SecurED: Secure Multiparty Edit Distance for Genomic Sequences

Jiahui Gao
Arizona State University
jgao76@asu.edu

Yagaagowtham Palanikumar
Arizona State University
ypalanik@asu.edu

Dimitris Mouris
Nillion
dimitris@nillion.com

Duong Nguyen
Arizona State University
duongnt@asu.edu

Ni Trieu
Arizona State University
nitrieu@asu.edu

Abstract

DNA edit distance (ED) measures the minimum number of single nucleotide insertions, substitutions, or deletions required to convert a DNA sequence into another. ED has broad applications in healthcare such as sequence alignment, genome assembly, functional annotation, and drug discovery. Privacy-preserving computation is essential in this context to protect sensitive genomic data. Nonetheless, the existing secure DNA edit distance solutions lack efficiency when handling large data sequences or resort to approximations and fail to accurately compute the metric.

In this work, we introduce SecurED, a protocol that tackles these limitations, resulting in a significant performance enhancement of approximately $2-24\times$ compared to existing methods. Our protocol computes a secure ED between two genomes, each comprising $1,000$ letters, in just a few seconds. The underlying technique of our protocol is a novel approach that transforms the established approximate matching technique (i.e., the Ukkonen algorithm) into exact matching, exploiting the inherent similarity in human DNA to achieve cost-effectiveness. Furthermore, we introduce various optimizations tailored for secure computation in scenarios with a limited input domain, such as DNA sequences composed solely of the four nucleotide letters.

Keywords

Applied Cryptography, Dynamic Programming, DNA Matching, Edit Distance, Genomics, Multiparty Computation

1 Introduction

The rise of genomic sequencing technologies has marked a new era in biological and healthcare research. The Human Genome Project [18], completed in the early 2000s, is a milestone of sequencing technologies to map the entire human genome. This motivates numerous fields of research and a wide range of applications including, personal medical treatments [4, 13], paternity test [8, 12, 38], cancer and infectious disease research [5, 39], etc. A fundamental challenge is measuring the similarity of sequences using Hamming distance, Jaccard similarity, Pearson correlation, and Euclidean distance. However, among these options, computing edit distance is often favored for several reasons across many scenarios.

Proceedings on Privacy Enhancing Technologies 2025(2), 479–493

https://doi.org/10.56553/popets-2025-0072

Edit distance (ED) [30] is given by the minimum number of insertions, deletions, and substitutions to convert one sequence into another. This metric can be intuitively understood as representing the number of evolutionary events that may occur between sequences. The Wagner-Fischer (WF) algorithm [42] is renowned for its efficacy in solving the edit distance problem using dynamic programming. This algorithm enables precise calculations of edit distances with manageable computational overhead and without reliance on any public reference. Dynamic programming (DP) [30, 42] solves complex problems by breaking them down into smaller subproblems. For edit distance, each sub-problem involves computing the edit distance between increasingly larger subsequences of the original inputs. The WF algorithm fills in a DP table $\mathcal{D}$ with each cell relying on the values of neighboring cells to determine the minimum edit cost for the transformation at that step.

An example is provided in Table 1 for sequences $\alpha = \underline{\text{A}}\text{TCG}\underline{\text{A}}$ and $\beta = \text{TCG}\underline{\text{TC}}$. The DP table $\mathcal{D}$ is structured as a 6×6 grid, progressing from the top-left to the bottom-right corner. In this table, each cell $\mathcal{D}_{i,j}$ represents the edit distance between the subsequences $\alpha_0 \dots \alpha_i$ and $\beta_0 \dots \beta_j$. The final entry, $\mathcal{D}_{5,5} = 3$ (highlighted with a yellow background) indicates that the edit distance between the original inputs α and β is 3. Additionally, the yellow-highlighted path captures the minimum steps required for this transformation.

Table 1: ED from DP.

$\mathcal{D}$	-	T	C	G	T	C
-	0	1	2	3	4	5
A	1	1	2	3	4	5
T	2	1	2	3	3	4
C	3	2	1	2	3	3
G	4	3	2	1	2	3
A	5	4	3	2	2	3

While the WF algorithm provides an efficient solution for ED, genomic data inherently contain sensitive information with significant ethical and legal implications. Traditional DP approaches do not address the privacy concerns that arise when multiple parties wish to compute the edit distance of their genome sequences. This scenario falls within the realm of secure multi-party computation (MPC), where parties collaboratively compute a function over their private inputs without revealing any information beyond the result. Customized MPC protocols include private set intersections and unions [22, 23, 34] to decision trees, histograms, and heavy hitters [3, 10, 11, 19, 20, 35, 36], whereas MPC compilers [9, 21, 27, 43] allow for private computation from high level programs.

In this work, we introduce SecurED, a protocol to enable multiple parties to compute the secure edit distance of their input sequences as $ED(\alpha, \beta)$ while maintaining their privacy. To establish a protocol for secure ED computation, a range of generic methods can be

employed, including garbled circuits and secret-sharing techniques. Zhu et al.[48] proposed a tailored garbled circuit approach, while [41] employed a secret-sharing-based approach. We delve into more details about these approaches in Section 2.1. Additionally, [1, 44] proposed secure solutions for an approximate edit distance, which are also discussed in Section 2.2. This work prioritizes exact solutions for ED, particularly due to the necessity of precise results in DNA-related computations such as parental testing (i.e., ensuring support for exact ED computation remains a high priority).

Technical Overview. When developing a privacy-preserving ED solution for genome sequences using dynamic programming (DP), efficiency relies on two key aspects. Firstly, the performance of generic MPC methods is closely tied to the efficiency of the underlying algorithm in the clear. However, computing the minimum value of adjacent cells in the DP table is expensive under MPC (as it requires comparisons), and careful consideration is needed to convert the cleartext algorithm to a privacy-preserving equivalent. Although state-of-the-art solutions [14] over clear data outperform general-purpose ED algorithms, they rely on different structures of the DP table and reveal the histogram of the DP computation. This poses significant security concerns when considering privacy-preserving edit distance solutions. Secondly, genome sequences possess distinct properties compared to sequences from a general alphabet, allowing for tailored optimizations with MPC in mind.

Thus, to make SecurED practical, we begin by tailoring the DP algorithm specifically for genome sequences. This customization is grounded in the observation that two arbitrary DNA sequences exhibit high similarity (with over 99.5% nucleotides being identical [18]). Consequently, when computing $ED(\alpha, \beta)$ in the DP table, likely, the minimum values leading to the optimal solution lie close to the diagonal. Thus, we can optimize this scenario by employing Ukkonen's algorithm [40], which computes only the entries around the diagonal, thereby reducing computational load.

Traditionally, Ukkonen's algorithm provides an approximation rather than an exact result. However, in the context of DNA sequences, achieving precise computations hinges on setting an appropriate threshold. This raises an intriguing question: *how can we determine the suitable threshold* T *for Ukkonen's algorithm to compute the edit distance accurately?* This work introduces a novel method that effectively identifies this threshold under MPC. To the best of our knowledge, SecurED marks the first instance of combining the Wagner-Fischer and Ukkonen algorithms for precise computation. Beginning with a loose upper bound for the threshold such as $\mathsf{T} = 0.1 \max(m, n)$ (where m and n are the sizes of the two sequences), we propose an efficient algorithm to determine a tighter threshold T' suitable for application to Ukkonen's algorithm.[1] Using this tighter threshold T', we identify the path in the DP table that closely approaches optimal (e.g., the yellow path in Table 1) and make the computation of edit distance significantly more lightweight, and thus feasible under MPC. To the best of our knowledge, revealing the value T' does not leak any significant information as it is highly correlated with the final edit distance result. We include a detailed discussion on this topic in Section 5.5 describing that the information leakage is minor and cannot be used to infer anything meaningful about the private DNA sequences in practice.

We implement our protocol in two MPC variants: (a) using a secret-sharing scheme where the DP table is shared among the participants and no party learns the other parties' inputs or the intermediate values but can reconstruct and learn the final outputs, and (b) using a garbled circuit approach where one of the participants creates the circuit using their input and the other participant can evaluate it. The two approaches introduce a computation/communication trade-off; when secret sharing is employed, the total communication cost is lower than that of the garbled circuits, albeit with more communication rounds. We delve into this trade-off in our evaluations in Section 6.2. Our SecurED protocol can be instantiated in the semi-honest setting, which we discuss in Section 3.1 and evaluate in Section 6.2.

To accelerate our solution further, we adopt the optimization proposed in [41], where the DP table is partitioned into small boxes. This optimization enables direct computation of the optimal value for each partition box, rather than processing one entry at a time. Depending on the size of the partition boxes, the method proposed in [41] provides a trade-off between computation and the number of rounds, both of which impact the end-to-end performance of secret-sharing protocols. However, despite this optimization, all computations (such as comparing minimum values among different numbers of edits) still need to be performed for each box to obtain the optimal value. In other words, [41] operates over the whole DP table, rendering it impractical. In SecurED, we refine this box optimization further by combining it with Ukkonen's algorithm; we observe that certain computations can be avoided, which occurs within a threshold of the main diagonal. We demonstrate a 4-fold improvement over our baseline protocol without box optimization by integrating this box optimization into our protocol.

Our contributions can be summarized as follows:

- We introduce SecurED, an efficient secure protocol for computing the exact edit distance of genome sequences. SecurED relies on the Ukkonen algorithm in conjunction with the Wagner-Fischer dynamic programming method, achieving exact computation rather than approximate. SecurED can be instantiated with semi-honest security and works with multiple parties.
- We propose a novel method to determine a tight upper bound T' for ED between two genome sequences. SecurED uses T' in the Ukkonen algorithm for edit distance, which operates with linear computational complexity in both time and space.
- We implement our SecurED protocol in both two-party and multi-party settings using MP-SPDZ [27] and evaluate it in terms of execution time and communication cost in both LAN and WAN settings. Additionally, we implement SecurED with both garbled circuits and secret sharing. Our protocol outperforms the state-of-the-art SS-based [41] and GC-based [48] works more than 24× and 2× in terms of running time, while at the same time our communication is reduced by more than 9× and 1.2×, respectively. Finally, we evaluate the threshold determination protocol in the same LAN and WAN settings and analyze its accuracy.

[1] We refer to T as *loose* and T' as *tight* in the sense of the gap to the true edit distance. More discussion is provided in Section 6.1 to illustrate this point further where we show the statistics for T, T', and the ground truth for ED.

2 Related Work

The two main approaches in the field of secure edit distance computation either focus on exact ED [2, 16, 37, 41, 48] or on approximate computation [1, 44, 47]. The latter approaches rely on the strong assumption of possessing a good public reference string for edit distance that enables parties to perform local computations (e.g., sequence alignment) to enhance performance. This work eliminates this assumption since determining the reference string is not reliable in many cases – the performance of ED protocols highly depends on a reference string as shown in [48]. Furthermore, we focus on the exact edit distance computation as it is vital for healthcare applications that require precise results. For comprehensiveness, this section provides an overview of existing works in both categories.

2.1 Focusing on Exact ED Computation

Dynamic programming is the most effective method for finding the exact edit distance between two sequences. To devise a privacy-preserving edit distance computation, secure computation can be applied on top of the dynamic programming algorithm. These techniques encompass Homomorphic Encryption (HE), Garbled Circuits (GC), and Secret Sharing (SS).

Based on HE, the first edit distance protocol was proposed by [2], where two parties maintain additive shares of the DP table. Another HE-based protocol was developed by [37], where the sender computes the encryption of the DP table, and the receiver can only learn the last entry, representing the value of the edit distance. In [16], encrypted data is outsourced to a cloud server, which performs dynamic programming under HE, and only the data owner can decrypt the final output. They also propose a method to divide the edit distance matrix into sub-boxes to reduce the depth of computation.

The work of [48] presents a generic edit distance framework by introducing a customized GC approach. Their improvement stems from the observation that, during edit distance computation, the difference of inputs for the minimum circuit is bounded, and an arithmetic circuit proves more efficient than a binary one.

