sciendo

Proceedings on Privacy Enhancing Technologies ; 2018 (4):104–124

Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin

Privacy-Preserving Search of Similar Patients in Genomic Data

Abstract: The growing availability of genomic data holds great promise for advancing medicine and research, but unlocking its full potential requires adequate methods for protecting the privacy of individuals whose genome data we use. One example of this tension is running Similar Patient Query on remote genomic data: In this setting a doctor that holds the genome of his/her patient may try to find other individuals with "close" genomic data, and use the data of these individuals to help diagnose and find effective treatment for that patient's conditions. This is clearly a desirable mode of operation. However, the privacy exposure implications are considerable, and so we would like to carry out the above "closeness" computation in a privacy preserving manner.

In this work we put forward a new approach for highly efficient secure computation for computing an approximation of the Similar Patient Query problem. We present contributions on two fronts. First, an approximation method that is designed with the goal of achieving efficient private computation. Second, further optimizations of the two-party protocol. Our tests indicate that the approximation method works well, it returns the exact closest records in 98% of the queries and very good approximation otherwise. As for speed, our protocol implementation takes just a few seconds to run on databases with thousands of records, each of length thousands of alleles, and it scales almost linearly with both the database size and the length of the sequences in it. As an example, in the datasets of the recent iDASH competition, after a one-time preprocessing of around 12 seconds, it takes around a second to find the nearest five records to a query, in a size-500 dataset of length-3500 sequences. This is 2-3 orders of magnitude faster than using state-of-the-art secure protocols with existing edit distance algorithms.

Keywords: Genomic privacy, cryptographic protocols, edit-distance

DOI 10.1515/popets-2018-0034

Received 2018-02-28; revised 2018-06-15; accepted 2018-06-16.

Gilad Asharov: Cornell Tech, NY. asharov@cornell.edu.

Shai Halevi: IBM Research, NY. shaih@alum.mit.edu.

Yehuda Lindell: Bar-Ilan University, Israel. Yehuda.Lindell@biu.ac.il.

Tal Rabin: IBM Research, NY. talr@us.ibm.com.

1 Introduction

Consider the task of a medical doctor who wants to compare a patient's DNA against a remote genomic database, e.g., to determine the patient's pre-disposition to various medical conditions. The database contains a list of individual genome sequences, each labeled with the medical conditions of that person. The doctor needs to find the few individuals in the database whose genome sequence (in the relevant segment) most resembles that of the patient, and learn the medical conditions of these individuals. We define resemblance (or closeness) in terms of edit distance.

This mode of operation is important for recognizing the subtype of cancer a patient might have. As each cancer is unique, comparing the genome of a patient will help pinpoint which mutations are behind the disease, and will also help to avoid painful treatments that would not cure the disease. According to the Global Alliance for Genomics and Health (GA4GH) institution [GA4], this mode of operation is expected to be used in a scale of hundreds of millions of patients within about a decade. Genome sequencing can help patients find out which treatments to select or avoid, and a more accurate prognosis and guidance to the most suitable clinical trial.

Sending the patient's DNA sequence to the database has severe privacy implication, thus, we would like to find an effective privacy preserving solution to this task. More specifically, we seek a solution to the following k-closest-match problem: We have a server that holds a database DB of genomic sequences $(S_1, \ldots, S_m)$, whose approximate length and position inside the human genome are known. The client (doctor) holds a sequence query Q, and wishes to find the identities of the k sequences in the DB that have the smallest edit distance from Q (where k is a public parameter). The goal is to perform this computation in a privacy-preserving manner (see Figure 1). We target security in the pres-

ence of an honest-but-curious adversary. Our work was motivated by the recent "secure genome analysis competition" run by iDASH [iDA16].

Unfortunately, the straightforward solution of computing the exact edit distance of Wagner-Fischer [WF74] (or even the near-linear-time approximation of Andoni and Onak [AO12]) using a secure-computation protocol would be prohibitively slow. Using state-of-the-art secure-computation techniques, such protocols would take many minutes (maybe even hours) per query, and certainly will not scale to large datasets and long sequences.

In this work, we develop an efficient privacy-preserving protocol for computing the k-match function from above. Our solution reduces the secure computation portion by following two principals: (1) Off-loading computation to the parties in the clear, even at the cost of increasing their local computation, and (2) Exploiting as much as possible the specific setting of the problem that we are solving. While targeting a somewhat restricted case study, the techniques that we develop can also be applied in other settings of computing k-closeness, as we will elaborate below.

These principles are demonstrated in our solution as follows. As for (1), our solution lets the client and server preprocess their respective inputs to the protocol. This preprocessing includes many edit-distance computations (linear in the size of the database), however, these are all carried out in the clear, and saves significant work for the secure computation portion. Moreover, this preprocessing is reusable and can be used by the server to answer an unlimited number of queries. As for (2), we develop an approximation function for the k-closeness problem that utilizes the application domain.

We show that the implementation of this approximation can handle databases with hundreds of records and sequences of length thousands of alleles. We ran our solution on a few databases of various sizes in regions featuring high divergence among individual genomes (variability of around 5%). Our experiments yielded excellent results both with respect to the accuracy and runtime (see §5 for details). Furthermore, our protocol was tested by external referees as part of the participation in the iDash competition in which we won the first place for the fastest runtime and for accuracy and efficiency.

Similar accuracy and performance results to the values that we report were confirmed. After a one-time preprocessing of around 12 seconds, our solution can answer many queries in about a second each, where each query consists of finding the five closest sequences

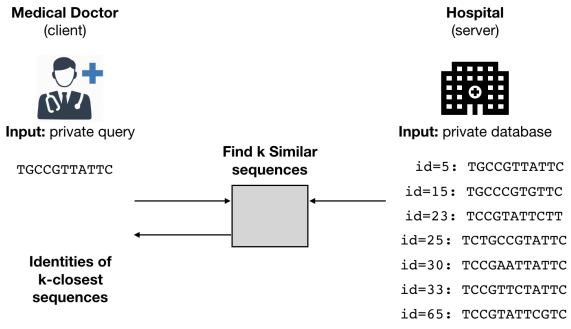

Fig. 1. *The problem statement. The client (Doctor) holds a single sequence, and it looks for the identities of the k-closest sequences in a remote database (Hospital). Privacy should be preserved for both the sequences in the database as well as the doctor's query.*

in a database of 500 sequences, each of length approximately 3470 nucleotides. Our solution scales well for larger databases, as we explain in §6. As for accuracy, our protocol returns the exact k-closest records in 98% of the queries, and returns a very close set to the exact one otherwise.

In summary, our contributions include the following:

- Developing new approximation function for edit distance that highly utilizes the application domain, that is, the distribution of genomic data.
- We design two-party protocol for computing this approximation function, and make further optimizations of the two-party protocol.

Applications beyond genomics. A large part of the paper is devoted for the development of the approximation function and understanding its properties. This approximation function heavily relies on the specific data distribution and is tailored to the application domain of finding similar patients in genomic data. In contrast, our secure protocol is suitable, as is, for other scenarios in which one has to compute distance of some query from a remote database. The protocol can also compute k-closest vectors for some query vector, where closeness is in terms of hamming distance, as long as each coordinate is over a small alphabet. This task has many applications, such as finding closest codeword for a given string, matching biometrics in a remote database, detecting abnormality in network logs, finding similar patients in structural medical database, and more.

Related work. The most relevant previous work is that of Wang et al. [WHZ+15], and the relevant concurrent works are those of Zhu and Huang [ZH17], and Al Aziz, Alhadid and Mohammed [AAM17]. All works deal with a similar problem to ours—computing edit-distances

while targeting genomic applications. [WHZ+15] targets regions of the genome with smaller divergence and is not sensitive enough to approximate well the distances in regions with higher diversity as addressed in this paper. Both concurrent works achieve significantly slower running times. We elaborate on these works in Appendix A.

Jha et al. proposed in [JKS08] some techniques for secure edit distance using garbled circuits, and shows that the overhead is acceptable only for small strings. (For example, handling 200-character strings takes about 2GB of bandwidth.) Using some further optimizations, they showed that 500-character string instances can be computed in almost an hour. Computing edit distance is also a common benchmark for analyzing improvements in general secure computation techniques and frameworks (see [HEKM11, HSE+11, ALSZ13], to state a few). These works compute accurate edit distance, and do not utilize the specific input distribution of genomic data.

Recent years saw a large body of work on using secure computation protocols for genomic data, some surveys include [NAC+15, ABOcS15].

Security implications of computing an approximation. Feigenbaum et al. [FIM+01] observed that computing an approximated version of a function may have security implications, in that the approximated version may leak information which is not revealed by the exact version. This concern applies to our solution, as well as to other works that compute approximation (e.g., [WHZ+15, AAM17]). For instance, when asking for the closest-5 sequences and the exact result returns ids {6,25,88,192,994}, our approximating protocol might return {6,25,97,192,994}, revealing information about patient 97 that was not supposed to be revealed by the exact computation. We elaborate on this in §7.

2 Overview

We develop a new (approximate) edit distance algorithm with the goal of achieving efficient private computation. In the following, we overview the ideas behind our approximation function.

The approximation function. We develop an efficient approximation algorithm that utilizes the distribution of genomic data. We heuristically expect (and empirically verify) that our algorithm provides an excellent approximation of the desired functionality.

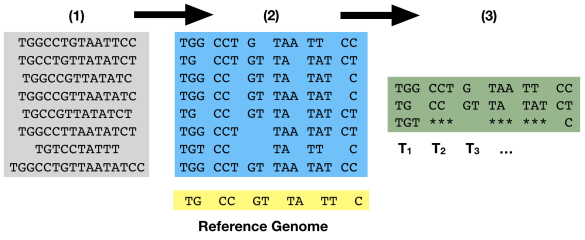

Fig. 2. *Flow of the preprocessing of the database at the server side. (1) The database. (2) The server compares all sequences to the reference genome, and breaks each sequence into blocks. (3) The number of different values in each block is small. The client has to compare its query to few values (in this contrived example, $v \leq 3$). *** denotes a fake string, and is used for padding.*

We first replace the edit-distance function with a block-wise approximation of it. Using a specially tailored method (that we describe below) we break the query Q into n blocks $(Q_1, \ldots, Q_n)$, and similarly break each sequence S_i in the database into blocks $(S_{i,1}, \ldots, S_{i,n})$, where the blocks are very small (typically, no more than 15 letters). Denoting by ED the edit-distance function, we first define the approximation to be:

$$\mathsf{ApproxED}(Q, S_i) \approx \sum_{\ell=1}^{n} \mathsf{ED}(Q_\ell, S_{i,\ell}). \qquad (2.1)$$

This approximation alone reduces the cost significantly, from $O(|Q| \cdot |S_i|)$ to $O(|Q| + |S_i|)$, as computing the distance of $\mathsf{ED}(Q_\ell, S_{i,\ell})$ when $|Q_\ell|, |S_{i,\ell}|$ are is relatively cheap, and so the computation of $\mathsf{ApproxED}(Q, S_i)$ can be done in linear time. As answering the query requires computing the edit distance between Q and *many* sequences $S_1, \ldots, S_m$, and given that we are dealing with genomic data we can further optimize the run-time.

We observe that in this setting, each block position has only a few distinct values (such as {TT, AGT, AGG}) that actually appear in that location. That is, after breaking all sequences in the database into blocks, there are only few possible combinations for each location. To be more precise, for each ℓ (where $\ell = 1, ..., n$) the cardinality of the set of values $T_\ell = \{S_{i,\ell}\}_{i=1}^{m}$ is *much smaller* than m. In our test of public genomic datasets we only had $v = \max_\ell\{|T_\ell|\} \leq 10$ (even for a dataset of size 500). This means that hundreds of edit distances can be computed at the cost of computing $\mathsf{ED}(Q_\ell, S_{i,\ell})$ for only 10 values, and saves a considerable amount of the work. In addition, in almost all cases ($> 99\%$), the block Q_ℓ of the query is also one of the values in the set T_ℓ, and so the edit distance values are from the set $\mathsf{ED}(u, S_{i,\ell})$ for all $u \in T_\ell$. We utilize these facts in order to speedup the computation.

