Privacy-Preserving Protocols for of Edit Distance and Other Dynamic Programming Algorithms

Anonymous submission

Abstract

The edit distance between two strings is the minimum number of delete, insert, and replace operations needed to convert one string into another. Computational biology tasks such as comparing genome sequences of two individuals rely heavily on the dynamic programming algorithm for computing edit distances as well as the algorithms for related string-alignment problems. A genome sequence may reveal a lot of sensitive information about an individual. Therefore, it is important to develop methods for analyzing and comparing such sequences that are both cryptographically secure and efficient for practical use.

We present several protocols for securely computing the edit distance between two strings so that the owner of each string does not learn anything about the other string except the edit distance. Our protocols are provably secure in the standard multi-party computation paradigm of modern cryptography. Experimental evaluation of our prototype implementation demonstrates that it can be feasibly applied to strings of up to several hundred characters in length, which is sufficient for many practical scenarios. We also discuss how to generalize our protocol to other problems for which there exist efficient dynamic programming algorithms.

1 Introduction

The ease of access to information due to the Internet has brought privacy concerns to the forefront [3, 22]. Therefore, there is considerable interest in developing techniques [2, 6, 19] and protocols to address these privacy concerns. Specifically, privacy-preserving protocols [4, 5, 12, 14] that allow multiple parties to perform computations without revealing their private inputs have been the subject of much interest.

One of the fundamental cryptographic primitives for designing privacy-preserving protocols is *secure function evaluation (SFE)*. A protocol for SFE enables two parties A and B with respective inputs x and y to jointly compute a function $f(x, y)$ while preserving the privacy of their respective inputs, *i.e.*, A does not learn anything from the protocol execution beyond what is revealed by her own input x and the result $f(x, y)$; a similar condition holds for B.

One of the seminal results in secure multi-party computation, due to Yao [23] and Goldreich, Micali, and Wigderson [9], is that for any efficiently computable (*i.e.*, probabilistic polynomial-time) function f, there exists an efficient protocol for securely evaluating f. Details of this result can be found in standard textbooks on secure multi-party computation [8, chapter 7]. Generic constructions, however, are not always practical. Therefore, there has been much interest in developing special-purpose constructions for specific problems such as privacy-preserving auctions, surveys, and so on [4, 5, 12, 14]. In this paper, we consider the problem of privately computing edit distances between two strings. The algorithm for computing edit distance and its variants are widely used in several areas, such as computational biology. We also demonstrate how the key ideas in our privacy-preserving protocol for computing edit distances are applicable to other dynamic programming algorithms.

The *edit distance* between two strings α and β (denoted by $\delta(\alpha, \beta)$) is the minimum number of delete, insert, and replace operations needed to convert α into β. We consider private computation of edit distance, *i.e.*, user Alice has α, user Bob has β, and they want to jointly compute $\delta(\alpha, \beta)$ without revealing their individual strings, and present three protocols for solving this problem. Our first protocol is a straightforward implementation of standard protocols for secure circuit evaluation. The key novel idea behind the other two protocols is to randomly split the table of values maintained by the dynamic programming algorithm between the two parties. Protocol 3 also exploits the structure of the table of values used by the algorithm for computing edit distance. Many other problems can be efficiently solved by dynamic programming algorithms [1, Chapter 15]. We discuss how the key ideas behind our protocols can be used to construct efficient privacy-preserving protocols for these problems.

Edit distance computation and related string alignment problems are the basic tasks in many computational biology algorithms (*e.g.*, see [11, Chapter 11]). Therefore, a privacy-preserving edit distance protocol can serve as a fundamental building block in many applications that are legally required to preserve privacy of individual genome sequences. We evaluate a prototype implementation of our protocols, and demonstrate that they can be feasibly applied to problems of realistic size.

To summarize, this paper makes the following contributions:

- We address the problem of securely computing edit distance between two strings. We present three protocols in Section 3. Section 4 also discusses how our protocols can be adapted to construct privacy-preserving protocols for problems that can be efficiently solved using dynamic programming.
- We have implemented all three protocols and present our evaluation in Section 6. Our experimental results demonstrate that, using one of our protocols, it is feasible to securely compute edit distances of strings of length up to a few hundred. Since genome sequences are only a few hundred base pairs long, our protocol can be applied to genome sequence comparison and related computational problems in molecular biology.