The work by [41] implements a secret-shared-based dynamic programming algorithm. They enhance the algorithm using the box technique [16] to strike a balance between computation and round complexity. Despite improving performance compared to the baseline dynamic programming approach in both semi-honest and malicious models, the efficiency of their protocol remains suboptimal for large inputs as they compute the entire DP table. In contrast, our protocol only computes a partial table using the Ukkonen algorithm, rendering it practical.

Comparison. For comparison, we ignore the works based on homomorphic encryption since their performance is impractical. Compared to the state-of-the-art GC-based approach [48] and SS-based approach [41], our protocol demonstrates a 2-fold and 20-fold improvement, respectively. These improvements come from significant reductions in computation achieved by our protocol for the basic dynamic programming algorithm. We provide detailed experimental comparisons in Section 6.2.

2.2 Focusing on Approximate ED Computation

An approximate secure edit distance protocol that leverages private set difference was proposed by [44]. Given a public reference genome sequence, they compress the input genome sequence into smaller sets by tracking the differences and approximate the ED by the difference between these two sets. Subsequently, they compress set S into an integer d_S, representing the sum of the evaluations of each element on a random binary hash function (i.e., $\mathcal{H}(\cdot) : \{0,1\}^* \mapsto \{-1,1\}$). They demonstrate that the expression $E[(d_A - d_B)^2]$ corresponds to the set difference $|Diff(A,B)|$ between sets A and B, serving as a reliable approximation for the edit distance. This expression can be computed by securely averaging $(d_A - d_B)^2$ over multiple iterations, where in each iteration, the hash function is independent and randomly sampled from a family of binary hash functions. Unfortunately, their protocol hinges on the assumption of having a public reference string and solely operates in the semi-honest setting.

The work by [1] enhances the approach of [44] by partitioning input genomes into smaller segments and approximating the edit distance through the summation of edit distances across all partitions. They observe that the potential edit distances for each partition are quite limited, allowing for significant computational savings. These small edit distances can be pre-computed, reducing the problem to a secure matrix-vector multiplication. To divide the genome into smaller segments, both the database and query sequences are aligned with a common reference genome sequence, followed by segmentation into fixed-length segments. Unfortunately, as in [44], [1] computes the approximate ED, operates only in the semi-honest setting, and relies on having the reference sequence public.

The work of [47] builds on HE and considers the outsourced setting, empowering clients to query databases securely. Similar to [1], the approximation is derived by summing the edit distances across segments of the original input sequences. Each small segment's distance is pre-computed and stored, allowing for querying with new inputs, facilitated by the finite possible variations for each segment. The proposed index technique employs HE when querying the edit distance for each pair of small segments.

Comparison. In summary, the existing protocols [1, 44, 47] are only secure in the semi-honest setting and do not support exact computation for edit distance, which is crucial for healthcare applications. Moreover, their performance heavily depends on large iterations, as seen in [44], and they require genome sequences to be aligned before computation. The absence of a clear method for selecting a suitable reference sequence significantly affects the efficiency and accuracy of these protocols. We discuss the performance comparison in detail in Section 6.2.3.

3 Preliminaries

Notation. We use $[\cdot]$ notation to refer to a set. For example, $[m]$ implies the set $\{1, \ldots, m\}$.

3.1 Security Model

There are mainly two adversarial models for secure multi-party computation. In the *semi-honest model*, all parties follow the protocol but try to learn additional information by analyzing the messages received during the execution of the protocol. In the *malicious model*, corrupted parties may deviate from the protocol in any way that can help to learn additional information. We consider the semi-honest adversarial model and rely on the underlying MPC framework.

Parameters: Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, N servers $P_1, \ldots, P_N$, alphabet $\mathcal{A}$, and length m, n.
Functionality:

- Wait for input $\alpha \in \mathcal{A}^m$ from $\mathcal{R}$.
- Wait for input $\beta \in \mathcal{A}^n$ from $\mathcal{S}$.
- Wait for no input from server $P_{i \in [N]}$.
- Give $\mathcal{R}$ and $\mathcal{S}$ the value $ED(\alpha, \beta)$.

Figure 1: Ideal Functionality for Secure Edit Distance (SecurED). We show in gray color the Ideal Functionality for Delegated SecurED.

3.2 Secure Edit Distance Computation

The secure edit distance computation for human genome sequences involves a two-party protocol where the sender $\mathcal{S}$ has a DNA sequence α and the receiver $\mathcal{R}$ has a DNA sequence β. They aim to learn the edit distance between α and β without revealing any additional information. In this scenario, the two sequences are denoted as $\alpha \in \mathcal{A}^m$ and $\beta \in \mathcal{A}^n$, each of lengths m and n, respectively, with alphabet $\mathcal{A} = \{\mathtt{A}, \mathtt{G}, \mathtt{T}, \mathtt{C}\}$. We assume that $m \geq n$ as in multiple places in the paper we write $m - n$ (e.g., in Theorem 4.1); this can trivially work in the case where $m < n$ where we need to take the absolute value $|m - n|$. Regardless, this does not affect the security of our protocol and it is used for simplicity. The possible edits include insertion, deletion, and substitution of a single character in the sequences. The distance between two sequences is defined as the minimum number of edits required to convert one string into the other. Fig. 1 (only black color) presents the ideal functionality of the secure edit distance.

Additionally, we delve into the delegated setting, where both $\mathcal{S}$ and $\mathcal{R}$ delegate the computation to untrusted cloud servers using the secret sharing scheme outlined in Section 3.5.1. Specifically, $\mathcal{S}$ and $\mathcal{R}$ secret-share their inputs to N servers (with $N \geq 2$), which conduct the dynamic programming on the shared values. The secret-shared result is subsequently transmitted back to the $\mathcal{R}$, who can reconstruct the input. The protocol maintains security under the assumption that all servers do not collude. The delegated ideal functionality is presented in Fig. 1 (both gray and black color).

3.3 Wagner Fischer (WF) Algorithm

The dynamic programming algorithm used to solve the edit distance between arbitrary sequences is commonly referred to as the Wagner-Fischer (WF) algorithm [42]. Given two strings α and β of lengths m and n, the edit distance $ED(\alpha, \beta)$ is computed by filling a matrix $\mathcal{D}$ of size $(m+1) \times (n+1)$. Each cell $\mathcal{D}_{i,j}$ in the matrix represents the minimum number of edits required to convert the prefix $\alpha_1 \ldots \alpha_i$ to $\beta_1 \ldots \beta_j$, with the final cell $\mathcal{D}_{m,n}$ holding the edit distance between the two strings α and β.

Initially, the cells in the first row $\mathcal{D}_{0,j}$ for $j \in [m]$ are initialized to represent the cost of inserting j characters to transform a string of size 0 into the string $\beta_1 \ldots \beta_j$. Similarly, the cells in the first column $\mathcal{D}_{i,0}$ for $i \in [n]$ are filled to represent the cost of deleting i characters. This initialization can be formalized as:

$$\mathcal{D}_{0,j} = \sum_{k=1}^{i} \omega_{ins}(\beta_k), \quad \mathcal{D}_{i,0} = \sum_{k=1}^{i} \omega_{del}(\alpha_k),$$

where ω_{ins} and ω_{del} denote the cost of insertions and deletions. As a result of the initialization, we have the edit distances between each string and the empty string; e.g., observe in Table 1 that $\mathcal{D}_{i,0} = \mathcal{D}_{0,i} = i$ for $i \in [m]$ ($m = n = 6$ in this case). Subsequently, each remaining matrix cell $\mathcal{D}_{i,j}$ is filled by either using the cost of the upper left cell ($\mathcal{D}_{i-1,j-1}$) in case α_i is equal to β_j, or by propagating the minimum cost of the three neighboring cells added to the cost of the current cell (i.e., $\mathcal{D}_{i-1,j-1}$ plus the cost of a substitution, $\mathcal{D}_{i-1,j}$ in addition to the cost of one deletion, and $\mathcal{D}_{i,j-1}$ plus one insertion cost). We formalize this in Eq. (1), where ω_{sub} represents the substitution cost:

$$\mathcal{D}_{i,j} = \begin{cases} \mathcal{D}_{i-1,j-1} & \text{if } \alpha_i = \beta_j, \\ min \begin{cases} \mathcal{D}_{i-1,j-1} + \omega_{sub}(\alpha_i, \beta_j) \\ \mathcal{D}_{i-1,j} + \omega_{del}(\alpha_i) \\ \mathcal{D}_{i,j-1} + \omega_{ins}(\beta_j) \end{cases} & \text{if } \alpha_i \neq \beta_j. \end{cases} \quad (1)$$

Without loss of generality, we assume a uniform cost for all operations, meaning that $\omega_{ins}(a) = \omega_{del}(a) = \omega_{sub}(a, b) = 1$ for all symbols $a, b \in \mathcal{A}$. Table 1 in Section 1 provides an end-to-end example of the application of the WF algorithm.

3.4 Ukkonen Algorithm for Approximate ED

Ukkonen altered the WF algorithm and diminished time and space requirements, albeit with a minimal chance of inaccuracies in the edit distance result [40]. In this approach, a threshold T is set, restricting computations to T diagonals extending in both directions from the main diagonal of the WF matrix. For the diagonal passes $\mathcal{D}_{i,j}$ (with i, j being in the diagonal), we refer to it as the k-th diagonal where $k = j - i$. We present this in Alg. 1: Lines 3-4 initialize the first row and column, while lines 8-9 update the value of $\mathcal{D}_{i,j}$ as per Eq. eq1. The primary difference from the original WF algorithm is that updates are limited to the T leading diagonals. Note that we have removed any control flow decisions from Alg. 1 to make it MPC-friendly – i.e., cmp is always added to $\mathcal{D}_{i,j}$ regardless of whether α_i is equal or not to β_i.

Another way to view this problem is to consider the matrix as a directed dependency graph with each cell $\mathcal{D}_{i,j}$ being a node and the operations (insertions, deletions, and substitutions) being weighted rightward, downward, and diagonal paths. Then, the goal is to move from $\mathcal{D}_{0,0}$ to $\mathcal{D}_{m,n}$ via the path with the least cost. Viewing this as a graph highlights the dependency structure inherent in edit distance calculations and shows how each operation impacts the traversal path. As we consider all weights to be equal, we see that if there is a directed path from $\mathcal{D}_{i,j}$ to $\mathcal{D}_{i',j'}$ then, $\mathcal{D}_{i',j'} \geq \mathcal{D}_{i,j} + |(j' - i') - (j - i)|$ [40]. We can extend this observation to see that if the path corresponding to the minimum number of edits from $\mathcal{D}_{0,0}$ to $\mathcal{D}_{m,n}$ and if any node in that path $\mathcal{D}_{i',j'} > \mathsf{T}$, then $|j' - i'| > \mathsf{T}$ indicating the path goes cross the T-th diagonal. In this case, restricting calculations to T diagonals might miss parts of the optimal path, leading to an approximation. In practice, there is no good way to learn the upper bound of the edit distance for two arbitrary inputs. That is why a predefined parameter T leads to an approximation. Of course, a larger T gives a better approximation at the cost of more diagonals needing to be evaluated.

Algorithm 1 Ukkonen's Approximate Edit Distance [40]

Inputs: $\alpha = \alpha_1 ... \alpha_m, \beta = \beta_1 ... \beta_n$, ▷ *Arrays* α, β.
$\mathsf{T} > |m - n|$ ▷ *Threshold* T.

1: **procedure** UkkonenED(α, β, T)
2: $\mathcal{D} \leftarrow$ matrix of shape $(m+1) \times (n+1)$ initialized with 0s.
3: $\mathcal{D}_{i,0} \leftarrow i \quad \forall i \in [0, \mathsf{T})$ ▷ *Initialize first column (up to* T*).*
4: $\mathcal{D}_{0,j} \leftarrow j \quad \forall j \in [0, \mathsf{T})$ ▷ *Initialize first row (up to* T*).*
5: $d \leftarrow -(\lfloor (\mathsf{T} - (n-m))/2 \rfloor)$ ▷ *First diagonal offset.*
6: **for** $i \leftarrow 1$ **to** m **do**
7: **for** $j \leftarrow max(1, d+i)$ **to** $min(\mathsf{T}+d+i, n)$ **do**
8: $cmp \leftarrow \alpha_i \neq \beta_j$ ▷ *0 (False) if* $\alpha_i = \beta_j$*, 1 (True), otherwise.*
9: $\mathcal{D}_{i,j} \leftarrow min\{\mathcal{D}_{i-1,j-1} + cmp, \mathcal{D}_{i-1,j} + 1, \mathcal{D}_{i,j-1} + 1\}$
10: **return** $\mathcal{D}_{m,n}$ ▷ *Edit distance result.*

3.5 Multi-party Computation

SecurED proposes an MPC-friendly edit distance algorithm from DP. To this end, we first discuss two widely used MPC methods.

