk using Algorithm 1 on *Client*. As noted in Section 2, the first and last columns of the matrix formed by all the left rotations of conc(X, \$) can be written as conc(X, \$) and conc(\$, X), respectively.

Let U and V be the array representations of strings conc(X, \$) and conc(\$, X), and $m := |U| = |V|$. We generate independent random permutations and use them to construct the shuffled first and last column arrays F and L, respectively, from U and V, as noted in Definition 2. The shuffled suffix array SA, which is the same as the permutation applied to the first column of the rotation matrix, can be used by the client to retrieve the original alignments from the candidate alignments returned by the clouds.

Definition 2. [Shuffled first and last columns] Given random permutations $\pi_f = \pi(m)$ and $\pi_l = \pi(m)$, the shuffled first and last columns are defined as

$$F[i] = U[\pi_f[i]], \quad L[i] = V[\pi_l[i]], \quad \forall i \in \{1, \dots, m\}.$$

The transformed suffix array SA is defined as

$$SA[i] = \pi_f[i], \quad \forall i \in \{1, \dots, m\}.$$

We also define arrays F' and L' to store the links between the shuffled columns F and L. These linking arrays are required to carry out the alignment algorithm between the two clouds because the first and last columns are shuffled independently. F' and L' are in-directions needed to shield the knowledge of F and L. For instance, if *CloudL* has the knowledge of F and L, $\mathcal{RO}$, it can deduce the alignment positions of the different reads by storing a directed graph whose nodes are the different list of rows of F encountered during the course of the algorithm. This graph can yield a subset of the different alignment positions. Similarly, if *CloudF* had access to both F and L, using the read order $\mathcal{RO}$ and the bases accessed in F at every iteration, it is possible to build the read. F' and L' give access to F and L when stored in different clouds. For example, given F and L', without, the knowledge of F', it is not possible to construct L. However, given the indices, it may be possible to construct F'. Even then, it is computationally expensive and may not be feasible to build F and L. We store L' in *CloudF* and F' and L in *CloudL*. Note that we do not store F or SA in *CloudF* or *CloudL*.

Definition 3. [Linking arrays F' and L'] Suppose we are given random permutations $\pi_f = \pi(m)$, $\pi_l = \pi(m)$ and T-ranking is used to define the first and last column arrays F and L. The array F' is such that for each $i \in \{1, \dots, m\}$, $F'[i]$ is the index/location in L of the ranked base to the immediate left of $F[i]$ in X. The array L' is such that for each $i \in \{1, \dots, m\}$, $L'[i]$ is the index/location in F of the ranked base $L[i]$.

Example 1. Let us revisit the example in Section 2. We have $X = A_0C_0C_1T_0C_2T_1A_1$ with T-ranking. Let $\pi_f = [5, 3, 2, 1, 8, 6, 4, 7]$ and $\pi_l = [2, 1, 5, 6, 8, 7, 3, 4]$ be the random permutations used to shuffle conc(X, \$) and conc(\$, X), respectively. From Definition 2, we have:

$$U = [A_0, C_0, C_1, T_0, C_2, T_1, A_1, \$],$$
$$V = [\$, A_0, C_0, C_1, T_0, C_2, T_1, A_1],$$
$$F = [C_2, C_1, C_0, A_0, \$, T_1, T_0, A_1],$$
$$L = [A_0, \$, T_0, C_2, A_1, T_1, C_0, C_1].$$

From Definition 3, we can also compute the linking arrays F' and L' as follows:

$$F' = [3, 7, 1, 2, 5, 4, 8, 6], \quad L' = [4, 5, 7, 1, 8, 6, 3, 2].$$

Algorithm 2 Transforming the Input Reads

```
1: procedure TransformReads($reads, g$)
2:     Divide the $reads$ into $g$ groups
3:     Create random bijections between reads in
       group 1 and reads in groups 2 to $g$
4:     Create read order $\mathcal{RO}$ for reads in group 1
5:     Use $\mathcal{RO}$ to create super-read $\mathcal{SR}_1$ for reads in
       group 1 satisfying Property 1
6:     Create $\mathcal{SR}_i$ for reads in groups $i = 2$ to $g$ using
       $\mathcal{SR}_1$ and the constructed bijections
7: end procedure
8: Store $\mathcal{RO}$ in CloudF, $\{\mathcal{SR}_i\}_{i=1}^{g}$ in CloudL, $readGroups$ and
   $readMappings$ with Client
```

Here, $F'[1]$ is the index/location in L of the base T_0, which is to the immediate left of $F[1] = C_2$ in X. Since T_0 is in the third position of L, we have $F'[1] = 3$. Similarly, $L'[1]$ is the index/location in F of the base $L[1] = A_0$. Since A_0 occurs in the fourth position of F, we have $L'[1] = 4$. We can recover F and L by reverse engineering the above steps. In order to do this, access to both F' and L' is required. We use T-ranking in F and L only for illustration, but do not store ranks of occurrence of the bases in either cloud unlike the conventional FM-index.

Transforming Reads: Next, we describe the procedure for transforming the reads on the client's secure computer. We randomly partition the set of reads into $g \geq 2$ groups of equal size. We then interleave the reads of each group using a randomized procedure while preserving the relative ordering of the symbols in the individual reads to construct a super-read $\mathcal{SR}_k$ for each read group k that satisfies Property 1. In other words, a super-read $\mathcal{SR}_k$ is an interleaved arrangement of the set of input reads in group k such that the jth symbol of each read $r_i[j]$ in group k occurs before the $(j+1)$th symbol of the same read $r_i[j+1]$ within $\mathcal{SR}_k$ for each i and j. The motivation for constructing a super-read for each group is to ensure that the clouds are oblivious to the individual contents of the reads in the group.

Property 1. A super-read $\mathcal{SR}$ of reads $r_1, \ldots, r_n$, each of length l, is a restrictive random permutation of $\mathsf{conc}(r_1, \ldots, r_n)$ such that for each $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, l-1\}$, $r_i[j]$ occurs before $r_i[j+1]$ in the permutation.

We now explain how such a super-read may be constructed in practice. A super-read $\mathcal{SR}$ satisfying Property 1 can be constructed using a random permutation of the following array of read indices of length $l \times n$:

$$\mathsf{conc}([\underbrace{1, \ldots, 1}_{l\ terms}], [\underbrace{2, \ldots, 2}_{l\ terms}], \ldots, [\underbrace{n, \ldots, n}_{l\ terms}]).$$

We call this random permutation the read order $\mathcal{RO}$ in the super-read $\mathcal{SR}$. Given a read order $\mathcal{RO}$, we can construct the super-read $\mathcal{SR}$ as follows. Suppose $\mathcal{RO}[i] = j$ for some $j \in \{1, \ldots, n\}$ and the read index j occurs exactly $k \in \{1, \ldots, l\}$ times in the subarray $\mathcal{RO}[1:i]$. Then we set $\mathcal{SR}[i] = r_j[k]$. We follow a similar procedure to specify each element of the super-read $\mathcal{SR}$ using $\mathcal{RO}$. We use the above procedure to create the super-read and read order for the first group of reads. We then construct random bijections between the reads in group 1 and the reads in each of the other $g-1$ groups. We use these mappings to form the super-reads for the remaining $g-1$ groups by transforming the read order and super-read of read group 1 using these bijections. We provide *CloudF* with the read order for group 1 and *CloudL* with the super-reads for all g groups.

Example 2. Suppose we have four reads $r_1 = ACC$, $r_2 = CCT$, $r_3 = TCT$, and $r_4 = CTA$ and divide them into two groups with $readGroups = [[1,4],[2,3]]$. Let $readMappings = \{1:[2],\ 4:[3]\}$ denote the random bijections from reads in group 1 to the reads in the remaining groups. Suppose $\mathcal{RO} = [1,4,4,1,4,1]$. Note that $\mathcal{RO}$ comprises only the read numbers 1 and 4 since $readGroups[1] = [1,4]$. We use the read order $\mathcal{RO}$ to construct a super-read $\mathcal{SR}_1 = [A, C, T, C, A, C]$ for group 1 that satisfies Property 1. Using the random bijection defined by $readMappings$, we construct the *implicit read order* for group 2 to be $\mathcal{RO}_2 = [2,3,3,2,3,2]$. We use this implicit read order for group 2 to build the super-read $\mathcal{SR}_2 = [C, T, C, C, T, T]$ for group 2.

Encoding Group Information: *CloudL* selects a list of large random primes $\mathcal{P}_k$ for each group k. Instead of communicating the group number k to *CloudF*, *CloudL* selects a random prime p'_k from $\mathcal{P}_k$ and sends a random number r' such that $r' \equiv 1 \pmod{p'_k}$ to *CloudF*. *CloudF* picks a large random number N and a polynomial $poly$ with positive coefficients that sum ($sumCoeff$) to N. *CloudF* then returns $polyVal = poly(r')$ and N to *CloudL*. Finally, *CloudL* assigns group number j if $poly(r') \pmod{p'}$ equals $N \pmod{p'}$ for some $p' \in \mathcal{P}_j$ and $j \in \{1, \ldots, g\}$. Because $poly(r') \pmod{p'_k}$ equals $N \pmod{p'_k}$ for the original group number k and chosen prime $p'_k \in \mathcal{P}_k$, the original group number k will be recovered by *CloudL*.