Let v be a known bound on the number of distinct values in each block. We denote the elements of the set T_ℓ as $(u_{\ell,1}, \ldots, u_{\ell,v})$; if the number of elements in T_ℓ is less than v, then dummy values are added (see Figure 2 for a demonstration). We define a bit variable $\chi_{\ell,j}$ that indicates whether or not $u_{\ell,j} = Q_\ell$, for $u_{\ell,j} \in T_\ell$. If the value Q_ℓ happens to be equal to one of the $u_{\ell,j}$'s, then for every $S_{i,\ell}$ we have

$$\mathsf{ED}(Q_\ell, S_{i,\ell}) = \sum_{j=1}^{v} \chi_{\ell,j} \cdot \mathsf{ED}(u_{\ell,j}, S_{i,\ell}). \tag{2.2}$$

Namely, in this case we can compute the values that are needed for Eq. (2.1) as a simple linear combination involving the (few) bits $\chi_{\ell,j}$ and the values $\mathsf{ED}(u_{\ell,j}, S_{i,\ell})$. Importantly, this means that the actual edit distance between Q_ℓ and the $S_{i,\ell}$ never has to be computed explicitly! That is, we approximate the edit distance between Q and S_i by computing

$$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot \mathsf{ED}(u_{\ell,j}, S_{i,\ell}). \tag{2.3}$$

The only part of this equation that depends on Q_ℓ is $\chi_{\ell,j}$, which is a simple equality comparison and so can be efficiently computed privately. We note that in the case where $Q_\ell \notin T_\ell$, the expression on the right-hand side of Eq. (2.2) is always zero, so these cases introduce more error to our approximation. Nevertheless, our empirical tests on real genomic data show that the effect of this added error is very minor (see Appendix D.2). Summing up, a query is solved by securely computing the approximate k-closest-match function, defined as:

$$\begin{aligned}\mathsf{ApproxClosest}_{k,m}(Q, \{S_1, \ldots, S_m\}) = i_1, \ldots, i_k, \\ \text{where } S_{i_1}, \ldots, S_{i_k} \text{ have the smallest} \\ \mathsf{ApproxED}(Q, S_i) \text{ values.}\end{aligned} \tag{2.4}$$

Ties are broken using the indexes i themselves. We remark that the actual output returned may be the indexes, or may be a label stating the medical conditions of the patients with the matching sequences.

As we will see, this function can be securely computed with extremely high efficiency, as many of the elements in this function can be computed locally in the clear by the owner of the dataset and the query holder (details below). This leaves the question of how to partition the sequence into blocks to yield not only run-time efficiency but also accuracy.

Partitioning into blocks. A crucial detail of our approximation is the method that we use to partition the sequences S_i and the query Q into blocks. The simplest possibility would be to partition them into fixed-length blocks, but this simplistic partitioning yields a poor approximation. For example, a small shift close to the beginning of the query (perhaps just inserting a single character) can lead to many misalignments in the consecutive blocks, causing this single error to be counted multiple times (see an example below).

To get a better partitioning method, we utilize specific features of our application domain,[1] specifically the existence of a public "reference genome" R. This reference genome was created by the "Reference Genome Consortium" with the purpose of being a representative of the human genome. As a result of its design principals it is somewhat close to both the query Q and the database sequences S_i. Our partitioning method begins with applying the simplistic partitioning above to the reference genome R, using a block-size parameter b. Then, each party separately aligns its input sequence(s) to the reference sequence (using the Wagner-Fischer algorithm [WF74]), and we use the fact that the alignments of Q vs. R and S_i vs. R are close, to induce a good alignment between Q and S_i. We stress that although R is broken into blocks of size b in a naive way, the alignment method used for breaking Q and the S_i sequences into blocks results in blocks of varying lengths. We denote by b' an upper bound on the size of blocks in $Q, S_1, \ldots, S_m$ (i.e., all of the blocks are of size $0, \ldots, \mathsf{b}'$).

Using this method for partitioning the blocks relatively to a publicly known sequence yields very good results: In our tests, our approximation algorithm returned exactly the k closest sequences in more than 98% of the runs, and a very good approximation in the remaining 2%, see more details in §5. Our experiments show similar results for all different regions of the genome that we checked.

In Example 2.1, we show the significance of breaking the sequences with alignment relative to a reference genome R, as opposed to breaking them into fixed-length blocks. Our partitioning allows flexibility with respect to the breaking points of the sequences, significantly improving the accuracy of the block-wise edit-distance approximation.

1 We remark that using an application-specific partitioning method is the best we can hope for: Any general-purpose partitioning that yields linear-time processing (and guarantees accuracy) will violate a conditional lower bound on the complexity of edit-distance calculations [BI15]. We stress that the alignment of Q and the S_i's to R is done locally *and in the clear* by each party. Description of this procedure and an estimate of its accuracy can be found in §4.

Fixed-Size Partitioning			**Total: 7**
$\boldsymbol{Q}$	TTTA	ATGG	TTAT
$\boldsymbol{S_i}$	TTAA	TAGT	TAGA
$\mathsf{ED}(Q_\ell, S_{i,\ell})$	**1**	**3**	**3**

Our Partitioning			**Total: 4**
R	TTTA	ATAG	TTAG
Q	TTTA	ATGG	TTAT
S_i	TTA	ATAG	TTAGA
$\mathsf{ED}(Q_\ell, S_{i,\ell})$	1	1	2

Example 2.1. *Comparing between block-wise edit distance approximation where the sequences are split according to fixed-size partitioning and our partitioning. In this example,* $S_i =$ TTAATAGTTAGA, $Q =$ TTTAATGGTTAT, *and the reference genome is* $R =$ TTTAATAGTTAG, *where we break the blocks to blocksize* $\mathsf{b} = 4$. *In the exact solution, the edit distance is* 4.

Efficient secure computation. Transforming the approximation procedure above into a secure protocol is not a straightforward application of generic transformations (e.g., Yao [Yao86] or GMW [GMW87]). Rather we use the specific form of our approximation to get a faster implementation.

The protocol begins with a preprocessing phase. The server first breaks all the genomes into blocks as described above, and then it computes the sets $T_\ell = \{u_{\ell,1}, \ldots, u_{\ell,v}\}$ for every $\ell = 1, \ldots, n$. Likewise, the client also breaks its query Q into blocks $Q_1, \ldots, Q_\ell$ according to the same reference genome R. Moreover, the server computes all the intra-block edit-distance values defining a matrix L_ℓ such that $L_\ell[j,i] = \mathsf{ED}(u_{\ell,j}, S_{i,\ell})$, for $\ell = 1, \ldots, n$, $j = 1, \ldots, v$ and $i = 1, \ldots, m$. That is, the value $L_\ell[j,i]$ represents the contribution of the ℓth block to the approximation of $\mathsf{ED}(Q, S_i)$, in case where that block of the query is the jth value in the set T_ℓ, i.e., $Q_\ell = u_{\ell,j}$.

Once these matrices are precomputed and held by the server (in the clear), the problem of computing Eq. (2.3) is reduced to securely computing matrix-vector multiplication. That is, the parties first compute shares of the vector of bits $\chi_{\ell,j}$ (recall that bit $\chi_{\ell,j}$ indicates whether or not $Q_\ell = u_{\ell,j}$), and the result is obtained by securely computing the product of a matrix held by the server and the vector of bits $\chi_{\ell,j}$ shared between the parties.

Since the $\chi_{\ell,j}$s must be secret, and are a function of private inputs, the vector needs to be computed using a secure protocol, and the output must be shares of the vector so that neither party learns it. In order to carry out this computation of shares of $\chi_{\ell,j}$, the parties engage in a standard secure computation protocol for computing a random XOR sharing of all the bits $\chi_{\ell,j}$, using an optimized variant of Yao's garbled circuits. Then for each j, ℓ the parties execute a 1-out-of-2 oblivious transfer protocol to get a random additive sharing of the value $\chi_{\ell,j} \cdot L_\ell[j,i] = \chi_{\ell,j} \cdot \mathsf{ED}(u_{\ell,j}, S_{i,\ell})$, which is of course accelerated using OT-extension [ALSZ13, KOS15]. This utilizes a method for securely multiplying a string and a shared bit using OT.

The parties then locally sum up their shares as per Eq. (2.3), thus obtaining an additive sharing of the approximate edit distance values $\mathsf{ApproxED}(Q, S_i)$ for every $i = 1, \ldots, m$. Finally a standard secure computation protocol, using an optimized variant of Yao's garbled circuits, yields the indexes of the k smallest values. In order to ensure that enough "wires" are allocated for each value, we assume a publicly known upper-bound d on the maximum edit distance. Since this has little effect on the efficiency of the solution, a coarse upper bound can be taken. We prove that:

Theorem 2.2 (informal)**.** *The protocol sketched above securely computes the function* $\mathsf{ApproxClosest}_{k,m}$ *from Eq. (2.4) in the semi-honest adversary model.*

Implementation and performance. We implemented our protocol using the C++ version of the Secure Computation API library (SCAPI) [EFLL12], and tested it on a few databases with hundreds of real genomic sequences. Furthermore, the protocol was evaluated by external referees as part of the iDASH competition.

In our tests, the most costly aspect was the preprocessing on the server side (which is performed in the clear, and only needed to be done once). We did not optimize this part and it took under 12 seconds for our 500-sequence database (with the length of each sequence ≈ 3500). We expect an optimized implementation to be much faster, as our implementation is somewhat naive.

For the online secure computation itself (which is done for every query), the overall number of non-XOR gates is only about 1M AND gates, and we use roughly the same number of OTs. Using efficient implementations of Yao's garbled circuits [KS08, ZRE15] and OT extensions [ALSZ13, KOS15], it took about 1 seconds to fully process each query and find its 5 closest sequences in the database. As a comparison, for the same sequence length, an accurate edit-distance computation of a query and a *single* sequence in the database is roughly ≈ 40 million gates, even when leveraging some upper bounds on the maximal possible distance (reducing the circuits

by 20× factor). Computing the task of finding the closest sequence in a set of 500 sequences, would result in a circuit of ≈ 20 billion gates. Thus, our solution is faster by a factor of approximately $20,000$.

Organization. The rest of this paper is organized as follows. We start with the secure computation protocol in §3. Followed by the description of how to break the sequences into blocks in §4. We report the accuracy of our protocol in §5, and the implementation in §6. We conclude with some discussions and extensions in §7 In the appendices we report on some of our experiments, as well as some supplementary data for decisions we made in our design.

3 Privacy Preserving Protocol

In this section we present our semi-honest secure protocol for computing the ApproxClosest function from Eq. (2.4): The client has a query string, the server has a database of records, and the client needs to learn the indices (or labels) of the k closest records to its query, as specified in Functionality 3.1 below. The functionality is given the parameters $\mathsf{b}, \mathsf{b}', v, R$ (see discussion in §7). Recall that R is the public reference genome, b is the block size for breaking up the reference genome R, b' is an upper bound on the size of the blocks in the query and sequences after alignment with the blocks of R, v is the maximum number of possible different values in a block, and d is an upper bound on the maximum edit distance.