3.5.1 Secret Sharing (SS). Secret sharing splits private inputs and intermediate results of a computation into seemingly random values, which are then distributed between non-colluding parties. One form of SS, called additive, splits an ℓ-bit value x as follows: the data owner selects $n-1$ random values $x_1, \ldots, x_{n-1}$ from $\{0,1\}^\ell$ and computes x_n as $x_1 + \ldots + x_{n-1} - x = x_n$. (For simplicity, we omit the modulo operation, denoting the share of party i $[\![x]\!]_i$.) To reconstruct the shared value $[\![x]\!]$, each party transmits its share to a combiner, who locally reconstructs the secret $x = x_1 + \ldots + x_n$.

One can directly perform addition, subtraction, and multiplication-by-constant operations on the shares without communication between parties. For instance, $[\![x+y]\!]$ can be computed as $[\![x]\!] + [\![y]\!]$ locally. To securely multiply two ℓ-bit values, we employ Beaver triples [6], where much of the communication and computation is offloaded into a (offline) preprocessing phase, which is independent of the inputs. During the offline phase, secret shared values $([\![a]\!], [\![b]\!], [\![c]\!])$ are generated such that $c = ab$. Then, in the online phase, parties locally compute $[\![\alpha]\!] = [\![x]\!] - [\![a]\!]$ and $[\![\beta]\!] = [\![y]\!] - [\![b]\!]$, where x and y denote the private inputs. Thus, parties collaboratively reconstruct α and β by exchanging their respective shares $[\![\alpha]\!]$ and $[\![\beta]\!]$. Finally, the product $[\![xy]\!]$ is calculated as $[\![c]\!] + \alpha[\![b]\!] + \beta[\![a]\!] + \alpha\beta$, which each party can evaluate locally. Utilizing addition and multiplication operations on secret-shared data enables computation while maintaining privacy, ranging from simple equality checks and finding minimum values, to any arithmetic circuit. This capability is crucial for computing the DP table for ED computation in a privacy-preserving way.

3.5.2 Garbled Circuits (GC). Garbled Circuits [24, 45] is another technique for two-party computation. The ideal functionality of GC is to take inputs x and y from the parties and compute a function f on them without revealing the inputs as $\mathcal{GC}(f, x, y)$. In this work, on top of our SS-based approach, we design a GC-tailored technique for computing the DP table, consisting of "less than" and "equal" circuits. The GC protocol involves a *garbler* and an *evaluator*: the garbler encodes a boolean function f (e.g., less than) into a circuit using two random keys per wire of the circuit; the evaluator obtains the corresponding keys of the input wires and evaluates the circuit to learn the corresponding output wire key. Finally, the evaluator utilizes a decoding table, which maps the final output wire keys to the actual values, to decode the final output $f(x, y)$.

4 Our Edit Distance Protocol

This section presents our main protocol for SecurED. Our goal is to compute the exact edit distance of two DNA sequences while maintaining the efficiency of approximate ED methods. Our starting point is Ukkonen's algorithm (Alg. 1) for approximate ED. Next, we introduce an efficient protocol for determining a precise edit distance upper bound. This enables us to select an appropriate threshold for Ukkonen's algorithm, optimizing its performance while maintaining the accuracy of exact matching. Finally, we discuss various optimizations that can be implemented when deploying our protocol with both garbled circuits and secret-sharing techniques.

4.1 From Approximate to Exact: Achieving Precision in Edit Distance Computation

Contrary to other sequences, genome sequences consisting of DNA nucleotides $\mathcal{A} = \{\mathsf{A}, \mathsf{G}, \mathsf{T}, \mathsf{C}\}$ exhibit properties that can be exploited when constructing an edit distance protocol [44]. Specifically:

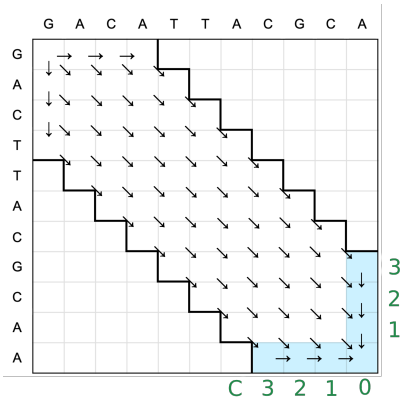
(1) For arbitrary human genome sequences, it has been observed that over 99.5% of nucleotides are identical.
(2) When considering the application of insertion, deletion, and substitution edits to transform one genome sequence into another, it has been found that over 95% of edits are non-adjacent. Furthermore, within this non-adjacent edit set, approximately 80% to 90% of the edits are substitutions.

The first observation suggests that the edit distance is significantly smaller than the input size. The second observation implies that the path leading to the optimal output is likely to be near the leading diagonal. These characteristics make Ukkonen's algorithm an excellent candidate for efficiently solving the edit distance problem over human genome sequences.

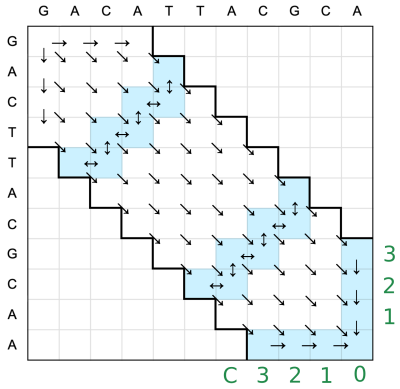
However, the Ukkonen algorithm with a threshold T may not yield an exact result. This is because, rather than computing all cells in the matrix, it only examines T diagonals adjacent to and including the leading diagonal. We observe that when we assign a uniform cost of 1 to any edit operation, it can be easily proved that if the actual edit distance does not exceed T, assessing only the T diagonals always yields the exact result. On the other hand, if the edit distance is greater than T and Ukkonen only examines T diagonals, it does not necessarily yield inexact results. For this to happen, the path of optimal edits (e.g., the yellow path in Table 1) may be very close to the leading diagonal but the result could be greater than T (e.g., due to many substitutions). Below, we formalize the condition for yielding exact computation in a theorem; our proof follows from [40, Corollary 1].

Theorem 4.1. *Considering the* $\mathsf{T} + (m-n)$ *leading diagonals in the DP matrix* $\mathcal{D} \in \mathbb{Z}^{(m+1)\times(n+1)}$ *for input sequences* $\alpha \in \mathcal{A}^m$ *and* $\beta \in \mathcal{A}^n$*, if the ground truth* $ED(\alpha, \beta) < \mathsf{T} + m - n$*, then the Ukkonen algorithm gives exact edit distance computation.*

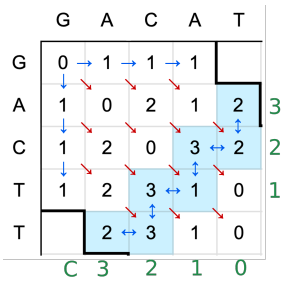
Proof. To demonstrate the accuracy of the ED computation without error, we need to prove that the path corresponding to



(a) Initial approach by restricting insertions/deletions to the ends.



(b) Segmented approach where diagonal switches can only be made at checkpoints.



(c) A snippet of (b). The substitution costs (red arrows) are stored in matric $\mathcal{D}$ while the costs to switch diagonals (blue arrows) are stored in array C. At each checkpoint (blue background), we accumulate the costs $\mathcal{D} + C$ to *accum*.

Figure 2: Overview of our upper bound T' algorithm. In (a), we show the array C that keeps track of the cost of insertions and deletions at the beginning and the end (shown with vertical and horizontal arrows). We only show C in the end (i.e., the cost of going from $\mathcal{D}_{m-2,n}$ to $\mathcal{D}_{m,n}$ is 2). Through the rest of matrix $\mathcal{D}$, we only count the substitutions (i.e., diagonal arrows). In (b), we use the same approach as (a) but we use multiple segments and at the end of each segment we introduce a "checkpoint". At each checkpoint, we synchronize the costs for additions and deletions. Lastly, in (c), we show a snippet of (b) in more detail.

the optimal solution never intersects a diagonal outside the range defined by the $\mathsf{T} + m - n$ leading diagonals. We prove Theorem 4.1 by contradiction. Suppose the optimal path contains at least one node, say $\mathcal{D}_{i,j}$, outside the range defined by the $\mathsf{T} + m - n$ leading diagonals. Let's consider the following two cases:

- $\mathcal{D}_{i,j}$ is located on the $(-\mathsf{T}/2)$-th diagonal (i.e., $i - j = -\mathsf{T}/2$),
- $\mathcal{D}_{i,j}$ is located on the $(m - n + \mathsf{T}/2)$-th diagonal (i.e., $i - j = m - n + \mathsf{T}/2$).

In both cases, the optimal path is divided into two halves. The first half begins at $\mathcal{D}_{0,0}$ and ends at $\mathcal{D}_{i,j}$, while the second starts at $\mathcal{D}_{i,j}$ and ends at $\mathcal{D}_{m,n}$. According to [40, Lemma 2], we have:

$$\mathcal{D}_{i,j} - \mathcal{D}_{0,0} \geq |j - i| \text{ and } \mathcal{D}_{m,n} - \mathcal{D}_{i,j} \geq |n - m - (j - i)|$$

Combining these two inequalities, we have:

$$\begin{aligned} \mathcal{D}_{i,j} - \mathcal{D}_{0,0} + \mathcal{D}_{m,n} - \mathcal{D}_{i,j} &\geq |j - i| + |m - n - (i - j)| \\ \Leftrightarrow \mathcal{D}_{m,n} - \mathcal{D}_{0,0} &\geq |\mathsf{T}/2| + |m - n + \mathsf{T}/2| \\ \Leftrightarrow ED(\alpha, \beta) &\geq |\mathsf{T}/2 + m - n + \mathsf{T}/2| \\ \Leftrightarrow ED(\alpha, \beta) &\geq \mathsf{T} + m - n. \end{aligned}$$

The assumption that the optimal solution path contains at least one node outside the range defined by the $\mathsf{T} + m - n$ leading diagonals leads to a contradiction. Thus, all nodes in the optimal path must lie within this range. Considering only the $\mathsf{T} + m - n$ leading diagonals the Ukkonen algorithm provides accurate computation if the true value of $ED(\alpha, \beta)$ is smaller than $\mathsf{T} + m - n$. □

The immediate question that follows is: *how can we efficiently determine such a threshold to enable exact edit distance computation from the Ukkonen algorithm?* The main contribution of SecurED is to address the above question by proposing an efficient threshold determination algorithm (which we present in Section 4.2). This approach transforms our main MPC-friendly construction into two phases. First, we determine a tight upper bound T' based on a predefined loose upper bound T. Second, we utilize T' as the threshold for the Ukkonen algorithm to compute the edit distance for genome sequences. Note that after being computed securely, the upper bound T' can be seen as publicly disclosed information, as it can also be inferred from the final output.

4.2 Edit Distance for Genomic Sequences: Determining Threshold Upper Bounds

A straightforward approach to selecting a threshold for Ukkonen's algorithm is to establish a predetermined upper limit. This method may suffice when analyzing entire genome sequences containing billions of nucleotides, as even a conservative estimate of $0.5\% - 1\%$, according to the first observation in Section 4.1, could serve as a robust upper bound. However, for applications such as patient queries or personalized medicine, where only specific segments of the genome are of interest for comparison [25, 28, 33], analyzing the entire genome sequences may not be practical. Instead, focusing on relevant portions of the sequence is more efficient and targeted.