The Alignment Algorithm: The alignment of the reads to each scrambled reference template chunk proceeds in a manner similar to the conventional BWT alignment algorithm. This procedure is outlined in Algorithm 3. The partial alignments for the reads in groups 2 to g are stored in the entry corresponding to the read in group 1 to which they are mapped.

CloudF sends (index, polyVal, sumCoeff) pairs corresponding to the current read number in the read order $\mathcal{RO}$ to *CloudL*. *CloudL* identifies the group number for each tuple sent by *CloudF*. It also transforms the indices of F to indices of L using the links F'. *CloudL* then collects all of the candidate indices of L corresponding to each group. It then determines which of these locations in L contains the current base of the corresponding group's super-read. *CloudL* encodes the group number and sends the (index, E(groupNum)) pairs for indices in L that contain the current base in the super-read to *CloudF*. *CloudF* converts the encoded group numbers into (polyVal, sumCoeff) pairs. *CloudF* then transforms the indices of L to the corresponding indices of F using the links L' and updates the entry in $\mathcal{A}$ of the current read number in $\mathcal{RO}$. The algorithm terminates when the entire read order $\mathcal{RO}$ is processed from right to left. After termination, *CloudF* returns the final list of ordered pairs of candidate alignments and encryptions for all reads to the client. Finally, *Client* computes the final alignment positions for each read after post-processing the candidate alignments using Algorithm 4.

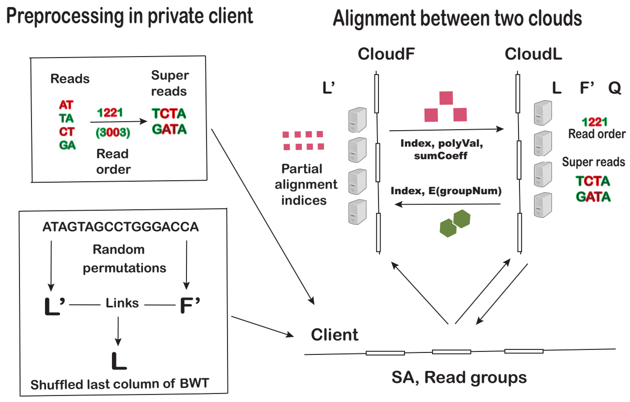

Figure 2: Outline of the alignment procedure between two clouds in the Two-Cloud Shuffled BWT Algorithm

Inexact Alignment: When none of the candidate indices of L contain the current base of the super-read for a particular read in group k, *CloudL* does not return any partial alignment indices for this read to *CloudF* during that iteration. Our Two-Cloud Shuffled BWT alignment algorithm can be adapted for the case of inexact alignment as follows: if *CloudL* does not encounter any candidate indices of L for a particular read group k, it simply returns all indices of L that contain the current base of $\mathcal{SR}_k$ to *CloudF*. Since this inexact alignment strategy may sometimes lead to an explosion in the number of candidate alignments, especially if the read contains a SNP among its first few bases, the post-processing algorithm filters these alignments depending on the maximum number of allowed mismatches specified.

Post-processing Candidate Alignments: The indices of F returned by the Two-Cloud Shuffled BWT algorithm are post-processed on *Client* using Algorithm 4. This algorithm begins by decrypting group numbers and separating the candidate alignments for each individual read. The algorithm then filters the alignments given a limit on the maximum number of mismatches allowed to determine final alignments for each read.

Optimization of Communication Between Clouds: In each chunk, our algorithm produces more inexact alignments than exact ones. This results in explosion of communication costs as both clouds exchange nearly half the template size of indices whenever there is an inexact alignment. In order to cut down on these costs, in addition to groups numbers, we also maintain a threshold Y for all reads and a cutoff $C_i, i \in \{1 \ldots n\}$ for each read. Suppose the threshold is the minimum of number of occurrences of all the bases in the reference template. We increase the cutoff for a read by 1 whenever *CloudL* transfers more than Y indices to *CloudF*. We communicate the cutoff by obfuscating them using the same scheme we use of the group numbers. Once the cutoff exceeds 2, we do not send any indices for that read from then on.

5 DISCUSSION OF THE PROPOSED APPROACH

Modern DNA sequencing technologies produce millions of reads. These reads contain recurring patterns (e.g., codons) in the human genome. Patterns in the human genome are public knowledge because the reference human genome template is publicly available. Consequently, private genome alignment algorithms should be carefully designed to ensure that these patterns do not risk exposure of SNPs or alignment positions. In our algorithm, we use two clouds to achieve our goal of privacy for SNPs. We achieve this by only providing selective information to each of the clouds without compromising the security of the reads during any given phase of execution. In addition, we introduce the following *five novel ideas* to achieve our correctness and privacy goals:

(1) **Shuffling First and Last Columns of the Rotation Matrix:** We list all of the left rotations of the template string in order to obtain the rotation matrix of characters. The ith entry in the first column of the rotation matrix is linked to the ith entry in the last column, whereas the ith entry in the last column of the rotation matrix is linked to the $(i-1)$th element of the first column. We shuffle the first and last columns of this rotation matrix independently and use the corresponding permutations to determine the new links between the shuffled columns. As noted in Section 2, we directly shuffle $\mathsf{conc}(X, \$)$ and $\mathsf{conc}(\$, X)$ to achieve this.

(2) **Constructing Super-reads:** In order to keep the reads private, we transform groups of reads into super-reads. Each super-read is constructed using a restrictive random permutation of the concatenation of the reads in its group (or alternatively, using a sequence of read numbers called the read order).

(3) **Dividing Reads into Groups and Encrypting Group Information:** We randomly partition the set of reads into $g \geq 2$ groups of equal size. We then create the super-read and read order for the first read group as detailed in Section 4. The random one-to-one mappings of the read numbers in group 1 to those of other groups ensures correctness of the algorithm when reads of all g groups are processed in clusters. *CloudL* aligns the next bases of the current set of g reads as determined by the read order and the g super-reads. It then combines the partial alignments for the current reads of all g groups and encrypts group information. This encryption precludes *CloudF* from developing pattern-based attacks. We provide more details below and in Section 6.

(4) **Partitioning and Scrambling the Template into Multiple Chunks:** After each iteration of the algorithm, *CloudF* gets a set of partial alignment indices from *CloudL*. These indices will be a subset of indices of F of the corresponding base. If group information is known, the number of partial alignment indices for a particular read indicate the number of occurrences of its processed substring. With the knowledge of the public reference template, the cloud could compare the number of occurrences of every possible realization of this pattern with the number of partial alignment indices to potentially deduce the pattern. To avoid this, we divide the template into several large chunks and concatenate a part

Algorithm 3 Outline of the Two-Cloud Shuffled BWT Alignment Algorithm

```
procedure TwoCloudShuffledBWT($L, F', L', \mathcal{RO}, \{\mathcal{SR}_k\}_{k=1}^{g}$)
    CloudF
    Initialization
        Get $L'$ and $\mathcal{RO}$ from Client
        Define dictionary $\mathcal{A}$ with $\mathcal{A}[readNum]$ = [ ] for each unique $readNum$ in $\mathcal{RO}$
        Set current iteration number $i \leftarrow |\mathcal{RO}|$

    Alignment
        if iteration number $i < |\mathcal{RO}|$ then
            Get list of (index, E(groupNum)) tuples from CloudL for read number $\mathcal{RO}[i]$
            Convert all indices of $L$ (first terms of tuples) to indices of $F$ using $L'$
            Convert second terms of tuples (E(groupNum)) to (polyVal, sumCoeff) tuples
            Update $\mathcal{A}[\mathcal{RO}[i]]$ to the list of (index, polyVal, sumCoeff) tuples and set $i \leftarrow i - 1$
        end if

    Communication
        Send list of (index, polyVal, sumCoeff) tuples from $\mathcal{A}[\mathcal{RO}[i]]$ to CloudL

    CloudL
    Initialization
        Get $F'$, $L$ and super-reads $\mathcal{SR}_k$, $k \in \{1, \ldots, g\}$, from Client

    Alignment
        Get list of (index, polyVal, sumCoeff) tuples from CloudF
        Identify group numbers from (polyVal, sumCoeff) pairs
        Convert all indices of $F$ (first terms of tuples) to indices of $L$ using $F'$
        Pool together candidate indices of $L$ for each group
        No candidate indices of $L$ for a group $\implies$ use all indices of $L$ as candidates
        for each group $k \in \{1, \ldots, g\}$ do
            Among the indices for group $k$, check for current base of the super-read $\mathcal{SR}_k[i]$ in $L$
            For each matching index of $L$, assemble the index and E(k) as a tuple
        end for

    Communication
        Send the list of (index, E(groupNum)) ordered pairs to CloudF
        if iteration number $i == 0$ then
            Return list ($index$, E($groupNum$)) of ordered pairs to Client
        end if

    Client
    Preprocessing
        Transform the reference template chunk using Algorithm 1 to get $F$, $L$, $F'$, $L'$, and $SA$
        Transform input reads using Algorithm 2 to get the super-reads $\{\mathcal{SR}_k\}_{k=1}^{g}$ and read order $\mathcal{RO}$
        Send $L'$, $\mathcal{RO}$ to CloudF, and $L$, $F'$, $\{\mathcal{SR}_k\}_{k=1}^{g}$ to CloudL

    Post-processing
        Post-process candidate alignments returned by CloudL using Algorithm 4
        Return final set of alignments $\mathcal{F}$ for each read given limit on number of mismatches
end procedure
```

or whole of a template chunk with another template chunk. This ensures *CloudF* cannot employ direct pattern attacks because the number of partial alignment indices now only corresponds to the number of patterns of a read substring in that particular template chunk.