Related work: The literature on privacy-preserving protocols is vast. To our knowledge, however, this is the first paper that investigates the problem of privately computing edit distances between two strings. In a related paper, Szajda *et al.* [21] consider distributed computation of the Smith-Waterman genome comparison algorithm [20]. Szajda *et al.* decompose the problem into several sub-problems, which are distributed to several participants. The intuition is that each participant solves a sub-problem and thus cannot infer the inputs for the original problem. However, Szajda *et al.* did not provide a formal proof of privacy of their protocol. In this paper, we give a *provably secure* protocol for privacy-preserving edit distance computation, with cryptographic security guarantees. Moreover, our protocol can be adapted to other dynamic programming algorithms.

2 Edit Distance between Two Strings

Let α and β be two strings over an alphabet Σ. Let the lengths of the two strings α and β (denoted by $\mid \alpha \mid$ and $\mid \beta \mid$) be n and m, respectively. The edit-distance between the two strings α and β (denoted by $\delta(\alpha, \beta)$) is the minimum number of edit operations (delete, insert, and replace) needed to transform α into β. We will describe a dynamic programming algorithm to compute $\delta(\alpha, \beta)$, which executes in time $O(nm)$. The description of the algorithm for computing edit-distance is based on the discussion in [11].

- Compute $D(i,0)$ and $D(0,j)$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ using equation 1.
- Compute $D(i,j)$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ in row major order using equation 3. In other words, we first compute all entries for row 1, then row 2, and so on.
- The edit distance $\delta(\alpha,\beta)$ is equal to $D(n,m)$.

Figure 1: Algorithm for computing edit distance.

Given a string α, let $\alpha[1 \cdots i]$ denote the first i characters of α. The dynamic programming algorithm maintains a $(n+1) \times (m+1)$ matrix $D(0 \cdots n, 0 \cdots m)$, where $D(i,j)$ is the edit distance between $\alpha[1 \cdots i]$ and $\beta[1 \cdots j]$.

For the base case, we have the following:

$$D(i,0) = i \ , \ 0 \leq i \leq n \quad (1)$$

$$D(0,j) = j \ , \ 0 \leq j \leq m \quad (2)$$

Next we describe a recursive relationship between the value $D(i,j)$ and the entries of D with indices smaller than i and j. The (i,j)-th entry $D(i,j)$ of the matrix is computed as follows:

$$D(i,j) = \min[D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)] \quad (3)$$

where $t(i,j)$ is defined to have value 1 if $\alpha(i) \neq \beta(j)$, and has value 0 if $\alpha(i) = \beta(j)$. The i-th character of a string α is denoted by $\alpha(i)$. The entire algorithm for computing edit distance is shown in Figure 1.

3 Privacy-Preserving Edit Distance Computation

Alice A has the string α and Bob B has the string β, with $\mid \alpha \mid = n$ and $\mid \beta \mid = m$. A and B want to jointly compute the edit distance $\delta(\alpha,\beta)$ between the two strings without revealing the strings.

3.1 Cryptographic toolkit

In our protocols below, we will employ several standard cryptographic techniques.

Oblivious transfer. The first technique is *oblivious transfer*, originally proposed by Rabin [18]. Informally, a 1-out-of-n oblivious transfer (we will denote it as OT_1^n) is a protocol between two parties, the Chooser and the Sender. The Sender's inputs into the protocol are n values $v_1, \ldots, v_n$. The Chooser's input is an index i such that $1 \leq i \leq n$. As a result of the protocol, the Chooser receives v_i, but does not learn anything about the rest of the Sender's values. The Sender learns nothing. Our protocols do not depend on a particular implementation of oblivious transfer; therefore, we simply assume that we have access to a cryptographic primitive implementing OT_1^n. In our implementations, we rely on Fairplay [15] and Naor-Pinkas oblivious transfer construction [16].

Secure circuit evaluation. We will also employ two standard methods for secure circuit evaluation: Yao's "garbled circuits" method and secure computation with shares. Consider any (arithmetic or Boolean) circuit C, and two parties, Alice and Bob, who wish to evaluate C on their respective inputs x and y. In Yao's

"garbled circuits" method, originally proposed in [23], Alice securely transforms the circuit so that Bob can evaluate it obliviously, *i.e.*, without learning Alice's inputs into the circuit or the values on any internal circuit wire except the output wires.