Motivation. When focusing on specific parts of the DNA sequence, such as thousands of nucleotides, it becomes challenging to find a precise upper bound that closely matches the ground truth. For instance, consider the dataset from the iDASH 2016 competition [26], consisting of sequences with lengths of around 3500. This dataset, extracted from chromosome 3 of the human genome within the high divergence region of gene ZNF717, exhibits significant variation in edit distance among pairs of genome sequences. Our experiment with the iDASH data shows that the edit distance ranges from 37 to 172, corresponding to approximately 1% to 5% of the total sequence length. This variability underscores the difficulty in establishing a solid upper bound for edit distance, particularly when analyzing specific segments of the DNA sequence.

Indeed, selecting an excessively large upper bound can result in unnecessary computational costs for cases with small edit distances, while on the flip side of things, a smaller threshold increases the

likelihood of approximation rather than exact computation for cases with larger edit distances. To mitigate this, we propose a refined method for determining the threshold. Our algorithm efficiently determines a tight upper bound T' for the edit distance based on a pre-defined loose upper bound T, such as 10% instead of 1% or 5%. This approach aims to strike a balance between computational efficiency and accuracy, ensuring that the selected threshold is appropriate for the specific characteristics of the data being analyzed.

High-Level Overview. We view the DP matrix $\mathcal{D}$ of dimensions $(m+1) \times (n+1)$ as a dependency graph with paths starting from $\mathcal{D}_{0,0}$ and ending at $\mathcal{D}_{m,n}$. Moving right, down, or diagonally along these paths represents a series of edits needed to convert one sequence into another. The shortest path in this graph corresponds to the edit distance, representing the fewest steps required. To find a reasonable upper bound (T') for the threshold edit distance, it suffices to identify a satisfactory or nearly optimal path, rather than aiming for perfection (the exact path), from $\mathcal{D}_{0,0}$ to $\mathcal{D}_{m,n}$. This can be achieved by confining specific edits to certain areas and computing fewer comparisons (i.e., to find the minimum of the neighboring cells) than required in Eq. (1). Note that computing the minimum is the main bottleneck in secure computation for determining the value of the edit distance.

Algorithm Description. We present our optimal threshold algorithm in Alg. 2 which takes a pair of genome sequences $\alpha \in \mathcal{A}^m$ and $\beta \in \mathcal{A}^n$, a loose upper bound of threshold $\mathsf{T} > m - n$, and returns a tight upper bound T'. Notably, we refer to T' as tight upper bound and optimal threshold interchangeably. For simplicity, we assume $m = n$ and denote the corresponding dynamic programming matrix as $\mathcal{D}_{(m+1)\times(n+1)}$ in the following section.

Algorithm 2 Find optimal threshold T'.

Inputs: $\alpha = \alpha_1 ... \alpha_m, \beta = \beta_1 ... \beta_n$, ▷ *Arrays α, β.*
$\mathsf{T} > |m-n|, x$ ▷ *Threshold T, segment size x.*

1: **procedure** FindThreshold($\alpha, \beta, \mathsf{T}, x$)
2: $n \leftarrow \mathtt{Size}(\alpha)$
3: $C_i \leftarrow \mathtt{Abs}(\lfloor T/2 \rfloor - i)\ \forall i \in [0, \mathsf{T})$ ▷ *Distance vector.*
4: $accum \leftarrow 0$ ▷ *Final accumulated upper bound*
5: $\mathcal{D} \leftarrow$ matrix of shape $(n+1) \times (n+1)$ initialized with 0s.
6: **for** $x_{START} \leftarrow 0$ **to** n by x **do** ▷ *For each segment.*
7: $x_{END} \leftarrow \mathtt{Min}(x_{START} + x, 2n)$ ▷
8: **for** $d \leftarrow -T/2$ **to** $T/2$ **do** ▷ *Traverse diagonals.*
9: **for** $i \leftarrow x_{START}$ **to** x_{END} **do** ▷ *Traverse segment.*
10: $j \leftarrow i + d$ ▷ *Given i and diagonal d, determine j.*
11: **if** $i = x_{START}$ **then** ▷ *Init. $\mathcal{D}$ at beginning of segment.*
12: $\mathcal{D}_{i,j} \leftarrow \alpha_i \neq \beta_j$
13: **if** $0 \leq j < n$ **and** $i + j < x_{END}$ **then**
14: $\mathcal{D}_{i,j} \leftarrow \mathcal{D}_{i-1,j-1} + (\alpha_i \neq \beta_j)$ ▷ *Substitutions cost.*
15: $\mathcal{L} \leftarrow$ array with T cells from $\mathcal{D}$ containing x_{END} ▷ *Checkpoint area: anti-diagonal passing from x_{END}.*
16: $total_k = \mathcal{L}_k + C_k\ \forall k \in [0, \mathsf{T}]$ ▷ *Costs at checkpoint.*
17: $d_{MIN} \leftarrow \mathtt{ArgMin}(total)$ ▷ *Index of minimum value.*
18: $accum \leftarrow accum + \mathtt{Min}(total)$ ▷ *Add minimum value.*
19: $C_i \leftarrow \mathtt{Abs}(i - d_{MIN})\ \forall i \in [0, \mathsf{T})$ ▷ *Dist. from d_{MIN} (Alg. 4).*
20: **return** $accum + \mathtt{Min}(C)$ ▷ *Final threshold T'.*

As shown in Fig. 2a, we begin by solely focusing on paths that follow diagonals, which follows the second observation in Section 4.1 that substitutions dominate edits for genome sequences. To illustrate, we consider the path corresponding to the first (or similarly last) diagonal $\lfloor -\mathsf{T}/2 \rfloor$ ($\lfloor \mathsf{T}/2 \rfloor$ respectively). Starting from cell $\mathcal{D}_{0,0}$, we execute $\lfloor \mathsf{T}/2 \rfloor$ insertions (or deletions) to reach the first (or last) diagonal. We account for these insertions and deletions in a separate array called C. In more detail, each element of array C represents the number of steps (cost) needed to convert the corresponding index to the leading diagonal and counts the cost of insertion and deletion at both ends. In Step 3 of Alg. 2, we initialize C with the offset from the leading diagonal, which is in the middle of our enumeration $[\lfloor -\mathsf{T}/2 \rfloor, \ldots, 0, \ldots, \lfloor \mathsf{T}/2 \rfloor]$. For instance, in Fig. 2b we maintain seven diagonals (thus, $\mathsf{T} = 7$), C is initialized as $[3, 2, 1, 0, 1, 2, 3]$; meaning that the cost of moving from the first diagonal to the leading one is 3.

Subsequently, for each of these T starting points, the path progresses diagonally to perform substitutions (if necessary). The substitution cost is stored in the matrix $\mathcal{D}$ – note that this is different from the array C that holds the costs of insertions and deletions. The first row and column of matrix $\mathcal{D}$ are initialized with zeros as $\mathcal{D}_{0,j} = \mathcal{D}_{i,0} = 0$. The value for $\mathcal{D}_{i,j}$ is updated for each step (Step 14 in Alg. 2) according to the following equation:

$$\mathcal{D}_{i,j} = \begin{cases} \mathcal{D}_{i-1,j-1} & \text{if } \alpha_i = \beta_j \\ \mathcal{D}_{i-1,j-1} + 1 & \text{if } \alpha_i \neq \beta_j \end{cases} \tag{2}$$

It is clear that the value of $\mathcal{D}_{i,j}$ only depends on the previous value $\mathcal{D}_{i-1,j-1}$ in the same diagonal.

Once we reach the end (highlighted blue area in Fig. 2a), for each diagonal (up to $\lfloor \mathsf{T}/2 \rfloor$ from each side), we execute deletions (or insertions) to return to the leading diagonal at the cell $\mathcal{D}_{n,n}$ (which incurs an extra C cost). This approach is equivalent to confining insertions/deletions to the beginnings and ends, permitting substitutions only in between – i.e., it only moves diagonally across $\mathcal{D}$, except in the beginning and at the end. We then incorporate to C the additional cost to return to the leading diagonal from each blue cell (i.e., $C = C + [3, 2, 1, 0, 1, 2, 3] = [6, 4, 2, 0, 2, 4, 6]$). Finally, the ultimate upper bound T' is determined by adding the updated C to the highlighted blue values of $\mathcal{D}$ and selecting the minimum among all T values.

This outcome is similar to determining the minimum edit distance involving substitutions only across T diagonals, incorporating insertions/deletions to both ends of α or β. These steps can be executed in $\mathcal{O}(n)$ time since each diagonal is independent (as we only care about substitutions) and computation for each diagonal can be made in parallel. However, this approach may not consistently yield a nearly optimal upper bound T' for the actual edit distance. Typically, in the WF algorithm, the path corresponding to edit distance involves taking multiple downward or rightward steps to select the best diagonals to traverse. As these steps only incur a single comparison (for whether the two characters we currently inspect are not equal to increase the cost by one), the main objective is to maximize the number of steps available before reaching the endpoint. In our approach in Fig. 2a, restricting insertions/deletions to the ends, while offering a tighter bound T' than T, may not align with the goal of the WF algorithm, i.e., maximizing the concept

of the aforementioned efficient steps. For instance, consider the genomes α = GACATTACGCA and β = GACTTACGCAA. The true edit distance between them would be 2, involving the insertion of A at the fourth position and the removal of A from the last position in β. However, the approach illustrated in Fig. 2a would result in 6:

```
G A C A T T A C G C A
G A C T T A C G C A A
0 0 0 1 1 2 3 4 5 6 6
```

that significantly deviates from the true value.

Checkpoints for Synchronization. To address this issue, we break down our algorithm into segments and iteratively apply it in smaller partitions, rather than across the entire matrix $\mathcal{D}$. This allows us to periodically switch to the better diagonal and determine the minimum accumulation, as depicted with a blue background in Fig. 2b. Checkpoints are chosen periodically along the diagonals based on segment length x as described below. Starting from the same initial point for C, let x denote the length of the segment, i.e., the number of cells the algorithm will visit before reaching a synchronization "checkpoint". Thus, the size of each segment determines how frequent the checkpoints will be. We discuss the choice of segment length to select checkpoints in Section 6.1 so that the T' is close to the ED. As before, $\mathcal{D}$ is filled out, according to Eq. (2). However, at the end of each segment (i.e., the blue checkpoint in Fig. 2b), we incorporate the offset cost from C for each diagonal into the corresponding cells in $\mathcal{D}$. We isolate the blue area of $\mathcal{D}$ in $\mathcal{L}$, and, at each checkpoint, we compute the minimum value of $\mathcal{L} + C$ among the T diagonals (Steps 15-17 in Alg. 2) and store it in *accum* along the corresponding diagonal (that the minimum value originated from), which we call d_{MIN}. We proceed as follows:

- Reset the accumulated substitutions for the next segment, i.e., for all diagonals set $\mathcal{D}$ to 0 if $\alpha_i = \beta_j$ and to 1, otherwise (Step 12 in Alg. 2).
- Update C to reflect the offset cost relative to d_{MIN} (i.e., $C_i = |i - d_{MIN}|$ shown in Step 19 of Alg. 2). For instance, if $\mathsf{T} = 7$ and $d_{MIN} = 2$, the updated C would be $[2, 1, 0, 1, 2, 3, 4]$.

We then move on to the next segment and repeat the process of adding cells of the updated C to the corresponding cells in $\mathcal{D}$, find the minimum, and add it to *accum*.

In other words, for every segment, the minimum value for each diagonal is computed by adding the number of substitutions within the current segment to the offset from it to the optimal diagonal from the previous segment. After iterating through this process and finishing the computation for the final segment (until reaching the bottom right), we then incorporate the cost to "return back" (offset) to the leading diagonal into the *accum* value obtained (Step 19 in Alg. 2).

Ridge Walking on Checkpoints. Our algorithm for finding the tight threshold relies on synchronizing the costs between the multiple diagonals on multiple checkpoints. We now explain our algorithm for this synchronization, called *ridge walking*. Ridge walking involves traversing along two anti-diagonals[2] in a "zigzag" pattern and collecting the cells of interest in the checkpoint. For example, the cells with a blue background in Fig. 2c are filled during a single iteration of ridge walking. (We exclude the first row and column in this diagram, so the starting cell here is $\mathcal{D}_{1,1}$ rather than $\mathcal{D}_{0,0}$). Values for each cell are shown for the first segment and the beginning of the second. The cells in $\mathcal{D}$ highlighted in blue depict the hamming distance for each diagonal, which are $[2, 3, 3, 1, 3, 2, 2]$. In this case, C (i.e., the cost of moving horizontally and vertically by additions and deletions in the beginning) is initialized as $[3, 2, 1, 0, 1, 2, 3]$. Thus, the total cost for each diagonal is given by $\mathcal{D} + C = [5, 5, 4, 1, 4, 4, 5]$. The minimum is 1, and the index for this value is 3 (across the leading diagonal). Consequently, *accum* is updated to 1, and C is updated as $[3, 2, 1, 0, 1, 2, 3]$, reflecting the offset for the diagonal with the minimum cost in the first segment. This iteration continues until the last segment, and *accum* provides the upper bound T' as the output.