As described in the Introduction we do not compute the exact edit distance between the query and the sequences in the database, but rather an approximation of this value which is amenable to an efficient secure computation. The exact function that we compute depends on our procedure for breaking the sequences and query into blocks, which we describe in detail in §4 below. That procedure computes the blocks $Q = (Q_1, \ldots, Q_n)$ and $S_i = (S_{i,1}, \ldots, S_{i,n})$ (where $n = \lceil |R|/\mathsf{b} \rceil$). For each block location we define a set, $T_\ell = \{S_{1,\ell}, \ldots, S_{m,\ell}\}$ of values that occur in that block position The approximate edit distance function that we compute is:

$$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell}), \quad \text{where,} \tag{3.1}$$

$$\Delta(Q_\ell, S_{i,\ell}) = \begin{cases} \mathsf{ED}(Q_\ell, S_{i,\ell}) & \text{if } Q_\ell \in T_\ell \\ 0 & \text{otherwise .} \end{cases}$$

We remark that although computing $\mathsf{ED}(Q_\ell, S_{i,\ell})$ also for blocks where $Q_\ell \notin T_\ell$ would improve accuracy, this improvement is minor. This is due to the fact that the case of $Q_\ell \notin T_\ell$ is rare, as verified empirically and discussed in Appendix D.2.

Observe that the ApproxED function depends on the parameters b, b' and v, as well as on the reference genome R (since this determines the blocks) and on the set $\mathcal{S} = \{S_1, \ldots, S_m\}$. The dependence on $\mathcal{S}$ is due to the fact that this determines the values in each set T_ℓ. Thus, formally, one should write $\mathsf{ApproxED}_{\mathcal{S},R,\mathsf{b},\mathsf{b}',v}(\cdot,\cdot)$; for clarity, we write ApproxED only, with the understanding that this dependence on the parameters is necessary for fully defining the function.

See Functionality 3.1 for a formal description of the ideal functionality computing the k closest records, based on our approximate edit distance function defined in Eq. (3.2). Observe that the functionality returns the indexes in lexicographic order; this ensures that which sequence is closest, second closest and so on, is not revealed.

Functionality 3.1: (Approximate) Closest k Records Functionality

- **Public parameters:** The database size m, output size $k < m$ and the parameters $\mathsf{b}, \mathsf{b}', v$ and R.
- **Private inputs:** The client holds a sequence query Q. The server holds a database DB of m sequences $(S_1, \ldots, S_m)$.
- **The functionality:**
  1. Let $\tilde{e}_i = \mathsf{ApproxED}(Q, S_i)$ be the approximate edit distance between Q and S_i, as defined in Eq. (3.2) for the parameters $\mathsf{b}, \mathsf{b}', v, R$ and set $\{S_1, \ldots, S_m\}$.
  2. Let I_k be the set of indexes of the k-smallest values in $\tilde{e}_1, \ldots, \tilde{e}_m$, breaking ties according to the lexicographic order.
- **Output:** The client outputs I_k (ordered lexicographically), the server has no output.

Securely computing the ideal functionality. Our protocol for realizing Functionality 3.1 consists of a local preprocessing stage, followed by two main protocol stages:

Preprocessing: In this stage the parties break their sequences into blocks, and the server computes several tables. We describe this stage in §3.1.

Stage I: computing additive sharing of the approximations. This stage is the crux of our protocol, and is described in §3.2. The client and the server interactively compute a secret sharing of the vector of approximated edit distances. Specifically, the parties compute an additive sharing (inside $\mathbb{Z}_d$) of the following vector L:

$$L \triangleq (\mathsf{ApproxED}(Q, S_1), \ldots, \mathsf{ApproxED}(Q, S_m)). \tag{3.2}$$

Stage II: computing the k-minimal values. In the second stage of the interaction, the client and the server compute the k minimal values of the secret-shared vector L, and learn the indices of these values. This stage is described in §3.3.

3.1 The One-Time Preprocessing Stage

The preprocessing stage relies on a procedure BreakToBlocks that the two parties use to break each of their respective sequences into blocks. That procedure is described in §4, and it has the property that for each block location there are only a few distinct values that occur there, and moreover that the two parties know a bound v on the number of values in each block. The $(Q_1, \ldots, Q_n) = \mathsf{BreakToBlocks}_{R,\mathsf{b}}(Q)$ procedure receives a sequence Q and returns its partitioning to blocks, based on the public reference sequence R and blocksize parameter b.

The client. On input the query Q, the client sets $(Q_1, ..., Q_n) := \mathsf{BreakToBlocks}_{R,\mathsf{b}}(Q)$.

The server. On input the database, $S_1, ..., S_m$, the server proceeds as follows:

1. Set $(S_{i,1}, ..., S_{i,n}) := \mathsf{BreakToBlocks}_{R,\mathsf{b}}(S_i)$ for $i = 1, \ldots, m$.
2. For each block location $\ell = 1, ..., n$, compute the set
$$T_\ell = \{S_{i,\ell} : i = 1, \ldots, m\} = \{u_{\ell,1}, \ldots, u_{\ell,v}\} \quad (3.3)$$
of all the values in the ℓth block. The server pads all sets T_ℓ to be of the same size v using some dummy values.[2]
3. For every block location $\ell = 1, \ldots, n$, every sequence S_i $(i = 1, \ldots, n)$, and every value $u_{\ell,j} \in T_\ell$ $(j = 1, \ldots, v)$, the server computes the edit distance between $u_{\ell,j}$ and $S_{i,\ell}$, setting $L_\ell[j, i] := \mathsf{ED}(u_{\ell,j}, S_{i,\ell})$. Below we denote the row $L_\ell[j, \cdot]$ by $L_{\ell,j}$, namely
$$L_{\ell,j} := \big(\mathsf{ED}(u_{\ell,j}, S_{1,\ell}), \ldots, (\mathsf{ED}(u_{\ell,j}, S_{m,\ell})\big) . \quad (3.4)$$
(Jumping ahead, each vector $L_{\ell,j}$ represents the contribution of the ℓ'th block to the final edit distances approximations, for the case where $Q_\ell = u_{\ell,j}$.)

The preprocessing of the server is done only once, and then multiple queries can be computed.

2 This is achieved by introducing also one more character to the alphabet and therefore each DNA character is represented using 3 bits, and not 2. This also increases the size of the circuits.

Computing Eq. (3.2). We observe that for each i, ℓ, the value $\Delta(Q_\ell, S_{i,\ell})$ from Eq. (3.2) can be expressed as
$$\Delta(Q_\ell, S_{i,\ell}) = \sum_{j=1}^{v} \chi_{\ell,j} \cdot \underbrace{\mathsf{ED}(u_{\ell,j}, S_{i,\ell})}_{=L_\ell[j,i]}, \quad (3.5)$$
where $\chi_{\ell,j}$ is 1 if $Q_\ell = u_{\ell,j}$, and 0 otherwise. Therefore we have for all i
$$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell}) = \sum_{\ell=1}^{n}\sum_{j=1}^{v} \chi_{\ell,j} \cdot L_\ell[j, i].$$
Thus, the vector of approximations $(\mathsf{ApproxED}(Q, S_i))_i$ can be computed as
$$\begin{aligned} L &= (\mathsf{ApproxED}(Q, S_1), \ldots, \mathsf{ApproxED}(Q, S_m)) \\ &= \sum_{\ell=1}^{n}\sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j} \ . \end{aligned} \quad (3.6)$$

3.2 Stage I: Computing Additive Sharing of the Vector L

After preprocessing, the client holds a vector of blocks $(Q_1, ..., Q_n)$, and the server holds all the (ordered) sets $T_1, \ldots, T_n$ and the edit-distance vectors $L_{\ell,j}$ for $\ell = 1, \ldots, n$ and $j = 1, \ldots, v$. Our goal in the first stage of interaction is to compute an additive sharing of the approximate-distance vector L. We use additive sharing (rather than XOR sharing), since this enables the parties to locally add their shares from all blocks in order to obtain additive sharing of the overall approximate edit distance. Formally, we need to realize the functionality described in Functionality 3.2.

Functionality 3.2: Additive Sharing of Approximate Edit-Distances, $L^c - L^s = L$

- **Parameters**: Let d be a public upper bound on $\max_{i \in m} \mathsf{ApproxED}(Q, S_i)$.
- **Input:** The client inputs the blocks $(Q_1, \ldots, Q_n)$. The server inputs the tables $T_\ell = \{u_{\ell,1}, \ldots, u_{\ell,v}\}_{\ell \in [n]}$ and vectors $\{L_{\ell,j}\}_{\ell \in [n], j \in [v]}$.
- **The functionality:**
  1. Let $L = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j} \in \mathbb{Z}_d^m$ ($L_{\ell,j}$, $\chi_{\ell,j}$ are defined in Eq. (3.4), Eq. (3.5), respectively);
  2. Choose a random vector $L^s \in \mathbb{Z}_d^m$ and set $L^c := L + L^s \bmod d$.
- **Output:** The client outputs L^c while the server outputs L^s.

The protocol for realizing Functionality 3.2 consists of two main steps:

– First, the parties compute XOR shares of the indicator bits $\chi_{\ell,j}$. That is, for every $\ell \in [n], j \in [v]$, the client and server receive random bits $\chi^c_{\ell,j}, \chi^s_{\ell,j}$, respectively, s.t. $\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = \chi_{\ell,j}$.
– Next they use oblivious transfer to convert their shares of $\chi_{\ell,j}$ (and the value $L_{\ell,j}$ held by the server) into additive shares of $\chi_{\ell,j} \cdot L_{\ell,j}$. That is, they interactively compute random vectors $L^c_{\ell,j}, L^s_{\ell,j}$ such that $L^c_{\ell,j} - L^s_{\ell,j} = \chi_{\ell,j} \cdot L_{\ell,j} \pmod d$.

Then the client and server locally sum their shares: The client computes $L^c = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j} \bmod d$, and the server computes $L^s = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^s_{\ell,j} \bmod d$. Hence

$$L^c - L^s = \sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j} - L^s_{\ell,j} = \sum_{\ell=1}^{n} \sum_{j=1}^{v} \chi_{\ell,j} \cdot L_{\ell,j} = L \ ,$$

where the computation is performed $\pmod d$.

Step 1: Indicator bits. This step realizes the Functionality 3.3:

Functionality 3.3:
Computing XOR sharing for the indicator bit ($\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = \chi_{\ell,j}$)

– **Input:** The client inputs the block Q_ℓ.
The server inputs $u_{\ell,j}$, which is the jth value in the set T_ℓ.
– **The functionality:** Let $\chi_{\ell,j} = 1$ if $Q_\ell = u_{\ell,j}$, and $\chi_{\ell,j} = 0$ otherwise.
Choose a random bit $\chi^s_{\ell,j}$ and set $\chi^c_{\ell,j} = \chi^s_{\ell,j} \oplus \chi_{\ell,j}$.
– **Output:** The client outputs $\chi^c_{\ell,j}$ while the server outputs $\chi^s_{\ell,j}$.

We realize Functionality 3.3 using a direct application of Yao's protocol. Let $Q_\ell = \sigma_1, \ldots, \sigma_t$ and $u_{\ell,j} = \tau_1, \ldots, \tau_t$, represent the inputs $Q_\ell, u_{\ell,j}$ (each padded to some bound b' and converted to binary using suffix-free encoding.[3] The server chooses a random bit $\chi^s_{\ell,j}$ (which will also be its output of the protocol), and we use a standard secure protocol (e.g., Yao's protocol) in which the client learns the output bit $\chi^c_{\ell,j} = \chi^s_{\ell,j} \oplus \bigwedge_{k=1}^{t} (\sigma_k \oplus \tau_k \oplus 1)$. Note that if $Q_\ell = u_{\ell,j}$ then $\sigma_k = \tau_k$ for every k and so $\chi^c_{\ell,j} = \chi^s_{\ell,j} \oplus 1$, resulting in $\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = 1$. In contrast, if $Q_\ell \neq u_{\ell,j}$ then there exists a k for which $\sigma_k \oplus \tau_k \oplus 1 = 0$ and so $\chi^c_{\ell,j} = \chi^s_{\ell,j}$, resulting in $\chi^c_{\ell,j} \oplus \chi^s_{\ell,j} = 0$. At the end of this stage, the client and the server hold the appropriate bits $\chi^c_{\ell,j}, \chi^s_{\ell,j}$ (resp.) for every $\ell = 1, \ldots, n$ and $j = 1, \ldots, v$.

Step 2: Additive sharing. This step realizes Functionality 3.4.

