Privacy-preserving edit distance computation using secret-sharing two-party computation

Hernán Darío Vanegas Madrigal[1], Daniel Cabarcas Jaramillo[1⋆], and Diego F. Aranha[2]

[1] Universidad Nacional de Colombia, Sede Medellín
{hdvanegasm, dcabarc}@unal.edu.co
[2] Aarhus University, Denmark
dfaranha@cs.au.dk

Abstract. The edit distance is a metric widely used in genomics to measure the similarity of two DNA chains. Motivated by privacy concerns, we propose a 2PC protocol to compute the edit distance while preserving the privacy of the inputs. Since the edit distance algorithm can be expressed as a mixed-circuit computation, our approach uses protocols based on secret-sharing schemes like Tinier and SPD$\mathbb{Z}_{2^k}$; and also daBits to perform domain conversion and edaBits to perform arithmetic comparisons. We modify the Wagner-Fischer edit distance algorithm, aiming at reducing the number of rounds of the protocol, and achieve a flexible protocol with a trade-off between rounds and multiplications. We implement our proposal in the MP-SPDZ framework, and our experiments show that it reduces the execution time respectively by 81% and 54% for passive and active security with respect to a baseline implementation in a LAN. The experiments also show that our protocol reduces traffic by two orders of magnitude compared to a BMR-MASCOT implementation.

Keywords: edit distance · secure MPC · secret-sharing schemes

1 Introduction

Given an alphabet of symbols Σ, the edit distance between two strings in Σ^* is the minimum cost of a sequence of editing operations (insertions, deletions, or substitutions) to transform one string into the other [28]. Intuitively, the smaller the edit distance between two strings, the more similar they are. Algorithms to compute the edit distance have been studied for many years and the most popular are based on dynamic programming, such as the Wagner-Fischer algorithm [29]. Such algorithms are useful in genomics, where the similarity between two gene sequences is used in disease diagnosis and treatment [32]. On a typical scenario, millions of *reads* from a subject's DNA are compared to a reference for alignment, with read lengths ranging from a few hundred to a few million bases [23].

Despite the benefits of computing similarities in genomic data, there are risks that come from revealing such information. One of the main risks is called *re-identification*, where a subject can be identified from its genomic data [21]. There

⋆ The author was partially supported by the CyTeD program grant 522RT0131.

are other concerns like *ancestry identification*, where an individual can identify their ancestors from genomic data; and the so-called *attribute disclosure attacks via DNA*, where an attacker can detect a sensible attribute about someone from their DNA sample and a database of attribute-related samples [13].

These concerns motivate the application of privacy-preserving computation in the following scenario: Alice and Bob are connected via a secure communication channel and each has a DNA chain represented as a list of nucleotides. They want to compute the edit distance of both chains, but without revealing their chain to each other. We will accomplish this task by evaluating the Wagner-Fischer (WF) algorithm without revealing the inputs. The WF algorithm is a dynamic programming solution to find the edit distance $d(A, B)$ between two chains $A = (a_1, \ldots, a_n)$ and $B = (b_1, \ldots, b_m)$. The core of the algorithm is to compute a matrix D for which the recursion holds (for $1 \leq i \leq n$ and $1 \leq j \leq m$):

$$D(i,j) = \min \begin{cases} D(i-1,j)+1, \\ D(i,j-1)+1, \\ D(i-1,j-1)+t(i,j) \end{cases}, \text{ with } t(i,j) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } a_i \neq b_j \\ 0 & \text{otherwise} \end{cases}.$$

In this algorithm, two operations have high relevance for security: (i) the computation of t requires a secure equality test between a pair of symbols in the chains; (ii) the computation of the minimum requires secure comparison between integers, which in turn needs the extraction of their most significant bit.

These challenges can be solved using various cryptographic techniques. Particularly, we will focus on *secure multi-party computation* (MPC). In an MPC protocol, a set of parties, each one holding part of the input of a function, want to compute such function while preserving the privacy of the inputs. To achieve their goal, the parties exchange messages and perform local computations. In the end, the parties may obtain the correct result, and the messages exchanged between them are guaranteed not to reveal any information about their inputs.

Most previous works in secure computation of edit distance employ garbled circuits as the MPC protocol because of their good performance in bit-wise operations [18, 12, 31]. Another class of MPC protocol based on *secret-sharing schemes* (SSS) is efficient for arithmetic operations [14], but it was rarely used for this problem [26]. Since recent advances in protocols based on SSS allow efficient transformation between an arithmetic and a binary domain [1, 25, 14, 11], and since the WF algorithm has significant mixed computation, designing an efficient MPC solution based on secret-sharing should be possible.

Related work. Most current and past research in secure computation of edit distance are based on homomorphic encryption (HE) and garbled circuits (GC) (see surveys at [12] and [22]). In the case of HE, Zheng et al. [32] propose an architecture where the data owners outsource the computation of edit distance. They ensure privacy by using a modified version of the Paillier cryptosystem [5], but their protocol allows one of the parties to know the DNA chain of the other party and compute the edit distance between blocks in the clear to improve performance. For dynamic programming approaches, Rane and Sun [24] compute

the minimum between three elements using HE. Cheon et al. [7] take the idea further and compute the minimum of a list of numbers to reduce circuit depth. However, they do not prove correctness and optimality of all their techniques, and focus on same-length strings. We extend their strategy to solve both problems.

Another technique actively used to compute edit distance is GC, and protocols derived from Yao's GC are widely used to implement dynamic programming approaches [12]. Jha et al. [18] is among the first works, and they use one circuit for each basic operation in the algorithm: increment, minimum, and equality test. Further theoretical and practical work in [12, 31] improves security, memory usage, communication complexity, and specialized hardware to increase parallelism. More recently, Zhu and Huang [33] use GC to compute the edit distance in both active and passive threat models, and claim to outperform the best previous GC-based protocols. As in our work, they consider the secure computation of the WF algorithm and exploit the structure of the minimization problem to find better bounds to improve performance, but they do not report performance measurements in the actively secure setting. Other works consider an approximation version of the edit distance problem to improve performance [3].

Compared to HE and GC, protocols based on secret-sharing techniques are less common for edit distance. Rane and Sun [24] use additive secret-sharing alongside HE, but they do not rely on secret-sharing to perform the operations. EPISODE by Schneider et al. [26] is the closest to our techniques. They use ideas from [2] to compute an approximation of the edit distance using the ABY framework [11]. ABY allows designing protocols using mixed-circuit computation against passive adversaries, so they can compute parts of their protocol in binary or arithmetic domains, moving secrets from one domain to another. There are significant differences between this and our work: (i) they only consider security against passive adversaries, while we also explore active adversaries; (ii) they improve performance by approximating the edit distance, while we focus on the exact problem; (iii) they consider a different security setup, aligning multiple sequences to a publicly known reference genome.

Contributions. We propose a 2PC protocol to compute the edit distance privately. More parties are possible, but the scenario naturally suggests two parties. We apply recently developed MPC protocols based on secret-sharing schemes such as SPD$\mathbb{Z}_{2^k}$ [9] and Tinier [15], and protocols such as daBits [25] and edaBits [14]. To the best of our knowledge, we are the first to propose a solution to secure edit distance using these techniques. We divide the WF algorithm into two parts: the preamble in charge of computing the matrix t, and the arithmetic section to compute the matrix D. We optimize each part separately.