Putting Everything Together. In the traditional Ukkonen's algorithm, the minimum of three values, as per Eq. (1), must be computed $\mathsf{T} \times n$ times. However, in our approach, with a segment length of x, we only need to compute the minimum of T values $2n/x$ times. This strategy renders our algorithm practical in a privacy-preserving setting since computing the minimum is resource-intensive and requires multiple rounds of communication. The choice of segment length x serves as a configurable parameter, enabling better upper bounds for the edit distance. The choice of x depends on the statistics and distribution of the data. Of course, when $x = 1$, the algorithm converges into the normal DP method. For the iDASH dataset, we have empirically found that the optimal value for x is around 60. More details about the choice of x can be found in Section 6.1. Alternatively, if the exact distance is needed, this *accum* could act as the tighter threshold T' and could be used to run Ukkonen's algorithm. The updated algorithm significantly improves the nearly optimal upper bound T', resulting in 3 instead of 6 for the example mentioned earlier.

In summary, our algorithm for determining the upper bound T' follows a greedy approach. It breaks down large genomic sequences into smaller segments, computes the substitution distance for each segment, and, during checkpointing, adjusts it with the offset cost from the leading diagonal. Subsequently, for the following segments, the cost adjustment is made relative to the diagonal that had the minimum cost-adjusted distance. The loose upper bound T can be chosen to be a much higher value compared to what is feasible with the regular Ukkonen algorithm. From T, we achieve the tight bound T'. T is required to be greater or equal to the true edit distance which is easy to select since two arbitrary DNA sequences have over 99.5% nucleotides being identical [18]. We choose $\mathsf{T} = 10\%$ and it works well in practice. More discussion can be found in Section 6. This flexibility enables our protocol to achieve the exact edit distance. The following lemma demonstrates that the output of Alg. 2 is indeed an upper bound for the true edit distance.

Lemma 4.2. *For input sequences $\alpha \in \mathcal{A}^m$ and $\beta \in \mathcal{A}^n$ with a pre-defined loose threshold T and a segmentation length $x > \mathsf{T}$, the output $\mathsf{T}' = \mathtt{FindThreshold}(\alpha, \beta, \mathsf{T}, x)$ from Alg. 2 is a tight upper bound for the edit distance $ED(\alpha, \beta)$.*

[2]We call anti-diagonals the diagonals that are parallel to the diagonal from bottom left to the top right.

Proof. The output T' from Alg. 2 indicates a path that converts α to β by applying insertion, deletion, and substitution in certain places. By the definition of edit distance which is the minimum number of applying these edits without restriction, $\mathsf{T}' \geq ED(\alpha, \beta)$. □

5 Complete Protocols

We now present our SecurED protocols using secret sharing and garbled circuits. At a high level, SecurED combines our novel method for determining the upper bound threshold T' with our modified privacy-preserving Ukkonen algorithm. Secret sharing involves distributing input data among participants, resulting in low total communication, albeit potentially requiring more communication rounds. On the other hand, garbled circuits facilitate arbitrary functionalities in a fixed number of rounds, but may lead to larger data transmission due to garble table encoding. We formally present both the SS-based and GC-based SecurED protocols in Fig. 3.

Parameters:

- Sender $\mathcal{S}$ and Receiver $\mathcal{R}$, alphabet $\mathcal{A}$, the threshold T, lengths m, n, and segment size x.
- Algorithm `FindThreshold()` described in Algorithm 2.
- Algorithm `UkkonenED()` described in Algorithm 1.

Input: A sequence $\alpha \in \mathcal{A}^m$ from $\mathcal{R}$, a sequence $\beta \in \mathcal{A}^n$ from $\mathcal{S}$

Secret-Sharing-Based Protocol:

(1) $\mathcal{R}$ secret shares $[\![\alpha]\!]$ to $\mathcal{S}$.
(2) $\mathcal{S}$ secret shares $[\![\beta]\!]$ to $\mathcal{R}$.
(3) Both parties compute $[\![\mathsf{T}']\!] \leftarrow$ `FindThreshold`$([\![\alpha]\!], [\![\beta]\!], \mathsf{T}, x)$, and reconstruct T'.
(4) Both parties compute $[\![ED]\!] \leftarrow$ `UkkonenED`$(\mathsf{T}', [\![\alpha]\!], [\![\beta]\!])$
(5) $\mathcal{S}$ and $\mathcal{R}$ exchange $[\![ED]\!]$ and reconstruct the output $ED(\alpha, \beta)$.

Garbled-Circuits-Based Protocol:

(1) Both parties compute $\mathsf{T}' \leftarrow \mathcal{GC}($`FindThreshold`$, \alpha, \beta, \mathsf{T}, x)$ as described by the $\mathcal{GC}$ functionality in Section 3.5.2.
(2) Both parties compute $ED \leftarrow \mathcal{GC}($`UkkonenED`$, \mathsf{T}', \alpha, \beta)$.

Figure 3: SecurED: Our Secure Edit Distance Protocol.

5.1 Generic Optimizations

We introduce optimizations for both garbled-circuit and secret-sharing-based protocols. In comparing the genomes α and β, given that nucleotides are limited to four types $\mathcal{A} = \{\mathsf{A}, \mathsf{C}, \mathsf{G}, \mathsf{T}\}$, they can be efficiently encoded as two bits $\{00, 01, 10, 11\}$. This enables efficient comparisons using small bit lengths or binary `XOR` operations.

Additionally, our `FindThreshold` algorithm (Alg. 2) requires comparing each α_i with each β_i in Step 12. This comparison is also necessary for the Ukkonen algorithm (Step 8, Alg. 1). Therefore, we can cache and reuse the comparison result. Regarding ridge walking on checkpoints, we do not need to store the row and column indices separately; we only need the indices of the start and end cells which we can obtain from the previous start and end cell indices in constant time. In our protocol for threshold determination, it is evident that maintaining the whole path is unnecessary. Thus, instead of matrix $\mathcal{D}$, we only keep an array ($\mathcal{L}$) that contains only the latest values of $\mathcal{D}$ for each diagonal. This adjustment reduces space complexity to $\mathcal{O}(\mathsf{T})$ instead of $\mathcal{O}(\mathsf{T} \times n)$. Combining

Figure 4: Optimization in [16]: $(\tau + 1)$ boxes where $\tau = 2, 3, 4$.

all these optimizations in conjunction with our novel threshold determination algorithm allows us to achieve practical DNA edit distance computation. Note that our implementation uses these optimizations but for presentation purposes, we have not included them in the main protocol.

5.2 Optimizing Secret Sharing with Boxes

The primary bottleneck of our edit distance algorithm based on secret-sharing is the number of rounds of interaction between two parties. Recall that each cell is filled based on the minimum of the three neighboring cells (above, left, and above left), as seen in Eq. (1). Computing comparisons one after the other for each cell to determine the minimum of the three surrounding cells results in a blow up in the number of communication rounds. [16] omits certain cells in the dynamic programming matrix from the computation to reduce the multiplicative depth of homomorphic encryption (HE). Similarly to [41], we can apply the HE approach by [16] to reduce the number of rounds by "ignoring" $\mathcal{D}_{i-1,j-1}$ and deducing the information from its surrounding cells. Let the equality between α_i and β_j be denoted by $\mathcal{E}_{i,j}$. As depicted in Fig. 4, with $\mathcal{D}_{i-1,j-1}$ being the white cell in the middle, we can compute $\mathcal{D}_{i,j}$ as:

$$\mathcal{D}_{i,j} = min \begin{cases} \mathcal{D}_{i-2,j-2} + \mathcal{E}_{i-1,j-1} + \mathcal{E}_{i,j} \\ \mathcal{D}_{i-1,j-2} + \mathcal{E}_{i,j-1} + 1 \\ \mathcal{D}_{i-1,j-2} + \mathcal{E}_{i,j} + 1 \\ \mathcal{D}_{i,j-2} + 2 \\ \mathcal{D}_{i-2,j-1} + \mathcal{E}_{i-1,j} + 1 \\ \mathcal{D}_{i-2,j-1} + \mathcal{E}_{i,j} + 1 \\ \mathcal{D}_{i-2,j} + 2 \end{cases}$$

Although this expansion, rather than storing and reusing $\mathcal{D}_{i-1,j-1}$, would increase the number of multiplications and comparisons (and thus the total communication), it decreases the communication rounds as all these operations can happen together. In Fig. 4, we illustrate this optimization where cells with a white background are omitted for different box sizes. We need to carefully consider the value τ as if we make it too large, although the number of rounds decreases, it significantly increases the bandwidth and the number of arithmetic operations. We employ this optimization in our secret-sharing implementation.

5.3 Garbled Circuits

For our garbled circuit approach, we leverage Free-`XOR` [29], half-gates [46], and fixed-key AES garbling optimizations [7]. We compose the circuit such that the additions in Alg. 1 are done in parallel. Note that the optimizations proposed in [48] are orthogonal and could be applied in SecurED to further accelerate our protocol. Our

implementation has the commonly used optimizations mentioned above, and it already surpasses [48].

5.4 Complexity

This section presents the complexity of our approach and examines the performance enhancements compared to existing methods [40, 41, 48]. Considering the input $\alpha \in \mathcal{A}^m$, $\beta \in \mathcal{A}^n$, and a loose upper bound T (such as $\mathsf{T} = 0.1 \cdot \max(m, n)$), our secure edit distance protocol consists of two phases. First, we determine the threshold T' using $\mathtt{FindThreshold}(\alpha, \beta, \mathsf{T})$, and then compute the edit distance $ED(\alpha, \beta)$ using $\mathtt{UkkonenED}(\alpha, \beta, \mathsf{T}')$.

The FindThreshold algorithm fills T diagonals of matrix $\mathcal{D}$ and updating C for each segment. The overall computation and space complexity is $\mathcal{O}(\mathsf{T} \cdot min(m, n))$. The diagonal-wise computation for $\mathcal{D}_{i,j}$ in Eq. (2) can be parallelized, resulting in a time complexity of $\mathcal{O}(min(m, n))$. Since we only need to learn the upper bound T', the space complexity can be reduced to $\mathcal{O}(\mathsf{T})$. For the second phase, the complexity of Ukkonen's algorithm with a threshold T' is $\mathcal{O}(\mathsf{T}' \cdot min(m, n))$ in time and $\mathcal{O}(\mathsf{T}' \cdot min(m, n))$ in space.

Comparison to other threshold determination algorithms. To the best of our knowledge, there is no previous work specifically focused on finding the threshold for the Ukkonen algorithm. In the original Ukkonen paper [40], an algorithm is provided to check whether a value T is larger than the true edit distance (see Test 1 and Test 2 in [40]). The main idea of this algorithm is to naively perform the Ukkonen algorithm with the threshold T and then check the validity of the output.

A binary search method can indeed be employed when given a loose upper bound T. This approach results in a computation complexity of $\mathcal{O}(\mathsf{T} \log \mathsf{T} \cdot min(m, n))$ and a space complexity of $\mathcal{O}(\mathsf{T} \log \mathsf{T})$. Thus, our FindThreshold algorithm reduces both complexities by $\mathcal{O}(\log \mathsf{T})$. Note that the majority of the computation in our algorithm involves comparisons from Eq. 2 rather than minimum computations done at the end of each segment, further enhancing performance.