Functionality 3.4: Computing additive sharing for $\chi_{\ell,j} \cdot L_{\ell,j}$

– **Parameters**: The edit-distance bound d.
– **Input:** the client has $\chi^c_{\ell,j}$, and the server has $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$.
– **The functionality:** Set $\chi_{\ell,j} = \chi^c_{\ell,j} \oplus \chi^s_{\ell,j}$. Choose a random vector $L^s_{\ell,j} \in \mathbb{Z}_d^m$ and set $L^c_{\ell,j} = L^s_{\ell,j} + \chi_{\ell,j} \cdot L_{\ell,j}$.
– **Output:** The client outputs $L^c_{\ell,j}$ and the server outputs $L^s_{\ell,j}$.

We realize Functionality 3.4 using 1-out-of-2 oblivious transfer, as described in Protocol 3.5. We recall the definition of 1-out-of-2 oblivious transfer functionality, denoted as $(\lambda, L^\sigma) = F_{\mathrm{OT}}((L^0, L^1), \sigma)$. The sender holds two strings $L^0, L^1 \in \mathbb{Z}_d^m$ and the receiver holds a bit $\sigma \in \{0, 1\}$. The receiver receives L^σ while the sender outputs the empty string λ.

In order to realize Functionality 3.4, the server chooses a random vector $L^0_{\ell,j}$, and its output share would always be $L^c_{\ell,j} = L^0_{\ell,j}$. In addition, it sets $L^1_{\ell,j} = L_{\ell,j} + L^0_{\ell,j}$. The output of the client would be $L^c_{\ell,j} = L^0_{\ell,j}$ in case $\chi_{\ell,j} = 0$ (and thus $L^c_{\ell,j} - L^s_{\ell,j} = 0$) or $L^c_{\ell,j} = L^1_{\ell,j}$ in case $\chi_{\ell,j} = 1$ (and thus $L^c_{\ell,j} - L^s_{\ell,j} = L_{\ell,j}$). Determining which one of the outputs the client receives is done using an oblivious transfer. We prove the security of the protocol in Theorem C.2.

Protocol 3.5: Realizing Functionality 3.4 (in the F_{OT}-hybrid model)

– **Parameters**: The edit-distance bound d.
– **Input:** Client inputs is $\chi^c_{\ell,j}$, server inputs is $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$.
– **The protocol:** (all additions are done $\pmod d$)
1. The server chooses a random vector $L^0_{\ell,j}$ and sets $L^1_{\ell,j} = L^0_{\ell,j} + L_{\ell,j}$.
2. The server and the client engage in a 1-out-of-2 oblivious transfer. The client as the receiver with the choice bit $\chi^c_{\ell,j}$, and the server as the sender with inputs:
   – $(L^0_{\ell,j}, L^1_{\ell,j}) = (L^0_{\ell,j}, L^0_{\ell,j}) + (0, L_{\ell,j})$ if $\chi^s_{\ell,j} = 0$,
   – $(L^1_{\ell,j}, L^0_{\ell,j}) = (L^0_{\ell,j}, L^0_{\ell,j}) + (L_{\ell,j}, 0)$ if $\chi^s_{\ell,j} = 1$.
   Let $L^c_{\ell,j}$ denote the output that the client receives from the OT protocol.
– **Output:** The server outputs $L^s_{\ell,j} = L^0_{\ell,j}$ and the client outputs $L^c_{\ell,j}$.

3 In our case the original strings were over a 4-ary alphabet, so to get suffix-free encoding we need to set at least $t = 2\mathsf{b}' + 1$.

Putting it all together – realizing Functionality. 3.2. We realize functionality 3.2 in Protocol 3.6 using Functionalities 3.3 and 3.4. The overview of the protocol was already presented in the beginning of this section (i.e., §3.2), and its security is proven in Theorem C.3.

Protocol 3.6: Realizing Functionality 3.2 (Using Functionalities 3.3 and 3.4)

- **Parameters**: Let d be a public upper bound on $\max_{i\in m} \mathsf{ApproxED}(Q, S_i)$.
- **Input:** The client inputs the blocks $(Q_1, \ldots, Q_n)$. The server inputs the tables $T_\ell = \{u_{\ell,1}, \ldots, u_{\ell,v}\}_{\ell\in[n]}$ and vectors $\{L_{\ell,j}\}_{\ell\in[n],j\in[v]}$.
- **The protocol:** (all additions are done mod d)
  1. For every $\ell = 1, \ldots, n$ and $j = 1, \ldots, v$:
     (a) Invoke Functionality 3.3, with client input Q_ℓ and server input $u_{\ell,j}$. Let $\chi^c_{\ell,j}$, $\chi^s_{\ell,j}$ be the outputs of the client and server, respectively.
     (b) Invoke Functionality 3.4 with client input $\chi^c_{\ell,j}$ and server input the bit $\chi^s_{\ell,j}$ and the vector $L_{\ell,j}$. Let $L^c_{\ell,j}, L^s_{\ell,j}$ be the output of the client and server, respectively.
  2. The client computes $L^c = \sum_{\ell=1}^{n}\sum_{j=1}^{v} L^c_{\ell,j}$, the server computes $L^s = \sum_{\ell=1}^{n}\sum_{j=1}^{v} L^s_{\ell,j}$.
- **Output:** The client outputs L^c, the server outputs L^s.

3.3 Stage II: Finding k Minimal Values

After computing an additive sharing of the approximate edit distances between Q and the m records $S_1, \ldots, S_m$, the parties engage in a protocol to find the k smallest distances. The full specification is found in Functionality 3.7.

The protocol to realize Functionality 3.7 is just a direct application of Yao's protocol, applied to the following circuit. The circuit outputs m bits, where the ith bit denotes whether S_i is in the k minimum set, and works as follows:

- Compute $L = (L_1, \ldots, L_m) = L^c - L^s \bmod d$.
- Repeat the following for k times:
  - Find the minimum in the list.
  - Compare the found minimum with each one of the elements in the list; When found, set the bit for that output array to 1.
  - OR each value with its set bit in the output array. Note that this makes the minimum value all-ones and therefore it will not be the minimum in the next iteration.

This requires about 5 non-XOR gates per input bit per layer (where k is the number of layers).

Functionality 3.7: Find the k-Minimal Values

- **Parameters:** number of records m, output size k, distance bound d.
- **Input:** Client and server hold $L^c, L^s \in \mathbb{Z}_d^m$, respectively.
- **The functionality:**
  1. Let $L = L^c - L^s \bmod d$, and denote $L = (L_1, \ldots, L_m)$
  2. Find the k smallest values in the sequence L, using the indexes $1, \ldots, m$ to break ties.

  Output: The client gets m bits $(\sigma_1, \ldots, \sigma_m)$, where $\sigma_j = 1$ if $L[j]$ is one of the k smallest values. The server has no output.

Protocol 3.8: Realizing Functionality 3.1 (using Functionalities 3.2 and 3.7)

- **Parameters:** Database size m, output size $k < m$, distance bound d. Moreover, reference genome R, v, block size b and a bound b'.
- **Input:** The client holds a sequence query Q. The server holds a database DB of m sequences $(S_1, \ldots, S_m)$.
- **The protocol:**
  1. The clients and the server perform the preprocessing stage. The client holds the blocks $Q_1, \ldots, Q_n$, and the server holds the tables $T_1, \ldots, T_n$ and the vector $\{L_{\ell,j}\}_{\ell\in[n],j\in[v]}$.
  2. The parties invoke Functionality 3.2, where the client inputs $Q_1, \ldots, Q_n$ and the server inputs the vector $\{L_{\ell,j}\}_{\ell\in[n],j\in[v]}$ and the tables $\{T_\ell\}_{\ell=1}^n$. The client receives vector L^c and the server receives the vector L^s.
  3. The parties invoke Functionality 3.7, where the client inputs L^c and the server inputs L^s. The client receives the m bits $(\sigma_1, \ldots, \sigma_m)$.
- **Output:** The client outputs $(\sigma_1, \ldots, \sigma_m)$.

Realizing Functionality 3.1 is now a straightforward application of the components above, as summarized in Protocol 3.8 below. We prove that the protocol securely realizes Functionality 3.1 in Theorem C.4.

3.4 Security Analysis

Below we sketch the security analysis of our protocol. We follow the standard definition of static semi-honest security in the standalone model (cf. [Gol04]; see also §C.2). We argue the security in a bottom-up fashion:

– We first instantiate the building blocks that we use. For Functionalities 3.3 and 3.7 we use Yao's protocol, and refer to [LP09] for deriving security in the presence of semi-honest security. As for F_{OT}, we refer to [Gol04] for semi-honest protocols that realize this functionality.
– Next, in Theorem C.2 we prove the security of Protocol 3.5 that realizes Functionality 3.4 (sharing of $\chi_{\ell,j} \cdot L_{\ell,j}$). In Theorem C.3 we prove that Protocol 3.6 privately realizes Functionality 3.2 (sharing of L).
– Finally, in Theorem C.4 we put everything together and prove security of the entire protocol (Protocol 3.8).

Note that only the last two bullets require new proofs, everything else holds by assumption on the components that we use. Moreover, we derive the security of the protocol using the standard stand-alone composition theorem [Can00, Gol04]. In Appendix C.2, we prove the following theorem:

Theorem 3.9 (Overall protocol)**.** *Protocol 3.8 realizes Functionality 3.1 against static corruptions in the semi-honest adversary model.*

4 Breaking Sequences into Blocks

As stated in the Introduction the main idea underlying our solution is to approximate the edit distance between two sequences S and Q by partitioning both sequences into n blocks each, then summing up the edit distances across all blocks, returning $\sum_{\ell=1}^{n} \mathsf{ED}(Q_\ell, S_\ell)$. In this section we describe the method that we use to partition the sequences into blocks.

The idea of approximating the edit distance by computing the edit distance on small blocks is appealing as it yields an extremely efficient secure computation. However, the simplest manner of breaking the sequence/query into equal size blocks did not yield a good approximation of the edit distance over the full sequence. Thus, the question arose whether we can enable both parties to break their sequences into blocks that would also yield a good approximation of the edit distance.

4.1 Utilizing a Public Reference Genome

In order to refine the method of breaking the query and sequences into blocks, we utilize a publicly known reference genome R and have both the server and the client break their sequences in relation to R. Utilizing the fact that we work on genomic data, the sequences are somewhat close to each other and also to the reference genome R. Thus, this enables us to break the sequences and query into blocks in a manner that yields a better "alignment" between the blocks of the query and the blocks of the sequences in the database, giving a better approximation of the exact edit distance. We use the public reference genome of [GRC].

The sequences and query are broken into blocks by computing an edit distance between the sequence/query and the reference genome. These are local edit distance computations on known data, and are thus much more efficient than any secure computation. Moreover, as we have seen, the preprocessing is re-usable, and for a large amount of queries this overhead becomes minor.

In our solutions we rely on a reference genome R of roughly the same length as the sequences that we want to break. To break a sequence S (or a query Q) into blocks, we run the Wagner-Fischer edit distance algorithm (for full details see Appendix B) to compute the edit distance between R and S. The algorithm also returns the PTR matrix that keeps the alignment between R and S. From the upper-left corner of the PTR to the lower-right corner it traces the path of how the minimum edit-distance can be obtained.

Let b be a parameter representing our desired block size (on the selection of b see §6; concretely, b is arbitrarily small, e.g., $\mathsf{b} = 5$). With S being recorded at the top of the matrix we break it as follows. We traverse the minimum edit distance path in PTR and whenever we have moved down b rows we break the sequence in that position into a block. Note, that the sizes of the blocks in this partition will vary. Most blocks are of size b, but some are shorter or longer. A full specification of the partitioning algorithm can be found in Algorithm 4.1.

In §5.1 we provide intuition for why this breaking into blocks algorithm yields a good approximation. In a nutshell, computing the full edit-distance between a sequence S and the reference genome R allows us to find the optimal alignment (and the minimal number of ways) of transforming S into R. Thus, this allows us to find the optimal partitioning of S into the blocks of R.

