EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs*

Thomas Schneider
TU Darmstadt
schneider@encrypto.cs.tu-darmstadt.de

Oleksandr Tkachenko
TU Darmstadt
tkachenko@encrypto.cs.tu-darmstadt.de

ABSTRACT

Nowadays, genomic sequencing has become much more affordable for many people and, thus, many people own their genomic data in a digital format. Having paid for genomic sequencing, they want to make use of their data for different tasks that are possible only using genomics, and they share their data with third parties to achieve these tasks, e.g., to find their relatives in a genomic database. As a consequence, more genomic data get collected worldwide. The upside of the data collection is that unique analyses on these data become possible. However, this raises privacy concerns because the genomic data uniquely identify their owner, contain sensitive data about his/her risk for getting particular diseases, and even sensitive information about his/her family members.

In this paper, we introduce EPISODE — a highly efficient privacy-preserving protocol for Similar Sequence Queries (SSQs), which can be used for finding genetically similar individuals in an outsourced genomic database, i.e., securely aggregated from data of multiple institutions. Our SSQ protocol is based on the edit distance approximation by Asharov et al. (PETS'18), which we further optimize and extend to the outsourcing scenario. We improve their protocol by using more efficient building blocks and achieve a 5–6× run-time improvement compared to their work in the same two-party scenario.

Recently, Cheng et al. (ASIACCS'18) introduced protocols for outsourced SSQs that rely on homomorphic encryption. Our new protocol outperforms theirs by more than factor 24 000× in terms of run-time in the same setting and guarantees the same level of security. In addition, we show that our algorithm scales for practical database sizes by querying a database that contains up to a million short sequences within a few minutes, and a database with hundreds of whole-genome sequences containing 75 million alleles each within a few hours.

CCS CONCEPTS

• **Security and privacy** → **Privacy-preserving protocols**; **Management and querying of encrypted data**; **Privacy protections**;

KEYWORDS

medical privacy, privacy-enhancing technologies, genomic research, edit distance, secure computation, outsourcing

1 INTRODUCTION

Numerous efforts by the research community, industries, and governments of different countries substantially reduced the costs of genome sequencing: the costs for sequencing a whole genome have fallen from 10 million USD to less than 1 000 USD in the last ten years [Wet17]. This leads to more genome data being collected, and services that use genome data are becoming increasingly popular, e.g., 23andMe[1], MyHeritage[2], and ancestry[3]. Common use cases for genome data are: (i) Similar Sequence Queries (SSQs) for finding genome sequences that are similar to the sequence of the analyzed person, (ii) Genome-Wide Association Studies (GWAS) for finding associations between diseases and genetic variants, and (iii) genealogical tests for determining ancestral ethnicity of the person.

In this work, we focus on SSQs. They can be used for finding (up to that time unknown) relatives, and for making better diagnoses and prescribing the most promising treatments using the medical history of people that are genetically similar to the patient [FPI16]. However, a data provider (e.g., a medical institution) commonly has a limited number of collected genome sequences which prevents a high-quality similar patient analysis, since the diversity and completeness of the database is crucial in genome analyses [PF16].

A further use case for SSQs is crime solving where only the DNA of the suspect is known. It has been shown in the past that some very complex criminal investigations can be solved using solely the DNA information [Jon10], also even if only the suspect's second-degree relatives are contained in the database, e.g., by reconstructing the family tree [Cor18]. However, no global DNA databases exist at the moment that would facilitate such investigations, since this would raise concerns about the privacy of the DNA donors. As a solution, we consider privacy-preserving aggregation of the DNA databases of multiple parties and privacy-preserving queries on the aggregated database for ensuring privacy of the DNA donors.

Despite the good uses of genomic data, their leakage causes severe privacy violations for the genomic data donors. This is due to the fact that genome data are unique for each individual and contain sensitive information about him/her and his/her relatives [HAHT15, Ayd16], e.g., ethnicity information and predispositions to particular diseases. The possession of this information by third parties can give rise to genetic discrimination, e.g., if a health insurance company would increase the client's fee based on his/her predispositions to diseases, or if an employer would reject the candidate's application based on the aforementioned reasons. To address this, we employ Secure Multi-Party Computation (SMPC)

*A summary of preliminary results of this paper has been published as short paper at WPES'18 [ST18] and the full results have been published at ASIACCS'19 [ST19]. This version contains improvements to the description of the protocols in §3.3.

[1] https://www.23andme.com
[2] https://myheritage.com
[3] https://www.ancestry.com

techniques for constructing highly efficient privacy-preserving protocols for distributed SSQs. Although there already exist solutions for privacy-preserving SSQs, the solutions are either inefficient or custom-tailored, i.e., it is far from trivial to extend these protocols to privacy-preserving aggregation of databases or thresholding distances to similar sequences.

1.1 Our Contributions

We design *EPISODE*, an efficient SSQ protocol that is by orders of magnitude more efficient than previous related works. EPISODE is designed for outsourcing but can be used for the two-party client/server model as well, e.g., for the same setting as in [AHLR18] where one party has a database and the other party has a query, or when each party possesses one or more databases (e.g., for crowd-sourced SSQs). We also show how multiple databases can be aggregated in the outsourcing scenario and describe related costs for each scenario. In addition, we describe a thresholding protocol for finding relatives using EPISODE.

Large-Scale Experiments. We conduct large-scale experiments on an outsourced database with up to one million genome sequences of small and medium lengths, and we show that our implementation has practical run-times on commodity hardware, e.g., secure evaluation of an SSQ protocol on one million sequences of length one thousand took only 8.3 minutes.

Whole-Genome Experiments. To the best of our knowledge, we are the first to conduct experiments on whole-genome sequences (sequence length n=75 million alleles) using the Edit Distance (ED) approximation of [AHLR18] that, unlike [WHZ+15], can handle high-divergence data and show practical run-times of just a few hours.

1.2 Outline

Section 2 describes related work in the field of privacy-preserving SSQs. In Section 3, we explain the necessary basics of genetics, SMPC, and the SMPC framework used in this work. Afterwards, Section 4 gives the system model and details the designed algorithms. Finally, we show the benchmarking results of our algorithms in different scenarios in Section 5 and conclude in Section 6.

2 RELATED WORK

In this section, we compare our Edit Distance (ED) and Similar Sequence Query (SSQ) algorithm with that in previous related work. Our SSQ protocol is benchmarked in the same system model of the papers we compare to unless stated otherwise.

Asharov et al. [AHLR18] introduced an approximation technique for ED that can handle high divergence data. Their contribution is twofold: (i) they construct an efficient and precise approximation for ED, and (ii) they use Look-Up Tables (LUTs) instead of direct computation of the ED, thus precomputing the most expensive parts of the computation in the clear.

The first contribution works as follows: the genome sequences in the database and query are split into blocks of small size (e.g., b=5) and padded to a somewhat greater size (e.g., b'=16). Because of the much smaller size of the blocks compared to a full sequence, the overhead for computing ED is also much smaller (ED requires $O(n^2)$ computation in the sequence length n).

For their second contribution, they utilize the fact that genes in the blocks are naturally distributed highly non-uniformly. For all sequences available in the clear, this allows to compute cross-sequence LUTs block-wise for all observed block values. Using this approach, the value of the block is compared with each element of the corresponding LUT instead of computing the ED directly. If the value is equal to one of the values in the LUT, the corresponding distance is selected. Asharov et al. empirically show that the probability of an element not being in the LUT is small, and the absence of a single element influences the overall distance only slightly. Moreover, the authors design a custom protocol for computing the k nearest edit distances between a client's query and a server's database containing parts of genome sequences. However, their protocol has the drawback that it is custom which makes it non-trivial to extend to other functionalities such as aggregation of databases. Their protocol works in a setting with two semi-honest parties, where the client inputs the query and the server inputs a database into a Secure Two-Party Computation (STPC) protocol. EPISODE runs by factor 5–6× faster than their protocol in the same two-party setting (see Section 5.2 for details).

Atallah et al. [AKD03] developed protocols for secure sequence comparison, and Atallah and Li [AL05] moved these protocols to the outsourcing scenario. In both works, the authors compute the ED, i.e., the number of additions, deletions, and substitutions needed to transform one string into another, with quadratic computation and communication overhead in the sequence length n, i.e., $O(n^2)$, which is a much larger overhead than ours of $O(n\omega)$, where ω is the LUT width, usually 20 or 30.

Jha et al. [JKS08] designed algorithms for privacy-preserving ED using Garbled Circuits (GCs). Their construction scales much worse than the recent solutions including ours, e.g., their algorithm runs in 658 s and requires 364 MB communication, whereas ours requires only 5.7 ms run-time (more than 100 000× faster) and 397 kB communication (more than 900× less) for the same sequence length n=200, and we set the width of the Look-Up Table (LUT) to ω=20.

Wang et al. [WHZ+15] propose an extremely efficient approach for approximating the ED using a set size difference metric. Their approach can process a genome-wide query over one million patients in about 3 hours, but, unfortunately, it works only for data with very small divergence (less than 0.5 % variability between individuals), which is not always true for genome data.