For the computation of t, we encode the nucleotides using a binary representation, and we propose a protocol to compute the equality test between a pair of nucleotides through bit-wise operations using Tinier. Once we compute t, we obtain binary shares of each possible value of the function, and we use daBits to transform such binary shares into arithmetic shares for the arithmetic section.

For the arithmetic part, we generalize the ideas presented by Cheon et al. [7] in two directions. First, we expand the recursions from the WF algorithm to

compute D not as the minimum of three numbers, but of a longer list of numbers. This allows us to divide D into sub-boxes, such that it takes fewer rounds to compute them. However, this strategy also increases the number of multiplications and comparisons in the protocol, raising a trade-off between execution time and communication. This trade-off is studied both theoretically and empirically. For comparison, Cheon et al. consider a sub-box that matches the size of D and focus only on equal-length chains. We generalize their work for sub-boxes of arbitrary size, which works for DNA chains with different lengths.

As part of this generalization, we propose an algorithm to automatically generate the equations to compute each sub-box. This algorithm arises from representing the recursions of the WF algorithm as a graph. Using this representation, we prove both the correctness and the optimality of the generation. We must point out that Cheon et al. use a different graphical method to compute their own equations, for which they do not prove correctness or optimality.

We perform experimental evaluations of our method using the MP-SPDZ framework, and analyze the performance trade-off as the size of the sub-box increases. We show that our algorithm has a significant reduction in the execution time in a LAN compared to a naive implementation of the WF algorithm. Additionally, we find that our protocol is competitive with the techniques currently used to solve the edit distance problem, like Yao's garbled circuits, and outperforms techniques like HE and BMR [4]. Moreover, we empirically show that protocols in $\mathbb{Z}_{2^k}$ are best-suited for our implementation and we give supporting arguments. Our complete source code can be found in a GitHub repository[3].

Organization. Section 2 covers a formal definition of edit distance, the WF algorithm, a background on MPC and other building blocks. In Section 3, we compute edit distance using MPC protocols based on secret-sharing and perform complexity analysis. In Section 4, we show an algorithm based on graph theory to obtain the minimal number of terms as parameters of the minimum function to compute edit distance correctly. Finally, in Section 5, we present a performance evaluation of our solution and compare it with the current state-of-the-art.

2 Preliminaries

2.1 The edit distance problem

The edit distance is the minimum-weight series of operations that transforms one string into the other. Formally, let $A \stackrel{\text{def}}{=} (a_1, \ldots, a_n)$ be a string over an alphabet Σ. We define the possible *editing operations* on A:

1. *delete* the i-th position to obtain $(a_1, \ldots, a_{i-1}, a_{i+1}, \ldots, a_n)$.
2. *insert* $b \in \Sigma$ at position $(i+1)$ to obtain $(a_1, \ldots, a_i, b, a_{i+1}, \ldots, a_n)$.
3. *change* position i to $b \in \Sigma$ to obtain $(a_1, \ldots, a_{i-1}, b, a_{i+1}, \ldots, a_n)$.

[3] https://github.com/hdvanegasm/sec-edit-distance

Given $A, B \in \Sigma^*$, the *edit distance problem* consists in finding the sequence of editing operations to transform A into B that minimizes the sum of the costs of the operations. We assume that each editing operation costs 1, and we are only interested in computing the minimum cost, and not in the operations.

To solve this problem, Wagner and Fischer propose a dynamic programming algorithm [29]. Let $A \stackrel{\text{def}}{=} (a_1, \ldots, a_n)$ and $B \stackrel{\text{def}}{=} (b_1, \ldots, b_m)$ be two strings in Σ^*. For $i \in [n]$ the set $\{1, 2, \ldots, n\}$, denote the sub-string $A^{(i)} \stackrel{\text{def}}{=} (a_1, a_2, \ldots, a_i)$ and the edit distance between $A^{(i)}$ and $B^{(j)}$ by $D(i, j)$[4]. The goal is thus to find $D(n, m)$. Wagner and Fischer propose Algorithm 1 and prove its correctness.

Algorithm 1 Edit distance algorithm

Input: two chains $A = (a_1, \cdots, a_n)$ and $B = (b_1, \cdots, b_m)$.
Output: an integer value with the edit distance between the chains A and B.

Let t be an $n \times m$ matrix with indexes starting from one.
for $(i, j) \in [n] \times [m]$ **do**
    if $a_i \neq b_j$ **then** $t(i, j) = 1$
    else $t(i, j) = 0$
Let D be an $(n + 1) \times (m + 1)$ zero-initialized matrix, indexes starting from zero.
for $i = 0$ to n **do** $D(i, 0) = i$
for $j = 0$ to m **do** $D(0, j) = j$
for $i = 1$ to n **do**
    for $j = 1$ to m **do**

$$D(i, j) = \min \begin{cases} D(i-1, j) + 1, \\ D(i, j-1) + 1, \\ D(i-1, j-1) + t(i, j) \end{cases} \tag{1}$$

return $D(n, m)$

2.2 Multi-party computation and secret-sharing schemes

In a secure multi-party computation (MPC) protocol, parties $P_1, \ldots, P_n$ jointly compute the value of $f(x_1, \ldots, x_n)$, where f is a fixed publicly known function and P_i holds the value x_i. During the computation, parties exchange messages and perform local computations such that there is no leakage of information about the parties' inputs, except for the function output.

The security of an MPC protocol can be stated and proven using techniques like universal composability (UC) [6]. One assumes the existence of an adversary that corrupts a subset of parties. An adversary can be *passive* or *active*. In the former, it tries to learn information from the exchanged messages but it does not deviate from the protocol specification. In the latter, the adversary can deviate from the protocol to obtain information about the parties' inputs or to prevent the honest parties from learning the correct output of the function.

[4] We will occasionally replace the parentheses with a subscript for the matrices D and t. That is, $D(i, j)$ will be written as $D_{i,j}$ and $t(i, j)$ as $t_{i,j}$.

A particular type of MPC protocols are those based on SSS, where a secret s is split into n parts, called *shares*, such that any subset of at most t shares reveal no information about s, but s can be completely reconstructed from any set of at least $t+1$ shares [8]. If s is secret-shared among the parties using shares $s^{(1)}, \ldots, s^{(n)}$, where P_i holds $s^{(i)}$, we denote this by $[\![s]\!] \stackrel{\text{def}}{=} \left(s^{(1)}, \ldots, s^{(n)}\right)$.

Even though our edit distance solution can be instantiated in various ways[5], we concretely consider two particular SSS-based MPC protocols: SPD$\mathbb{Z}_{2^k}$ [9] with algebraic domain $\mathbb{Z}_{2^k}$, and Tinier [15] with algebraic domain $\mathbb{Z}_2$. To make clear the domain of computation in which the shares live, we distinguish shares of SPD$\mathbb{Z}_{2^k}$ from those of Tinier by respectively denoting them as $[\![s]\!]_{2^k}$ and $[\![s]\!]_2$. Both secret-sharing schemes are linear, meaning that additions and multiplications by public constants can be done without communication. However, the product of secret values is more involved, requiring communication among the parties and calls to subprotocols (like Open to reveal values to other parties). For security against active adversaries, both protocols use information-theoretic MACs to authenticate secret-shared values.