(5) **Inexact Alignment:** If a read contains a SNP, none of candidate indices of the shuffled last column L may contain its

Algorithm 4 Post-processing Candidate Alignments

```
1: procedure POSTPROCESSING($X$, $reads$, $\mathcal{A}$, $SA$,
        $readGroups$, $readMappings$, $misLimit$)
2:     Initialize $\mathcal{F} = \{j : \emptyset \text{ for } j \in \{1, \ldots, n\}\}$
3:     for $readNum$ in $readGroups[1]$ do
4:         for $(index, pv, sc)$ in $\mathcal{A}[readNum]$ do
5:             Let $i \leftarrow \mathsf{D}(pv, sc)$ be the group number
6:             if $i == 1$ then
7:                 $j \leftarrow readNum$
8:             else
9:                 $j \leftarrow$ read number in group $i$
                       corresponding to $readNum$
10:            end if
11:            Set $k \leftarrow SA[index]$
12:            if $\texttt{diff}(X[k : k + l - 1], r_j) \leq misLimit$
13:                $\mathcal{F}[j] \leftarrow \mathcal{F}[j] \cup \{k\}$
14:            end if
15:        end for
16:    end for
17: end procedure
18: Return final set of alignments $\mathcal{F}$ to $Client$
```

SNP base (i.e., the current base of its group's super-read). We let *CloudL* return an empty set of partial alignments and continue with the alignment for the rest of the read. During post-processing, we filter these inexact alignments based on the specified mismatch limit.

Since the reference human genome template is public knowledge, one may question the need to transform the first and last columns of the rotation matrix. Consider an alternative approach where *CloudF* only contains the read order $\mathcal{RO}$ in the super-read, while *CloudL* contains the unshuffled first and last columns F and L of the BWT matrix along with the super-read $\mathcal{SR}$. Although *CloudL* does not know the read order $\mathcal{RO}$, it can deduce the alignment positions of the different reads by storing a directed graph whose nodes are the different list of row indices of F encountered during the course of the algorithm. An edge from node u to node v of the graph indicates that during some iteration and for some read, the list of matching row indices of L was the subset v of the row indices u of L. The leaves of this graph yield a subset of the different alignment positions. Moreover, *CloudL* can traverse the graph to locate the relative position of any SNPs within a read. Therefore, it is crucial to transform the public reference template and only provide selective information to the two clouds.

Computational Complexity: The pre-processing steps for the reference template and the computation of F', L', and SA require $O(m)$ space and memory, where m denotes the size of each template chunk. The pre-processing steps for the n reads, including the construction of the read-order $\mathcal{RO}$ and the g super-reads $\mathcal{SR}_1, \ldots, \mathcal{SR}_g$, require $O(nl)$ space and memory. The per-iteration cost of our alignment algorithm is equal to $O(m)$ per read, which is the same as the per-iteration cost of the traditional BWT alignment algorithm that uses the FM-index (although the actual realized cost may be significantly smaller). Practical implementations of the BWT alignment algorithm, such as Bowtie [38], implement several optimizations that improve the efficiency of the basic alignment algorithm. While some of these techniques might be adapted to our two-cloud alignment algorithm as well, we must take care to ensure that they do not compromise security of our approach.

6 CORRECTNESS AND PRIVACY GUARANTEES

In this section, we prove that our Two-Cloud Shuffled BWT algorithm recovers all exact and inexact alignments whenever a read does not contain a SNP in its first position. We also analyze the security of the algorithm and provide argumentation for why it does not reveal the presence of SNPs in an individual's genome.

6.1 Proof of Correctness

We begin with the following definitions of exact and inexact alignments.

DEFINITION 4. [Exact alignment] A read r of length l is said to align exactly with the template X at position p if the substring $X[p : (p + l - 1)]$ is identical to the string r.

DEFINITION 5. [Inexact alignment] Given a mismatch limit $M \geq 1$, a read r of length l is said to align inexactly with the template X at position p if $X[p : (p + l - 1)]$ and r differ by at least one and at most M elements.

We now prove that the Two-Cloud Shuffled BWT algorithm recovers all exact alignments of reads in the original template X. While without loss of generality we consider a simplified setting with one template chunk, two reads, two read groups, and trivial encodings for ease of exposition, our arguments readily extend to the more general setting of Algorithm 3.

THEOREM 2. [Correctness of the Two-Cloud Shuffled BWT Algorithm] Consider the setting with one template chunk X of length m, two reads each of length l ($n = 2$), two read groups ($g = 2$), and with trivial encodings, i.e., $\mathsf{E}(i) \equiv i$ and the corresponding (polyVal, sumCoeff) $\equiv (i, i)$ for $i = 1, 2$. Assume without loss of generality that reads r_1 and r_2 are in read groups 1 and 2, respectively. Suppose read r_1 has an exact alignment with the template X at position p. Then the index p is an element of the set of final alignments $\mathcal{F}[1]$ determined by *Client* after post-processing using Algorithm 4.

PROOF. Given random permutations $\pi_f = \pi(m)$ and $\pi_l = \pi(m)$, let the shuffled first and last columns F and L and the shuffled suffix array SA be defined as in Definition 2. Let the linking arrays F' and L' be defined as in Definition 3. Note that the super-read $\mathcal{SR}_k$ for group $k \in \{1, 2\}$ is simply the array $r_k[1 : l - 1]$ since there is exactly one read in each group.

We proceed by induction. For the base case, we argue that the set of partial alignments $\mathcal{A}[1]$ returned by *CloudL* after one base of reads r_1 and r_2 are processed includes the tuple $(j, 1, 1)$, where the index j is such that $F[j] = X[p+l-1] = r_1[l]$. Then, as the induction hypothesis, we assume that after $k \geq 1$ bases of reads r_1 and r_2 are processed, the set of partial alignments $\mathcal{A}[1]$ includes the tuple $(j, 1, 1)$, where the index j is such that $F[j] = X[p+l-k] = r_1[l+1-k]$. We then demonstrate that after one more iteration of the Two-Cloud Shuffled BWT algorithm is executed (i.e., after $k + 1$ bases of reads r_1 and r_2 are processed), the tuple $(q, 1, 1)$ is an element of the set of partial alignments $\mathcal{A}[1]$, where the index q is such that $F[q] = X[p+l-k-1] = r_1[l-k]$. Finally, we conclude by induction that when the Two-Cloud Shuffled BWT algorithm terminates (i.e., after all l bases of reads r_1 and r_2 are processed), the tuple $(q, 1, 1)$ is an element of the set of partial alignments $\mathcal{A}[1]$, where the index q is such that $F[q] = X[p] = r_1[1]$. This readily implies that the exact alignment at index p for read r_1 is recovered by the secure client computer after post-processing.

Base Case. We begin by showing that the base case of our induction hypothesis holds. Note that the set of partial alignments $\mathcal{A}[1]$ returned by *CloudL* after one base of reads r_1 and r_2 are processed includes all pairs $(j, 1, 1)$ with indices $j \in \{1, \ldots, m\}$ of F such that $F[j] = r_1[l]$. Since read r_1 aligns with the template exactly at position p by assumption, we have $X[p + l - 1] = r_1[l]$. Consequently, the set $\mathcal{A}[1]$ includes the tuple $(j, 1, 1)$ such that $F[j] = X[p + l - 1] = r_1[l]$.