The authors of [AAAM17] designed two approximations for ED: a set intersection method based on [PSSZ15] and a banded alignment-based algorithm that relies on GCs. The drawbacks of these methods are: (i) neither algorithm achieves good accuracy on long genome sequences, and (ii) the authors do not show which security parameters are used and do not detail communication requirements of their algorithms.

Zhu and Huang [ZH17] design efficient algorithms for ED, which are, however, much slower than the approximations of ED. Their benchmarks of ED on two sequences of 4 000 nucleotides with a security parameter of 127 bits took 7.08 s run-time and 2.04 GB communication. In contrast, our algorithm requires only 65 ms run-time (108× faster) and 8 MB communication (261× less) for the same setting and the width of the LUT ω=20.

Table 1: Notation used in our paper.

Parameters	
ω	Look-up table width
N	Number of sequences
n	Sequence length
b	Block size
b'	Padded block size
t	Number of blocks
ℓ	Block bit-length
ψ	Number of data providers
β	Bit-length for the distance values s.t. no overflow occurs
A, C, G, T	Nucleotides: Adenine, Cytosine, Guanine, Thymine
K, M, G, T	Powers of 10: kilo (10^3), mega (10^6), giga (10^9), tera (10^{12})
Notation from [DSZ15]	
$l[i]$	Operator for referencing element #i in list l
$l.e$	Operator for accessing element e in list l
$x \wedge y$ and $x \oplus y$	Bit-wise *AND* and *XOR* operation
A, B, Y	Sharing types: **A**rithmetic, **B**oolean, **Y**ao
$\langle x \rangle_i^t$	Share of value x in sharing type t held by party i
$\mathrm{Shr}_i^t(x)$	Sharing function for value x by party i in sharing type t
$\mathrm{Rec}(\langle x \rangle_0^t, \langle x \rangle_1^t)$	Reconstruction function for value x from both shares
$\langle z \rangle^t = \langle x \rangle^t \odot \langle y \rangle^t$	Operations on shares, $\odot\colon \langle x \rangle^t \times \langle y \rangle^t \mapsto \langle z \rangle^t$
$\langle x \rangle^t = s2t(\langle x \rangle^s)$	Conversion from sharing type s to sharing type t
$\langle 0 \rangle^t, \langle 1 \rangle^t, \langle n \rangle^t$	Secret-shared constant 0, 1, and n, respectively
$\langle \mathbf{F}(\cdot) \rangle^t$	Secret-shared constant of locally computed function **F**
System Model	
T_0, T_1	Semi-trusted third parties that perform SMPC
$P_1, \ldots, P_\psi$	Data providers that contribute genomic data
C	Client

Mahdi et al. [MHM17] securely computed the Hamming distance for SSQ, which is an error-prone metric for measuring the distance between genome sequences, because the sequences are compared bit-wise and, thus, any deletions and additions, which cause shifts in the genome sequence, result in severe errors.

The authors of [CHW18] design a protocol for privacy-preserving SSQs based on [AHLR18] using homomorphic encryption in an outsourcing scenario with two non-colluding, semi-honest parties, which provides the same security guarantees as our model. As we show in Section 5.1, our protocol outperforms theirs by more than factor 24 000× in terms of run-time and by at least factor 16× in terms of communication in the same setting.

The privacy of other applications of genomics were also addressed in the literature, e.g., outsourcing of genome data storing [SLH+17], pattern matching [WSLH17], genome sequence queries [DHSS17], and Genome-Wide Association Studies (GWAS) [BMA+18, TWSH18, BKLS18, CWB18], see [MMDC19] for a good survey on genomic privacy.

3 PRELIMINARIES

In this section, we explain the basics underlying our constructions. Our notation is summarized in Table 1.

3.1 Genomic Primer

Deoxyribonucleic Acid (DNA) is contained in the cells of each living individual and encodes genome information. Based on DNA, individuals develop different phenotype traits — observable differences between individuals, e.g., hair or eye color. The basic components that form DNA are called nucleotides. There exist four of them: Adenine (A), Cytosine (C), Guanine (G), and Thymine (T), which can be encoded in $\log_2 4 = 2$ bits. In our model, however, we require a dummy character for padding blocks of alleles to the predefined global block size, which yields $\lceil \log_2 5 \rceil = 3$ bits. DNA consists of multiple long sequences called chromosomes, which are built as sequences of pairs of nucleotides (e.g., "AT CG AA ...").

The human DNA consists of 3.5 billion base pairs from which only 0.1 % vary among individuals [Eur17]. The variations of single nucleotides in specific regions (called loci, singular locus) of a chromosome are called alleles. The genes, each allele of which is represented to some minimal degree in the population (e.g., more than 1 %), are called Single-Nucleotide Polymorphisms (SNPs).

3.2 Similar Sequence Queries (SSQs)

The approach of SSQs is used for finding the sequences that are most similar to the analyzed query. This approach can be used, for example, for finding individuals that are genetically very similar to a patient in order to better analyze the health conditions of the patient. This leads to more precise medical diagnoses based on the additional information provided by genetically similar individuals.

A precise SSQ algorithm requires the computation of the Edit Distance (ED) [Lev66], which measures how different two sequences are by finding the minimum number of deletions, additions, and substitutions that are required to transform one string into another. ED has $O(n^2)$ computation complexity, where $n = \mathbf{max}(n_s, n_q)$, n_s is the length of the sequence, and n_q is the length of the query. There exist other distance metrics for measuring the similarity of the sequences, but they are generally suboptimal, e.g., Hamming distance is not a good choice because it compares sequences bit by bit and thus any additions and deletions in the genome lead to large errors. To avoid heavy computations of ED, a few approximations have been developed, e.g., [WHZ+15, AHLR18]. For more details see Section 2.

This work focuses on the ED approximation of Asharov et al. [AHLR18], since, in contrast to [WHZ+15], it can handle high-divergence data (the authors of [AHLR18] empirically show this for up to 10 % variability between individual genomes). An example of computing this ED approximation is given in Figure 1. It works as follows: first, the sequences in the database are aligned to a public reference genome and split into blocks of predefined size. Then, the statistical distribution of the sequences in the database is used to construct a Look-Up Table (LUT) containing the most frequently observed block values and their distances to each other. Afterwards, the value of the block i in the query is compared to each entry of the i-th row of the LUT (the comparison is performed only once for one database), and based on the comparison result pre-computed distances are either selected as output or set to 0. Here, the sum of all outputs yields either the correct distance or 0 in the case of an error (this outcome is rare and influences the overall result only very slightly [AHLR18]). The last step is to sum up the distances of all blocks which results in the approximated distance between the query and the sequence. After computing the ED to all sequences in the database, they are used to find the indices of the k most similar sequences.

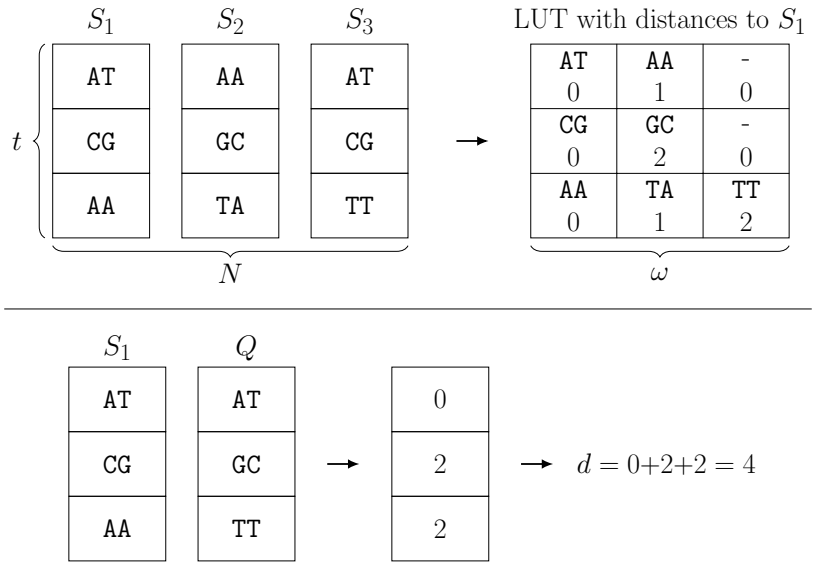

Figure 1: Example for computation of the Edit Distance approximation of [AHLR18]. Here, a Look-Up Table (LUT) for Sequence S_1 is precomputed in the clear based on the distribution of values in all sequences S_1, S_2, and S_3 each containing t blocks of size b=b'=2 alleles (top). In more detail, a LUT contains precomputed distances to all observed block values, e.g., in the third row block AA in S_1 has distances 0 to itself, 1 to TA in S_2, and 2 to TT in S_3. After the LUT construction, the LUT and the pre-computed distances for S_1 are used for computing the Edit Distance d between query Q and S_1 (bottom).