To securely compute a function f using a protocol based on an SSS, the function is considered as an arithmetic circuit. Initially, the parties distribute shares of their inputs. Then, using the protocols mentioned above, the parties evaluate the circuit so that each party holds a secret-shared value of the intermediate steps. At the end, the parties reconstruct the final result of the computation.

2.3 Domain conversions and comparisons

In our protocol for the edit distance, we need to compute two main operations securely: domain conversion and integer comparisons. These two operations can be done efficiently using daBits and edaBits. In the domain conversion, the goal is to convert binary Tinier shares $[\![x]\!]_2$ into arithmetic SPD$\mathbb{Z}_{2^k}$ shares $[\![x]\!]_{2^k}$, where $x \in \{0,1\}$. For that case, we use a daBit, which is a tuple $([\![r]\!]_2, [\![r]\!]_{2^k})$, where $r \in \mathbb{Z}_2$ is chosen at random. In [25], they propose a protocol to generate daBits that is secure against malicious adversaries, which is improved later in [1]. We can perform a domain conversion using techniques presented in [10] which can be adapted to the case where daBits are generated in a pre-processing phase.

To compute integer comparisons efficiently, we use edaBits. An edaBit is a tuple of m binary secret-shared random bits $([\![r_{m-1}]\!]_2, \ldots, [\![r_0]\!]_2)$ along with shares $[\![r]\!]_{2^k}$, such that $r = \sum_{i=0}^{m-1} r_i \cdot 2^i$. In [14], a protocol is proposed to generate edaBits that is secure against active adversaries. Also, a protocol is presented to compare integers using edaBits by expressing comparisons in terms of the extraction of the most significant bit of their binary representation.

It is worth mentioning that although SPD$\mathbb{Z}_{2^k}$ and Tinier are different protocols, they are compatible to compute domain conversions and comparisons using daBits and edaBits. This is because the protocols used to generate such random material model the MPC protocols as an arithmetic black box. Also, the protocols to generate daBits and edaBits are independent of the methods used by the MPC protocols based on secret-sharing schemes to authenticate shared values.

[5] Any MPC protocol that implements an $\mathcal{F}_{\text{edaBits}}$ functionality as described in [14].

3 A privacy-preserving solution using secret sharing

In this section, we present an efficient strategy to compute the edit distance using an SSS-based MPC protocol. Although the strategy works for any protocol based on linear secret-sharing schemes, we will aim at schemes whose computation domain is $\mathbb{Z}_{2^k}$. We build our protocol upon MPC schemes that implement an $\mathcal{F}_{\text{edaBits}}$ functionality, thus, its security follows from the security of the underlying scheme. We divide the task into two distinctive parts of the Algorithm 1, the *preamble* (lines 1–4), and the *arithmetic* part (lines 5–10).

The preamble of Algorithm 1 computes matrix t by comparing every pair of nucleotides. We propose to compare nucleotides in an efficient way using a binary domain. We will encode the nucleotides of a DNA chain using two elements of $\mathbb{Z}_2$ as $A \mapsto 00$, $C \mapsto 01$, $G \mapsto 10$, and $T \mapsto 11$. We denote the sharing of the nucleotide $N = \langle b_0, b_1 \rangle \in \mathbb{Z}_2^2$ as $[\![N]\!]_2 \stackrel{\text{def}}{=} \langle [\![b_0]\!]_2, [\![b_1]\!]_2 \rangle$. We extend the XOR operations to nucleotides $N = \langle b_0, b_1 \rangle$ and $N' = \langle b'_0, b'_1 \rangle$ by $N \oplus N' \stackrel{\text{def}}{=} \langle b_0 \oplus b'_0, b_1 \oplus b'_1 \rangle$, and extend it to shares in a natural way. Notice that $N = N'$ iff $N \oplus N' = 0$. To determine if a nucleotide is zero, we use the logical OR among its components. Denoting by $S(N) \stackrel{\text{def}}{=} b_0 \vee b_1 = (b_0 + b_1 + b_0 b_1) \mod 2$, we have that $N = N'$ iff $S(N \oplus N') = 0$. Hence, denoting by $N \stackrel{?}{=} N'$ a bit that indicates whether $N = N'$ or not, we can obtain a Boolean share of the assertion as

$$\left[\!\left[N \stackrel{?}{=} N'\right]\!\right]_2 = 1 - [([\![b_0]\!]_2 + [\![b'_0]\!]_2) + ([\![b_1]\!]_2 + [\![b'_1]\!]_2) + ([\![b_0]\!]_2 + [\![b'_0]\!]_2)([\![b_1]\!]_2 + [\![b'_1]\!]_2)].$$

Using this approach, we can compute the matrix t using mn multiplications in $\mathbb{Z}_2$. In total, we need to transmit $4nm$ bits through invocations of the Open protocol. Since the computation of each entry of t is independent of all other entries, we can compute them in parallel and the computation of the matrix only costs one round. Once the matrix t is computed, each entry is a binary share, so we will have to transform it to an arithmetic share for the next part.

3.1 Arithmetic part

After computing the matrix t, we use daBits to transform its entries into arithmetic shares. For the arithmetic part, we thus assume that the parties hold shares $[\![t(i,j)]\!]_{2^k}$ for each index (i,j). Also, following Algorithm 1, $D(i,0) = i$, for all $i \in [n]$, and $D(0,j) = j$, for all $j \in [m]$. Our goal is to compute shares of the bottom-right corner of the matrix D, namely, $[\![D(n,m)]\!]_{2^k}$.

It is possible to compute the entries of D using well-known protocols to compute comparisons between two signed integers (c.f [10]). The problem of this approach is the sequential dependency between the positions of the matrix. This dependency prevents us from parallelizing the process, which increases the number of rounds. To overcome this limitation we compute only some selected entries of the matrix. Our approach builds upon the ideas in [7] to compute the

edit distance using homomorphic encryption. We generalize their idea, fixing some issues, and we apply it to secret-sharing based protocols.

Let A and B be two DNA chains with lengths n and m, respectively. The matrix D will then have $n+1$ rows and $m+1$ columns. Applying Equation (1) from Algorithm 1 recursively for $D(i-1,j)$, $D(i,j-1)$, and $D(i-1,j-1)$, and removing identical formulas, we obtain that $D(i,j)$ is equal to the minimum of

$$\begin{array}{ll} D(i-2,j)+2, & D(i-2,j-1)+t(i-1,j)+1, \\ D(i-2,j-1)+3, & D(i-1,j-2)+3, \\ D(i-2,j-2)+t(i-1,j-1)+2, & D(i,j-2)+2, \\ D(i-1,j-2)+t(i,j-1)+1, & D(i-2,j-1)+t(i,j)+1, \\ D(i-1,j-2)+t(i,j)+1, & D(i-2,j-2)+t(i,j)+t(i-1,j-1). \end{array}$$

We can then remove some redundant formulas, which can be proven to be greater or equal to some other formula in the set. After systematically removing all of them, we obtain that $D(i,j)$ is equal to the minimum of the following list:

$$\begin{array}{ll} D(i-2,j)+2, & D(i-2,j-1)+t(i-1,j)+1, \\ D(i-2,j-1)+t(i,j)+1, & D(i-2,j-2)+t(i-1,j-1)+t(i,j), \\ D(i,j-2)+2, & D(i-1,j-2)+t(i,j-1)+1, \\ D(i-1,j-2)+t(i,j)+1. & \end{array}$$