5 Accuracy of Our Approximation

We examine the accuracy of our approximation both theoretically and empirically. In §5.1 we provide a theoretical analysis of our approximation algorithm. While

Algorithm 4.1: $\mathsf{BreakToBlocks}_{R,\mathsf{b}}(S)$ – **Partition a Sequence into Blocks**

- **Parameters:** A reference sequence $R = (\rho_1, \ldots, \rho_r)$, block-size parameter b.
- **Input:** A sequence $S = (\sigma_1, \ldots, \sigma_s)$
  1. Invoke $\mathsf{ED}(R, S)$, and store the table PTR.
  2. Start at the top-left corner of PTR for each multiple of b, i.e. $\mathsf{b}, 2\mathsf{b}, \ldots$ find the index $j_1, j_2, \ldots$ such that $(i\mathsf{b}, j_i)$ is on the minimum edit-distance path. (If there is more than one pair for the same value $i \cdot \mathsf{b}$, store the index j that is closest to $i \cdot \mathsf{b}$.)
  3. Denote $n = \lceil r/\mathsf{b} \rceil$ (observe that the previous steps defines exactly $n-1$ indexes). Let $j_1, \ldots, j_{n-1}$ be the stored indexes, set $j_0 = 0$ and $j_n = s$. Define the blocks $S_\ell = (\sigma_{1+j_{\ell-1}}, \ldots, \sigma_{j_\ell})$ for every $1 \leq \ell \leq n$.
- **Output:** Output the blocks $S_1, \ldots, S_n$.

the bounds in this analysis are somewhat coarse, the analysis still shows that the algorithm is accurate if the reference genome is "close" to the database, or when all sequences in the database are equally-far from the reference genome. In §5.2, we show an empirical evaluation of our algorithm on real genomic datasets, validating that in practice this assumption does hold on various datasets and various different regions of the genome.

5.1 Theoretical Analysis

Intuition for accuracy. The following disregards the case of a "block miss", which we discuss in Appendix D. That is, we focus now on the quality of approximating the edit-distance between two sequences X and Y by computing block-wise distances, where partitioning into block is performed with respect to a common reference genome as described in Algorithm 4.1.

In more detail, suppose we have two sequences X and Y and we wish to compute their edit-distance $\mathsf{ED}(X, Y)$. Our approximation algorithm first breaks each one of the sequences into blocks $X_1, \ldots, X_n$ and $Y_1, \ldots, Y_n$. Then, it computes $\mathsf{ApproxED}(X, Y) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell)$. First, it is clear that

$$\mathsf{ED}(X, Y) \leq \mathsf{ApproxED}(X, Y) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell) \ .$$

This is because there are many ways one can "transform" sequence X into sequence Y. In our case, we transform each block of X into the corresponding block of Y, and sum the number of operations these block-wise transformation consumes. The term $\mathsf{ED}(X, Y)$ minimizes over all possible ways to transforms X into Y, including that specific aforementioned possibility.

Let $R = (R_1, \ldots, R_n)$ be the strings of the reference genome, after breaking it into b-size blocks. For every $\ell \in \{1, \ldots, n\}$, it holds that

$$\mathsf{ED}(X_\ell, Y_\ell) \leq \mathsf{ED}(X_\ell, R_\ell) + \mathsf{ED}(R_\ell, Y_\ell) \ .$$

This holds from a similar reasoning as before: There are many ways to transform X_ℓ into Y_ℓ. The optimal way consumes $\mathsf{ED}(X_\ell, Y_\ell)$ operations, whereas the right hand-side is just one possible way – transforming X_ℓ into R_ℓ, and then transforming R_ℓ into Y_ℓ.

Moreover, we claim that $\mathsf{ED}(X, R) = \sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, R_\ell)$. In order to see that, first note that the term $\mathsf{ED}(X, R)$ is the number of minimal operations that are required for transforming X to R. Moreover, the term $\sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, R_\ell)$ is a specific way to make this transformation, by taking the optimal transformation for transforming X_1 into R_1, then the optimal transformation of X_2 into R_2, etc. While other partitions of X into blocks could have added restrictions when considering block-by-block alignments, the specific partitioning of X that we are considering does not add such restrictions as it was constructed from the optimal alignment path of $\mathsf{ED}(X, R)$. From a similar reason, it also holds that $\mathsf{ED}(Y, R) = \sum_{\ell=1}^{n} \mathsf{ED}(Y_\ell, R_\ell)$.

Putting it all together, since $\sum_{\ell=1}^{n} \mathsf{ED}(X_\ell, Y_\ell) \leq \sum_{\ell=1}^{n} (\mathsf{ED}(X_\ell, R_\ell) + \mathsf{ED}(R_\ell, Y_\ell)) = \mathsf{ED}(X, R) + \mathsf{ED}(Y, R)$, we conclude the following upper bound:

$$\mathsf{ED}(X, Y) \leq \mathsf{ApproxED}(X, Y) \leq \mathsf{ED}(X, R) + \mathsf{ED}(Y, R) \ .$$

While this bound is coarse, still it provides some meaningful insights:

- First, if the reference genome has relatively the same distance from X and Y as any other sequence in the database (i.e., all values $\mathsf{ED}(X, Y)$, $\mathsf{ED}(X, R)$ and $\mathsf{ED}(Y, R)$ are similar), this is a 2-approximation.
- Second, if the two sequences X and Y are obtained by adding random mutations to the reference genome in different distinct locations, then this approximation is in fact, exact. Our experiments show that in practice, this is more likely to be the case.

5.2 Empirical Evaluation

We empirically evaluate the accuracy of our approximation protocol. We specifically target "high-divergence" regions of the genome, since we seek to verify that we still get good results even for such regions. We tested our approach on various datasets and on different chromosomes:

Dataset	# Samples	Length	Avg. Δ (stdev)	Max-Δ Pair	Max-Δ Ref	Variability
ZNF717	501	3470	91.33 (94.24)	175	184	$\leq 5.04\%$
TEKT4P2	51	2087	27.49 (28.55)	54	57	$\leq 2.58\%$
CDC27P1	101	714	12.41 (13.42)	33	50	$\leq 4.62\%$
CDC27P2	101	1950	29.31 (30.46)	65	68	$\leq 3.33\%$
ABHD17AP5	15	1570	3.01 (3.51)	6	18	$\leq 0.38\%$

Table 1. *Our datasets. Avg. Δ is the average of all edit distances between pairs in the datasets. Max-Δ Pair – is the maximal edit distance among all pairs in the dataset. Max Δ-Ref – is the maximal distance between a sequence and the reference genome. Variability is the maximal distance divided by the size of the region.*

Our main dataset: ZNF717. Our main dataset was provided by the organizers of the iDash competition [iDA16]. It contains relatively many (501) gene sequences extracted from the publicly available 1000 Genomes Project [Int18]. It was extracted from human chromosome 3 (75785026-75788496), of length just under 3500, within the coding region of gene ZNF717. The iDASH organizers explained the choice of this particular gene by its high divergence among individual genomes.

Other datasets. From a set of 170 complete sequences, and with the help of the iDash organizers, we extracted several other regions with high-divergence. For each region we chose only a subset of the samples to make the task more challenging. That is, we *excluded* samples that were identical to each other within that region (since the approximation is exact in such a case). The following datasets were extracted:

- **TEKT4P2:** Chromosome 21 (9907190–9909277) of size under 2100.
- **CDC27P1:** Chromosome 2 (133019901–133020615) of size under 750.
- **CDC27P2:** Chromosome Y (10027986-10029907) of size under 1950.
- **ABHD17AP5:** Chromosome 22 (22720578–22722138) of size under 1570, within the coding region of gene ABHD17AP5. Here the region has very low variability. We therefore extracted only 15 samples, testing our algorithm also for a "toy" database.

The datasets, including some basic properties, are given in Table 1. We remark that looking the datasets have a nice variety in the type of queries we examined, where there are queries in which the set of k closest sequences is easily recognizable (as the distance between the $k+1$'th closest element and the query is significantly greater than the distance between the query and the k'th closest element), and in most cases the set is much harder to be recognized (these two distances are very small or even identical).

Accuracy. The main results of our accuracy test are summarized in Table 2. Our algorithm performs remarkably well on all tested datasets: It returned the *exact result* in almost all tests, and very close results otherwise (most of the cases, a result with the same edit distance as the edit distance of the k'th element, or one farther).

We ran the following experiment for each one of the datasets: We chose $\approx 10\%$ random sequences from the dataset as queries and the rest of the dataset was set to be the database. We ran the preprocessing phase of our protocol, and compared the set of sequences that the protocol returned to the correct values, for different threshold parameters – $k = 1, 3, 5$ and 10 (that is, finding the closest sequence, the set of three closest sequences, etc.). We repeated the experiments 10 times, for independent random choice of queries. The block size was set to $\mathsf{b} = 3$, while similar results are obtained to other choices of this parameter.

Table 2 summarizes the accuracy results of our approximation algorithm in the different datasets for different k. The table consists of the following columns:

- Dataset.
- k is the threshold parameter – how many sequences to return.
- Average ED is the (true) average edit distance between the query and the set of the kth closest sequence in the database.
- Average-Δ is the average of how much farther the farthest record returned by the algorithm was than the k'th-closest record. As the numbers are so low this represents that when the algorithm returns a record that it should have not returned, it returns a record that is very close to the one it should have returned.
- Precision: Among the set of the true k closest elements, how many (correct) elements did the approximation algorithm return. This is the standard notion of precision: number of true positives (records that are supposed to be returned) over the sum of

Dataset	k	Ave. ED	Ave. -Δ	Precision
ZNF717	1	2.07	0	100%
	3	2.96	0	100%
	5	4.68	0.01	98.85%
	10	28.98	0.25	97.48%
TEKT4P2	1	13.14	0	100%
	3	16.29	0.80	96.66%
	5	18.5	0.73	96.66%
	10	21.39	0.60	97.33%
CDC27P1	1	2.81	0.02	95.91%
	3	4.39	0.18	94.56%
	5	5.47	0.33	94.28%
	10	6.87	0.57	96.94%
CDC27P2	1	13.08	0	100%
	3	16.75	0	100%
	5	18.27	0.03	99.67%
	10	20.55	0	99.67%
ABHD17AP5	1	0.92	0	100%
	3	2.75	0.6	86.67%
	5	3.17	0.2	92%
	10	4.92	0	98%

Table 2. *Accuracy of our algorithm, for the various datasets and various choices of k.*

Block Size (b)	b′	# Values	Average-Δ (stdev)
3	10	6	0 (0)
5	12	8	0.01 (0.1)
8	15	10	0 (0)
12	19	10	0.01 (0.1)

Table 3. *Algorithm accuracy as function of block size b. Average(stdev) edit-distance between query and 5'th closest sequence is 4.15(8.37).*

true positives and false positive (wrong records that were returned). We break ties according to the lexicographically order. This implies that in case of a tie in which the approximation algorithm returned a sequence with the right edit distance but greater id than the lexicographically smallest one, we count it as an error.

It is important to note that our approximation algorithm returns fairly accurate results even for relatively small databases. For instance for both datasets ABHD17AP5 (15 sequences) and TEKT4P2 (51 sequences) the algorithm always succeeds to identify the closest record in the database, and in case it is wrong for larger k's it always returns records that are very close to those it should have returned. This is an important property, as for several rare diseases the sample set that a real hospital holds can be rather small (couple of dozens patients).

	b = 5		b = 8	
Dataset	Max-b′	Max-v	Max-b′	Max-v
ZNF717	12	8	15	10
TEKT4P2	5	7	8	10
CDC27P1	7	4	12	4
CDC27P2	8	4	12	4
ABHD17AP5	5	2	8	2

Table 4. *The maximum number of block size and number of different values in each table observed in the various datasets.*

The ZNF717 dataset. The dataset in which we had the most number of samples, as well as the most varied types of queries is the dataset ZNF717. This was the database chosen by the iDash organizers [iDA16], who are domain experts for this task. We report some more detailed results for that dataset.