Inductive Step. Suppose that after $k \geq 1$ bases of reads r_1 and r_2 are processed, set $\mathcal{A}[1]$ includes the tuple $(j, 1, 1)$, where the index j is such that $F[j] = X[p+l-k] = r_1[l+1-k]$. We wish to show that after one more iteration of the Two-Cloud Shuffled BWT algorithm is executed (i.e., after $k + 1$ bases of reads r_1 and r_2 are processed), the tuple $(q, 1, 1)$ is an element of the set of partial alignments $\mathcal{A}[1]$, where the index q is such that $F[q] = X[p + l - k - 1] = r_1[l - k]$. By assumption, the list of ordered pairs $\mathcal{A}[1]$ sent by *CloudF* to *CloudL* includes the tuple $(j, 1, 1)$, where the index j is such that $F[j] = X[p + l - k] = r_1[l + 1 - k]$. *CloudL* first decrypts the group number to identify that the index j is among the list of indices of F for group 1. It then transforms the index j of F to the index $F'[j]$ of L, which by Definition 3 yields the index v of L such that

$$L[v] = X[p + l - k - 1] \tag{1}$$

since $F[j] = X[p+l-k]$. *CloudL* then checks whether $L[v]$ contains the same base as $\mathcal{SR}_1[l - k] := r_1[l - k]$. Since read r_1 has an exact alignment at position p of X by assumption, we have from (1) that

$$L[v] = X[p + l - k - 1] = r_1[l - k]. \tag{2}$$

Therefore, $L[v]$ contains the same base as $\mathcal{SR}_1[l - k]$ and *CloudL* sends the tuple $(v, 1, 1)$ among its list of ordered pairs to *CloudF*. Finally, *CloudF* transforms the index v of L to the index $L'[v]$ of F, which by (2) and Definition 3 yields the index q of F such that

$$F[q] = X[p + l - k - 1].$$

Therefore, when the set of partial alignments $\mathcal{A}[1]$ is updated by *CloudF* after $k + 1$ bases of reads r_1 and r_2 are processed, it contains the tuple $(q, 1, 1)$ such that $F[q] = X[p + l - k - 1] = r_1[l - k]$. This proves our inductive step.

Inductive Argument. By induction, when the Two-Cloud Shuffled BWT algorithm terminates after all l bases of reads r_1 and r_2 are processed (i.e., $k = l - 1$), the tuple $(q, 1, 1)$ is an element of the set of partial alignments $\mathcal{A}[1]$, where the index q is such that $F[q] = X[p] = r_1[1]$. Therefore, the set of candidate alignments returned by *CloudF* to the client includes the tuple $(q, 1, 1)$ such that $F[q] = X[p] = r_1[1]$. This readily implies that the exact alignment at index p for read r_1 is recovered by the secure client computer after post-processing. ◻

We next show that our algorithm recovers all inexact alignments for reads with SNPs so long as these SNPs do not occur at the first read index. For ease of exposition, we consider the same setting as Theorem 2 but when read r_1 has an inexact alignment with one mismatch.

Corollary 3. [Alignment with single SNP] *Consider the setting in Theorem 2 with mismatch limit $M = 1$. Suppose read r_1 has an inexact alignment with the template X at position p with one mismatch. Assume further that $r_1[1] = X[p]$ and read r_1 does not align exactly with template X. Then the index p is an element of the set of final alignments $\mathcal{F}[1]$ determined by Client after post-processing using Algorithm 4.*

Proof. Let $i \in \{2, \ldots, l\}$ denote the index of read r_1 such that $r_1[i] \neq X[p+i-1]$. Such an index i exists because r_1 aligns inexactly with X at position p with one mismatch. Note that $r_1[j] = X[p+j-1]$ for each $j \neq i$. Theorem 2 thus implies that *CloudL* returns a non-empty list of partial alignments to *CloudF* after each of the bases $r_1[j], j \in \{i + 1, \ldots, l\}$, are processed. By assumption, r_1 does not align exactly with X. Consequently, there is an index $q \in \{2, \ldots, i\}$ such that after the base $r_1[q]$ is processed by the alignment algorithm, *CloudL* returns an *empty* list of partial alignments for read r_1 to *CloudF* (note that $q \neq i$ necessarily). Since the substrings $r_1[1 : q - 1]$ and $X[p : p + q - 2]$ match exactly, when the next base $r_1[q - 1]$ of r_1 is processed by the alignment algorithm, *CloudL* returns a *non-empty* list of partial alignments for $r_1[q - 1]$ in X to *CloudF*. Similarly, by Theorem 2, for any index $j \in \{1, \ldots, q - 1\}$, after the base $r_1[j]$ is processed by the alignment algorithm, *CloudL* returns a *non-empty* list of partial alignments for $r_1[j : q - 1]$ in X to *CloudF*. Therefore, at termination, *CloudL* returns the list of partial alignments for $r_1[1 : q - 1]$ in X to *CloudF*. Since the substrings $r_1[1 : q - 1]$ and $X[p : p + q - 2]$ match exactly, Theorem 2 then implies that the index p is an element of the set of candidate alignments $\mathcal{A}[1]$ returned by *CloudF* to the *Client*. Because r_1 aligns inexactly with X at position p with one mismatch, the index p is an element of the set of final alignments $\mathcal{F}[1]$ determined by *Client* after post-processing using Algorithm 4. ◻

6.2 Security Analysis

In this section, we formally analyze security guarantees for our algorithm. We begin by discussing the privacy guarantees afforded by our modified BWT structure. In particular, we argue that *CloudF* and *CloudL* cannot simply use pattern analysis to identify the individual contents of the reads. We assume throughout that *CloudF* and *CloudL* are non-colluding clouds. In addition, each client independently generates the data structures used in the clouds. In order to make sure that only a legitimate client can interact with a given data structure in the cloud, authentication mechanisms can be used.

Security Analysis for CloudF. We shuffle the first and last columns of the rotation matrix independently preserving the links between indices of the elements. Storing L' and F' in different clouds ensures that neither cloud knows the indices of L that corresponds to indices of F and vice versa. Additionally, neither cloud has access to the shuffled suffix array. This ensures that the final alignments cannot be deduced by both *CloudF* and *CloudL*. Randomly scrambling large template chunks provides security from pattern attacks. For example, it is known that the substring *TAT* occurs 239268 times in the human genome. Similar counts can be computed for all substrings of size three (codons) in the reference human genome. Therefore, a possible privacy leak while computing partial alignments for the last three bases of each read is that *CloudF* can compare the number of partial alignments for a read with the counts of different codons in the reference template. *CloudF* can then use this information to infer the shuffled first column array F and the contents of the individual reads. We preclude this privacy leak by randomly scrambling and splitting the reference template into chunks. This ensures the counts of each codon in each template chunk is a random fraction of the total codon count in the reference template. By re-numbering reads, generating different read groups, read mappings across groups, and super-reads for each scrambled template chunk, we ensure that *CloudF* cannot directly add up the number of partial alignments across each template chunk to identify the contents of individual reads.

Suppose *CloudF* can use associativity rules to partition the indices of F into four disjoint groups—one for each base A, C, T, and G. Additionally, suppose *CloudF* randomly assigns the symbols W, X, Y, and Z to these four groups. Note that there are 4! possible bijections from $\{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$ to $\{A, C, T, G\}$, i.e., 4! valid assignments for the tuple (W, X, Y, Z). To motivate our analysis, consider a single scrambled template chunk of length m. After any iteration of our Two-Cloud Shuffled BWT algorithm, each row of $\mathcal{A}$ contains the combined partial alignments for reads in g groups. Without loss of generality, we focus on trying to identify read r_1 using the information only in *CloudF* (or only in *CloudL*). Suppose read r_1 belongs to read group 1. Because the partial (candidate) alignments for read r_1 are combined with the partial alignments for corresponding reads in groups $2-g$ in $\mathcal{A}[1]$ after each iteration of the Two-Cloud Shuffled BWT algorithm, it may not be possible to uniquely deduce read r_1 (or any of the corresponding reads in groups $2-g$) in terms of the symbols W, X, Y, and Z. We explain this in detail using the following example.

Example 3. Suppose $g = 2$ groups, reads r_1 and r_2 are in groups 1 and 2, and read r_2 is mapped to read r_1. Additionally, suppose the last three bases of r_1 are WXW and the last two bases of r_2 are WZY in terms of the symbols W, X, Y, and Z. Note that *CloudF* does not know the true mapping from $\{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$ to $\{A, C, T, G\}$. The partial alignments $\mathcal{A}[1]$ will contain indices of F corresponding to both symbols W and Y. Since *CloudF* has no way of deducing which symbol corresponds to which read (r_1 or r_2), read r_1 has the two possible "super-patterns" W and Y in terms of the symbols W, X, Y, and Z. Once the next base (from the right) of reads r_1 and r_2 are processed by the Two-Cloud Shuffled BWT alignment algorithm in Algorithm 3, the list of partial alignments $\mathcal{A}[1]$ will contain indices of F corresponding to the symbols X and Z. Consequently, read r_1 will have the four possible super-patterns XW, XY, ZW, and ZY after this iteration of our alignment algorithm. Next, after the third-last base of reads r_1 and r_2 are processed by our algorithm, the list of partial alignments $\mathcal{A}[1]$ will only contain rows corresponding to the symbol W since the third-last base of both r_1 and r_2 is W. Consequently, read r_1 will only have the four possible super-patterns WXW, WXY, WZW, and WZY after this iteration. Note that the number of patterns associated with each super-pattern is $O(1)$ because there are only $O(1)$ possible bijections from $\{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$ to $\{A, C, T, G\}$ (even after excluding duplicate patterns arising from different super-patterns).