We will explain this process further in Section 4. Notice that by systematically substituting occurrences of $D(i-1,j-1)$, we completely removed it from the equation. We can repeat recursively to write $D(i,j)$ in terms of formulas that include the value of positions of the matrix that lie on the border of a rectangle inside the matrix which bottom right corner is $D(i,j)$. More specifically and following the notation of [7, Section 4.3], define the $(\tau+1)$-box for $D(i,j)$ as the set comprised of the union of the following sets, with τ a positive integer:

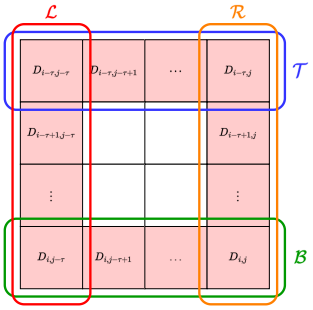
$$\mathcal{T} \stackrel{\text{def}}{=} \{D_{i-\tau,j-\tau}, D_{i-\tau,j-\tau+1}, \ldots, D_{i-\tau,j}\}\,,\ \mathcal{B} \stackrel{\text{def}}{=} \{D_{i,j-\tau}, D_{i,j-\tau+1}, \ldots, D_{i,j}\}\,,$$
$$\mathcal{L} \stackrel{\text{def}}{=} \{D_{i-\tau,j-\tau}, D_{i-\tau+1,j-\tau}, \ldots, D_{i,j-\tau}\}\,,\ \mathcal{R} \stackrel{\text{def}}{=} \{D_{i-\tau,j}, D_{i-\tau+1,j}, \ldots, D_{i,j}\}\,.$$

With these definitions, not just $D(i,j)$ but all the elements in $\mathcal{B} \cup \mathcal{R}$ can be written as the minimum of formulas that depend on positions in $\mathcal{T} \cup \mathcal{L}$. Figure 1a shows the positions in the $(\tau+1)$-box, with the sets $\mathcal{T}$, $\mathcal{R}$, $\mathcal{B}$ and $\mathcal{L}$ highlighted.

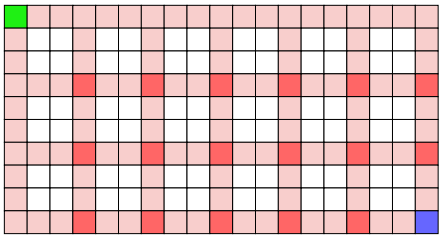
Continuing with the example for $\tau = 2$ and using the new notation for the borders of the box, we can compute positions $D(i-1,j)$ and $D(i,j-1)$ as for $D(i,j)$ using the following equations in terms of the positions in $\mathcal{T} \cup \mathcal{L}$:

$$D(i-1,j) = \min \begin{cases} D(i-2,j)+1 \\ D(i-2,j-1)+t(i-1,j) \\ D(i-2,j-2)+t(i-1,j-1)+1 \\ D(i-1,j-2)+2 \end{cases}, \tag{2}$$

$$D(i,j-1) = \min \begin{cases} D(i,j-2)+1 \\ D(i-1,j-2)+t(i,j-1) \\ D(i-2,j-2)+t(i-1,j-1)+1 \\ D(i-2,j-1)+2 \end{cases}. \tag{3}$$



(a) Positions inside a $(\tau+1)$-box.



(b) Division of the edit distance matrix into boxes.

Fig. 1: Positions in a $(\tau+1)$-box relative to $D(i,j)$ and its use to divide the edit distance matrix into boxes.

In [7], the authors use only one $(\tau+1)$-box, with $\tau = n$, to compute the edit distance of two chains of the same length. Going beyond the ideas in [7] we leave τ as a hyperparameter that can be specified by the user allowing a trade-off between the number of rounds and the data sent during the secure computation. Figure 1b shows an example of the complete matrix D divided into boxes. The green position corresponds to $D(0,0)$ and the blue position corresponds to $D(n,m)$. The light red positions are the positions that are needed to compute the position $D(n,m)$, and the dark red positions are the most expensive position to compute inside each $(\tau+1)$-box. Our approach also allow us to compute the edit distance between two DNA chains that do not have the same length.

To compute each minimum, we use the protocol MIN_q proposed in [10, 27]. It computes the minimum of a list of q numbers in $O(\log_2 q)\cdot(c_r+2)$ rounds, where c_r is the number of rounds of a comparison. It requires $q-1$ comparisons and $2q-2$ multiplications. As an example, for $\tau = 2$, we can compute securely a share of the position $D(i,j)$ by executing MIN_7 with the following list of arguments:

$$\begin{array}{ll}
[\![D(i-2,j)]\!]_{2^k}+2, & [\![D(i-2,j-1)]\!]_{2^k}+[\![t(i-1,j)]\!]_{2^k}+1 \\
[\![D(i-2,j-1)]\!]_{2^k}+[\![t(i,j)]\!]_{2^k}+1, & [\![D(i-2,j-2)]\!]_{2^k}+[\![t(i-1,j-1)]\!]_{2^k}+[\![t(i,j)]\!]_{2^k} \\
[\![D(i,j-2)]\!]_{2^k}+2, & [\![D(i-1,j-2)]\!]_{2^k}+[\![t(i,j-1)]\!]_{2^k}+1 \\
[\![D(i-1,j-2)]\!]+[\![t(i,j)]\!]_{2^k}+1 &
\end{array}$$

The shares $[\![D(i-1,j)]\!]_{2^k}$ and $[\![D(i,j-1)]\!]_{2^k}$ can be written similarly following the Equations (2) and (3), and using the protocol MIN_4.

Our method traverses left-to-right and top-to-down the $(\tau+1)$-boxes, computing, for each box, the positions in $\mathcal{B}\cup\mathcal{R}$ from the positions in $\mathcal{T}\cup\mathcal{L}$. At the end of the protocol, the parties will hold shares $[\![D(n,m)]\!]_{2^k}$, which is the share of the edit distance. They can then reveal it using the OPEN protocol.

We analyze the complexity of the arithmetic part of the protocol, assuming the stated complexity of the MIN_q functionality [27, Section 13.1.1]. In Section 4, we will present a method to calculate the formulas in each minimum computation and we will prove that the number of formulas in the minimum computation is bounded by $O(\tau\cdot 2^{3\tau})$. Assuming that τ divides both m and n, we need to

compute nm/τ^2 boxes. Given that we need to compute $2\tau - 1$ positions in each $(\tau+1)$-box, we are required to compute $\frac{nm}{\tau^2} \cdot (2\tau - 1)$ positions from D in total.

Note that all the positions in $\mathcal{B} \cup \mathcal{R}$ within one box can be computed in parallel since there is no dependency between them. This makes the term $2\tau - 1$ disappear from the round count. For two DNA chains of lengths n and m, we compute the edit distance in $O\left(\frac{nm}{\tau^2} \cdot (3\tau + \log_2 \tau) \cdot (c_r + 2)\right)$ rounds, $O\left(\frac{nm}{\tau^2} \cdot (\tau^2 \cdot 2^{3\tau} - \tau)\right)$ comparisons, and $O\left(\frac{nm}{\tau^2} \cdot (\tau^2 \cdot 2^{3\tau+1} - 2\tau)\right)$ multiplications.