Alice does this by generating two random keys for each circuit wire, one representing 0 on that wire, the other representing 1. The keys encoding Alice's own inputs into the circuit she simply sends to Bob. The keys encoding Bob's inputs are transferred to Bob via the OT_1^2 protocol. For each of Bob's input wires, where Bob acts as the chooser using his circuit input bit as his input into OT_1^2, and Alice acts as the sender with the two wire keys for that wire as her inputs into OT_1^2. Alice produces the "garbled" truth table for each circuit gate in such a way that Bob, if he knows the wire keys encoding the values on the gate input wires, can decrypt exactly one row of the garbled truth table and obtain the key encoding the value of the output wire. Yao's protocol maintains the invariant that for every circuit wire, Bob learns *exactly one* wire key.

Because wire keys are random and the mapping from wire keys to values is not known to Bob (except for the wire keys corresponding to his own inputs), this does not leak any information about actual wire values. The circuit can thus be evaluated "obliviously." To save space, we omit the details. A complete description of Yao's method and security proofs can be found in [13].

The second standard method is *secure computation with shares* (SCWS). Details of this method can be found in [8, Chapter 7]. This protocol maintains the invariant that, for every circuit wire w, Alice learns a random value s and Bob learns $b_w - s$, where b_w is the bit value of the wire. Therefore, Alice's and Bob's shares add up to b_w, but because the shares are random, neither party knows the actual wire value. For each output wire of the circuit, Alice and Bob combine their shares to reconstruct the circuit output. Either Yao's "garbled circuits" method, or SCWS can be used to securely and privately evaluate any circuit C.

Additively homomorphic encryption. Let (G, E, D, M) be a public-key encryption scheme, where G is the key generation function, E and D are the encryption and decryption functions, and M is the message space respectively. We will assume that:

- The encryption scheme is *semantically secure* [10]. Informally, this means that the ciphertext leaks no useful information about the plaintext even after the adversary has previously observed many plaintext-ciphertext pairs on plaintexts of his choice.
- There exists a computational function g such that for all $m \in M$ and $\alpha \in M$, $m_1 \in E(m)$ implies that $g(m_1, \alpha) \in E(m\alpha)$. With any semantically secure encryption scheme, encrypting the same message twice will yield different ciphertexts, so $E(m)$ denotes the set of ciphertexts that can be obtained by encrypting m. [1]
- There exists a computable function f such that for all messages m_1 and m_2, the following property holds:

$$f(E(m_1), E(m_2)) \quad = \quad E(m_1 + m_2)$$

There are several encryption scheme that satisfy these properties, of which Paillier's encryption scheme is perhaps the most famous [17]. Since we will use the encryption scheme as a black-box cryptographic primitive, we omit the details of the scheme.

We present three protocols. Protocol 1 is the most straight forward and uses the standard method for secure circuit evaluation. However, protocol 1 generates very large circuits. Protocol 2 uses homomorphic

[1] Of course, to successfully decrypt two different messages m and m', sets $E(m)$ and $E(m')$ should be disjoint.

	Number of rounds	Circuits generated by the protocol
Protocol 1	Uses 1 round	Circuit for problem of size (n, m)
Protocol 2	Uses nm rounds	Uses circuit for "minimum-of-three"
Protocol 3	Uses approximately $\frac{nm}{k^2}$ rounds	Circuit for problem of size (k, k)

Figure 2: Characteristics of various protocols for problem of size (n, m).

encryption in combination with the SWCS protocol. Protocol 2 generates very simple circuits, but uses multiple rounds. Protocol 3 exploits the structure of the problem, i.e., divides the matrix D into a grid of size k and only computes values on the grid. Figure 2 shows the characteristics for the various protocols for problem of size (n, m), i.e., the two strings are of size n and m.

3.2 Protocol 1

Recall that the edit-distance algorithm maintains a $(n + 1) \times (m + 1)$ matrix $D(0 \cdots n, 0 \cdots m)$, where $D(i, j)$ is the edit distance between $\alpha[1 \cdots i]$ and $\beta[1 \cdots j]$. Let α and β be two strings over an alphabet. Note that α and β can be expressed as bit strings $bit(\alpha)$ and $bit(\beta)$ of length qn and qm, where q is equal to $\lceil \log_2(\mid \Sigma \mid) \rceil$.