In what follows, we study how the number of super-patterns and patterns of a read grow as more of its bases are processed by the alignment algorithm.

Lemma 4. Let S_j denote the number of possible super-patterns for read r_1 in terms of the symbols W, X, Y, and Z after j of its bases have been processed by the alignment algorithm 3. Let $N_j \in \{1, 2, 3, 4\}$ denote the number of unique symbols in $\mathcal{A}[1]$ after j bases of r_1 are processed. We have

$$S_{j+1} = N_{j+1} S_j, \quad \forall j \in \{0, 1, \ldots, l-1\},$$

with $S_0 := 1$. Consequently, the number of possible patterns P_j for r_1 (in terms of A, C, T, and G) after j bases of it are processed is given by

$$P_{j+1} = O(1) N_{j+1} P_j, \quad \forall j \in \{0, 1, \ldots, l-1\},$$

with $P_0 := 1$.

As the number of groups $g \geq 2$ increases, so does the probability that $N_j > 1$ for each $j \in \{1, \ldots, l\}$. Lemma 4 then clearly implies that the number of super-patterns (and patterns) is expected to grow exponentially with the number of bases of a read processed. Therefore, the number of patterns and super-patterns for a read will be very large with high probability for reads of moderate length (≥ 100 bps). We formalize this in our next result.

Lemma 5. Let N_j be defined as in Lemma 4. Suppose each read is a random string of length l from the alphabet $\{A, C, T, G\}$ and the $g \geq 2$ read groups are formed uniformly at random. Then

$$\mathbb{P}(N_j > 1) = 1 - \frac{1}{4^{g-1}}.$$

Proof. We have

$$\begin{aligned}\mathbb{P}(N_j > 1) &= 1 - \mathbb{P}(N_j = 1)\\ &= 1 - \frac{4}{4^g} = 1 - \frac{1}{4^{g-1}},\end{aligned}$$

where $\mathbb{P}(N_j = 1) = \frac{4}{4^g}$ follows from the fact that there are 4 different situations in which all groups can correspond to the same symbol (W, X, Y, or Z), whereas there are 4^g possible g-tuples of group symbols. □

Next, we identify conditions under which a candidate bijection from $\{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$ to $\{A, C, T, G\}$ can be eliminated using pattern analysis on the reference template.

Lemma 6. Consider the setting with one template chunk. Let $\psi : \{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\} \to \{A, C, T, G\}$ denote a candidate bijection from $\{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$ to $\{A, C, T, G\}$. Let $\mathcal{SP} := \{\sigma_1 \sigma_2 \cdots \sigma_j\}$, with $\sigma_k \in \{\texttt{W}, \texttt{X}, \texttt{Y}, \texttt{Z}\}$, denote the set of possible super-patterns for read r_1 after j of its bases are processed by the alignment algorithm. Furthermore, let $\mathcal{P} := \{s_1 s_2 \cdots s_j : s_k = \psi(\sigma_k)\}$, denote the corresponding set of possible patterns for read r_1 when the mapping ψ is applied to the set of super-patterns $\mathcal{SP}$. Then, we can eliminate the bijection ψ if the number of times *every* pattern $s_1 s_2 \cdots s_j$ in $\mathcal{P}$ occurs in the public reference template is different than the number of partial candidate alignments observed in $\mathcal{A}[1]$ for each of the g groups after j bases of r_1 are processed.

Although we can expect the counts of k-mers in the reference template to be unique, the following result indicates that it might still not be easy for *CloudF* to eliminate a specific bijection ψ. This is because we divide the template into several chunks, re-number and re-organize the reads into different groups, and generate different read order and super-reads for each individual template chunk.

Lemma 7. Consider the setting with multiple template chunks. Let $\psi : \{\mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\} \rightarrow \{A, C, T, G\}$ denote a candidate bijection from $\{\mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$ to $\{A, C, T, G\}$. Let $\mathcal{SP} := \{\sigma_1\sigma_2 \cdots \sigma_j\}$, with $\sigma_k \in \{\mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\}$, denote the set of possible super-patterns for read r_1 in a particular template chunk after j of its bases are processed by the alignment algorithm. Furthermore, let $\mathcal{P} := \{s_1s_2 \cdots s_j : s_k = \psi(\sigma_k)\}$, denote the corresponding set of possible patterns for read r_1 when the mapping ψ is applied to the set of super-patterns $\mathcal{SP}$. Then, we can eliminate the bijection ψ if the number of times *every* pattern $s_1s_2 \cdots s_j$ in $\mathcal{P}$ occurs in the entire public reference template is *less than* the number of partial candidate alignments observed in $\mathcal{A}[1]$ for any of the g groups after j bases of r_1 are processed.

While Lemma 7 focuses on the case of exact alignments, it can be easily adapted to the setting where some of the reads in a cluster have inexact alignments in a particular template chunk. When the counts of different k-mers in the reference template are on the same order of magnitude and the number of template chunks is on the order of 10, it becomes less likely that *CloudF* can use Lemma 7 to eliminate a particular bijection. Along with the fact that the number of super-patterns grows exponentially with high probability, this provides argumentation for why the individual contents of the reads (and therefore, SNPs) cannot be deduced by *CloudF* easily.

Security Analysis for CloudL. We store the last column array L, the linking array F', and the set of super-reads in *CloudL*. Because each super-read is constructed using a restrictive random permutation of the concatenation of the reads in its group, *CloudL* cannot readily deduce the contents of each individual read. This ensures privacy for the set of reads and possible SNPs. If a read contains a SNP, *CloudF* may send an empty set of indices of F for its corresponding read group during a particular iteration. Since *CloudL* does not know the read order and the read number corresponding to this iteration, it cannot deduce the read in which a SNP is present. For the same reason, *CloudL* cannot deduce the reads even though it has the knowledge of which indices of F correspond to each read group at any particular iteration. Furthermore, it is not possible to find out the actual alignment position of any of the reads as we keep the shuffled suffix array SA private from both *CloudF* and *CloudL*.

7 EMPIRICAL RESULTS

7.1 Setup of Computational Experiments

We implemented our methods for preprocessing the reads and the reference genome on the secure client computer, alignment across both clouds, and postprocessing on the client in Python 3.8. We run our algorithm using a high throughput cluster computing facility. We used the cluster computing facility to record accuracy, network bandwidth and memory footprints, run times for our algorithm. The communication between the two clouds are implemented using Message Passing Interface (MPI). MPI is a portable message-passing standard designed to function on parallel computing architectures. We simulated a total of $150K$ $150bp$ and $350bp$ PacBio reads from all the chromosomes of the human genome [67] using SimLoRD [70] read simulation software with 0%, 1% (0.1% indels and 0.9% SNPs), and 2% (0.2% indels and 1.8% SNPs) error probabilities. The rest of SimLoRD's parameters are set to default. We use Chromosome 11 as a representative chromosome to report our results and of the $100K$ reads, 4000 align in chromosome 11. We divide the alignment into a set of jobs, each of our jobs involves a template chunk of size $1M$ and a read batch with 1000 reads. Our algorithm uses the two clouds only for alignment. Our client is also located in the cluster computing facility. The client performs all the preprocessing steps and transfers our index structures to the MPI processes. We analyze the scalability of our algorithm, empirically compare our approach with the state-of-the-art aligners Bowtie [40], BWA [43], Minimap2 [42], and Balaur [60], and empirically assess the security of our algorithm on various test instances.

7.2 Discussion of Results

Scaling Analysis: We evaluate the effect of varying template length, read length, number of reads, and number of read groups on the time to preprocess the reference template and the reads. Figure 3 shows that the template and read preprocessing times scale linearly with the template and read lengths and the number of reads. It also demonstrates that the read preprocessing times decrease with increasing read groups. Note that the alignment times do not change significantly with increasing number of read groups because the total number of operations executed by the alignment algorithm remains the same for any number of read groups. Our alignment algorithm is also massively parallelizable. We can run alignment for multiple template chunks and read batches in parallel and retrieve alignments with full accuracy and privacy. With the power of cloud computing, parallelization and little communication between the clouds, our algorithm can achieve rapid alignment speeds and high performance.