In comparison with a straightforward implementation of the arithmetic part, we find that our method reduces the number of rounds by a factor of τ. However, the higher the τ, the higher the number of multiplications and comparisons, which increases the data sent in the protocol execution. This is a trade-off that should be considered according to the specific protocol and the network speed.

4 Automatic generation of formulas for edit distance

In this section, we present an algorithm to produce a correct and minimal set of formulas necessary to compute any positions in $\mathcal{B} \cup \mathcal{R}$ of the $(\tau+1)$-box in terms of the positions in the sets $\mathcal{T} \cup \mathcal{L}$. For example, with $\tau = 2$, applying Equation (1) recursively without removing any formula inside the minimum, we obtain Equation (3.1), which contains the formulas

$$D(i-2, j-1) + t(i-1, j) + 1 \quad \text{and} \quad D(i-2, j-1) + 3.$$

Since $t(i-1, j) \in \{0, 1\}$, for any $D(i-2, j-1)$ and $t(i-1, j)$, it holds that

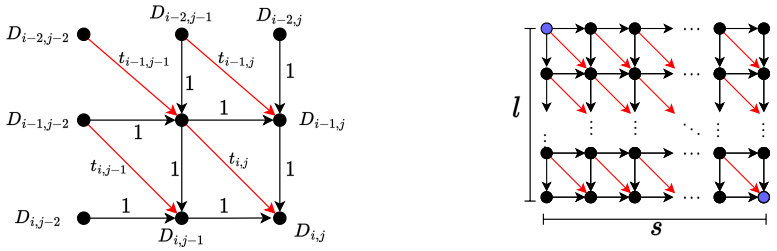
$$D(i-2, j-1) + t(i-1, j) + 1 \leq D(i-2, j-1) + 3.$$

Hence, we can remove $D(i-2, j-1)+3$ from the set of formulas without changing the overall result of the minimum function.

We use a directed labeled graph with colored edges to represent the direct dependencies among the entries of the matrix D as shown in Figure 2a. The vertices are the entries of the matrix and each edge represents the dependency given by Equation (1), labeled by the term to add in the formula. We color an edge black if the label is 1, and red otherwise. We will refer to the graph G constructed in this way for a $(\tau+1)$-box as the *dependency graph*. A similar abstraction was considered by Ukkonen in [28, Section 2] without colors.

The formulas inside a minimum to compute one entry of $\mathcal{B} \cup \mathcal{R}$ in terms of one in $\mathcal{T} \cup \mathcal{L}$ correspond to paths in the dependency graph.

Definition 1. *Let $V \in \mathcal{B} \cup \mathcal{R}$, $W \in \mathcal{T} \cup \mathcal{L}$ and P a path from W to V. The formula induced by P, is $f_P \stackrel{def}{=} W + a$, where a is the sum of all the labels of the edges in the path P. Each formula f_P will be called an unrolled formula from W to V and the set of all such formulas will be called the set of unrolled formulas from W to V.*



(a) Dependency graph for $\tau = 2$. (b) Delanoy graph.

Fig. 2: Graphs used in the automatic generation of formulas.

We can compute the position $V \in \mathcal{B} \cup \mathcal{R}$ in the matrix D as

$$V = \min\{f_P \mid P \text{ is a path from } W \text{ to } V \text{ in } G,\ \forall W \in \mathcal{T} \cup \mathcal{L}\},$$

which gives an equivalent representation of the unrolled formulas.

We now define the quantities that are key for removing redundant formulas.

Definition 2. *Let P and Q be two paths in the dependency graph G. We define $r_{P,Q}$ as the number of red edges in P that are not in Q. Also, we define b_P as the number of black edges in the path P.*

The following proposition states a criterion for removing redundant formulas.

Proposition 1. *Let $U \in \mathcal{T} \cup \mathcal{L}$, $W \in \mathcal{B} \cup \mathcal{R}$, let $\mathcal{P}_{U,W}$ be the set of all paths form U to W in a dependency graph G, and let P be any path in $\mathcal{P}_{U,W}$. We can remove the formula induced by P from the set of unrolled formulas from U to W without changing the overall value of the minimum, if $r_{Q,P} + b_Q \leq b_P$, for some $Q \in \mathcal{P}_{U,W} \setminus \{P\}$.*

Proof. Let $P \in \mathcal{P}_{U,W}$ be any path, and suppose there exists some $Q \in \mathcal{P}_{U,W} \setminus \{P\}$ such that $r_{Q,P} + b_Q \leq b_P$. We can write the formula induced by P as

$$f_P \stackrel{\text{def}}{=} D_U + \sum_i t_i + \sum_{i=1}^{r_{P,Q}} t_i^{(P)} + b_P.$$

The terms denoted by t_i are the red labels shared by both P and Q, and the terms denoted by $t_i^{(P)}$ are the red labels that are in P but not in Q. Similarly, denoting by $t_i^{(Q)}$ the terms that are in Q but not in P, we can write the formula induced by Q, and it follows that:

$$f_Q \stackrel{\text{def}}{=} D_U + \sum_i t_i + \sum_{i=1}^{r_{Q,P}} t_i^{(Q)} + b_Q \leq D_U + \sum_i t_i + r_{Q,P} + b_Q$$
$$\leq D_U + \sum_i t_i + \sum_{i=1}^{r_{P,Q}} t_i^{(P)} + r_{Q,P} + b_Q$$
$$\leq D_U + \sum_i t_i + \sum_{i=1}^{r_{P,Q}} t_i^{(P)} + b_P = f_P.$$

Hence, we can remove the formula induced by P without changing the overall value of the minimum over the set of unrolled formulas.

Using Proposition 1, we can formulate an algorithm to produce a subset of the set of unrolled formulas for $W \in \mathcal{B} \cup \mathcal{R}$ without changing the result of the minimum. The details of the algorithm are presented in Algorithm 2.

Algorithm 2 Optimal set of paths

Input: a dependency graph G, two endpoints $U \in \mathcal{T} \cup \mathcal{L}$ and $W \in \mathcal{B} \cup \mathcal{R}$.
Output: a reduced set of paths $\mathcal{S}$ such that the minimum over all the formulas induced by paths in $\mathcal{S}$ is equal to the minimum over all the formulas induced by paths in $\mathcal{P}_{U,W}$.

```
Generate the set P_{U,W}.
S ← ∅.
for P ∈ P_{U,W} do
    r ← True
    for Q ∈ P_{U,W} \ {P} do
        if r_{Q,P} + b_Q ≤ b_P then
            r ← False
            break
    if r = True then Append P to S
return S
```

To generate the expression to compute $D(i', j') \in \mathcal{B} \cup \mathcal{R}$ in a $(\tau + 1)$-box, we run the algorithm several times, with $D(i', j')$ fixed as the end point and for each of the vertices in $\mathcal{T} \cup \mathcal{L}$ as end points, and then we take the union of the resulting formulas as argument of the minimum function. Note that when we change the starting point of the paths, the induced formulas are not comparable, because they depend on different entries of the matrix D that can take any value.

We prove that Algorithm 2 is optimal with respect to the following definition.