For managing LUTs in the outsourcing scenario, we store the LUTs of all data providers and use them in the ED computation of the corresponding genome sequences in the respective databases, which is a very promising approach in terms of efficiency. The efficiency of this approach grows with N/ψ (the number of sequences N divided by the number of institutions ψ), and in a real-world scenario we expect a small to medium number of data providers ψ that contribute a large number of sequences N. We discuss further LUT management options in Section 4.5.

Family Search from the Similar Sequence Query Protocol. Our SSQ protocol can also be extended for finding one's family. Consider a scenario where a large number of individuals possess their digital genome sequences. They are willing to contribute their data to a common database that can be used to perform family search. For this, they secret-share their genome sequences and compute distances to a public LUT (this can be prepared by a public authority and is the same as the reference genome), which are then used in the SSQ protocol. However, instead of computing k-Nearest Neighbors (k-NN), we can blind the indices that correspond to a big distance to the query (greater than some threshold T, e.g., at most 5 % difference). The result of this protocol is the set of indices of all similar sequences in the database.

3.3 Secure Multi-Party Computation (SMPC)

SMPC allows parties $P_1, \ldots, P_n$ to securely compute a function $f(x_1, \ldots, x_n)$ on their respective inputs without revealing the inputs to each other, i.e., one or more parties learn the result of f, but no intermediate values.

The first approaches to SMPC were proposed in the late 1980s (see, e.g., [Yao86, GMW87]). Although SMPC was first believed to be impractical, with the further progress on SMPC optimization and computer hardware improvements it is nowadays possible to solve complex problems using SMPC within seconds or minutes. SMPC can be conducted considering different adversary models. The two most common adversary models are passive (honest-but-curious) and active (malicious) adversaries. Whereas passive adversaries follow the protocol specification but try to learn as much information as possible from the information they obtain, active adversaries can arbitrarily deviate from the protocol. In this work, as in most previous works in this area [WHZ+15, AAAM17, AHLR18, ZH17, MHM17, CHW18] — to name just a few — we concentrate on protocols with security against passive adversaries that are much more efficient than actively secure protocols and provide sufficient security for settings where curious insiders want to learn additional information from the protocol runs without actively interfering with it. The major difficulty in the use of SMPC is the need of extensive knowledge of cryptography, circuit design, and algorithm complexity for constructing efficient privacy-preserving protocols.

3.3.1 Oblivious Transfer. Oblivious Transfer (OT) is an important building block of many SMPC protocols. In 1-out-of-2 OT, the sender has two messages m_0 and m_1 as input, and the receiver inputs a choice bit c. As output, the receiver receives the message of its choice m_c without learning m_{1-c}, and the sender does not learn c.

Public key-based OT protocols, e.g., [NP01], achieve thousands of OTs per second and OT extension allows using mainly symmetric key primitives resulting in millions of OTs per second [IKNP03,

ALSZ13]. There also exist other variants of OT, such as Random Oblivious Transfer (R-OT) [NNOB12, ALSZ13] and Correlated Oblivious Transfer (C-OT) [ALSZ13]. In C-OT, the sender has m as input, and the receiver has b as input. The outputs of the parties are as follows: the sender receives a random m_0 as output, and the receiver receives $m_0 + bm$ as output, where $m = m_1 - m_0$ in $\mathbb{Z}_{2^\ell}$ and ℓ is the bit-length of the values. This variant of OT improves the communication of the protocol (especially for large ℓ), where instead of $\kappa + 2\ell$ bits only $\kappa + \ell$ bits are sent in the C-OT extension, where κ is the symmetric security parameter, see [ALSZ13].

3.3.2 ABY Framework. We use the ABY framework [DSZ15], which implements state-of-the-art optimizations for Secure Two-Party Computation (STPC), i.e., SMPC with n=2 parties, and is secure against passive adversaries. It enables privacy-preserving algorithms using three different STPC protocols called sharings: Arithmetic sharing (a generalization of the GMW protocol [GMW87] to unsigned integers), Boolean sharing (the Boolean GMW protocol [GMW87]), and Yao sharing (Yao's Garbled Circuits (GCs) [Yao86]). It also enables mixing these protocols to use particular STPC protocols for the parts of the computed function where they perform best. For notation used in this paper, please refer to Table 1.

Arithmetic Sharing. Arithmetic (also called Additive) sharing is performed on integer numbers in a ring $\mathbb{Z}_{2^\ell}$. Values are shared locally by subtracting random numbers as one-time-pads from the initial values, and afterwards one of the shares is sent to the other party. The reconstruction of the shares for the outputs is also straightforward: the parties exchange the shares and compute the sums of the single shares in the corresponding ring which yields the corresponding cleartext values. The main advantage of Arithmetic sharing over other sharings is that it allows local computation of addition mod 2^ℓ and cheap computation of multiplication mod 2^ℓ using Multiplication Triples (MTs) [Bea96] that can efficiently be precomputed using OT extension [DSZ15]. The drawback of this sharing is that it does not allow to trivially perform other more complicated operations. For example, secure comparisons are very expensive in Arithmetic sharing.

Boolean Sharing. In ABY, Boolean sharing stands for the GMW protocol [GMW87]. This sharing is represented as a Boolean circuit where shares represent wires in the circuit. Similarly to Arithmetic sharing, the values are shared by performing the XOR operation with a random value. Reconstruction can be performed by applying XOR on both shares (the parties, again, exchange the shares), which will eliminate the random value and yield the cleartext value. The Boolean circuit is constructed by using XOR and AND gates (any computable function can be converted to a Boolean circuit using XORs and ANDs only). There is, however, a large difference in efficiency of evaluating these gates. Whereas XOR gates can be evaluated locally (due to the associative property of XOR), AND gates require communication during the evaluation. For secure evaluation of AND gates in ABY, Beaver's MTs [Bea96] that are precomputed using OT extension [ALSZ13] are utilized. The communication requirements of GMW can be further reduced at the cost of slightly higher computation [DKS+17]. Moreover, each AND-layer in the circuit adds an additional communication round to the protocol. Consequently, we are interested in shallow circuits for Boolean sharing.

Yao Sharing. Yao's Garbled Circuits (GCs) [Yao86] are denoted as Yao sharing in ABY. Yao sharing includes all state-of-the-art enhancements, such as point-and-permute [MNPS04], free-XOR [KS08], fixed-key AES garbling [BHKR13], and half-gates [ZRE15]. Yao sharing, similar to Boolean sharing, can be used to securely evaluate a Boolean circuit. It is split into two phases: an input-independent setup phase and an input dependent online phase. In the setup phase, the party called garbler garbles the circuit and sends it to the other party called evaluator. The parties then proceed to the online phase, where only the evaluator's inputs have to be obliviously transferred via precomputed OTs [Bea96] and the evaluator can compute the garbled result locally. For reconstructing the results on the evaluator's side, the garbler sends the output keys for the corresponding shares to the evaluator, and for the garbler's side, the evaluator sends the output keys to the garbler. Yao sharing has a constant number of rounds, i.e., it does not depend on the circuit depth, and therefore generally is better suited for high-latency networks than Boolean sharing. On the other hand, Yao sharing requires more computation and communication than Boolean sharing.

SMPC Protocol Conversion. It is clear from the description of the aforementioned sharings that the choice of particular sharing is not trivial even for relatively simple tasks. To solve this problem, ABY allows to mix the protocols by implementing efficient algorithms for converting between the three different sharing types. Although conversions imply some costs, they may result in better overall performance as shown in [DSZ15]. The partitioning can even be done automatically [BDK+18].

OT-Based Multiplication. We extend the OT-based multiplication algorithm of [AHLR18] for multiplying an additively secret-shared value $\langle v\rangle^A$ in Arithmetic sharing by a secret-shared bit $\langle b\rangle^B$ in Boolean sharing. Observe that we want to compute $\langle b\rangle^B \cdot \langle v\rangle^A = (\langle b\rangle_0^B \oplus \langle b\rangle_1^B)(\langle v\rangle_0^A + \langle v\rangle_1^A)$, which we reformulate as

$$\langle b\rangle_0^B\langle v\rangle_0^A + \langle b\rangle_0^B \cdot (-1)^{\langle b\rangle_1^B} \cdot \langle v\rangle_1^A + \langle b\rangle_1^B \cdot (-1)^{\langle b\rangle_0^B} \cdot \langle v\rangle_0^A + \langle b\rangle_1^B\langle v\rangle_1^A.$$

Whereas $\langle b\rangle_0^B\langle v\rangle_0^A$ and $\langle b\rangle_1^B\langle v\rangle_1^A$ can be computed locally by the respective parties, for the computation of $\langle b\rangle_i^B \cdot (-1)^{\langle b\rangle_{i-1}^B} \cdot \langle v\rangle_{i-1}^A$: $i \in \{0,1\}$ interaction is required between P_0 and P_1. Note though that the right part, i.e., $(-1)^{\langle b\rangle_i^B} \cdot \langle v\rangle_i^A$, is computed locally. For the remaining two multiplications, we utilize two C-OTs [ALSZ13], where P_i inputs $\langle b\rangle_i^B$ and P_{1-i} inputs $(r, (-1)^{\langle b\rangle_{i-1}^B} \cdot \langle v\rangle_{i-1}^A + r)$, where r is a random value, and vice versa. P_{1-i} then sets its share to $-r$. As a result, the parties compute a valid Arithmetic share $\langle b\cdot v\rangle^A$ of value $b \cdot v$. Compared to the state-of-the-art multiplication protocol by Patra et al. [PSSY20], the online communication in our protocol is only marginally higher (their 2ℓ vs. our $2\ell + 2$), whereas our communication in the setup phase is by a factor of $(2.5 + 2.5\ell/\kappa)\times$ lower for generic (non-amortized) multiplication. With the optimization technique of the online communication rounds of C-OT from [BDST20], our protocol requires the same round complexity of one round as the protocol in [PSSY20].