Run Times and Memory Footprint: We compare the run times of our preprocessing step on the *Client* with state-of-the-art read aligners such as Bowtie [40], BWA [43] and Minimap2 [42] and the private read aligner Balaur [60]. Table 1 demonstrates that the amount of client work done by our algorithm is either lower or comparable to the existing state-of-the-art approaches. Note that the run time for Balaur preprocessing is extrapolated to Chromosome 11 based on random template substrings of length ranging from $1M$ to $10M$ (we could not index Chromosome 11 directly using Balaur since its execution was killed by the system). The run time for each of our alignment jobs took about three and half minutes on the cluster. This means that aligning to the entire human genome may require a couple of days of runtime. However, our implementation is only a prototype that is not fully optimized. For example, unlike Balaur, we have not considered any preprocessing that yields an approximate region of the template to which each read aligns. We align each read to the whole reference template instead of relatively much small regions of it. By adding additional

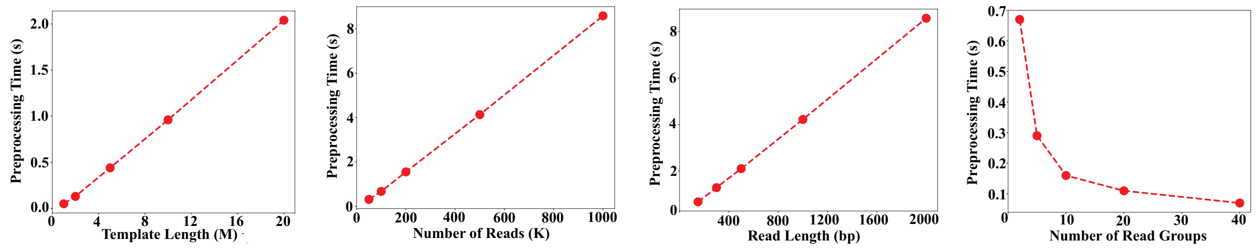

Figure 3: Preprocessing times for varying template and read dimensions. Default parameters across the plots are: template length = $10M$**, number of reads** = $100K$**, read length** = 150**bp, and number of read groups** = 2**.**

preprocessing step to our algorithm to yield approximate alignment regions for each read, the run times may reduce significantly. For example, if the alignment is being done to determine genetic risks based on a set of SNPs, then one can align reads only to parts of the genome around the SNP locations and thereby disregard large parts of the genome to reduce the runtime. There are many other possible strategies for reducing the genomic regions to which a read may have to be aligned. In addition, in our current implementation, the time for processing groups takes about 40% of the total time in CloudL. Most of these computations can be pruned and/or parallelized, but not yet implemented in the current prototype. This will also significantly reduce the runtime for aligning to the full human genome. In addition, in our current implementation, the time for processing groups takes about 40% of the total time in *CloudL*. A lot of this computation can be pruned and/or parallelized. If run sequentially, the total CPU time of our prototype is several days and thus not scalable to full human genome. Our algorithm's memory footprint is also less or comparable to existing cloud-bases private read aligners. For example, Chen et al. [15] requires about 6TB for storing its index files for the entire genome. Balaur [60] takes about 3.7GB to store index files for Chromosome 11 of length 135M. Our algorithm sends only 1GB of index files to the clouds for Chromosome 11. State-of-the art aligners such a Bowtie and BWA without privacy have a peak memory footprint of 2.3 GB for single-end mapping and about 3 GB for paired-end mapping of the entire human genome [43]. These memory footprints are similar to other approaches and are small for the memory availability of modern day cloud computing platforms.

Network Latency. In our experimental setup and implementation with MPI, the latency was an average of 10 millisenconds for transferring 100K indices or 250KB of data. If we run the experiments over a wide-area network, the network latencies may be larger. However, since all the jobs in our algorithm are independent and since there are a large number of jobs, even a large network latency can be hidden by interleaving computations from multiple jobs.

Correctness Analysis: Out of the true alignments in Chromosome 11, for 1% error rates, our algorithm retrieves 99.7% of the original alignments for both read lengths 150bp and 350bp. Note that our algorithm guarantees recovery of all alignments with 1 SNP when the reference template contains no exact alignments. Our algorithm

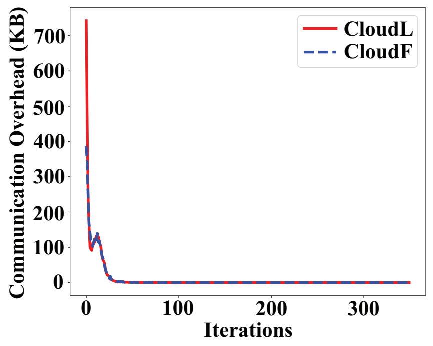

Figure 4: Average number of indices (averaged over $1K$ **150 bp reads and chunks of Chromosome** 11**) transferred between *CloudL* to *CloudF* (solid line) and *CloudF* to *CloudL* (dashed line) during the course of the alignment algorithm.**

retrieves 98.8% and 99.1% of the original alignments for 150bp and 350bp reads simulated with 2% error rates. Out of the remaining reads, only a handful of them could have aligned in Chromosome 11.

Client-Cloud Communication: For Chromosome 11 and $100K$ reads, our algorithm transfers less than 1GB data in total to both clouds. The data the client receives from the clouds is a few kilo bytes. Data transfer costs can be further reduced if tailored approaches are used to store the index files. With good network bandwidth and throughput, large index files can be transferred within seconds.

Communication Between Clouds: Figure 4 plots the number of indices transferred between *CloudF* and *CloudL* during the course of Algorithm 3. For each job, 330K indices on average per read is communicated by each cloud to the other during the execution of our alignment algorithm. Each index can be represented using 20 bits for template size of 1M. The total network usage in bits is the integral of the curve in the Figure 4 times 20. This translates to roughly 6.5MB of communication per read per job without any compression mechanisms. Therefore, for the 1000 reads in each job, the total communication from each cloud is on the order of 6.5GB.

Table 1: Comparison to state-of-the-art aligners for template size of 1M from chromosome 11 and 1000K reads of length 150. The alignment accuracies for Bowtie, BWA-MEM, Balaur are Q10% scores.

Read aligner	Bowtie	BWA-MEM	Minimap2	Balaur	Our algorithm
Preprocessing Times	7.37s	8.5s	0.7s	7.2s	1.5s
Client Communication	NA	NA	NA	370MB	100MB
Alignment Runtimes	7.5s	0.5s	2s	10.0s	208s
Alignment Accuracy (1% error)	95.9%	96.8%	95.2%	96.0%	99.7%
Alignment Accuracy (2% error)	94.5%	96.8%	96.2%	94.4%	98.8%

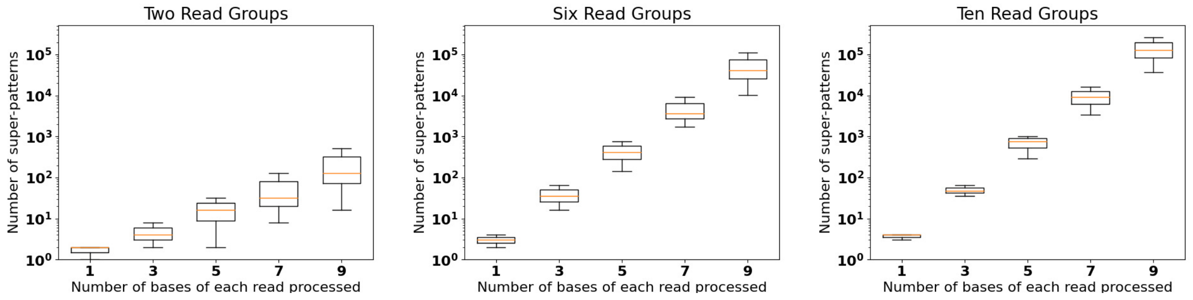

Figure 5: Number of super-patterns for increasing number of bases of each read processed

Note that the amount of data transferred between the two clouds reduces drastically during the course of our alignment algorithm. Since communication between *CloudF* and *CloudL* is curtailed for inexact alignments, our algorithm does not yield significant junk alignments. We allowed up to one inexact alignment for each read which results in spike in the number of indices transferred when this step is triggered and is shown in Figure 4. Note that the number of lookups of the bases in the last column L and the number of group encodings follow similar trends as in Figure 4. The number of lookups in our algorithm increases only by a constant factor compared to read alignment using the traditional FM-index structure.

Security Analysis: We use a sample reference template of size $4M$ and 120 randomly sampled reads for our security analysis. We divide the template into 10 equal chunks and align the 120 reads to only the first template chunk. Figure 5 shows that the number of super-patterns for each read increases exponentially with its number of bases processed, as noted in Lemma 4. We check the conditions in Lemma 7 to determine if any of the 24 possible bijections can be eliminated for read group counts 2, 4, 6, 8, and 10. We were not able to eliminate any bijections using Lemma 7 for the cases with 4, 6, 8, and 10 read groups when the last 10 bases of each read was processed by our alignment algorithm. However, for group size 2, we could rule out up to 20 bijections before the last 10 bases of each read was processed. Note that our algorithm remains fully private until all but one bijection is eliminated—in which case *CloudF* can determine F uniquely. However, even in this case, *CloudF* cannot determine the individual contents of the reads uniquely and can only identify a list of candidate patterns for each read.