Definition 3. *(Optimality). Let $U, W \in V(G)$. A set $\mathcal{S} \subseteq \mathcal{P}_{U,W}$ is optimal if, for all $P \in \mathcal{S}$, there exists an assignment of the red variables $(t_{i,j})$ such that, for all $Q \in \mathcal{S} \setminus \{P\}$, it holds that $f_P < f_Q$.*

Proposition 2. *(Optimality of Algorithm 2). Let $U, W \in V(G)$ be such that $U \in \mathcal{T} \cup \mathcal{L}$ and $W \in \mathcal{B} \cup \mathcal{R}$. Algorithm 2 returns an optimal set of paths $\mathcal{S} \subseteq \mathcal{P}_{U,W}$ (in the sense of Definition 3) such that the minimum over all the formulas induced by paths in $\mathcal{S}$ is the same as the minimum over all the formulas in the set of unrolled formulas from U to W.*

Proof. From Proposition 1, we know that the algorithm returns a set of paths whose induced formulas does not change the result of the minimum function. It remains to show that this set is optimal[6].

Let $\mathcal{S} \subseteq \mathcal{P}_{U,W}$ be the set of paths returned by Algorithm 2. Let $P \in \mathcal{S}$ be an arbitrary path. We can write f_P as

$$f_P = D_U + \sum_i t_i^{(P)} + b_P, \tag{4}$$

where $t_i^{(P)}$ are the labels of the red edges in P. Let us consider the following assignment of the red variables $(t_k) \in \{0,1\}^*$ for the dependency graph G: if the edge labeled as t_k is in the path P, set $t_k = 0$, and otherwise set $t_k = 1$. Given this assignment, it holds that

$$f_P = D_U + \sum_i t_i^{(P)} + b_P = D_U + b_P. \tag{5}$$

Now, let $Q \in \mathcal{S} \setminus \{P\}$. We can similarly expand f_Q with t_i being the labels of the red edges that are in both P and Q, and $t_i^{(Q)}$ the labels of the red edges that are in Q but not in P. It follows that

$$f_Q = D_U + \sum_i t_i + \sum_{i=1}^{r_{Q,P}} t_i^{(Q)} + b_Q = D_U + r_{Q,P} + b_Q > D_U + b_P = f_P. \tag{6}$$

The last inequality follows since both P and Q are paths returned by the algorithm, when the paths P and Q were selected in the iterations, the path P was not removed. Therefore, it holds that $r_{Q,P} + b_Q > b_P$.This shows that the algorithm is optimal in the sense of Definition 3. □

To compute an upper bound for the number of formulas generated by our approach, we consider the graph presented in Figure 2b. This graph is similar to the dependency graph in a $(\tau+1)$-box, but it has some additional edges. The number of all paths from the top-left corner to the bottom-right corner of this graph is given by the Delanoy number [30, Definition 1.2.8]:

$$\mathcal{D}(l,s) = \sum_{i=0}^{\min\{l,s\}} \binom{l}{i}\binom{s}{i} \cdot 2^i. \tag{7}$$

[6] We will not consider here the case $|\mathcal{P}_{U,W}| = 1$, since Algorithm 2 returns the only path in $\mathcal{P}_{U,W}$, which is trivial. Henceforth, we will consider only $|\mathcal{P}_{U,W}| > 1$. The case $\mathcal{P}_{U,W} = \emptyset$ is also not considered due to the definition of optimality.

This is an upper bound on the number of paths from the top-left corner to the bottom-right corner of the dependency graph because the dependency graph is a subgraph of the graph in Figure 2b. Since the number of paths is maximum for bottom-right corner of the box, it follows that an upper bound for the number of formulas inside the minimum function to compute $D(i, j)$ in a $(\tau + 1)$-box is

$$\sum_{k=1}^{\tau} [\mathcal{D}(\tau, \tau - k) + \mathcal{D}(\tau - k, \tau)] + \mathcal{D}(\tau, \tau). \tag{8}$$

Furthermore, it can be proven that

$$\sum_{k=1}^{\tau} [\mathcal{D}(\tau, \tau - k) + \mathcal{D}(\tau - k, \tau)] + \mathcal{D}(\tau, \tau) = O(\tau \cdot 2^{3\tau}). \tag{9}$$

5 Experiments

We now evaluate the performance of our private edit-distance solution. All the experiments were implemented in the MP-SPDZ framework [19] and were executed on a single AWS EC2 instance of type `c6a.4xlarge`[7], which has an AMD EPYC 7R13 Processor with 16 virtual cores and 32 GB of RAM. Some of the experiments were run without network limitation, so the communication speed is close to running the processes in the same machine. In order to measure the impact of the network, we simulate a local area network (LAN) architecture with 1.6 GBps of bandwidth and 0.3 milliseconds of latency, using the `tc`[8] command from the Linux operating system. For all of our experiments, we consider a bit-length of 16, which allows 16-bit integer computations. We select such number of bits because the edit distance between two chains of length n is upper-bounded by n. Hence, 16 bits is the least number of bits multiple of 8 that allows us to represent the integer numbers needed for the computation. Additional results can be found in the full version of the paper available on the IACR ePrint Archive.

We also compared the performance of the computation of the preamble in a binary domain with respect to a traditional implementation using an arithmetic domain. For two DNA chains of length 1,000, using Semi2^k for passive security, it reduces the data sent by approximately 18.68%, and using SPD$\mathbb{Z}_{2^k}$ for active security, it reduces the data sent by approximately 18.48%. These are percentages of the total data sent of the whole algorithm, including the arithmetic part.

We compared the performance of our solution on a field-domain protocol and a ring-domain protocol. For the ring-domain, we use SPD$\mathbb{Z}_{2^k}$ which has active security, and Semi2^k as its corresponding passive secure version. For the field-domain, we use MASCOT [20] to guarantee active security and Semi as its corresponding passive secure version. As an example, on a 1020 long DNA-chain Semi2^k sends 85% less data than Semi and SPD$\mathbb{Z}_{2^k}$ sends 86% less data than

[7] https://aws.amazon.com/ec2/instance-types/c6a/
[8] https://man7.org/linux/man-pages/man8/tc.8.html

MASCOT.[9] This improvement is explained by the advantages of $\mathbb{Z}_{2^k}$ protocols to perform operations like truncations and reductions modulo 2^k.

5.1 The Effect of the Box Size, τ

As we saw in Section 3.1, the size of the $(\tau+1)$-box affects the number of rounds, multiplications, and comparisons. We evaluate this trade-off by measuring data sent and the execution time of the protocol. We tested for DNA chains of length 1,020, on the passively secure protocol Semi2^k and on the actively secure one SPD$\mathbb{Z}_{2^k}$. We execute each protocol for $\tau \in \{1, 2, 3, 4, 5\}$. We report performance of the whole protocol (including pre-processing) and of the online phase alone.

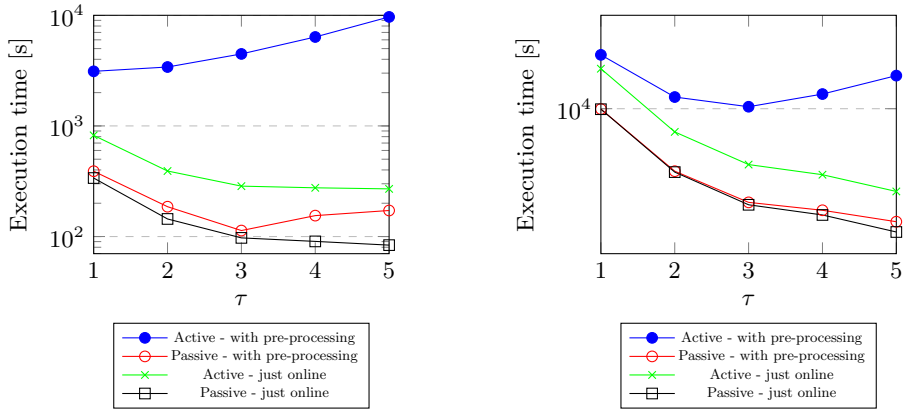
Table 1 and Figure 3a present the results. Since MPC protocols are sensitive to the network speed, we repeated the box size experiments simulating a LAN network as discussed at the beginning of this section. Table 2 shows the results of the LAN experiments and Figure 3b the corresponding graphical representation.