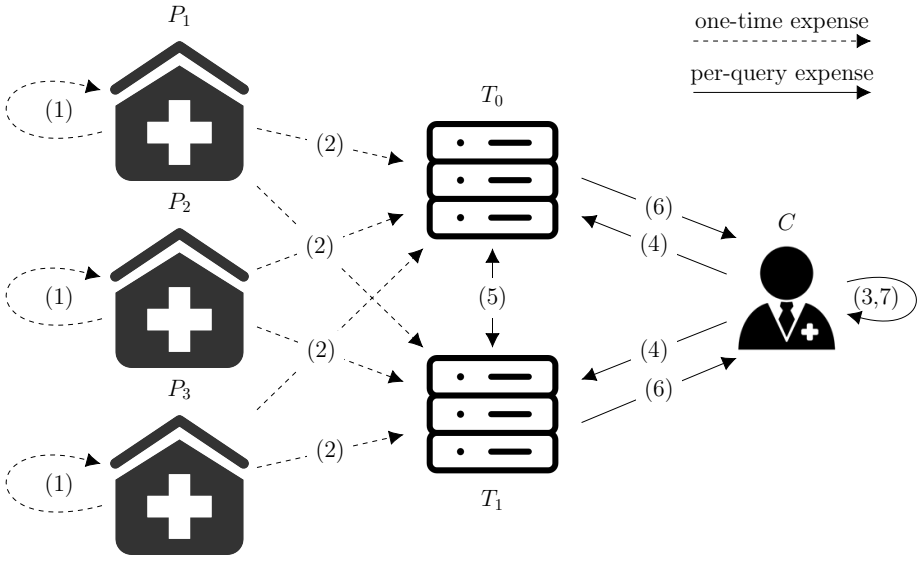

Figure 2: Privacy-preserving Similar Sequence Query system model with three medical institutions I_1, I_2, and I_3 that contribute their secret-shared genomic data to two Semi-Trusted Third Parties T_0 and T_1, and a client C who queries the secret-shared database. The communication between all parties is protected with a secure channel, e.g., TLS. See Section 4.1 for more details.

Note that the original publication [ST18, ST19] erroneously depicted a multiplication of two *Boolean* shares, which does not work for the multiplication of a Boolean with an arithmetic share. The description is fixed in this version of the paper.

Single Instruction Multiple Data (SIMD) gates. Wrapping of secret-shared data in container classes is very memory-consuming. Moreover, this adds additional overhead for managing and initialization of memory to the protocol. As a solution to this problem, SIMD gates have emerged [SZ13] and are also implemented in ABY. These gates are constructed as gates for evaluating arrays of secret-shared values rather than single values. This approach significantly reduces the RAM requirements and the online run-times of the protocols.

k-Nearest Neighbors. For finding the k most similar sequences in the Similar Sequence Query (SSQ) protocol, we utilize ABY's functionality for computing k-Nearest Neighbors (k-NN) [JLL+19], which improves over the work of Songhori et al. [SHSK15] in terms of the circuit size. This k-NN implementation is by about a factor of 5× more efficient than the one used in [AHLR18], e.g., for a database of size 500 [AHLR18] required 505 825 AND gates for computing the k-NNs, whereas we require only 92 500 AND gates.

4 OUR PRIVACY-PRESERVING SSQ PROTOCOL

Here, we describe the system model of our privacy-preserving protocol for Similar Sequence Queries (SSQs), the protocol itself, and we analyze its security.

4.1 System Model

The main idea of our protocol is to secret-share the database aggregated from data of multiple data providers between two non-colluding Semi-Trusted Third Parties (STTPs). We depict our system model in Figure 2. The communication between all parties is performed over a secure channel (e.g., TLS). Note that our protocol alternatively can be run directly between a server and a client with approximately the same efficiency (the outsourcing scenario is beneficial for data aggregation, but has the same efficiency in the querying phase). In our protocol, we have the following parties:

- Data providers (e.g., medical institutions) $P_1, \ldots, P_\psi$ that securely contribute their genome sequences to the outsourced database in a secret-shared form.
- A client C who privately queries the database with a genome sequence for finding the most similar sequences in the outsourced database.
- Two non-colluding Semi-Trusted Third Parties (STTPs) T_0 and T_1 who obliviously compute the Similar Sequence Query (SSQ) protocol on the client's query and outsourced database.

We choose two STTPs because it is the most practical and affordable model in a real-world setting, since each STTP has to be operated and maintained by different teams, and the servers must have completely different software stacks, which in total implies high costs. For running the two STTPs, one must choose two distinct organizations that have a high motivation to not collude, e.g., if they significantly loose in value/reputation if caught cheating. We can think of the following organizations: (i) health ministry, (ii) research institutes, or (iii) cloud service providers. This

outsourcing model has been widely used in the literature, e.g., in [CDC+17, ADS+17, TWSH18].

Our protocol consists of the following steps (see Figure 2):

A **Initialization**

(1) Each data provider P_i locally secret-shares its genomic data and Look-Up Table.

(2) I_i sends the shares to the Semi-Trusted Third Parties T_0 and T_1, respectively.

B **Querying**

(3) Client C locally secret-shares its query Q.

(4) C sends its secret-shared query to T_0 and T_1.

(5) T_0 and T_1 obliviously compute the SSQ protocol on the secret-shared database and the client's secret-shared query using STPC.

(6) T_0 and T_1 send the resulting output shares containing secret-shared indices of the most similar sequences in the database to C.

(7) C locally reconstructs the result from the output shares.

In contrast to querying the database (querying phase), aggregating the database from different sources (initialization phase) is a one-time expense. Data providers contribute their data only once, and any changes in the outsourced database are required for updates only. Since data providers can send their secret-shared sequences in any order and from different preprocessing sets (though using the same global parameters), the database can be updated without any further preprocessing steps on the STTPs' and data providers' side. Additional data providers in the protocol do not add any significant overhead because of the following: (i) the initialization phase is a one-time expense, (ii) the initialization phase is computed in parallel for all data providers, (iii) the STTPs do not apply any further preprocessing steps but only store the received shares, which has a very small overhead.

Client's Communication and Computation. Our model significantly reduces the amount of communication and computation performed by the client compared to the direct application of SMPC. More detailed, the client sends only 2× the amount of information compared to the non-private cleartext protocol. Moreover, the client does not require cryptographic operations in the protocol but only very efficient XOR and addition mod 2^ℓ operations. This makes our protocol even applicable for weak clients using mobile devices.

4.2 Privacy-Preserving Approximated Edit Distance

Our protocol for securely computing the Edit Distance (ED) between a genome sequence stored in the outsourced database and a client's query utilizes the idea of Asharov et al. [AHLR18] for improving the efficiency of computation by approximating ED using Look-Up Tables (LUTs) (see Section 3.2).

We extend the two-party protocol of [AHLR18] to the outsourcing scenario and carefully optimize the implementation using a mix of different sharings and minimize costly operations, such as conversions between sharings and the operations that require interaction/heavy computations. Our detailed algorithm is given in Algorithm 1 and its data representation is given in Figure 3. The Similar Sequence Query (SSQ) algorithm is given in Algorithm 2.

$\langle distance \rangle^A \leftarrow \textbf{ED}(seq, query)$

```
1 :  row_dist ← ∅
2 :  for i = 1 to seq.length do
3 :    ⟨row_sum⟩^A ← ⟨0⟩^A
4 :    for j = 1 to seq.LUT.width do
5 :      //For multiple ED computations, eq values are computed only once
6 :      //Next 2 lines are equal to e_{iω+j} == q_i ? d_{iω+j} : 0
7 :      ⟨eq⟩^B ← ⟨seq[i].LUT[j].value⟩^B == ⟨query[i]⟩^B
8 :      ⟨dist⟩^A ← OTM(⟨eq⟩^B, ⟨seq[i].LUT[j].dist⟩^A)
9 :      ⟨row_sum⟩^A ← ⟨row_sum⟩^A + ⟨dist⟩^A
10 :   row_dist.insert(⟨row_sum⟩^A)
11 :  for i = 2 to seq.length do
12 :    ⟨row_dist[1]⟩^A ← ⟨row_dist[1]⟩^A + ⟨row_dist[j]⟩^A
13 :  return ⟨row_dist[1]⟩^A
```

Algorithm 1: Privacy-preserving Edit Distance (ED) algorithm between a sequence *seq* containing a Look-Up Table with genomic variants and the corresponding distances, and a client's query containing genomic variants. OTM denotes Oblivious Transfer-Based Multiplication.