Comparison to other exact protocols. Previous works for secure edit distance [41, 48] built on the Wagner-Fischer (WF) algorithm, which has a computation complexity of $\mathcal{O}(m \cdot n)$. In contrast, our UkkonenED algorithm has an overall complexity of $\mathcal{O}(\mathsf{T}' \cdot min(m, n))$. In the case of genome sequences, T' is typically much smaller than $max(m, n)$, significantly enhancing the performance of the protocol. For conducting fair comparisons against [41, 48], we selected datasets (iDASH) where each pair of genome sequences has a large edit distance. Note that this choice favors [41, 48] as it requires larger T' values for our protocol, while their protocol remains unaffected. Yet, we demonstrate a substantial improvement in terms of running time, ranging from 2 to 24 times faster compared to existing protocols. The detailed performance comparison between our protocol and [41, 48] is presented in Table 2. Further details are discussed next (Section 6).

5.5 Discussion on Leakage

We now discuss the security of our methodology. Our protocol reveals the value of T' from FindThreshold (Step (3) for secret-sharing-based protocol and Step (1) for garbled-circuits-based protocol in Fig. 3) and uses it as the input for the following steps. Ideally, for each party, it would be possible to simulate the value of T' given the input of that party and its output from the protocol [31].

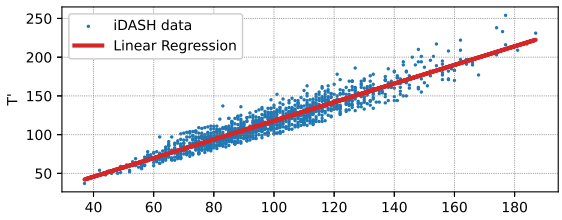


Figure 5: Distribution of T' and edit-distance over iDASH dataset. The T' is based on the initial $\mathsf{T} = 10\%$ and segmentation length of 50.

To further analyze the security of our two-step protocol, we explored the iDASH dataset.[3] We use an initial threshold of 10% and a segment length of 50 (these values are empirically found in Section 6.1). For each pair of DNA sequences in the iDASH dataset, we compute the value of T' and the ground truth edit distance and examine their correlation by computing the Pearson correlation coefficient. The coefficient value is 0.95, suggesting a strong linear relationship between T' and the edit distance. We already know from Lemma 4.2 that $\mathsf{T}' \geq ED$. We investigate this further by analyzing the unified gap $G = (\mathsf{T}' - ED)/ED$. The Pearson correlation coefficient of G and ED is 0.06, indicating a negligible linear relationship. Based on these observations, we propose the following hypothesis:

It is possible to estimate the value of T' given the true edit distance, and the error for this estimation is small and independent of each computation.

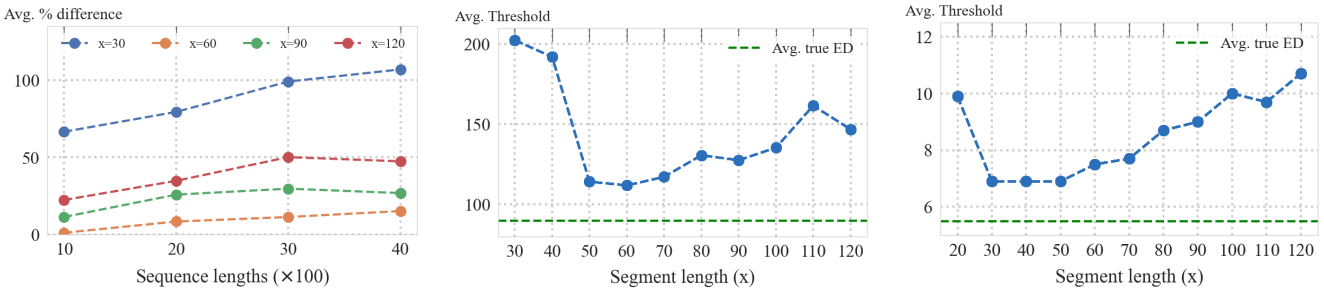
To test this hypothesis, we applied linear regression over the iDASH dataset. As shown in Fig. 5, the regression line fits the data tightly. Given the input of ED, the estimated value of T' has an error that is only 7% overall. Thus, we can effectively estimate T' given only the output of the protocol (and vice versa). This indicates that any information leaked by T' is minor. Notably, we also tried to assess this information leakage of our protocol using different T and T' values but we could not leak anything meaningful in practice. To the best of our knowledge, there does not exist any practical attack over the leakage of edit distance or any other similarity measurements.

Regardless, although we do not have formal proof, we also were unable to leak any significant information. In conjunction with our empirical results, we believe our protocol is secure in practice.

6 Experimental Evaluation

We implement our SecurED protocols using the MP-SPDZ [27] MPC library. Our implementation includes both secret-sharing-based and garbled-circuit-based approaches in the two-party setting.[4] Although our implementation can straightforwardly support the semi-honest and delegated setting by selecting appropriate parameters on MP-SPDZ, we report the numbers for this setting to ensure a fair comparison with existing protocols.

[3] We computed the edit distance between each pair of the iDASH dataset (1275 pairs).
[4] Our code is open-source at https://github.com/asu-crypto/secureED.



(a) Avg. % difference of T′ and true ED. **(b) Avg. T′ on iDASH data.** **(c) Avg. T′ on PGP data.**

Figure 6: Performance of `FindThreshold` with different sequence and segment lengths.

Our reported results are gathered as an average of ten runs.

We conducted a series of experiments on a single server equipped with AMD EPYC 74F3 processors and 256GB of RAM. All parties were run on the same network, with a simulated network connection using the Linux `tc` command. We simulated a LAN setting with 1ms round-trip latency and 2 Gbps network bandwidth, as well as a WAN setting with 40ms round-trip latency and 200 Mbps network bandwidth.

Our experiments utilize the iDASH 2016 dataset to ensure fair comparisons with existing work. The iDASH dataset is widely used in privacy-preserving edit distance protocols [1] and their associated applications [15, 32, 47]. Additionally, for our `FindThreshold` experiment, we use a dataset generated from genome sections of two individuals sourced from the Personal Genomes Project [17]. We evaluate the performance of our protocol across varying values of sequence length $m = n \in \{1000, 2000, 3000, 4000\}$.[5] Short sequence lengths are typically employed for exact edit distance in applications related to disease identification and tests focused on specific genome segments, such as paternity tests [4]. As we mentioned in the discussion above Theorem 4.1, a T larger than the edit distance is preferred as it always guarantees correct results. On the other hand, a smaller T marginally increases the efficiency of our framework but may result in inexact computation. The iDASH dataset exhibits high variance, with maximum edit distances between any two sequences ranging from 5.6–7.6% of the sequence length. For all experiments, we use set $\mathsf{T} = 10\%$, which has been proven to work effectively. However, in general, the edit distance between arbitrary DNA sequences should be much lower than this value – typically below 0.5% of the sequence length [18]. We verified this using the PGP dataset, which has low variance and confirmed that our algorithm performs well with $\mathsf{T} = 10\%$ across both high- and low-variance datasets.

6.1 Performance Evaluation of Optimal Threshold Protocol

We begin by presenting the efficacy of our upper bound determination protocol (`FindThreshold`). To ensure the accuracy of the tight upper bound T', we exhaustively consider all pairs of genomic sequences within the iDASH dataset. Subsequently, we assess the average error, defined as $|e - \mathsf{T}'|/e$, where e represents the true edit distance (the optimal bound) and T' denotes the output from our `FindThreshold`.

We conducted experiments with various segment lengths x to determine the optimal value with the lowest average error compared to the true edit distance. Smaller segment lengths result in more frequent checkpointing (see Section 4.2). Exploring segment lengths x within the set 30, 60, 90, 120, we assessed the error rate across different genome lengths $m, n \in \{1000, 2000, 3000, 4000\}$. As shown in Fig. 6a, the impact of the sequence length (m and n) on performance is relatively low. We observe that for segment lengths $x = 60$ and 90, the accuracy remains relatively unaffected for increasing sequence lengths (1000 to 4000) and also achieves the most accurate results. On the other hand, for $x = 30$ and 120 the accuracy varies more, which points towards 60 being a good segment length. Note that due to the greedy nature of our approach, the accuracy level is not inversely proportional to the segment length. In fact, setting small segment lengths results in poorer performance.

To find the optimal segment length for the iDASH dataset, we investigated segment lengths ranging from 30 to 120 in increments of 10, as depicted in Fig. 6b. We compared the resulting upper bounds with the true edit distance of all pairs of complete genomes in iDASH which is represented by the green dashed line. We found that a segment length of 60 yields the best upper bound for the iDASH dataset on average, with only a 24% deviation from the true value. Additionally, we evaluated the accuracy of our protocol on genome sections from the Personal Genome Project (PGP) [17] dataset in Fig. 6. This dataset exhibits less variability, with significantly lower average edit distances compared to iDASH. We conducted experiments with 60 sections, each of length 1000, and tested various segment lengths. Among these, segment lengths of 40 and 50 yielded the best results, achieving an average upper bound of 6.9 compared to the true edit distance of 5.5.

The optimal segment length depends on the dataset characteristics and can be determined through experimentation with pairs of small sequences. Importantly, the effectiveness of segment lengths remains consistent regardless of the sequence length. Based on our experiments with various genome datasets, we conclude that setting the segment length x around 50 is appropriate for our protocol `FindThreshold`.

[5] For sequence length 4000, since the iDASH samples are only of length $\sim$ 3500, we replicate at the end. For instance, if a genome sample is of length 3425, we repeat letters 1 to 625 in the end to reach 4000.

Table 2: Performance comparison with related works. The variable τ represents the box size optimization. Cells with a "-" denote trials that are not supported or not reported by the paper or take longer than 8 hours.

Setting	Sequence Length n	Garbled Circuits (GC)		Secret Sharing (SS)		SS with $\tau = 3$		SS with $\tau = 4$	
		[48]	SecurED	WF	SecurED	[41]	SecurED	[41]	SecurED
LAN (sec.)	1000	-	0.704	26270	1082	7595	296.8	6364	262.5
	2000	-	2.650	-	3581	29349	978.6	-	902.7
	3000	-	6.960	-	6865	-	1882	-	1729
	4000	23.72	11.43	-	11393	-	3079	-	3044
WAN (sec.)	1000	-	8.230	-	42803	-	11463	-	9971
	2000	-	27.55	-	-	-	-	-	-
	3000	-	56.14	-	-	-	-	-	-
	4000	178.0	79.62	-	-	-	-	-	-
Data (MB)	1000	-	255.2	1214	125.3	2641	276.2	4207	404.0
	2000	-	948.7	6001	434.4	13172	935.0	21091	1402
	3000	-	1983	13409	866.8	30514	1843	48826	2720
	4000	4188	3370	-	1440	-	3033	-	4798

The algorithm is very effective in providing a tight upper bound threshold relative to the length of the sequence, which is the highest possible ED. We show the comparison for T, T', and ED for each pair of sequences from the iDASH dataset in Fig. 7. It only contributes 10% of the computational cost and 25% of the communication cost to our main edit distance protocol. However, it significantly enhances the performance of our final protocol since we only need to execute UkkonenED with a much smaller threshold. On average, using the segment length that yielded the optimal results, T' was found to be $40 - 60\%$ of T. In turn, this introduces about $50 - 70\%$ saving in execution time and bandwidth for the second phase (Ukkonen) depending upon the underlying MPC protocol.

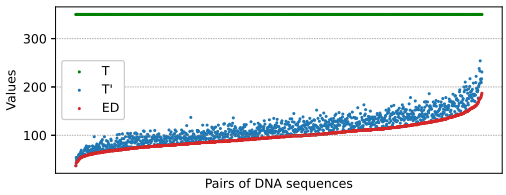


Figure 7: Loose upper bound T, tight upper bound T', and edit-distance ED for each pair of DNA sequences from the iDASH dataset. The data is sorted by ED and the T' is permuted accordingly for better visualization. Note, T is 10% since we used sequences of length 3500.