We observe that the block-size parameter does not effect much the accuracy of the algorithm. Nevertheless, it does effect the performance of the protocol, as larger block size means less blocks to process, and therefore overall less workload. Table 3 summarizes the performance and accuracy results of our approximation algorithm as a function of the blocksize parameter b, when returning the closest 5 (approximate) distances. The table consists of the following columns:

- b′ is the largest actual blocksize obtained for any of the sequences (i.e., after breaking the sequences into blocks, some blocks can be larger than b.)
- # Values is the largest number of distinct values found in any block (i.e., the parameter v should upper bound this value).
- Average-Δ is how much farther is the farthest record returned by the algorithm than the true k'th-closest record.

In this experiment, we chose $\approx 1\% \approx 5$ sequences as query sequences, and the other records were chosen to be the database. We repeated this choice 100 times, and the "Average-Δ" values are computed over these 100 runs, together with the standard deviation (in parenthesis).

On the parameters b′ and v. Our overall aim is to compute edit-distance in genomic setting with higher efficiency. Besides exploring the accuracy of our algorithm, we also wish to explore the values of b′ and v as these two parameters are important also for the efficiency of our protocol. The parameter b′ is important for realizing Functionality 3.3 using Yao's circuit. The parameter v reflects the amount of times we will invoke the

Dataset	DB Size	Length	Server CPU Time	Query CPU Time		#AND gates		#AND gates
			Preprocessing (s)	Server (s)	Client (s)	Compare	k-min	Naive
ZNF717	500	3470	11.86	1.22	0.48	1000800	505825	$\approx 20 \cdot 10^9$
TEKT4P2	50	2087	0.69	0.45	0.23	603360	44948	$\approx 400 \cdot 10^6$
CDC27P1	100	714	0.46	0.17	0.09	207360	95618	$\approx 230 \cdot 10^6$
CDC27P2	100	1950	0.91	0.45	0.23	554400	95618	$\approx 875 \cdot 10^6$
ABHD17AP5	15	1570	0.37	0.32	0.19	450720	11080	$\approx 30 \cdot 10^6$

Table 5. *Running times for the various datasets. In all runs $k = 5$, $\mathsf{b} = 5$, $\mathsf{b}' = 12$, $v = 15$, and bandwidth is smaller than 80MB.*

DB Size	$v =$ # values	Preprocessing (s)	Query Server CPU			Bandwidth (MB)	# AND-gates
			Compare (s)	OTs (s)	k-min (s)		
1000	25	30	1.51	4.36	0.16	180	1399480
2000	30	61.8	2.1	11.7	0.31	340	2035415
4000	35	119	2.8	28.2	0.6	660	3149350

Table 6. *Running times for varying DB sizes with fake data for $k = 5$, $\mathsf{b} = 4$ and $\mathsf{b}' = 16$.*

underlying subprotocols. In Table 4, we show how these two parameters appear in the datasets. The experiment is the same as in Table 2. The column Max-v reflects the number of different values in each block as was seen in the experiment, whereas the Column Max-b' represent the largest block size appeared after breaking all sequences into block. We intentionally do not call these parameters as v and b', in order to distinguish between the values that were observed in the experiments and the parameters of the protocol, where the latter should upper bound these values.

6 Evaluation and Performance

We implemented our protocol over the C++ version of the Secure Computation API library (SCAPI) [EFLL12]. We use the state-of-the-art improvements, include Yao with free-XOR technique [KS08] and half-gates [ZRE15], and the recent improvements in OT-extension [ALSZ13, KOS15]. Table 6 presents the performance results for varying database size, these numbers were obtained by running the protocol on a single x86_64 machine using the loopback device for client-server communication.

In our implementation, the most costly aspect was the pre-processing on the server side (which only needs to be done once per database). This part requires many edit distance computations (in the clear), and we did not attempt to optimize it.

Table 5 shows the running times for all datasets, with distance bound $d = 512$ (while the maximal edit distance between pair never exceeded 190), $\mathsf{b} = 5$, $\mathsf{b}' = 12$ and $v = 15$. We chose these parameters somewhat arbitrarily, such that they satisfy the conditions of Table 4. For each dataset, we exclude one sequence and took it as the query, while all other sequences were set to be the database. We repeated this process for every sequence in the dataset. As a result, e.g., the ZNF717 (which contains 501 total sequences) reflects the average of 501 different executions, where in each execution the DB size is 500. The bandwidth never exceeded 80 MB in all executions. We also consider the expected number of gates using the naive solution (with the optimization that considers the bound of maximal edit-distance between a pair in the DB). We expect roughly 10 minutes per 1 billion gates using GMW [KOS15].

Simulated dataset. We wanted to test the scalability of our secure protocol when processing databases with many more records. As there is a lack of availability of such a large genomic dataset, we used fake data. Due to the fact that the data was simulated we ran the protocol just to test the runtime and not the accuracy.

We checked 100 queries with databases of size $m = 1000, 2000$ and 4000 records. Each record is of size roughly 3470 nucleotides. In all these cases, we ran with $k = 5, \mathsf{b} = 4$, $\mathsf{b}' = 16$ and we allowed v to increase with the size of the database. We chose these parameters quite arbitrarily and conservatively as in the case of real genomic data. In Table 6, we report the maximal allowed size of the tables (# values, i.e., v in the protocol), the times needed for the preprocessing, answering a query, the bandwidth and the number of AND-gates.

The reference genome and accuracy. Our theoretical analysis shows that the reference genome must be somewhat "close" to the two sequences that are being compared, and this is necessary for achieving high ac-

curacy. The analysis also suggests that with a random reference genome, our algorithm will be completely inaccurate. We emphasize that the reference genome is never chosen by the protocol, and there is a consortium that is devoted for that [GRC]. Our empirical results show that this reference genome yields great approximation results on all tested databases.

To demonstrate the decline in accuracy with a "bad" reference genome, we give here some experimental result: We consider dataset CDC27P1, and its associated reference genome. We synthetically add noise to its reference genome by iterating over its letters, and at each position leave the letter unchanged with probability 83%, and otherwise randomly adding $1-3$ characters, substitute the current character or remove it. This increases the distance between the sequences in the database and the reference genome to around 150 (instead of 50), and decreases the accuracy from around $\approx 95\%$ to $\approx 90\%$. When increasing the noise even further and leaving the letter unchanged with probability 50%, the distance from the reference genome is increased to 330 and the accuracy is degraded to 62%.

7 Extensions and Discussions

Extending the protocol for other settings. We described Protocol 3.8 in the context of genomic data. This protocol can be used also in more general settings. As for example, consider the following problem: A client holds a vector $x \in \{0,1\}^n$, and the server holds m vectors $S_1, \ldots, S_m \in \{0,1\}^n$, and assume that $m \ll 2^n$. The client should receive the identities of the k-closest vectors in the database, where closeness is measured in terms of hamming distance. As here we are working over a small alphabet (bits) which is also public (i.e., T_i is always $\{0,1\}$, for every "block"), the client can simply share the indicator bits and there is no need for secure computation for that. This protocol results in $2n$ OTs (of vectors of length m), which can be fast using OT extensions. Garbled circuit is then needed only for computing the k-min values out of the m results. Moreover, this protocol is *accurate*, and can also easily be adjusted to weighted hamming distance, in which different coordinates have different weights, or also be generalized for larger alphabets.

Leakage from approximated results. We prove the security of our protocol according to the ideal-real simulation paradigm in the semi-honest settings, where the simulator receives the output of the approximation function. This follows the same spirit as the *liberal definition* for security of approximation in [FIM+01]. A stronger security notion called *functional-privacy* was also introduced in [FIM+01], and requires simulation of the approximated function from the output of the exact function. That is, a (possibly randomized) approximation function g' is *functional-private with respect to a function* g, if there exists a simulator $\mathcal{S}$ such that for every input x in the domain, $\mathcal{S}(g(x))$ is distributed identically to $g'(x)$. Notably, this is a property of the approximation function and the task to be computed, and not of the protocol.

Our approximation function is not functional-private, yet our protocol is fully simulatable given the result of the approximation function. An interesting question is whether an efficient secure protocol can be designed for some approximation function for this task, while the approximation function is also functional-private. We believe that using differential private techniques can transform our approximation function to be functional-private (by adding noise to the results), however, at the expense of degrading its accuracy.

We further note that being non-functional private does not render our protocol useless. In fact, in real-world applications, this task would serve as a building-block and not as a stand-alone system. In some cases, it is likely that the function to be computed using our approximation would be functionally-private, even though our approximation by itself is not. In order to see that, consider the task that motivated our work in the introduction: a medical doctor would like to examine whether a particular treatment would succeed for her client, based on the medical conditions of patients in a remote database. Assume that the vast majority of patients in the exact k-set share the same medical conditions. This is a reasonable assumption, as otherwise such a system would return arbitrary results. Based on our k-closest approximation function, one can build a protocol that first finds the (approximated) closest set, and then determines the results according to the majority of elements in the returned set. As our k-closest approximation recognizes almost all elements in the exact k-closest set, the output of our approximation and the exact function would be the same, and thus this function would be functionally-private. We believe that other tasks can be based on our system and result in functionally-private approximation.

We focused in this work on quantifying the accuracy of our approximation. We compared the identifiers that were returned by our approximation to the identifiers that were returned by the exact function (see Columns "Precision" in Table 2 and "Average-Δ" in Ta-

ble 2). These measurements would be helpful for one who would like to use our approximation as a subprotocol.

The semi-honest model and limitations of the exact functionality. Our solution targets the semi-honest model of security.The goal is to enjoy the benefits of genomic medicine without violating federal laws addressing privacy issues and legislations (such as the Health Insurance Portability and Accountability Act (HIPAA) [HIP]) that safeguard medical information.

It is not hard to see that when deviating from the semi-honest model, by engaging in multiple executions with adversarially chosen queries, a malicious doctor can choose its queries adversarially and learn significant information about each individual in the database. We stress that such attacks can be launched on an "ideal functionality" computing the exact functionality as well. That is, even if an incorruptible trusted party computed the function for the parties, it would still be possible to carry out such an attack. Thus, such an application can only be used safely by parties who trust that they will both behave semi-honestly.

Securely computing the parameters. Our protocol requires fixing several parameters such as (1) the block size b for breaking up the reference genome R, (2) an upper bound b' on the size of the blocks in the query and sequences (after alignment with R), (3) the maximum number of possible different values v in a block, and (4) an upper bound d on the maximum edit distance. At first, one may think that these parameters leak information about the database. However, as we discuss below, they can actually be determined from publicly available data and therefore do not leak *any* information about the database or the query.

The reference genome R that we use is GRCh37, which is publicly available to both the client and the server, and it can be found online (e.g., [GRC]). As it is public knowledge, it leaks no information whatsoever about any individual in the database, nor the query. Our experiments and theoretical analysis (§5) show that b has a minor effect on the accuracy of our approach, and we therefore choose it somewhat arbitrarily. The parameters b', v and d are related to the "variability" of the range in consideration (i.e., the average ratio between the distance of sequences from the reference genome and the length of the range). It is possible to extract these parameters as well from public datasets, and one can create a database mapping between genomic regions and the parameters v and b' (while taking into account the size of the database) similarly to other characteristics that are available for each position in the genome and are publicly available in genomic browsers such as the NCBI browser [NCB].

By conservative choices of the parameters based on public data, no specific database under consideration would exceed the parameters with very high probability. Moreover, the server can monitor whether its actual database satisfies the parameters prior to answering any query.