$ids \leftarrow \textbf{SSQ}(sequences, query, k)$

```
1 :  dists ← ∅
2 :  ids ← ⟨1⟩^Y, ..., ⟨sequences.size⟩^Y
3 :  for i = 1 to sequences.size do
4 :    ⟨dist⟩^A ← ED(sequences[i], query))
5 :    dists.append(A2Y(⟨dist⟩^A))
6 :  ids ← k-NN(dists, ids, k)
7 :  return ids
```

Algorithm 2: Privacy-preserving Similar Sequence Query (SSQ) algorithm between a query and sequences of genomic data in the outsourced database. *k*-NN denotes the *k*-Nearest Neighbors algorithm.

More detailed, we improve the protocol of [AHLR18] by using more lightweight GMW [GMW87] instead of GCs [Yao86] for comparisons, Correlated Oblivious Transfers (C-OTs) [ALSZ13] instead of general OTs, and a more efficient k-NN algorithm in Yao sharing. Although we use a more efficient C-OTs, we require two C-OTs instead of one OT, which is due to the fact that the Semi-Trusted Third Parties (STTPs) do not possess the LUTs in cleartext. However, the cost for OTs remain approximately the same as in the protocol of [AHLR18] for large databases (less than 1 % overhead for a database with 10 000 sequences). In contrast to [CHW18], we compute most of the functionalities using generic protocols which enables arbitrary extensions of our protocol. A possible and cheap extension of our protocol would be a thresholding protocol that reveals only those sequence indices that have distances smaller than some threshold T. This protocol has the advantage that it dispenses with the need of finding the k most similar sequences, which improves the complexity from $O(kN)$ to $O(N)$.

We optimize the SSQ algorithm by mixing different SMPC protocols. First, the blocks of the query and LUT are secret-shared in Boolean sharing. Boolean shares are then used to compare the block values of the query with the block values of the LUT, namely, the block value i of the query with each of the ω block values of the LUT in the row i. Afterwards in Arithmetic sharing, shared distances or zeros are chosen depending on the comparison results between the query and LUT. For this, we use two C-OTs for multiplying the comparison result $r \in \{0, 1\}$ with the distances in Arithmetic sharing. Since all distances are valid for the sequence (each block yields either a valid block distance or zero) and only need to be summed up for resulting in the total distance between the query and sequence, we perform — free in Arithmetic sharing — addition operations for all distances in the sequence.

4.3 Privacy-Preserving Similar Sequence Queries

In this work, we consider a system model where the client wants to find k genome sequences that are most similar to its query among the sequences stored in the outsourced database. Here, we proceed as follows: the distances to single sequences are first calculated using the Similar Sequence Query (SSQ) algorithm, and afterwards, the distances are used along with the corresponding IDs for finding the k closest distances using the k-NN algorithm (see Algorithm 2).

Choice of the Algorithm for Finding k Most Similar Sequences. Generally, we can think of two possible methods for efficiently finding the k most similar sequences: the k-Nearest Neighbors (k-NN) algorithm and sorting networks. The first has a small AND-size for small k, but a large AND-depth of $O(n)$, because the algorithm is difficult to parallelize, whereas the second have a logarithmic AND-depth and AND-size of $O(n \log^2 n)$, which is independent of k. Since the resulting circuit size of a sorting network is by an order of magnitude larger for small k, it is practically less efficient even in Boolean sharing than k-NN in Yao sharing, so we use the efficient algorithm for finding indices of the closest distances (k-NN with precomputed distances) in Yao sharing that is already implemented in ABY [JLL+19]. In EPISODE, we utilize the highly efficient k-NN implementation in ABY with $Nk(2\ell + \lceil \log_2 N \rceil)$ AND gates, where N is the number of sequences, k is the number of most similar sequences, and ℓ is the bit-length of the distances. Since the distances are shared in Arithmetic sharing, we first convert them to Yao sharing.

Communication. Our SSQ algorithm consists of three parts: comparison, OTs, and k-NN. The first part requires $6t\psi\omega\kappa b'$ bits to be transferred (see Table 1 on p. 3 for the explanation of all parameters). For our C-OT-based protocol, $2t\omega(N \cdot \mathbf{NPoT}(\lceil \log_2(tb') \rceil + \kappa))$ bits of communication are required. $\mathbf{NPoT}(x)$ (Next Power of Two) denotes a function that takes a rational number as input and outputs the smallest number that is a power of two and is equal or greater than x. For the last part (the k-NN algorithm), we require $2Nk\kappa(2\lceil \log_2(tb') \rceil + \lceil \log_2 N \rceil)$ bits of communication. The total communication is approximately $2t\omega(3\psi\kappa b' + N \cdot \mathbf{NPoT}(\lceil \log_2(tb') \rceil)) + 2Nk\kappa(2\lceil \log_2(tb') \rceil + \lceil \log_2 N \rceil)$ bits.

4.4 Security Analysis

Our protocol is based on the outsourcing protocol described in [KR11], which gives a generic construction and a security proof for turning an N-party SMPC protocol into a secure outsourcing scheme where data is oursourced to N non-colluding Semi-Trusted Third Parties (STTPs).

We instantiate SMPC with the $N = 2$-party ABY framework [DSZ15]. Our protocols implement the algorithm for SSQs of [AHLR18]. Our protocols are even secure against malicious data providers and clients (who all send a single message in the protocol), and secure against semi-honest STTPs. Since data providers do not receive any outputs, their malicious input cannot affect the privacy of the protocol. Moreover, any changes to the client's single input message correspond to a different input to the ideal functionality, which yields security against malicious clients.

Theorem 4.1. *Assume that the protocols implemented in ABY [DSZ15] are secure against semi-honest adversaries and the two STTPs are semi-honest and non-colluding. Then our protocols securely implement the algorithm of Asharov et al. [AHLR18].*

Proof (sketch). The proof follows immediately from the proof in [KR11] and the fact that the protocols run between the two STTPs are secure against semi-honest adversaries and operate on secret-shared data. More detailed, consider shares $\langle x \rangle_i^t$ and $\langle x \rangle_{1-i}^t$ of an input value x shared by Party P_i in sharing type $t \in \{A, B, Y\}$ corresponding to Arithmetic, Boolean, and Yao sharing, respectively. Party P_{1-i} gets $\langle x \rangle_{1-i}^t$ from P_i. Party P_{1-i} cannot derive any information from $\langle x \rangle_{1-i}^t$ without knowing $\langle x \rangle_i^t$. Similarly, the security of the STPC protocols and STPC protocol conversions that ABY is based on guarantees that no information can be derived from the intermediate shares (i.e., secret-shared result of any operation on shares). The C-OT-based multiplication (which is a straightforward extension of [AHLR18]) is performed on secret-shared values and thus does not reveal any information about the cleartext values. The STTPs do not learn any new information from the secret-shared outputs. By aggregating the joint database, the data providers have no outputs and thus cannot infer any information from the protocol. In view of the above arguments: (i) no semi-honest STTP can obtain any new information on the genome data from the shares, (ii) no actively corrupted server can obtain any information on the genome data contained in the client's query or other server's database, and (iii) no actively corrupted client can obtain new information on the genome data contained in the database of any of the servers.