Table 1: Effect of changing τ on our solution without any network limitations.

		Preprocessing+online		**Online phase only**	
Security	τ	**Data [MB]**	**Time [s]**	**Data [MB]**	**Time [s]**
Passive (Semi2^k)	1	1,214.0	389.5	54.4	338.0
	2	1,794.3	186.1	60.7	144.2
	3	2,641.2	113.3	78.9	97.2
	4	4,207.0	154.6	106.7	90.4
	5	7,110.3	172.0	167.6	83.6
Active (SPD$\mathbb{Z}_{2^k}$)	1	166,629	3,114.9	695.5	821.4
	2	241,689	3,407.9	348.8	391.3
	3	350,062	4,474.0	309.9	285.5
	4	551,783	6,366.6	389.3	275.8
	5	924,881	9,665.0	546.4	269.7

The results confirm the analysis in Section 3. The number of rounds decreases as an inverse linear function of τ, while the number of multiplications increases exponentially as a function of τ. The execution time is thus the sum of two functions of τ, an inverse linear induced by the number of rounds, and an exponential induced by the number of multiplications. The specific constants depend on the protocol and on the network, and they affect differently the offline and the online phase. For example, for the actively secure protocol, if we consider the offline phase (Figure 3a), the exponential term dominates the execution time, suggesting no benefit for increasing τ. However, looking only at the online phase (Figure 3a), as τ increases, the execution time decreases and eventually seems to flatten. The effect of the rounds in the total time is even

[9] All these experiments use daBits and edaBits and box-size $\tau = 3$.



(a) No network limits. (b) LAN.

Fig. 3: Effect of the box size τ.

Table 2: Effect of τ on our solution on a LAN.

		Preprocessing+online		**Online phase only**	
Security	τ	**Data [MB]**	**Time [s]**	**Data[MB]**	**Time[s]**
Passive (Semi2^k)	1	1,214.0	9,941.0	54.4	9,923.5
	2	1,794.3	3,891.9	60.7	3,837.8
	3	2,641.2	2,429.5	78.9	2,342.2
	4	4,207.0	2,155.8	106.7	2,008.3
	5	7,110.3	1,813.7	167.6	1,552.7
Active (SPD$\mathbb{Z}_{2^k}$)	1	166,629	22,552.1	695.5	18,298.5
	2	241,689	11,925.8	348.8	7,044.2
	3	350,062	10,300.2	309.9	4,300.1
	4	551,783	12,474.0	389.3	3,689.2
	5	924,881	16,482.1	546.4	2,864.7

more acute for the passively secure protocol (Figure 3a), where the execution time first decreases and then increases with an optimal value on $\tau = 3$. This is because multiplications are relatively cheap for such protocol, yet we expect the graph to eventually increase because the effect of the exponential part induced by the number of multiplications dominates asymptotically. The positive effect of increasing τ is also enhanced on a slower network, as shown on the simulated LAN results in Figure 3b.

Considering both the online and offline phases together, selecting the best value of τ for the passive and active security setting in the LAN configuration respectively reduces the execution time by 81% and 54% in comparison with a baseline implementation using $\tau = 1$.

5.2 Comparison between garbled circuits and secret-sharing

We compare the performance of our implementation using garbled circuits and using protocols based on secret-sharing schemes with $\mathbb{Z}_{2^k}$ domain. Due to the constraints of the AWS EC2 instance, we use DNA chains of length 210 for this experiment. As previously, we use a bit-length of 16 and we consider both a network with no limitations and a simulated LAN. We set $\tau = 1$ for garbled circuits and $\tau = 3$ for the protocols based on secret-sharing schemes, because garbled circuits have better performance when $\tau = 1$, as observed in a preliminary exploration. As τ increases, the number of formulas inside the minimum function grows exponentially and the number of addition gates increases too, which is specially costly for GC-based protocols.

Table 3 presents the results of the experiment. Notice that the secret-sharing based protocols perform better than the GC protocols regarding the data sent. In particular, they send approximately 67-99% less data than the GC protocols. This is due to the high number of arithmetic operations of the algorithm, which implies a high number of gates to garble and send through the network.

For passively secure protocols, the higher data sent of the GC protocols is compensated by their constant-round communication. This can be observed in the similar execution time without network limitations, and even more acutely in the faster execution on LAN.

Table 3: Performance of our solution using GC and SSS-based protocols.

Network	Security	Protocol	Time [s]	Data sent [MB]
No limit	Passive	Yao's GC	2.6	345.8
		Semi2^k	4.9	113.1
	Active	BMR-MASCOT	5,968.6	2.09×10^6
		SPD$\mathbb{Z}_{2^k}$	140.1	14,893.8
LAN	Passive	Yao's GC	2.7	345.8
		Semi2^k	103.0	113.1
	Active	BMR-MASCOT	9,034.0	2.09×10^6
		SPD$\mathbb{Z}_{2^k}$	368.5	14,893.8

For actively secure protocols, GC is one order of magnitude slower than the protocols based on secret-sharing schemes, and the former sends two orders of magnitude more data than the latter. This can be explained by the edit distance algorithm and by the mechanism to provide active security in BMR[10]. The edit distance algorithm has a heavy arithmetic component and many additions that require AND gates. In the case of BMR, the underlying MPC protocol to compute the offline phase is MASCOT [20], which uses MACs to ensure security

[10] Although there are other alternatives for actively secure GC protocols, we choose BMR because it is the only available GC-based protocol for malicious adversaries in MP-SDPZ. This allows us to make comparisons in the same "ground".

against malicious adversaries. So, garbling an AND gate needs to compute multiplications with the assistance of MASCOT which requires the generation of multiplication triples and puts an additional overhead. This overhead does not appear in protocols based on secret-sharing schemes because additions do not need communication.

5.3 Comparison with protocols based on homomorphic encryption

We compare the results of our experiments with a relevant homomorphic encryption (HE) solution. In [7], Cheon et al consider experiments with DNA chains of length 8 at 80 bits of security. According to the results in Table 6 in that paper, their method spends 27.54 seconds on key generation and 16.45 seconds on encryption on an Intel Xeon i7 2.3GHz, 192GB. In their report, they state that it takes 27.5 seconds to obtain the edit distance using the Halevi-Shoup library (HElib) [17] along with the techniques presented in [16]. In our case, considering both the pre-processing and the online phase on a LAN, using $\tau = 2$, we can compute the edit distance of two chains of length 8 in 0.3 seconds using Semi2^k protocol for passive security and in 5.92 seconds using the SPD$\mathbb{Z}_{2^k}$ protocol for active security. Furthermore, Cheon et al. estimate that their method for DNA chains of length 100 with a security parameter of 62 bits, computes the edit distance in 1 day and 5 hours. Our method computes the edit distance of two chains of length 100 in 96.69 seconds on a LAN using the SPD$\mathbb{Z}_{2^k}$ protocol. In terms of security, a 62 bits for HE is insufficient for the current recommended security levels in cryptography.