6.2 Secure Edit Distance Protocols

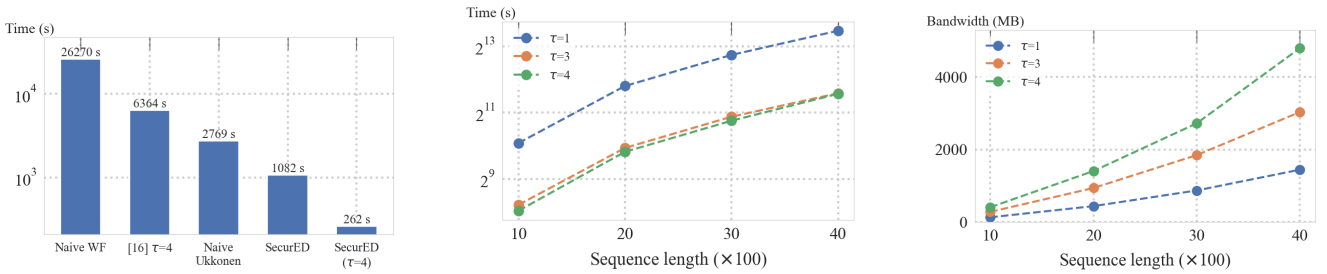
We present an end-to-end performance comparison between our SecurED protocol and existing works [41, 48] in the semi-honest setting in Table 2. For the evaluation of [41], we utilized their provided code available at https://github.com/hdvanegasm/sec-edit-distance. However, since the implementation of [48] is not publicly available, we obtained the reported numbers from [48, Table 1]. To ensure a fair comparison, we use the same LAN and WAN network settings of [48] for our experiments. We do not have the performance results for lower sequence lengths for [48]. We were also unable to execute the algorithm proposed by VCA [41] for larger sequence lengths on LAN and WAN, as the execution time exceeded eight hours.

6.2.1 Secret-sharing. Implementing the secure version of the naive WF algorithm with secret sharing becomes impractical due to the significant number of rounds required, reaching into the millions, even for genomes as short as 1000. In our experiments on a LAN network, the algorithm terminated after over 8 hours. To reduce the number of rounds, as discussed in Section 5.2, [41] employed $\tau + 1$ boxes. We opted to restrict τ to 4 since bandwidth increases exponentially for higher τ values. When $\tau = 4$, they managed to execute the algorithm in approximately 35 minutes on a LAN with 0.3 ms latency. We replicated their implementation and tested it on a 1 ms latency LAN, where it took about 1 hour and 46 minutes. This duration is considered as our baseline for the $\tau + 1$ box approach.

A naive implementation of Ukkonen's algorithm based on secret sharing encounters a significant bottleneck due to the high threshold necessary to ensure accurate results, leading to a substantial number of communication rounds, albeit far fewer than the general WF algorithm. On a LAN, the baseline Ukkonen algorithm with a threshold of 10% of $max(m, n)$ terminated in approximately 46 minutes. Our implementation, encompassing both phases of the main protocol, had a running time of about 18 minutes. We further optimized it with $\tau = 5$ boxes and our end-to-end time for SecurED decreased to only 4 minutes and 20 seconds.

When compared to the naive WF algorithm, our approach demonstrates a 100× improvement, attributed to a combination of factors including Ukkonen's bounds, threshold optimization, $\tau + 1$ boxes, and implementation optimizations such as binary XORs and reusing comparisons. Compared to the state-of-the-art secret sharing protocol [41], our method demonstrates more than 24.24× running time improvement and more than 9.56× reduction in terms of communication cost. We further evaluated the performance of our approach across different sequence lengths, reporting our execution time and bandwidth in Fig. 8.

6.2.2 Garbled-circuit. Implementing the same protocols with garbled circuits [45] improved the performance of the protocol compared to our secret sharing-based approach. For a sequence length of

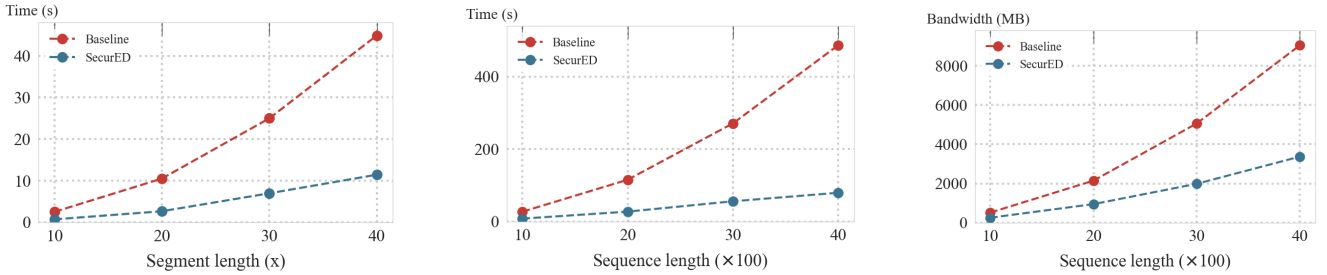


(a) Running times of different SecurED schemes on the sequence length 1000.

(b) Running time of our SecurED.

(c) Bandwidth cost of our SecurED.

Figure 8: Running time and Bandwidth Cost of Secret Sharing based SecurED in LAN.



(a) Running time in LAN.

(b) Running time in WAN.

(c) Bandwidth Cost.

Figure 9: Running time and Bandwidth of our SecurED protocol based on Garbled Circuits compared with the naive Ukkonen approach.

4000 on a LAN, our algorithm required approximately 11.4 seconds compared to the state-of-the-art GC-based protocol [48], which achieved 23.7 seconds, effectively halving the time with standard optimizations. As mentioned earlier, integrating [48]'s customized garbled circuit approach could further enhance running time performance. To ensure an unbiased comparison, we consider the general Ukkonen algorithm with a large threshold (10%) as our baseline and evaluate our performance against it for different sequence lengths. We present the results in Fig. 9.

6.2.3 Comparison to Approximation protocols. The work of [48] provides a comparison between exact computation protocols and approximation protocols, which we briefly discuss here for completeness. As introduced in Section 2.2, the current best approximation protocols are [1, 44]. In [44], the approximation is derived by transforming the edit distance problem into the set difference problem. Meanwhile, in [1], the approximation edit distance is computed as the summation of edit distances for each segment. However, the correctness of both methods heavily relies on the distribution of genome sequences, and their accuracy lacks formal proofs.

Another issue with approximation protocols is their sensitivity to the choice of the reference sequence. [48] demonstrates that with a randomly generated reference sequence, both [1, 44] exhibit poor accuracy of 75% and 59%, respectively, in terms of root mean square relative error. In contrast, our approach is independent of any reference genome and does not have such restrictions. Additionally, DP-based exact computation protocols such as [41, 48], as well as ours, allow for finding the path to convert sequence α to sequence β, whereas approximation protocols [1, 44] only provide the value of the edit distance.

7 Conclusion

This work has presented SecurED, a novel MPC protocol that overcomes the limitations of existing secure DNA edit distance solutions. The core innovation of our protocol is a novel transformation of the established approximate matching technique, specifically the Ukkonen algorithm, into an exact matching algorithm. We have leveraged the inherent similarity found in human DNA and introduced a privacy-preserving algorithm to find an optimal threshold for two given DNA sequences. We implemented several optimizations tailored to secure computation in DNA sequences composed solely of the four nucleotide letters. Finally, we have instantiated SecurED with secret-sharing and garbled circuits and have achieved a speedup of approximately $2-24\times$ over previous works. SecurED successfully computes secure edit distance between genomes, each containing 1000 letters, in a few seconds. An open question is to prove whether revealing the threshold introduces any leakage and explore whether a concrete attack exists that could exploit the threshold to recover input data. We also consider developing an efficient construction for the malicious setting as part of our future work.

Acknowledgments

We would like to thank the anonymous reviewers and editors for their valuable feedback and suggestions, which have significantly

improved our work. This work was partially supported by NSF awards #2101052, #2115075, and ARPA-H SP4701-23-C-0074.

References

[1] Gilad Asharov, Shai Halevi, Yehuda Lindell, and Tal Rabin. 2018. Privacy-Preserving Search of Similar Patients in Genomic Data. *Proceedings on Privacy Enhancing Technologies* 2018, 4 (Oct. 2018), 104–124. https://doi.org/10.1515/popets-2018-0034

[2] Mikhail J. Atallah and Jiangtao Li. 2004. Secure Outsourcing of Sequence Comparisons. In *PET 2004: 4th International Workshop on Privacy Enhancing Technologies (Lecture Notes in Computer Science, Vol. 3424)*, David M. Martin Jr. and Andrei Serjantov (Eds.). Springer, Berlin, Heidelberg, Germany, Toronto, Canada, 63–78. https://doi.org/10.1007/11423409_5

[3] Jianli Bai, Xiangfu Song, Xiaowu Zhang, Qifan Wang, Shujie Cui, Ee-Chien Chang, and Giovanni Russello. 2023. Mostree: Malicious Secure Private Decision Tree Evaluation with Sublinear Communication. In *Proceedings of the 39th Annual Computer Security Applications Conference* (Austin, TX, USA) *(ACSAC '23)*. Association for Computing Machinery, New York, NY, USA, 799–813. https://doi.org/10.1145/3627106.3627131

[4] Pierre Baldi, Roberta Baronio, Emiliano De Cristofaro, Paolo Gasti, and Gene Tsudik. 2011. Countering GATTACA: efficient and secure testing of fully-sequenced human genomes. In *ACM CCS 2011: 18th Conference on Computer and Communications Security*, Yan Chen, George Danezis, and Vitaly Shmatikov (Eds.). ACM Press, Chicago, Illinois, USA, 691–702. https://doi.org/10.1145/2046707.2046785

[5] Allan Balmain, Joe Gray, and Bruce Ponder. 2003. The genetics and genomics of cancer. *Nature Genetics* (2003), 238–244. https://doi.org/10.1038/ng1107

[6] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–432.

[7] Mihir Bellare, Viet Tung Hoang, Sriram Keelveedhi, and Phillip Rogaway. 2013. Efficient Garbling from a Fixed-Key Blockcipher. In *2013 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, Berkeley, CA, USA, 478–492. https://doi.org/10.1109/SP.2013.39

[8] Marina Blanton and Fattaneh Bayatbabolghani. 2017. Improving the Security and Efficiency of Private Genomic Computation Using Server Aid. *IEEE Security Privacy* 15, 5 (2017), 20–28. https://doi.org/10.1109/MSP.2017.3681056

[9] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A Framework for Fast Privacy-Preserving Computations. In *ESORICS 2008: 13th European Symposium on Research in Computer Security (Lecture Notes in Computer Science, Vol. 5283)*, Sushil Jajodia and Javier López (Eds.). Springer, Berlin, Heidelberg, Germany, Málaga, Spain, 192–206. https://doi.org/10.1007/978-3-540-88313-5_13

[10] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. 2021. Lightweight Techniques for Private Heavy Hitters. In *2021 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, San Francisco, CA, USA, 762–776. https://doi.org/10.1109/SP40001.2021.00048

[11] Lennart Braun, Adrià Gascón, Mariana Raykova, Phillipp Schoppmann, and Karn Seth. 2024. Malicious Security for Sparse Private Histograms. Cryptology ePrint Archive, Paper 2024/469. https://eprint.iacr.org/2024/469

[12] Fons Bruekers, Stefan Katzenbeisser, Klaus Kursawe, and Pim Tuyls. 2008. Privacy-Preserving Matching of DNA Profiles. Cryptology ePrint Archive, Report 2008/203. https://eprint.iacr.org/2008/203

[13] Wylie Burke and Bruce M. Psaty. 2007. Personalized Medicine in the Era of Genomics. *JAMA* 298, 14 (10 2007), 1682–1684. https://doi.org/10.1001/jama.298.14.1682

[14] Diptarka Chakraborty, Debarati Das, Elazar Goldenberg, Michal Koucký, and Michael Saks. 2020. Approximating Edit Distance Within Constant Factor in Truly Sub-quadratic Time. *J. ACM* 67, 6 (Oct. 2020), 1–22. https://doi.org/10.1145/3422823

[15] Ke Cheng, Yantian Hou, and Liangmin Wang. 2018. Secure Similar Sequence Query on Outsourced Genomic Data. In *ASIACCS 18: 13th ACM Symposium on Information, Computer and Communications Security*, Jong Kim, Gail-Joon Ahn, Seungjoo Kim, Yongdae Kim, Javier López, and Taesoo Kim (Eds.). ACM Press, Incheon, Republic of Korea, 237–251. https://doi.org/10.1145/3196494.3196535