4.5 Data Aggregation from Multiple Data Providers

We see three possible approaches of aggregating data in the outsourced database:

(1) The most intuitive approach is to attach a Look-Up Table (LUT) to each genome sequence, which was used in [CHW18]. This approach dispenses with the need of LUT management for the Similar Sequence Query (SSQ) protocol and when updates occur. Since each genome sequence has its own LUT and is thus independent of other sequences, the data provider only has to upload the secret-shared new sequence and the corresponding secret-shared LUT to the

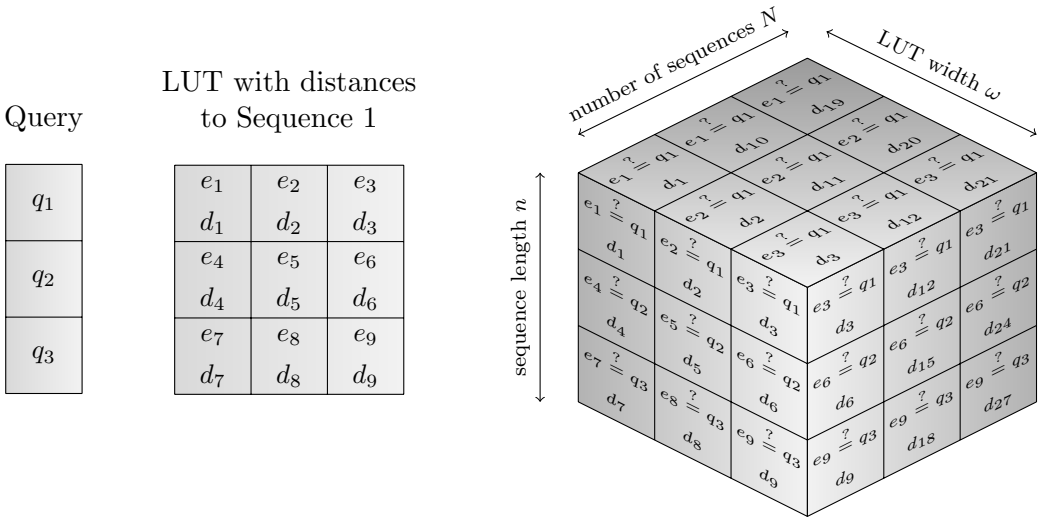

Figure 3: Data representation of the Similar Sequence Query algorithm. For computing the Edit Distance to three sequences, the values of the blocks in the client's query $\{q_1, q_2, q_3\}$ are compared with the precomputed values in the secret-shared Look-Up Tables (LUTs) $\{e_1, \ldots, e_9\}$. Based on the comparison, we process the precomputed distances $\{d_1, \ldots, d_9\}$, $\{d_{10}, \ldots, d_{18}\}$, $\{d_{19}, \ldots, d_{27}\}$ to the Sequences 1, 2, and 3, respectively. For example, for Sequence 1 in the first block e_1 is compared with q_1; if they are equal, the precomputed distance d_1 is returned, and 0 otherwise. The computation of the LUT is parallelized using Single Instruction Multiple Data gates.

Table 2: Run-time and communication comparison of our algorithm with that of Asharov et al. [AHLR18] with sequence length n=3 470, number of nearest sequences k=5, block length b=4, padded block length b'=16, and number of data providers ψ=1 for different parameters N (number of sequences) and ω (Look-Up Table width). The preprocessing stage is not included in the total run-time. A plot is given in Figure 4 in Appendix A.

N	ω	Run-time (localhost / LAN) in s			Communication in MB		
		[AHLR18]	Ours	Improvement	[AHLR18]	Ours	Improvement
1 000	25	6.03 / -	1.07 / 1.89	5.6× / -	180	130	1.3×
2 000	30	14.11 / -	2.20 / 4.07	6.4× / -	340	268	1.2×
4 000	35	31.60 / -	4.83 / 9.02	6.5× / -	660	571	1.1×

Table 3: Run-time comparison of our Edit Distance algorithm with that of Cheng et al. [CHW18] with sequence length n, Look-Up Table width ω=20, and block size b=2.

n	Run-time (LAN)		
	[CHW18]	Ours	Improvement
10	1.2 s	2.1 ms	571×
20	2.2 s	3.0 ms	733×
30	3.4 s	3.1 ms	1 096×
40	4.7 s	3.1 ms	1 516×
50	6.0 s	3.2 ms	1 875×

Semi-Trusted Third Parties (STTPs). This approach is in particular effective if there is a very large number of participating data providers in the protocol. The best-case scenario for this setting is when ψ=N, i.e., each institution uploads a single genome.

(2) A more realistic approach is to keep one LUT for each database. This approach has the advantage that the number of data providers is commonly not very large, e.g., ten big institutions is a realistic scenario. Since the STTPs know how many and which sequences came from which data provider, this approach does not violate privacy of the protocol by using predefined LUTs for particular genome sequences. Using this approach, only ψ comparisons with LUTs have to be performed, which is a small overhead if the number of sequences N is large. Due to the performance advantages over other options, we choose this approach in our SSQ protocol.

(3) The least realistic approach is to aggregate LUTs of multiple data providers. For this, the institutions count the frequencies of *all possible* alleles for a block in their database (the most commonly used block size in [AHLR18] is 16). For example, a frequency table for a 16-allele block would yield

4^{16}=4.3 billion values. These values first have to be aggregated from multiple databases. Afterwards, the w (width of the LUT) maxima have to be found from the dataset, e.g., using the k-Nearest Neighbors (k-NN) algorithm. This approach is performed for each block (there are 15 million blocks in a whole-genome sequence). Therefore, this approach is impractical and we do not discuss it further.

5 EVALUATION

We implemented all our protocols using the ABY framework [DSZ15]. We run our two Semi-Trusted Third Parties (STTPs) each on a standard PC equipped with an Intel Core i7-4770K 3.5 GHz processor and 32 GB RAM. They are connected via a 1 Gbit/s network with an average latency of 0.1 ms. All protocols are instantiated with a symmetric security parameter of 128 bit. We intentionally exclude the evaluation part for the preprocessing (extensively discussed in [AHLR18]) and aggregation of the databases (extensively discussed in [TWSH18]). Furthermore, the initialization phase (aggregation of the databases) is a negligible *one-time* expense and securing the communication channels using TLS does not add any significant overhead. Correctness and accuracy of the algorithms of Asharov et al. on real genomic data was shown in [AHLR18, Sect. 5]. Therefore, we use artificial data in our benchmarks because the performance of SMPC depends only on the size of the data, but not on concrete values.

5.1 Comparison with Cheng et al. [CHW18]

The most recent related work that covers privacy-preserving Edit Distance (ED) computations in the outsourcing scenario is [CHW18]. Unfortunately, the authors give the exact run-times only for the ED computation between two sequences in Table 3 of this paper. We compare our ED run-times in the system setting of [CHW18], i.e., the outsourcing scenario, with the run-times of their most efficient protocol. We achieve a significant run-time improvement over [CHW18] of 500× to 1 800× (see Table 3).

We can increase this even further when many sequences are computed in parallel. For this, we (optimistically for [CHW18]) approximate the benchmarking results of [CHW18, Figure 4 (a)] in Table 4 and in Figure 4 in Appendix A. As a result of the comparison, our algorithm outperforms that of Cheng et al. [CHW18] by more than factor 24 000× in run-time and by more than factor 16× in communication.

5.2 Comparison with Asharov et al. [AHLR18]

Here, we compare our privacy-preserving algorithm for ED and Similar Sequence Query (SSQ) with [AHLR18].

In Table 2 and in Figure 4 in Appendix A, we compare our algorithms in the benchmark setting of [AHLR18, Table 3], where both parties, client and server, are run on one machine (our protocol can trivially be applied for direct STPC between the client and server). In addition, we benchmark our algorithm in the LAN setting.

The authors of [AHLR18] do not detail their hardware setting, whereas we benchmark our algorithm on commodity hardware. Our algorithm outperforms that of Asharov et al. [AHLR18] by factor 5-6× in run-time which is due to more light-weight building blocks. Furthermore, our algorithm is still by factor 3× faster even on a LAN, compared to the localhost benchmarks of [AHLR18]. The communication of our algorithm is slightly lower. This is due to the more efficient C-OTs instead of general OTs (see Section 4.2) and a more efficient algorithm for finding k most similar sequences (see Section 3.3.2).

5.3 Batching the Execution for Large-Scale Benchmarks

For some of our large-scale benchmarks, we split the execution into multiple steps (i.e., we evaluate subcircuits instead of the entire circuit) because our STTPs ran out of RAM. We split our algorithms in a black-box way, i.e., without modifying the primitive protocols such as distance computation and k-Nearest Neighbors (k-NN). For the distance computation, this only increases the number of communication rounds because the computation is independent for each SNP, whereas for the k-NN algorithm we have to perform additional computation because the result depends on all input elements, which, however, turns out to be very cheap in terms of computation and does not add any significant overhead to the overall protocol. For example, for one million input genome sequences in total in the k-NN protocol with k=10 and 100 000 batch size, we have to perform 10 iterations with 100 000 input size and one iteration of k-NN on the joint output of size $10 \cdot k = 100$. Since k-NN has linear complexity in the number of inputs, the total overhead in the circuit size is $\sim 100/1\,000\,000 = 0.000\,1\,\%$, which is negligible. This also does not violate privacy, because the data and intermediate results all remain in secret-shared form and, hence, leak no information.

5.4 Large-Scale Benchmarks

For our large-scale benchmarks on thousands to millions of genomes, we define global parameters of the block size b=5, padded block size b'=16, number of data providers ψ=10, and the width of the LUT ω=30 (for better accuracy). The results of the benchmarks are given in Table 5 and in Figure 5 in Appendix A. As can be seen in the table, practical large-scale privacy-preserving SSQs on sequences of medium lengths are possible. For the sequence lengths n=1 K–10 K and any number of sequences, and n=100 with the number of sequences N=100 K, the run-times are always in the order of seconds. For all other parameters, the run-times are in the order of minutes (even for databases with N=1 M sequences). A few minutes is a reasonable delay in practice for SSQ which shows real-world applicability of our protocol to large-scale SSQ.