It gets harder to eliminate candidate bijections using Lemma 7 when the number of bases processed for each read increases. Firstly, with increasing pattern lengths, the computational cost for *CloudF* to perform pattern analysis increases. Secondly, patterns of larger sizes occur with less frequency than patterns with smaller sizes. Along with the fact that the number of partial alignments for each read decreases significantly with increasing number of bases processed, this ensures that it becomes harder to eliminate candidate bijections when the iteration count of the alignment algorithm increases. We also expect pattern analysis to be harder when the template size and the number of template chunks increase.

8 CONCLUSION AND FUTURE WORK

Conventional security algorithms may be inadequate when there is a need for secure read alignment due to privacy risks posed by the valuable information a human genome can provide. This becomes a major concern for researchers and precludes them from using public clouds resources. Many of the existing approaches for private read alignment compromise correctness of the algorithm to provide security. In this work, we present a workable approach to the private read alignment problem using two non-colluding semi-honest clouds. Our approach enables researchers to harness the computation and storage capabilities of cloud computing platforms without having to trade-off individuals' privacy. Our algorithm also produces read alignments with full accuracy akin to modern alignment tools. As part of future work, we would like to investigate approaches to optimize the way we store our index structures, including our shuffled BWT data structure, to improve the memory footprints, speed and efficiency of our private read alignment algorithm.

ACKNOWLEDGMENTS

This work was done in part with support from the University of Wisconsin-Madison and from the National Science Foundation Grant EFRI CEE 1933402. Initial computational experiments were run using the compute resources and assistance of the UW-Madison Center For High Throughput Computing (CHTC) in the Department of Computer Sciences. The CHTC is supported by UW-Madison, the Advanced Computing Initiative, the Wisconsin Alumni Research Foundation, the Wisconsin Institutes for Discovery, and the National Science Foundation, and is an active member of the OSG Consortium, which is supported by the NSF and the U.S. Department of Energy's Office of Science.

REFERENCES

[1] Mohamed Ibrahim Abouelhoda, Stefan Kurtz, and Enno Ohlebusch. 2004. Replacing suffix trees with enhanced suffix arrays. *Journal of Discrete Algorithms* 2, 1 (2004), 53–86.

[2] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. 2017. Secure approximation of edit distance on genomic data. *BMC Medical Genomics* 10, 2 (2017), 55–67.

[3] Stephen F Altschul, Warren Gish, Webb Miller, Eugene W Myers, and David J Lipman. 1990. Basic local alignment search tool. *Journal of Molecular Biology* 215, 3 (1990), 403–410.

[4] Stephen F Altschul, Thomas L Madden, Alejandro A Schäffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J Lipman. 1997. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. *Nucleic Acids Research* 25, 17 (1997), 3389–3402.

[5] Sebastian Angel, Hao Chen, Kim Laine, and Srinath Setty. 2018. PIR with compressed queries and amortized query processing. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 962–979.

[6] Sebastian Angel and Srinath Setty. 2016. Unobservable communication over fully untrusted infrastructure. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 551–569.

[7] Mikhail J Atallah, Florian Kerschbaum, and Wenliang Du. 2003. Secure and private sequence comparisons. In *Proceedings of the 2003 ACM Workshop on Privacy in the Electronic Society*. 39–44.

[8] Mikhail J Atallah and Jiangtao Li. 2005. Secure outsourcing of sequence comparisons. *International Journal of Information Security* 4, 4 (2005), 277–287.

[9] Erman Ayday, Jean Louis Raisaro, and Jean-Pierre Hubaux. 2013. Personal use of the genomic data: Privacy vs. storage cost. In *2013 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2723–2729.

[10] Joshua Baron, Karim El Defrawy, Kirill Minkovich, Rafail Ostrovsky, and Eric Tressler. 2013. 5PM: Secure pattern matching. *Journal of Computer Security* 21, 5 (2013), 601–625.

[11] M Bhuyan, V Deka, and S Bordoloi. 2013. Burrows Wheeler Based Data Compression and Secure Transmission. *International Journal of Research in Engineering and Technology* 2, 2 (2013).

[12] Marcelo Blatt, Alexander Gusev, Yuriy Polyakov, Kurt Rohloff, and Vinod Vaikuntanathan. 2020. Optimized homomorphic encryption solution for secure genome-wide association studies. *BMC Medical Genomics* 13, 7 (2020), 1–13.

[13] Michael Burrows and David Wheeler. 1994. A block-sorting lossless data compression algorithm. In *Digital SRC Research Report*. Citeseer.

[14] Barbara Cheifet. 2019. Where is genomics going next? *Genome Biology* 20, 1 (2019), 1–8.

[15] Yangyi Chen, Bo Peng, XiaoFeng Wang, and Haixu Tang. 2012. Large-Scale Privacy-Preserving Mapping of Human Genomic Sequences on Hybrid Clouds. In *Network and Distributed System Security (NDSS) Symposium*. 1–18.

[16] Vinicius V Cogo, Alysson Bessani, Francisco M Couto, and Paulo Verissimo. 2015. A high-throughput method to detect privacy-sensitive human genomic data. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*. 101–110.

[17] Anthony J Cox, Markus J Bauer, Tobias Jakobi, and Giovanna Rosone. 2012. Large-scale compression of genomic sequence databases with the Burrows–Wheeler transform. *Bioinformatics* 28, 11 (2012), 1415–1419.

[18] Wei Dai, Yarkın Doröz, and Berk Sunar. 2015. Accelerating SWHE based PIRs using GPUs. In *International Conference on Financial Cryptography and Data Security*. Springer, 160–171.

[19] Emiliano De Cristofaro, Sky Faber, and Gene Tsudik. 2013. Secure genomic testing with size-and position-hiding private substring matching. In *Proceedings of the 12th ACM workshop on Workshop on privacy in the electronic society*. 107–118.

[20] C Peter Devadoss and B Sankaragomathi. 2019. Near lossless medical image compression using block BWT–MTF and hybrid fractal compression techniques. *Cluster Computing* 22, 5 (2019), 12929–12937.

[21] Changyu Dong and Liqun Chen. 2014. A fast single server private information retrieval protocol with low communication cost. In *European Symposium on Research in Computer Security*. Springer, 380–399.

[22] Maria Fernandes, Jérémie Decouchant, Marcus Völp, Francisco M Couto, and Paulo Esteves-Verissimo. 2019. DNA-SeAl: sensitivity levels to optimize the performance of privacy-preserving DNA alignment. *IEEE Journal of Biomedical and Health Informatics* 24, 3 (2019), 907–915.

[23] Paolo Ferragina and Giovanni Manzini. 2000. Opportunistic data structures with applications. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*. IEEE, 390–398.

[24] Joseph Yossi Gil and David Allen Scott. 2012. A bijective string sorting transform. *arXiv preprint arXiv:1201.3077* (2012).

[25] Alan E Guttmacher and Francis S Collins. 2003. Welcome to the genomic era. *New England Journal of Medicine* 349, 10 (2003), 996–998.

[26] Melissa Gymrek, Amy L McGuire, David Golan, Eran Halperin, and Yaniv Erlich. 2013. Identifying personal genomes by surname inference. *Science* 339, 6117 (2013), 321–324.

[27] Steve Hoffmann, Christian Otto, Stefan Kurtz, Cynthia M Sharma, Philipp Khaitovich, Jörg Vogel, Peter F Stadler, and Jörg Hackermüller. 2009. Fast mapping of short sequences with mismatches, insertions and deletions using index structures. *PLoS Computational Biology* 5, 9 (2009), e1000502.

[28] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. 2008. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy (SP)*. IEEE, 216–230.

[29] Hui Jiang and Wing Hung Wong. 2008. SeqMap: mapping massive amount of oligonucleotides to the genome. *Bioinformatics* 24, 20 (2008), 2395–2396.

[30] W James Kent. 2002. BLAT—the BLAST-like alignment tool. *Genome Research* 12, 4 (2002), 656–664.

[31] Miran Kim and Kristin Lauter. 2015. Private genome analysis through homomorphic encryption. In *BMC Medical Informatics and Decision Making*, Vol. 15. BioMed Central, 1–12.

[32] Miran Kim, Yongsoo Song, Xiaoqian Jiang, and Arif Harmanci. 2021. SHiMMer: Privacy-Aware Alignment of Genomic Sequences with Secure and Efficient Hidden Markov Model Evaluation. *Research Square* (2021).

[33] Manfred Kufleitner. 2009. On bijective variants of the Burrows-Wheeler transform. *arXiv preprint arXiv:0908.0239* (2009).

[34] M Oğuzhan Külekci. 2012. On scrambling the Burrows–Wheeler transform to provide privacy in lossless compression. *Computers & Security* 31, 1 (2012), 26–32.

[35] Stefan Kurtz, Adam Phillippy, Arthur L Delcher, Michael Smoot, Martin Shumway, Corina Antonescu, and Steven L Salzberg. 2004. Versatile and open software for comparing large genomes. *Genome Biology* 5, 2 (2004), 1–9.