[16] Jung Hee Cheon, Miran Kim, and Kristin E. Lauter. 2015. Homomorphic Computation of Edit Distance. In *FC 2015 Workshops (Lecture Notes in Computer Science, Vol. 8976)*, Michael Brenner, Nicolas Christin, Benjamin Johnson, and Kurt Rohloff (Eds.). Springer, Berlin, Heidelberg, Germany, San Juan, Puerto Rico, 194–212. https://doi.org/10.1007/978-3-662-48051-9_15

[17] G M Church. 2005. The Personal Genome Project. *Molecular Systems Biology* 1, 1 (2005), 2005.0030. https://doi.org/10.1038/msb4100040 arXiv:https://www.embopress.org/doi/pdf/10.1038/msb4100040

[18] International Human Genome Sequencing Consortium. 2001. Initial sequencing and analysis of the human genome. *Nature* 409 (2001), 860–921. https://doi.org/10.1038/35057062

[19] Emma Dauterman and Henry Corrigan-Gibbs. 2023. Lightweb: Private web browsing without all the baggage. In *Proceedings of the 22nd ACM Workshop on Hot Topics in Networks* (Cambridge, MA, USA) *(HotNets '23)*. Association for Computing Machinery, New York, NY, USA, 287–294. https://doi.org/10.1145/3626111.3628207

[20] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. 2023. Verifiable Distributed Aggregation Functions. *Proceedings on Privacy Enhancing Technologies* 2023, 4 (July 2023), 578–592. https://doi.org/10.56553/popets-2023-0126

[21] Daniel Demmler, Thomas Schneider, and Michael Zohner. 2015. ABY - A Framework for Efficient Mixed-Protocol Secure Two-Party Computation. In *ISOC Network and Distributed System Security Symposium – NDSS 2015*. The Internet Society, San Diego, CA, USA. https://doi.org/10.14722/ndss.2015.23113

[22] Jiahui Gao, Ni Trieu, and Avishay Yanai. 2024. Multiparty private set intersection cardinality and its applications. *Proceedings on Privacy Enhancing Technologies* 2024, 2 (July 2024), 73–90.

[23] Gayathri Garimella, Benny Pinkas, Mike Rosulek, Ni Trieu, and Avishay Yanai. 2021. Oblivious Key-Value Stores and Amplification for Private Set Intersection. In *Advances in Cryptology – CRYPTO 2021, Part II (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, Cham, Switzerland, Virtual Event, 395–425. https://doi.org/10.1007/978-3-030-84245-1_14

[24] Oded Goldreich, Silvio Micali, and Avi Wigderson. 1987. How to Play any Mental Game or A Completeness Theorem for Protocols with Honest Majority. In *19th Annual ACM Symposium on Theory of Computing*, Alfred Aho (Ed.). ACM Press, New York City, NY, USA, 218–229. https://doi.org/10.1145/28395.28420

[25] The Huntington's Disease Collaborative Research Group. 1993. A novel gene containing a trinucleotide repeat that is expanded and unstable on Huntington's disease chromosomes. *Cell* (1993). https://doi.org/10.1016/0092-8674(93)90585-e

[26] iDASH. 2016. Privacy and Security Workshop 2016. http://www.humangenomeprivacy.org/2016/.

[27] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *ACM CCS 2020: 27th Conference on Computer and Communications Security*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, Virtual Event, USA, 1575–1590. https://doi.org/10.1145/3372297.3417872

[28] Bat-Sheva Kerem, Johanna M. Rommens, Janet A. Buchanan, Danuta Markiewicz, Tara K. Cox, Aravinda Chakravarti, Manuel Buchwald, and Lap-Chee Tsui. 1989. Identification of the Cystic Fibrosis Gene: Genetic Analysis. *Science* 245, 4922 (1989), 1073–1080. https://doi.org/10.1126/science.2570460

[29] Vladimir Kolesnikov and Thomas Schneider. 2008. Improved Garbled Circuit: Free XOR Gates and Applications. In *ICALP 2008: 35th International Colloquium on Automata, Languages and Programming, Part II (Lecture Notes in Computer Science, Vol. 5126)*, Luca Aceto, Ivan Damgård, Leslie Ann Goldberg, Magnús M. Halldórsson, Anna Ingólfsdóttir, and Igor Walukiewicz (Eds.). Springer, Berlin, Heidelberg, Germany, Reykjavik, Iceland, 486–498. https://doi.org/10.1007/978-3-540-70583-3_40

[30] Vladimir Levenshtein. 1966. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics Doklady* 10, 8 (1966), 707–710.

[31] Yehuda Lindell. 2016. How To Simulate It - A Tutorial on the Simulation Proof Technique. Cryptology ePrint Archive, Report 2016/046. https://eprint.iacr.org/2016/046

[32] Md Safiur Rahman Mahdi, Md Momin Al Aziz, Dima Alhadidi, and Noman Mohammed. 2019. Secure Similar Patients Query on Encrypted Genomic Data. *IEEE Journal of Biomedical and Health Informatics* 23, 6 (2019), 2611–2618. https://doi.org/10.1109/JBHI.2018.2881086

[33] Yoshio Miki, Jeff Swensen, Donna Shattuck-Eidens, P. Andrew Futreal, Keith Harshman, Sean Tavtigian, Qingyun Liu, Charles Cochran, L. Michelle Bennett, Wei Ding, Russell Bell, Judith Rosenthal, Charles Hussey, Thanh Tran, Melody McClure, Cheryl Frye, Tom Hattier, Robert Phelps, Astrid Haugen-Strano, Harold Katcher, Kazuko Yakumo, Zahra Gholami, Daniel Shaffer, Steven Stone, Steven Bayer, Christian Wray, Robert Bogden, Priya Dayananth, John Ward, Patricia Tonin, Steven Narod, Pam K. Bristow, Frank H. Norris, Leah Helvering, Paul Morrison, Paul Rosteck, Mei Lai, J. Carl Barrett, Cathryn Lewis, Susan Neuhausen, Lisa Cannon-Albright, David Goldgar, Roger Wiseman, Alexander Kamb, and Mark H. Skolnick. 1994. A Strong Candidate for the Breast and Ovarian Cancer Susceptibility Gene BRCA1. *Science* 266, 5182 (1994), 66–71. https://doi.org/10.1126/science.7545954 arXiv:https://www.science.org/doi/pdf/10.1126/science.7545954

[34] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin M Case. 2024. Delegated Private Matching for Compute. *Proceedings on Privacy Enhancing Technologies* 2024, 2 (July 2024), 49–72.

[35] Dimitris Mouris, Christopher Patton, Hannah Davis, Pratik Sarkar, and Nektarios Georgios Tsoutsos. 2025. Mastic: Private Weighted Heavy-Hitters and Attribute-Based Metrics. *Proceedings on Privacy Enhancing Technologies* 2025, 1 (July 2025), 1–30.

[36] Dimitris Mouris, Pratik Sarkar, and Nektarios Georgios Tsoutsos. 2024. PLASMA: Private, Lightweight Aggregated Statistics against Malicious Adversaries. *Proceedings on Privacy Enhancing Technologies* 2024, 3 (July 2024), 4–24.

[37] Shantanu Rane and Wei Sun. 2010. Privacy preserving string comparisons based on Levenshtein distance. In *2010 IEEE International Workshop on Information*

Forensics and Security. 1–6. https://doi.org/10.1109/WIFS.2010.5711449

[38] Liyan Shen, Xiaojun Chen, Dakui Wang, Binxing Fang, and Ye Dong. 2018. Efficient and Private Set Intersection of Human Genomes. In *2018 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. 761–764. https://doi.org/10.1109/BIBM.2018.8621291

[39] Ben Tran, Janet E. Dancey, Suzanne Kamel-Reid, John D. McPherson, Philippe L. Bedard, Andrew M.K. Brown, Tong Zhang, Patricia Shaw, Nicole Onetto, Lincoln Stein, Thomas J. Hudson, Benjamin G. Neel, and Lillian L. Siu. 2012. Cancer Genomics: Technology, Discovery, and Translation. *Journal of Clinical Oncology* 30, 6 (2012), 647–660. https://doi.org/10.1200/JCO.2011.39.2316 PMID: 22271477.

[40] Esko Ukkonen. 1985. Algorithms for approximate string matching. *Information and Control* 64, 1 (1985), 100–118. https://doi.org/10.1016/S0019-9958(85)80046-2 International Conference on Foundations of Computation Theory.

[41] Hernán Vanegas, Daniel Cabarcas, and Diego F. Aranha. 2023. Privacy-Preserving Edit Distance Computation Using Secret-Sharing Two-Party Computation. In *Progress in Cryptology - LATINCRYPT 2023: 8th International Conference on Cryptology and Information Security in Latin America (Lecture Notes in Computer Science, Vol. 14168)*, Abdelrahaman Aly and Mehdi Tibouchi (Eds.). Springer, Cham, Switzerland, Quito, Ecuador, 67–86. https://doi.org/10.1007/978-3-031-44469-2_4

[42] Robert A Wagner and Michael J Fischer. 1974. The String-to-String Correction Problem. *Journal of the ACM (JACM)* 21, 1 (1974), 168–173.

[43] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[44] Xiao Shaun Wang, Yan Huang, Yongan Zhao, Haixu Tang, XiaoFeng Wang, and Diyue Bu. 2015. Efficient Genome-Wide, Privacy-Preserving Similar Patient Query based on Private Edit Distance. In *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, Denver, CO, USA, 492–503. https://doi.org/10.1145/2810103.2813725

[45] Andrew Chi-Chih Yao. 1986. How to Generate and Exchange Secrets (Extended Abstract). In *27th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Toronto, Ontario, Canada, 162–167. https://doi.org/10.1109/SFCS.1986.25

[46] Samee Zahur, Mike Rosulek, and David Evans. 2015. Two Halves Make a Whole - Reducing Data Transfer in Garbled Circuits Using Half Gates. In *Advances in Cryptology – EUROCRYPT 2015, Part II (Lecture Notes in Computer Science, Vol. 9057)*, Elisabeth Oswald and Marc Fischlin (Eds.). Springer, Berlin, Heidelberg, Germany, Sofia, Bulgaria, 220–250. https://doi.org/10.1007/978-3-662-46803-6_8

[47] Yandong Zheng, Rongxing Lu, Jun Shao, Yonggang Zhang, and Hui Zhu. 2019. Efficient and Privacy-Preserving Edit Distance Query Over Encrypted Genomic Data. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*. 1–6. https://doi.org/10.1109/WCSP.2019.8927885

[48] Ruiyu Zhu and Yan Huang. 2022. Efficient and Precise Secure Generalized Edit Distance and Beyond. *IEEE Transactions on Dependable and Secure Computing* 19, 1 (2022), 579–590. https://doi.org/10.1109/TDSC.2020.2984219

Appendix

For completeness, we present the building blocks for Alg. 2. Alg. 3 demonstrates the procedure to find the minimum of two values without branching and how it can be used in an array to find the array minimum. Note that `ArgMin` can be computed similarly by additionally keeping track of the indices. We use Alg. 4 for computing the absolute value without branching over encrypted data.

Algorithm 3 Minimum Value.

procedure Min(x, y) ▷ *Minimum of two values.*
  $cmp \leftarrow x < y$ ▷ *1 (True) if $x < y$, 0 (False), otherwise.*
  return $cmp \times x + (1 - cmp) \times y$
procedure Min(arr) ▷ *Minimum of an array.*
  if size(arr) = 1 **then** ▷ *If arr has a single value.*
    return arr_0
  else
    $reduced \leftarrow []$ ▷ *Empty array.*
    for every two elements $(x, y) \in arr$ **do**
      append Min(x, y) to $reduced$
    if size(arr) $\mod 2 \neq 0$ **then** ▷ *If size is odd.*
      append last element of arr to $reduced$
    return Min($reduced$) ▷ *Recursive call.*

Algorithm 4 Absolute Value.

procedure Abs(x)
  $cmp \leftarrow x < 0$ ▷ *1 (True) if $x < 0$, 0 (False), otherwise.*
  return $cmp \times (-x) + (1 - cmp) \times x$