5.5 Whole-Genome Benchmarks

For our whole-genome benchmarks, we set the genome sequence length to n=75 M (the same as in [WHZ+15]) and the LUT width to $\omega \in \{10, 20\}$. As shown in [CHW18], a LUT width reduction slightly reduces accuracy, but significantly reduces the communication and computation of the protocol. The results of our benchmarks are given in Table 6 and in Figure 6 in Appendix A.

As can be seen in the table, running our protocol on a few hundred whole-genome sequences is practical. For example, a protocol run on up to N=1 000 sequences takes just a few hours. However, if we extrapolate the results to the dataset of [WHZ+15] with N=1 M sequences, we would require months to execute the protocol. Thus,

Table 4: Run-time and communication comparison of our Similar Sequence Query algorithm with that of Cheng et al. [CHW18] with sequence length n=500, Look-Up Table width ω=20, block size b=5, number of blocks t=20, number of most similar sequences k=10, and number of sequences N. A plot is given in Figure 4 in Appendix A

N	Run-time (LAN)			Communication		
	[CHW18]	Ours	Improvement	[CHW18]	Ours	Improvement
100	25 min	62 ms	24 193×	50 MB	3 MB	16×
200	50 min	96 ms	31 250×	100 MB	4 MB	25×
300	80 min	132 ms	36 363×	150 MB	5 MB	30×
400	105 min	171 ms	36 842×	200 MB	6 MB	33×
500	135 min	207 ms	39 130×	250 MB	7 MB	35×

Table 5: Large-scale benchmarks of our Similar Sequence Query algorithm for N sequences of length n, LUT width ω=30, number of data providers ψ=10, number of most similar sequences k=10, block size b=5, and padded block size b'=16. A plot is given in Figure 5 in Appendix A.

N	n	Run-time	Communication
1 K	100	0.5 s	21.1 MB
10 K	100	3.6 s	138.9 MB
100 K	100	24.2 s	1.4 GB
1 M	100	4.0 min	14.9 GB
1 K	1 K	1.2 s	129.5 MB
10 K	1 K	5.9 s	457.8 MB
100 K	1 K	1.1 min	3.9 GB
1 M	1 K	8.3 min	38.8 GB
1 K	10 K	8.5 s	1.2 GB
10 K	10 K	22.4 s	3.5 GB
100 K	10 K	4.1 min	26.6 GB
1 M	10 K	39.6 min	257.2 GB

Table 6: Run-times and communication for Similar Sequence Query on whole-genome genome sequences based on the computation of sub-sequences of smaller lengths with the following parameters: number of sequences N, Look-Up Table widths ω, sequence length n=75 M, block size b=5, padded block size b'=16, number of most similar sequences k=10, number of data providers ψ=10, number of blocks t=15 M, and bit-length of the distances β=$\lceil \log_2(tb') \rceil$=28. A plot is given in Figure 6 in Appendix A.

N	ω	Run-time	Communication
10	10	2.9 h	2.3 TB
100	10	3.2 h	2.4 TB
1 000	10	6.8 h	3.5 TB
10 000	10	1.2 d	14.3 TB
10	20	5.7 h	4.6 TB
100	20	6.3 h	4.8 TB
1 000	20	12.5 h	7.0 TB
10 000	20	2.4 d	28.6 TB

we propose either to use our protocol for whole-genome runs with relatively small databases (a few hundred sequences) or to use high-performance hardware.

6 CONCLUSION

In this work, we designed, implemented, and evaluated EPISODE, a scalable protocol for distributed privacy-preserving Similar Sequence Queries (SSQs), which outperforms the state of the art by orders of magnitude. Our protocol for SSQ is based on the approximation of Edit Distance (ED) computation of [AHLR18]. SSQ is performed on two Semi-Trusted Third Parties (STTPs) that obliviously compute indices of the k most similar sequences to the client's query. Our protocol is not only scalable, but it also substantially reduces the amount of communication and computation of the client. We implement our protocol using a mix of generic SMPC protocols and Correlated Oblivious Transfer (C-OT), which (i) improves the efficiency of our SSQ protocol by computing its parts using techniques that are most efficient for the particular tasks, which gives a greater than 20 000× speed-up compared to the most recent work of Cheng et al. [CHW18], and (ii) extend the protocol of Asharov et al. [AHLR18] for outsourcing while reducing its communication and computation overhead.

Acknowledgements. This work has been co-funded by the DFG as part of project E4 within the CRC 1119 CROSSING, and by the German Federal Ministry of Education and Research (BMBF) and the Hessen State Ministry for Higher Education, Research and the Arts (HMWK) within CRISP.

REFERENCES

[AAAM17] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. Secure approximation of edit distance on genomic data. In *BMC Medical Genomics*, 2017.

[ADS+17] Gilad Asharov, Daniel Demmler, Michael Schapira, Thomas Schneider, Gil Segev, Scott Shenker, and Michael Zohner. Privacy-preserving interdomain routing at Internet scale. In *Privacy Enhancing Technologies Symposium (PETS)*, 2017.

[AHLR18] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. Privacy-preserving search of similar patients in genomic data. In *Privacy Enhancing Technologies Symposium (PETS)*, 2018.

[AKD03] Mikhail J. Atallah, Florian Kerschbaum, and Wenliang Du. Secure and private sequence comparisons. In *ACM Workshop on Privacy in the Electronic Society (WPES)*, 2003.

[AL05] Mikhail J. Atallah and Jiangtao Li. Secure outsourcing of sequence comparisons. In *International Journal of Information Security*, 2005.

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

[Ayd16] Erman Ayday. Cryptographic solutions for genomic privacy. In *Financial Cryptography and Data Security (FC)*, 2016.

[BDK+18] Niklas Büscher, Daniel Demmler, Stefan Katzenbeisser, David Kretzmer, and Thomas Schneider. HyCC: Compilation of hybrid protocols for practical secure computation. In *CCS*, 2018.

[BDST20] Lennart Braun, Daniel Demmler, Thomas Schneider, and Oleksandr Tkachenko. MOTION - a framework for mixed-protocol multi-party computation. Cryptology ePrint Archive, 2020. https://ia.cr/2020/1137.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In *ACM Symposium on Theory of Computing (STOC)*, 1996.

[BHKR13] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Philip Rogaway. Efficient garbling from a fixed-key blockcipher. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[BKLS18] Dan Bogdanov, Liina Kamm, Sven Laur, and Ville Sokk. Implementation and evaluation of an algorithm for cryptographically private principal component analysis on genomic data. *Computational Biology and Bioinformatics*, 2018.

[BMA+18] Charlotte Bonte, Eleftheria Makri, Amin Ardeshirdavani, Jaak Simm, Yves Moreau, and Frederik Vercauteren. Towards practical privacy-preserving genome-wide association study. *BMC Bioinformatics*, 2018.

[CDC+17] Marco Chiesa, Daniel Demmler, Marco Canini, Michael Schapira, and Thomas Schneider. SIXPACK: Securing internet exchange points against curious onlookers. In *International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, 2017.

[CHW18] Ke Cheng, Yantian Hou, and Liangmin Wang. Secure similar sequence query on outsourced genomic data. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[Cor18] Zoë Corbyn. How taking a home genetics test could help catch a murderer, 2018.

[CWB18] Hyunghoon Cho, David J Wu, and Bonnie Berger. Secure genome-wide association analysis using multiparty computation. *Nature biotechnology*, 2018.

[DHSS17] Daniel Demmler, Kay Hamacher, Thomas Schneider, and Sebastian Stammler. Privacy-preserving whole-genome variant queries. In *Cryptology and Network Security (CANS)*, 2017.

[DKS+17] Ghada Dessouky, Farinaz Koushanfar, Ahmad-Reza Sadeghi, Thomas Schneider, Shaza Zeitouni, and Michael Zohner. Pushing the communication barrier in secure computation using lookup tables. In *Network and Distributed System Security Symposium (NDSS)*, 2017.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - a framework for efficient mixed-protocol secure two-party computation. In *Network and Distributed System Security Symposium (NDSS)*, 2015.

[Eur17] European Bioinformatics Institute. Genome reference consortium human build 38, Ensembl release 91. http://dec2017.archive.ensembl.org/Homo_sapiens/Info/Annotation, 2017.

[FPI16] Roger Allan Ford and W. Nicholson Price II. Privacy and accountability in black-box medicine. *Michigan Telecommunications and Technology Law Review*, 2016.

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game. In *ACM Symposium on Theory of Computing (STOC)*, 1987.

[HAHT15] Mathias Humbert, Erman Ayday, Jean-Pierre Hubaux, and Amalio Telenti. On non-cooperative genomic privacy. In *Financial Cryptography and Data Security (FC)*, 2015.

[IKNP03] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *Annual International Cryptology Conference (CRYPTO)*, 2003.

[JKS08] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2008.

[JLL+19] Kimmo Järvinen, Helena Leppäkoski, Elena Simona Lohan, Philipp Richter, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. PILOT: Practical privacy-preserving Indoor Localization using OuTsourcing. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2019.