6 Conclusion

We presented an MPC approach based on secret sharing to securely compute the edit distance via the Wagner-Fischer algorithm. Our method leverages the equations in the algorithm to compute selected positions of the edit distance matrix as minimum of several integers. This modification reduces the number of rounds but increases the number of multiplications and comparisons, inducing a performance trade-off. Using graph theory, we develop an algorithm to automatically generate all the equations needed to compute the required positions in the matrix. We prove that the algorithm returns correct and optimal equations. Our solution is competitive with GC-based solutions on a passive security model, and much faster if active security is required, demonstrating the effectiveness of the secret-sharing approach for bit-wise computations.

We identify two research problems for future work. The first is to find an optimal box size, given environment parameters such as bandwidth, latency, chain lengths, and local computational power. The second is to generalize the graph theory techniques from Section 4 to other dynamic programming problems.

References

[1] Abdelrahaman Aly et al. *Zaphod: Efficiently Combining LSSS and Garbled Circuits in SCALE*. Cryptology ePrint Archive, Paper 2019/974. 2019.

[2] Gilad Asharov et al. "Privacy-Preserving Search of Similar Patients in Genomic Data". In: *PETS*. Vol. 2018. 4. 2018, pp. 104–124.

[3] Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. "Secure approximation of edit distance on genomic data". In: *BMC medical genomics* 10 (2017), pp. 55–67.

[4] Donald Beaver, Silvio Micali, and Phillip Rogaway. "The Round Complexity of Secure Protocols (Extended Abstract)". In: *STOC*. ACM, 1990, pp. 503–513.

[5] Emmanuel Bresson, Dario Catalano, and David Pointcheval. "A Simple Public-Key Cryptosystem with a Double Trapdoor Decryption Mechanism and Its Applications". In: *ASIACRYPT*. Vol. 2894. LNCS. Springer, 2003, pp. 37–54.

[6] Ran Canetti. "Universally Composable Security: A New Paradigm for Cryptographic Protocols". In: *FOCS*. IEEE, 2001, pp. 136–145.

[7] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. "Homomorphic Computation of Edit Distance". In: *Financial Cryptography Workshops*. Vol. 8976. LNCS. Springer, 2015, pp. 194–212.

[8] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure multiparty computation*. Cambridge University Press, 2015.

[9] Ronald Cramer et al. "SPD$\mathbb{Z}_2^k$: Efficient MPC mod 2^k for Dishonest Majority". In: *CRYPTO (2)*. Vol. 10992. LNCS. Springer, 2018, pp. 769–798.

[10] Ivan Damgård et al. "New Primitives for Actively-Secure MPC over Rings with Applications to Private Machine Learning". In: *IEEE Symposium on Security and Privacy*. IEEE Computer Society, 2019, pp. 1102–1120.

[11] Daniel Demmler, Thomas Schneider, and Michael Zohner. "ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation". In: *NDSS*. The Internet Society, 2015.

[12] Tamara M. Dugan and Xukai Zou. "A Survey of Secure Multiparty Computation Protocols for Privacy Preserving Genetic Tests". In: *CHASE*. IEEE, 2016, pp. 173–182.

[13] Yaniv Erlich and Arvind Narayanan. "Routes for breaching and protecting genetic privacy". In: *Nature Reviews Genetics* 15.6 (2014), pp. 409–421.

[14] Daniel Escudero et al. "Improved Primitives for MPC over Mixed Arithmetic-Binary Circuits". In: *CRYPTO (2)*. Vol. 12171. LNCS. Springer, 2020, pp. 823–852.

[15] Tore Kasper Frederiksen et al. "A Unified Approach to MPC with Preprocessing Using OT". In: *ASIACRYPT (1)*. Vol. 9452. LNCS. Springer, 2015, pp. 711–735.

[16] Craig Gentry, Shai Halevi, and Nigel P. Smart. "Homomorphic Evaluation of the AES Circuit". In: *CRYPTO*. Vol. 7417. LNCS. Springer, 2012, pp. 850–867.

[17] Shai Halevi and Victor Shoup. *Design and implementation of HElib: a homomorphic encryption library*. Cryptology ePrint Archive, Paper 2020/1481.
[18] Somesh Jha, Louis Kruger, and Vitaly Shmatikov. "Towards Practical Privacy for Genomic Computation". In: *IEEE Symposium on Security and Privacy*. IEEE, 2008, pp. 216–230.
[19] Marcel Keller. "MP-SPDZ: A Versatile Framework for Multi-Party Computation". In: *CCS*. ACM, 2020, pp. 1575–1590.
[20] Marcel Keller, Emmanuela Orsini, and Peter Scholl. "MASCOT: Faster Malicious Arithmetic Secure Computation with Oblivious Transfer". In: *CCS*. ACM, 2016, pp. 830–842.
[21] Marie Oestreich et al. "Privacy considerations for sharing genomics data". en. In: *EXCLI Journal* (2021), pp. 1243–1260.
[22] Satsuya Ohata. "Recent Advances in Practical Secure Multi-Party Computation". In: *IEICE Trans. Fundam. Electron. Commun. Comput. Sci.* 103-A.10 (2020), pp. 1134–1141.
[23] Alexander Payne et al. "BulkVis: a graphical viewer for Oxford nanopore bulk FAST5 files". In: *Bioinformatics* 35.13 (Nov. 2018), pp. 2193–2198.
[24] Shantanu Rane and Wei Sun. "Privacy preserving string comparisons based on Levenshtein distance". In: *WIFS*. IEEE, 2010, pp. 1–6.
[25] Dragos Rotaru and Tim Wood. "MArBled Circuits: Mixing Arithmetic and Boolean Circuits with Active Security". In: *INDOCRYPT*. Vol. 11898. LNCS. Springer, 2019, pp. 227–249.
[26] Thomas Schneider and Oleksandr Tkachenko. "EPISODE: Efficient Privacy-PreservIng Similar Sequence Queries on Outsourced Genomic DatabasEs". In: *AsiaCCS*. ACM, 2019, pp. 315–327.
[27] Tomas Toft. "Primitives and Applications for Multi-party Computation". PhD thesis. Aarhus University, 2007.
[28] Esko Ukkonen. "Algorithms for Approximate String Matching". In: *Inf. Control.* 64.1-3 (1985), pp. 100–118.
[29] Robert A. Wagner and Michael J. Fischer. "The String-to-String Correction Problem". In: *J. ACM* 21.1 (1974), pp. 168–173.
[30] Douglas B West. *Combinatorial mathematics*. Cambridge Uni Press, 2020.
[31] Chuan Zhao et al. "Secure Multi-Party Computation: Theory, practice and applications". In: *Inf. Sci.* 476 (2019), pp. 357–372.
[32] Y. Zheng et al. "Efficient and Privacy-Preserving Edit Distance Query Over Encrypted Genomic Data". In: *WCSP*. IEEE Computer Society, 2019, pp. 1–6.
[33] Ruiyu Zhu and Yan Huang. "Efficient and Precise Secure Generalized Edit Distance and Beyond". In: *IEEE Trans. Dependable Secur. Comput.* 19.1 (2022), pp. 579–590.