8 Conclusions

In this work we described a privacy preserving protocol for answering Similar Patient Queries (SPQ) on genome data. Our protocol was designed to operate in settings with high divergence between individuals. We developed an efficient method for approximating the edit distance that provides very good accuracy even in regions of the genome with ≈5% variability, while at the same time being 2-3 orders of magnitude faster than exact calculation. Our work was motivated by the 2016 iDASH competition for computing on genome data, in which our solution won the first place. In particular, for the 500-record dataset used in that competition, we can answer SPQ in under 1.2 seconds per query (after about 12 seconds of one-time pre-processing of the database).

Acknowledgment

Some of the work was done while the first author was a postdoctoral researcher at IBM TJ Watson research center, supported by NSF grant No. 1017660; currently supported by a Junior Fellow award from the Simons Foundation. The second and forth authors are supported in part by the Defense Advanced Research Projects Agency (DARPA) and Army Research Office(ARO) under Contract No. W911NF-15-C-0236. The third author is supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS) and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

We thank Shalev Keren, Meital Levy and Assi Barak for the implementation of our protocol. We thank Diyue Bu and Haixu Tang for their support and help in extracting the datasets, and Haixu Tang, Diyue Bu, XiaoFeng Wang, Shuang Wang, Xiaoqian Jiang, and Lei Wang for organizing the iDASH competition and helping us with all our questions and requests. We also than Robin Hui for the proof in Section 5.1.

References

[AAM17] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure approximation of edit distance on genomic data. *BMC Medical Genomics*, 10(2):41, Jul 2017.

[ABOcS15] Mete Akgün, A. Osman Bayrak, Bugra Ozer, and M. Şamil Sağıroğlu. Privacy preserving processing of genomic data: A survey. *Journal of Biomedical Informatics*, 56:103 – 111, 2015.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *ACM Conference on Computer and Communications Security*, pages 535–548. ACM, 2013.

[AO12] Alexandr Andoni and Krzysztof Onak. Approximating edit distance in near-linear time. *SIAM J. Comput.*, 41(6):1635–1648, 2012.

[BBC+11] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, October 17-21, 2011*, pages 691–702, 2011.

[BI15] Arturs Backurs and Piotr Indyk. Edit distance cannot be computed in strongly subquadratic time (unless SETH is false). In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 51–58, 2015.

[Can00] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[EFLL12] Yael Ejgenberg, Moriya Farbstein, Meital Levy, and Yehuda Lindell. SCAPI: the secure computation application programming interface. *IACR Cryptology ePrint Archive*, 2012:629, 2012. A link to the library: http://crypto.biu.ac.il/about-scapi.

[FIM+01] Joan Feigenbaum, Yuval Ishai, Tal Malkin, Kobbi Nissim, Martin Strauss, and Rebecca N. Wright. Secure multiparty computation of approximations. In *ICALP*, volume 2076 of *Lecture Notes in Computer Science*, pages 927–938. Springer, 2001.

[GA4] GA4GH. GA4GH Strikes Formal Collaborations with 15 International Genomic Data Initiatives. https://www.ga4gh.org/news/sAhZCeJjS96QHhVPIYwwWA. article. [Online; accessed June-2018].

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In *ACM Symposium on Theory of Computing, STOC*, pages 218–229, 1987.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GRC] GRCh37. NCBI: The National Center for Biotechnology Information. The GRCh37 Reference Genome Sequence. https://www.ncbi.nlm.nih.gov/projects/genome/guide/human/index.shtml. [Online; accessed June-2018].

[HEKM11] Yan Huang, David Evans, Jonathan Katz, and Lior Malka. Faster secure two-party computation using garbled circuits. In *20th USENIX Security Symposium, San Francisco, CA, USA, August 8-12, 2011, Proceedings*, 2011.

[HIP] HIPAA. Centers for Medicare and Medicaid Services. Are you a covered entity? https://goo.gl/sdkm13. [Online; accessed June-2018].

[HSE+11] Yan Huang, Chih-Hao Shen, David Evans, Jonathan Katz, and Abhi Shelat. Efficient secure computation with garbled circuits. In *Information Systems Security - 7th International Conference, ICISS 2011, Kolkata, India, December 15-19, 2011, Procedings*, pages 28–48, 2011.

[iDA16] iDASH - integrating Data for Analysis, Anonimization, and SHaring, 2016. Webpage at https://idash.ucsd.edu/genomics, 2016 competition at http://www.humangenomeprivacy.org/2016/.

[Int18] International Genome Sample Resource. IGSR and the 1000 genomes project. http://www.internationalgenome.org/, Accessed Mar-2018.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *2008 IEEE Symposium on Security and Privacy (S&P 2008), 18-21 May 2008, Oakland, California, USA*, pages 216–230, 2008.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *Advances in Cryptology - CRYPTO*, pages 724–741, 2015.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming, 35th International Colloquium, ICALP*, pages 486–498, 2008.

[LP09] Yehuda Lindell and Benny Pinkas. A proof of security of yao's protocol for two-party computation. *J. Cryptology*, 22(2):161–188, 2009.

[LRU14] Jure Leskovec, Anand Rajaraman, and Jeffrey D. Ullman. *Mining of Massive Datasets, 2nd Ed.* Cambridge University Press, 2014.

[NAC+15] Muhammad Naveed, Erman Ayday, Ellen W Clayton, Jacques Fellay, Carl A Gunter, Jean-Pierre Hubaux, Bradley A Malin, and XiaoFeng Wang. Privacy in the genomic era. *ACM Computing Surveys (CSUR)*, 2015.

[NCB] NCBI. Genome Data Viewer. https://www.ncbi.nlm.nih.gov/genome/gdv/browser/. [Online; accessed June-2018].

[NW70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[WF74] Robert A. Wagner and Michael J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1), January 1974.

[WHZ+15] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. Efficient

genome-wide, privacy-preserving similar patient query based on private edit distance. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, pages 492–503, New York, NY, USA, 2015. ACM.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *Symposium on Foundations of Computer Science, FOCS*, pages 162–167, 1986.

[ZH17] Ruiyu Zhu and Yan Huang. Efficient privacy-preserving general edit distance and beyond. Cryptology ePrint Archive, Report 2017/683, 2017. http://eprint.iacr.org/2017/683.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole - reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology - EUROCRYPT*, pages 220–250, 2015.

Appendix

A Related Work

We provide some more in depth comparison with the related work of [WHZ+15] and the concurrent works of [ZH17] and [AAM17].

The work of [WHZ+15]. The most relevant prior work to ours is by Wang et al. [WHZ+15] (building on earlier work of Baldi et al. [BBC+11]) that designs a privacy-preserving protocol for supporting the Similar Patient Query functionality. Wang et al. developed an approximation protocol for edit distance that enables computation in a large scale of the *whole* genome.

The approach of Wang et al. relies on the fact that the genomes that they are examining have little divergence.Using this fact, Wang et al. show how to approximate the edit distance by just considering the set of indexes where the two sequences differ from the reference genome, and running a set-intersection protocol. The above assumptions are valid in some instances, especially when the edit-distance result is very close to that of the hamming-distance. The approximation was shown to be very accurate in these cases, and the computation of a single comparison between two sequences of total 100K variations is performed in several hundreds of seconds.

However, some regions have high divergence and the differences are also caused by insertions and deletions, which are more difficult to deal with in computing edit distance. For example in some regions that affect the immune system the distances between two individuals may be up to 5–7% of the size of the region, and about 25% of the differences between two sequences are due to insertions or deletions.

We implement the approximation function of Wang et al., and examined its accuracy. The approximation function is pretty accurate ($> 97\%$) for databases ABHD17AP5, TEKT4P2, but its accuracy in CDC27P2 is 93.23% (compared to 98.49% of our algorithm), and for CDC27P1 its accuracy is 66.16% (compared to 97.74% of our algorithm). The experiments where all tested with the same randomness for choosing the challenged queries, and with the same settings as in §5.2. In all these tests, our approximation was always more accurate than [WHZ+15].

We remark that while [WHZ+15] deal with 100K variations (i.e., the sets to be intersected are of size 100K) in few hundreds of seconds (depending on the accuracy level), we compare a single query to a database of 500 sequences of size 3470 size each at less than a second (after one-time preprocessing of around 12 seconds). Our problem contain much more characters to check (roughly factor of 17×) and is still much faster, showing the power of pushing most of the computation to preprocessing.

The work of Zhu and Huang [ZH17]. The concurrent work of Zhu and Huang computes edit-distance using garbled circuits, while designing specific gate-level "gadgets" to accelerate computations of edit-distance and related tasks, such as weighted edit-distance and Needleman-Wunsch [NW70]. This approach enables computation of accurate edit-distance and not approximation as ours, and also works for other domains rather than genomic data. Zhu and Huang reported a running time of 3.7 seconds for a single comparison between two sequences in the ZNF717 dataset ($\approx$ 3470-long strings, see §5 for more information regarding the dataset). Using their approach, answering a single query to find the 5 closest sequences in a database of 500 sequences would take 1850 seconds (more than 30 minutes).[4]

The work of [AAM17]. The work of [AAM17] proposed an approximating algorithm for computing the distances between the query and each sequence of the database based on "shingling", a technique used to identify lexically similar documents in data mining [LRU14]. They first break all sequences into small blocks of con-

4 We emphasize that we did not implement their protocol, and the only running times that they reported for the same database as ours is a single comparison.

secutive letters (e.g., the sequence ATTGTTA will be broken into "shingling" {ATTG,TTGT,TGTT,GTTA}). Then, they approximate closeness between a query and a sequence as the number of elements in the intersection of their shingling set, which is implemented using private-set-intersection protocol. This yields a somewhat fast approximation algorithm with low accuracy: around 50% accuracy when looking for the closest-1 set, less than 60% when querying for the closest-5 and around 25% when looking for the closest-10. In order to improve the accuracy of their algorithm, [AAM17] first apply their fast approximation algorithm to find $c \cdot k$ closest sequences (for some constant $c \geq 1$), and then proceed to computing accurate edit-distance between these candidates and the query using garbled circuits. They also present optimizations of the garbled circuit for the case of edit-distance (introducing some error). They report different variants of their algorithms, where the one with accuracy that is comparable to ours has a running time of more than 2000 seconds.

B The Wagner-Fischer Algorithm

The WF algorithm [WF74] is based on dynamic programming, for computing the edit distance between two sequences $A = (\alpha_1, \ldots, \alpha_a)$ and $B = (\beta_1, \ldots, \beta_b)$.

The algorithm proceeds by preparing an $(a+1)$-by-$(b+1)$ matrix $D[\cdot,\cdot]$, where entry (i,j) is the edit-distance between the i-prefix of A and the j-prefix of B. The first row and column are initialized by $D[i,0] = i$, $D[0,j] = j$ for all $0 \leq i \leq a$ and $0 \leq j \leq b$. Then for $1 \leq i \leq a$ and $1 \leq j \leq b$ the algorithm iteratively sets

$$D[i,j] = \begin{cases} \text{if } \alpha_i = \beta_j: & D[i-1,j-1] \\ \text{otherwise}: & \min \begin{cases} D[i-1,j-1]+1 \\ D[i-1,\ j]+1 \\ D[i\ ,j-1]+1 \end{cases} \end{cases}$$

where the first line relates to a "match", the second to "substitution", the third to "delete" and the last to "insert". and finally it returns the answer $D[a,b]$.

This procedure can be augmented to return not only the edit distance itself but also the sequence of operations that transforms A to B in $D[a,b]$ steps. Specifically, together with D we also prepare a matrix of pointers $PTR[\cdot,\cdot]$ (with the same dimension as D), that for each entry (i,j) points to the previous entry from which $D[i,j]$ received its value. Specifically, we initialize $PTR[0,0] = \perp$, $PTR[i,0] = (i-1,0)$ for all $1 \leq i \leq a$ and $PTR[0,j] = (0,j-1)$ for all $1 \leq j \leq b$, and then for $1 \leq i \leq a$ and $1 \leq j \leq b$ we iteratively set

$$PTR[i,j] = \begin{cases} (i-1,j-1) & \text{if } D[i,j] \leq D[i-1,j-1]+1 \\ (i-1,\ j) & \text{if } D[i,j] = D[i-1,j]+1 \\ (i\ ,j-1) & \text{if } D[i,j] = D[i,j-1]+1 \end{cases}$$

where the first case corresponds to a match or substitution, the second corresponds to a delete, and the last case corresponds to an insert. When more than one condition applies, we break ties toward the main diagonal. Namely, we prefer $(i\ ,j-1)$ to the other options when $j > i$, prefer $(i-1,\ j)$ when $i > j$, and prefer $(i-1,j-1)$ when $i = j$.