[Jon10] Thomas Jones. The rise of DNA analysis in crime solving, 2010.

[KR11] Seny Kamara and Marina Raykova. Secure outsourced computation in a multi-tenant cloud. In *IBM Workshop on Cryptography and Security in Clouds*, 2011.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2008.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, 1966.

[MHM17] Md Safiur Rahman Mahdi, Mohammad Zahidul Hasan, and Noman Mohammed. Secure sequence similarity search on encrypted genomic data. In *IEEE/ACM Conference on Connected Health: Applications, Systems and Engineering Technologies*, 2017.

[MMDC19] Alexandros Mittos, Bradley Malin, and Emiliano De Cristofaro. Systematizing genome privacy research: A privacy-enhancing technologies perspective. 2019.

[MNPS04] Dahlia Malkhi, Noam Nisan, Benny Pinkas, and Yaron Sella. Fairplay-secure two-party computation system. In *USENIX Security Symposium*, 2004.

[NNOB12] J. B. Nielsen, P. S. Nordholt, C. Orlandi, and S. S. Burra. A new approach to practical active-secure two-party computation. In *Annual International Cryptology Conference (CRYPTO)*, 2012.

[NP01] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Symposium on Discrete Algorithms (SODA)*, 2001.

[PF16] Alice B. Popejoy and Stephanie M. Fullerton. Genomics is failing on diversity. *Nature News*, 2016.

[PSSY20] Arpita Patra, Thomas Schneider, Ajith Suresh, and Hossein Yalame. ABY2.0: Improved mixed-protocol secure two-party computation. In *USENIX Security*, 2020.

[PSSZ15] Benny Pinkas, Thomas Schneider, Gil Segev, and Michael Zohner. Phasing: Private set intersection using permutation-based hashing. In *USENIX Security Symposium*, 2015.

[SHSK15] Ebrahim M. Songhori, Siam U. Hussain, Ahmad-Reza Sadeghi, and Farinaz Koushanfar. Compacting privacy-preserving k-nearest neighbor search using logic synthesis. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[SLH+17] João Sá Sousa, Cédric Lefebvre, Zhicong Huang, Jean Louis Raisaro, Carlos Aguilar-Melchor, Marc-Olivier Killijian, and Jean-Pierre Hubaux. Efficient and secure outsourcing of genomic data storage. *BMC Medical Genomics*, 2017.

[ST18] Thomas Schneider and Oleksandr Tkachenko. Episode: Efficient privacy-preserving similar sequence queries on outsourced genomic databases. In *Workshop on Privacy in the Electronic Society (WPES)*, 2018.

[ST19] Thomas Schneider and Oleksandr Tkachenko. EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs. In *ASIACCS*. ACM, 2019.

[SZ13] Thomas Schneider and Michael Zohner. GMW vs. Yao? Efficient secure two-party computation with low depth circuits. In *Financial Cryptography and Data Security (FC)*, 2013.

[TWSH18] Oleksandr Tkachenko, Christian Weinert, Thomas Schneider, and Kay Hamacher. Large-scale privacy-preserving statistical computations for distributed genome-wide association studies. In *ACM Asia Conference on Computer and Communications Security (ASIACCS)*, 2018.

[Wet17] Kris A. Wetterstrand. DNA sequencing costs: Data from the NHGRI Genome Sequencing Program (GSP), 2017. http://www.genome.gov/sequencingcostsdata.

[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 *ACM SIGSAG Conference on Computer and Communications Security (CCS)*, 2015.

[WSLH17] Bing Wang, Wei Song, Wenjing Lou, and Y. Thomas Hou. Privacy-preserving pattern matching over encrypted genetic data in cloud computing. In *IEEE Conference on Computer Communications (INFOCOM)*, 2017.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *Foundations of Computer Science (FOCS)*, 1986.

[ZH17] Ruiyu Zhu and Yan Huang. Efficient privacy-preserving general edit distance and beyond. Cryptology ePrint Archive, Report 2017/683, 2017. https://ia.cr/2017/683.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2015.

A VISUALIZED BENCHMARKS

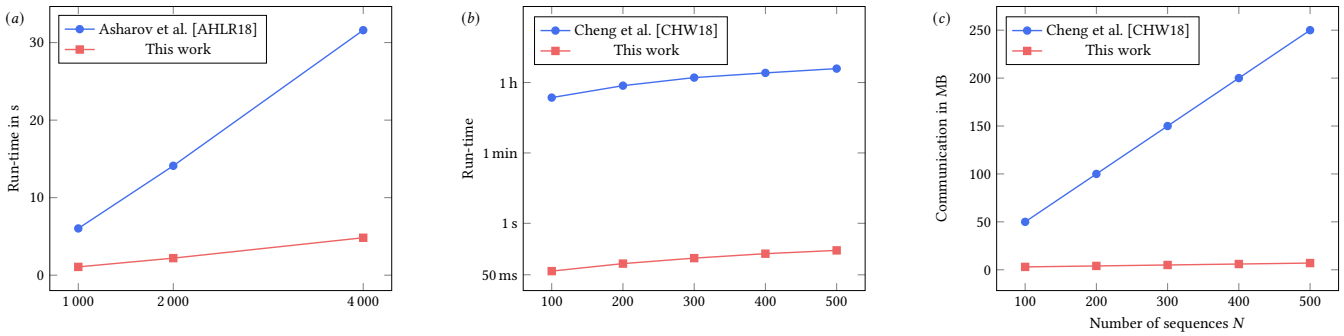

Figure 4: (a) A run-time comparison of our Similar Sequence Query algorithm with that of Asharov et al. [AHLR18] with sequence length n=3 470, number of nearest sequences k=5, block length b=4, padded block length b'=16, and number of data providers ψ=1 performed locally for different numbers of sequences N and LUT widths ω. (b,c) A run-time and communication comparison of our SSQ algorithm with that of Cheng et al. [CHW18] with n=500, b=5, k=10, ω=20, and number of blocks t=20. Numbers are given in Tables 2 (a) and 4 (b,c).

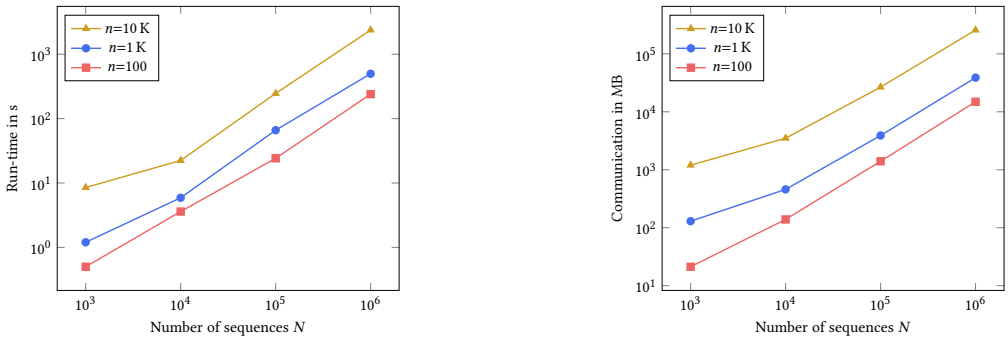

Figure 5: Large-scale benchmarks of our Similar Sequence Query algorithm for N sequences of length n, LUT width w=30, number of data providers ψ=10, number of most similar sequence queries k=10, block size b=5, and padded block size b'=16. Numbers are given in Table 5.

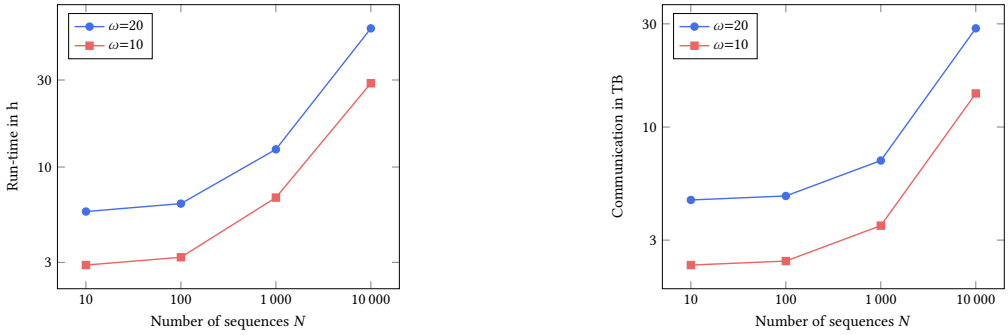

Figure 6: Run-times and communication for Similar Sequence Query on whole-genome genome sequences based on the computation of sub-sequences of smaller lengths with the following parameters: number of sequences N, Look-Up Table widths ω, sequence length n=75 M, block size b=5, padded block size b'=16, number of most similar sequences k=10, number of data providers ψ=10, number of blocks t=15 M, and bit-length of the distances β=$\lceil \log_2(tb') \rceil$=28. Numbers are given in Table 6.