[36] Tak Wah Lam, Wing-Kin Sung, Siu-Lung Tam, Chi-Kwong Wong, and Siu-Ming Yiu. 2008. Compressed indexing and local alignment of DNA. *Bioinformatics* 24, 6 (2008), 791–797.

[37] Christoph Lambert, Maria Fernandes, Jérémie Decouchant, and Paulo Esteves-Verissimo. 2018. MaskAl: Privacy Preserving Masked Reads Alignment using Intel SGX. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. 113–122. https://doi.org/10.1109/SRDS.2018.00022

[38] Ben Langmead. 2010. Aligning short sequencing reads with Bowtie. *Current Protocols in Bioinformatics* 32, 1 (2010), 11–7.

[39] Ben Langmead and Steven L Salzberg. 2012. Fast gapped-read alignment with Bowtie 2. *Nature Methods* 9, 4 (2012), 357–359.

[40] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg. 2009. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology* 10, 3 (2009), 1–10.

[41] Kristin Lauter, Adriana López-Alt, and Michael Naehrig. 2014. Private computation on encrypted genomic data. In *International Conference on Cryptology and Information Security in Latin America*. Springer, 3–27.

[42] Heng Li. 2018. Minimap2: pairwise alignment for nucleotide sequences. *Bioinformatics* 34, 18 (2018), 3094–3100.

[43] Heng Li and Richard Durbin. 2009. Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics* 25, 14 (2009), 1754–1760.

[44] Heng Li and Richard Durbin. 2010. Fast and accurate long-read alignment with Burrows–Wheeler transform. *Bioinformatics* 26, 5 (2010), 589–595.

[45] Heng Li and Nils Homer. 2010. A survey of sequence alignment algorithms for next-generation sequencing. *Briefings in bioinformatics* 11, 5 (2010), 473–483.

[46] Heng Li, J Ruan, and R Durbin. 2008. MAQ: Mapping and Assembly with Qualities. *Version 0.6* 3 (2008), 508.

[47] Ruiqiang Li, Yingrui Li, Karsten Kristiansen, and Jun Wang. 2008. SOAP: short oligonucleotide alignment program. *Bioinformatics* 24, 5 (2008), 713–714.

[48] Ruiqiang Li, Chang Yu, Yingrui Li, Tak-Wah Lam, Siu-Ming Yiu, Karsten Kristiansen, and Jun Wang. 2009. SOAP2: an improved ultrafast tool for short read alignment. *Bioinformatics* 25, 15 (2009), 1966–1967.

[49] Tobias P Loka, Simon H Tausch, Piotr W Dabrowski, Aleksandar Radonić, Andreas Nitsche, and Bernhard Y Renard. 2018. PriLive: privacy-preserving real-time filtering for next-generation sequencing. *Bioinformatics* 34, 14 (2018), 2376–2383.

[50] Bradley Malin and Latanya Sweeney. 2000. Determining the identifiability of DNA database entries. In *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 537.

[51] Bradley Malin and Latanya Sweeney. 2001. Re-identification of DNA through an automated linkage process. In *Proceedings of the AMIA Symposium*. American Medical Informatics Association, 423.

[52] Giovanni Manzini. 1999. The Burrows-Wheeler transform: theory and practice. In *International Symposium on Mathematical Foundations of Computer Science*. Springer, 34–47.

[53] Guillaume Marçais, Arthur L Delcher, Adam M Phillippy, Rachel Coston, Steven L Salzberg, and Aleksey Zimin. 2018. MUMmer4: A fast and versatile genome alignment system. *PLoS Computational Biology* 14, 1 (2018), e1005944.

[54] Colin Meek, Jignesh M Patel, and Shruti Kasetty. 2003. Oasis: An online and accurate technique for local-alignment searches on biological sequences. In *Proceedings of the 2003 VLDB Conference*. Elsevier, 910–921.

[55] Michael L Metzker. 2010. Sequencing technologies—the next generation. *Nature Reviews Genetics* 11, 1 (2010), 31–46.

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

[57] Saul B Needleman and Christian D Wunsch. 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology* 48, 3 (1970), 443–453.

[58] Zemin Ning, Anthony J Cox, and James C Mullikin. 2001. SSAHA: a fast search method for large DNA databases. *Genome Research* 11, 10 (2001), 1725–1729.

[59] Dale R Nyholt, Chang-En Yu, and Peter M Visscher. 2009. On Jim Watson's APOE status: genetic information is hard to hide. *European Journal of Human Genetics* 17, 2 (2009), 147–149.

[60] Victoria Popic and Serafim Batzoglou. 2017. A hybrid cloud read aligner based on MinHash and kmer voting that preserves privacy. *Nature Communications* 8, 1 (2017), 1–7.

[61] Collin Preston, Ziya Arnavut, and Basar Koc. 2015. Lossless compression of medical images using Burrows-Wheeler transformation with inversion coder. In *37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. IEEE, 2956–2959.

[62] Gianluca Reali, Mauro Femminella, Emilia Nunzi, and Dario Valocchi. 2018. Genomics as a service: A joint computing and networking perspective. *Computer Networks* 145 (2018), 27–51.

[63] Somak Roy, Christopher Coldren, Arivarasan Karunamurthy, Nefize S Kip, Eric W Klee, Stephen E Lincoln, Annette Leon, Mrudula Pullambhatla, Robyn L Temple-Smolkin, Karl V Voelkerding, et al. 2018. Standards and guidelines for validating next-generation sequencing bioinformatics pipelines: a joint recommendation of the Association for Molecular Pathology and the College of American Pathologists. *The Journal of Molecular Diagnostics* 20, 1 (2018), 4–27.

[64] Stephen M Rumble, Phil Lacroute, Adrian V Dalca, Marc Fiume, Arend Sidow, and Michael Brudno. 2009. SHRiMP: accurate mapping of short color-space reads. *PLoS Computational Biology* 5, 5 (2009), e1000386.

[65] Mikael Salson, Thierry Lecroq, Martine Léonard, and Laurent Mouchard. 2009. A four-stage algorithm for updating a Burrows–Wheeler transform. *Theoretical Computer Science* 410, 43 (2009), 4350–4359.

[66] Frederick Sanger, Steven Nicklen, and Alan R Coulson. 1977. DNA sequencing with chain-terminating inhibitors. *Proceedings of the National Academy of Sciences* 74, 12 (1977), 5463–5467.

[67] Valerie A. Schneider, Tina Graves-Lindsay, Kerstin Howe, Nathan Bouk, Hsiu-Chuan Chen, Paul A. Kitts, Terence D. Murphy, Kim D. Pruitt, Françoise Thibaud-Nissen, Derek Albracht, Robert S. Fulton, Milinn Kremitzki, Vince Magrini, Chris Markovic, Sean McGrath, Karyn Meltz Steinberg, Kate Auger, Will Chow, Joanna Collins, Glenn Harden, Tim Hubbard, Sarah Pelan, Jared T. Simpson, Glen Threadgold, James Torrance, Jonathan Wood, Laura Clarke, Sergey Koren, Matthew Boitano, Heng Li, Chen-Shan Chin, Adam M. Phillippy, Richard Durbin, Richard K. Wilson, Paul Flicek, and Deanna M. Church. 2016. Evaluation of GRCh38 and de novo haploid genome assemblies demonstrates the enduring quality of the reference assembly. *bioRxiv* (2016). https://doi.org/10.1101/072116 arXiv:https://www.biorxiv.org/content/early/2016/08/30/072116.full.pdf

[68] Suyash S Shringarpure and Carlos D Bustamante. 2015. Privacy risks from genomic data-sharing beacons. *The American Journal of Human Genetics* 97, 5 (2015), 631–646.

[69] Temple F Smith and Michael S Waterman. 1981. Identification of common molecular subsequences. *Journal of Molecular Biology* 147, 1 (1981), 195–197.

[70] Bianca K Stöcker, Johannes Köster, and Sven Rahmann. 2016. SimLoRD: simulation of long read data. *Bioinformatics* 32, 17 (2016), 2704–2706.

[71] Doug Szajda, Michael Pohl, Jason Owen, Barry G Lawson, and Virginia Richmond. 2006. Toward a Practical Data Privacy Scheme for a Distributed Implementation of the Smith-Waterman Genome Sequence Comparison Algorithm. In *Network and Distributed System Security (NDSS) Symposium*.

[72] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 492–503.

[73] Chathura Widanage, Weijie Liu, Jiayu Li, Hongbo Chen, XiaoFeng Wang, Haixu Tang, and Judy Fox. 2021. HySec-Flow: Privacy-Preserving Genomic Computing with SGX-based Big-Data Analytics Framework. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 733–743.

[74] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th Annual Symposium on Foundations of Computer Science (sfcs 1986)*. IEEE, 162–167.

[75] Yongan Zhao, Xiaofeng Wang, and Haixu Tang. 2018. A secure alignment algorithm for mapping short reads to human genome. *Journal of Computational Biology* 25, 6 (2018), 529–540.