The PTR table lets us trace on optimal path, starting from $PTR[a,b]$ and following the pointers to get both the alignment of the sequences A, B, as well as the corresponding operations (match, substitute, insert, delete).

C Security Definitions and Proofs

We provide security proofs according to the standard definition of secure protocols (cf. [Gol04]) in the semi-honest model. We briefly describe the definition, and proceed to the security proofs.

C.1 Definitions

For two distribution ensembles $X = \{X_s\}_s$ and $Y = \{Y_s\}_s$, we let $X \overset{c}{\approx} Y$ denote computationally-indistinguishability. Let $f : (\{0,1\}^*)^2 \to (\{0,1\}^*)^2$ be a probabilistic function, and write $f = (f_0, f_1)$, where each $f_i(x_0,x_1) = y_i$ for $i \in \{0,1\}$. Let π be a protocol between parties P_0 and P_1. We let $\mathsf{view}_i(x_0,x_1)$ denote the distribution of the view of party P_i in the protocol execution of π, which consists of the random tape of P_i and all the messages it receives throught the execution. Likewise, we denote by $\mathsf{output}_i(x_0,x_1)$ to denote its output distribution of that execution.

Definition C.1. *Let π, f be as above. We say that π securely realizes f in the presence of a semi-honest adversary, if there exist $\mathcal{S}_0, \mathcal{S}_1$ such that for every $x_0, x_1 \in \{0,1\}^n$ and for every $i \in \{0,1\}$ it holds that $\{\mathcal{S}_i(x_i, f_i(x_0,x_1)), f(x_0,x_1)\} \overset{c}{\approx} \{(\mathsf{view}_i(x_0,x_1), \mathsf{output}(x_0,x_1)\}$.*

Modular composition. The sequential composition theorem [Can00] is a tool for analyzing the security of a protocol in a modular way. Let π_f be a protocol for securely computing f that uses a subprotocol π_g for computing g. The theorem states that it suffices to consider the execution of π_f in a hybrid model where a trusted third party is used to ideally compute g. We rely on this composition theorem in our proof, and refer the reader to [Can00] for the formal statement of the theorem.

C.2 Security Proofs

We prove the security of our protocols. As in our case the parties are a client and a server, we denote the simulators as $\mathcal{S}_s$ (simulating corrupted server) and $\mathcal{S}_c$ (client).

Theorem C.2 (Sharing of $\chi_{\ell,j} \cdot L_{\ell,j}$)**.** *Protocol 3.5 securely realizes Functionality 3.4 in the F_{OT}-hybrid model against static corruptions in the semi-honest adversary model.*

Proof: We separate between the case of a corrupted client and a corrupted server. First, recall that Functionality 3.4 receives the inputs $\chi^c_{\ell,j}, \chi_\ell, j^s$ and $L_{\ell,j}$ from the parties, reconstructs $\chi_{\ell,j} = \chi^c_{\ell,j} \oplus \chi_\ell, j^s$. It then chooses a random output for the server, $L^s_{\ell,j}$ and then deterministically sets the output of the client to be $L^c_{\ell,j} = L^s_{\ell,j} + \chi_{\ell,j} \cdot L_{\ell,j}$.

In order to simulate a corrupted server, note that the protocol is just an execution of an OT protocol, and there are no other messages beyond that single invocation. The server has no output from the OT, and therefore its view is just its randomness, which is solely $L^0_{\ell,j} = L^s_{\ell,j}$. The simulator $\mathcal{S}_s$ receives as input the output of the server, i.e., $L^s_{\ell,j}$, and just outputs this value. According to the definition of the functionality it is guaranteed that the two outputs of the parties, i.e., $L^s_{\ell,j}, L^c_{\ell,j}$ guarantee $L^c_{\ell,j} - L^s_{\ell,j} = \chi_{\ell,j} \cdot L_{\ell,j}$. In the real execution, the OT guarantees that the output of the client $L^c_{\ell,j} = L^s_{\ell,j} + \chi_{\ell,j} \cdot L_{\ell,j}$ For a corrupted client, the simulator $\mathcal{S}_c$ receives as input some string $L^c_{\ell,j}$ as the output of the client. The view of the client in the protocol consists of just the message it receives from F_{OT}, and therefore the simulator outputs it. The theorem follows. ∎

Theorem C.3 (Sharing of L)**.** *Protocol 3.6 securely realizes Functionality 3.2 against static corruptions in the semi-honest adversary model.*

Proof: Correctness is easy by inspection. As for security, assume the case of a corrupted client. The simulator $\mathcal{S}_c$ receives as input a random L^c as the output of the corrupted client, and the output of the server guarantees $L^s = L^c - L$. The view of the client during the execution is the set of shares $\{\chi^c_{\ell,j}\}_{\ell,j}$ and the vectors $\{L^c_{\ell,j}\}_{\ell,j}$. The simulator chooses the set of bits $\{\chi^c_{\ell,j}\}_{\ell,j}$ uniformly at random, and also chooses the vectors $\{L^c_{\ell,j}\}_{\ell,j}$ at random from $\mathbb{Z}_d^{n \cdot v}$ under the constraint that $\sum_{\ell=1}^{n} \sum_{j=1}^{v} L^c_{\ell,j} = L^c$, and outputs these values. As the intermediate values $\{\chi^s_{\ell,j}\}_{\ell,j}$ are hidden from the distinguisher, the bits $\{\chi^c_{\ell,j}\}_{\ell,j}$ that the client receives from the invocations of Functionality 3.4 are distributed uniformly. Moreover, as the values $\{L^s_{\ell,j}\}_{\ell,j}$ are also hidden from the distinguisher, the vectors $\{L^c_{\ell,j}\}_{\ell,j}$ are all random under the constraint that they sum-up to the output of the client. Therefore, the join distribution of the view of the client and the output of all parties in the real execution is identical to the distribution of the output of the simulator and the output of the functionality in the ideal execution.

The case of a corrupted server is proven analogously. ∎

Theorem C.4 (Overall protocol)**.** *Protocol 3.8 realizes Functionality 3.1 against static corruptions in the semi-honest adversary model.*

Proof: Functionality 3.1 is deterministic, and therefore we can prove separately correctness and privacy. Correctness of the protocol is trivial given the definition of the underlying functionalities, i.e., Functionalities 3.2 and 3.7.

Except for the input and output values, the only other messages that the parties see in Protocol 3.8 are the vectors L^c, L^s that are returned by the intermediate Functionality 3.2, and that these vectors are individually uniform, irrespective of the input and output. Thus, in the case of a corrupted client the simulator $\mathcal{S}_c$ just chooses L^c uniformly at random, and in the case of a corrupted server $\mathcal{S}_s$ chooses L^s uniformly at random. ∎

As a conclusion, we derive security for Protocol 3.8 in the plain model using the composition theorem of [Can00, Gol04] in the stand-alone settings, while combining Theorems C.4, C.2, and C.3 together with the security of Yao's Protocol [LP09].

D Block Misses

Our approach ignores blocks of the client that do not appear in the database. This introduces some error to our approximation, but we empirically verify that it is minor. First, in §D.1 we study the frequency of block-misses and see that they rarely occur. Second, in §D.2 we study the error that is introduced with each such block-miss, and see that it is indeed very small.

D.1 Frequency of Block-Misses

Our approach ignores blocks of the clients that do not appear in the database. Intuitively, assuming that m independent samples from some distribution (i.e., the ℓth block of each one of the sequences) occur in some very small set (i.e., T_ℓ), then the probability that an additional independent sample (i.e., Q_ℓ) will occur in the same set is close to 1.

In Table 7, we report the frequency of block misses and show that it is a relatively rare event. The data below is on the real database (ZNF717), where we chose 30 sequences at random to be the queries, and the other 470 sequences to be the DB. We then built the tables T_ℓ, and run our protocol. Overall, more than 99.95% of the blocks of the queries do appear as one of the blocks in the DB. In fact, for more than half of the queries, all their blocks appear in the DB, and the maximal number of block misses per query that was observed is 3. In Table 7 we observe similar results even for small databases, and even when the number of queries and the number of sequences in the database are close.

DB Size	#Queries	Average # Hits per Query (stdev)	Max # of Misses Qer query
276	224	1155.91 (0.62)	2
340	160	1155.90 (0.61)	3
400	100	1155.88 (0.62)	3
434	66	1155.95 (0.49)	2
470	30	1156.51 (0.66)	2

Table 7. *Frequency of number of block of the queries Q_ℓ that appear or do not appear in the corresponding set T_ℓ of the* DB*, as a function of the* DB *size. The* DB *is the real database, where random number of elements where chosen to be the* DB *and the rest where chosen to be the queries. Block size b is always* 3*, and so the number of blocks is always* 1157*.*

D.2 Error Introduced by Block Miss

Even though that block misses are relatively rare events, it is still a question what to do in case they occurs. Assume that $Q_\ell \notin T_\ell$ for some block $\ell \in \{1, \ldots, n\}$. In the following, we compare between two possible approaches:

– The first approach is to compute the accurate distance between $\mathsf{ED}(Q_\ell, S_{i,\ell})$ for every $S_{i,1}, \ldots, S_{i,m}$. This introduces some additional complexity to the protocol, as we have to hide, both to the client and to the server, on which blocks Q_ℓ it holds that $Q_\ell \notin T_\ell$, as well as to compute edit-distances (of small blocks) in the online time. The resulting approximation function is as follows: $\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell})$ where,

$$\Delta(Q_\ell, S_{i,\ell}) \quad = \quad \mathsf{ED}(Q_\ell, S_{i,\ell}) \ . \tag{D.1}$$

– The second approach is the one we chose: we simply ignore these blocks. This results in the following approximation function:
$\mathsf{ApproxED}(Q, S_i) = \sum_{\ell=1}^{n} \Delta(Q_\ell, S_{i,\ell})$, where

$$\Delta(Q_\ell, S_{i,\ell}) \quad = \quad \begin{cases} \mathsf{ED}(Q_\ell, S_{i,\ell}) & \text{if } Q_\ell \in T_\ell \\ 0 & \text{otherwise} \end{cases} \tag{D.2}$$

In Table 8, we show that the accuracy improvement is minor when choosing the first approach. This justifies our choice, as the overhead in computation for computing full edit-distances in case of block misses is significant. The experiment is the same as in Table 2, with the same setting and same random choices of queries.

	Approach I: Eq. (D.1)		Approach II: Eq. (D.2)	
Dataset	Average-Δ (stdev)	Precision	Average-Δ (stdev)	Precision
ZNF717	0.25 (0.82)	96.44%	0.25 (0.82)	97.48%
TEKT4P2	0.54 (2.59)	99.28%	0.60 (3.23)	97.33%
CDC27P1	0.15 (0.69)	97.66%	0.57 (1.48)	96.94%
CDC27P2	1.16 (4.48)	99.02%	0 (0)	99.67%
ABHD17AP5	0 (0)	100%	0 (0)	98%

Table 8. *Accuracy loss for block-misses. Comparing between Approach I: computing $\Delta(Q_\ell, S_{i,\ell}) = \mathsf{ED}(Q_\ell, S_{i,\ell})$ in case $Q_\ell \notin T_\ell$ (as in Eq. (D.1)), and Approach II: $\Delta(Q_\ell, S_{i,\ell}) = 0$ in case $Q_\ell \notin T_\ell$ (as in Eq. (D.2)). The datasets, experiments and random choices are the same as in Table 2, for $k = 10$.*