The base case and recursive equation for computing $D(i, j)$ were given in equation 1. Let $C_{D(i,j)}$ be the circuit for computing $D(i, j)$ with inputs corresponding to bit representation of $\alpha[1, \cdots, i]$ and $\beta[1, \cdots, j]$. Assume that we have computed $C_{D(i-i,j)}$, $C_{D(i,j-1)}$, and $C_{D(i-1,j-1)}$. Using the recursive equation given above one can compute the circuit $C_{D(i,j)}$. The circuit $C_{D(i,j)}$ computes $D(i, j)$ by combining (i) the equality testing circuit for $t(i, j)$, (ii) three "add-1" circuits, and (iii) two "select-smaller-value" circuits. The inputs to the circuit $C_{D(i,j)}$ are bit representations of $\alpha[1, \cdots, i]$, $\beta[1, \cdots, j]$ and the outputs of circuits $C_{D(i,j-1)}$, $C_{D(i-1,j)}$, and, $C_{D(i-1,j-1)}$. Once we have the circuit representation $C_{D(n,m)}$ for the edit distance problem, we can compute $C_{D(n,m)}(\alpha, \beta)$ in a privacy-preserving manner using standard algorithms for secure circuit evaluation (see section 3.1).

3.3 Protocol 2

This protocol will rely on secure circuit evaluation with random shares (SCWS). Alice and Bob will each maintain a $(n + 1) \times (m + 1)$ matrix D_A and D_B, respectively. The protocol will maintain the invariant that every value in the matrix D is randomly shared between Alice and Bob, that is, for all $0 \leq i \leq n$ and $0 \leq j \leq m$ we have that

$$D(i, j) \quad = \quad D_A(i, j) + D_B(i, j)$$

Before the protocol starts, Alice picks an instance of the additively homomorphic encryption scheme (G, E, D, M), and generates public and private keys using G. We assume that Bob has Alice's authentic public key.

Alice fills in $D_A(i, 0)$ and $D_A(0, j)$ with random values and sends it to Bob. Bob fills $D_B(i, 0)$ with $i - D_A(i, 0)$ and $D_B(0, j)$ with $j - D_A(0, j)$.[2]

[2]Each random value must be at least 80 bits longer than any value which might appear in a matrix cell, *i.e.*, at least $\log(n+m)+80$ bits long. This ensures that adding this random value to the actual value statistically hides the latter. (With modular arithmetic over a small group, the random shares would have been shorter and perfectly, rather than statistically, hiding, but in this protocol we will be adding random values homomorphically under encryption, and the homomorphic encryption schemes in question do not allow modular arithmetic with short moduli on plaintexts.) In the rest of this protocol, all arithmetic is over integers.

Recall that the recursive equation for computing $D(i,j)$ was

$$D(i,j) \quad = \quad \min[D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)]$$

where $t(i,j)$ is defined to have value 1 if $\alpha(i) \neq \beta(j)$, and has value 0 if $\alpha(i) = \beta(j)$. The i-th character of a string α is denoted by $\alpha(i)$. Let $x(i,j) = D(i-1,j)+1$, $y(i,j) = D(i,j-1)+1$ and $z(i,j) = D(i-1,j-1)+t(i,j)$ be the three terms appearing in recursive equation shown above. Assume that Alice and Bob have computed random shares for $D(i-1,j)$ and $D(i,j-1)$ and $D(i-1,j-1)$. Next we will show how to compute random shares for three terms $x(i,j)$, $y(i,j)$ and $z(i,j)$ involved in the recursive equation.

- *Computing random shares of $x(i,j)$:*
  The random share $x_A(i,j)$ of A for the term $x(i,j)$ is $D_A(i-1,j)$. The random share $x_A(i,j)$ of B for the term $x(i,j)$ is $D_B(i-1,j)+1$.

- *Computing random shares of $y(i,j)$:*
  The random share $y_A(i,j)$ of A for the term $y(i,j)$ is $D_A(i,j-1)$. The random share $y_A(i,j)$ of B for the term $y(i,j)$ is $D_B(i,j-1)+1$.

- *Computing random shares of $z(i,j)$:*
  Computing the random shares $z_A(i,j)$ and $z_B(i,j)$ of the term $z(i,j)$ is tricky. For this computation, we will rely on additive homomorphism of the encryption scheme. Alice encrypts $D_A(i-1,j-1)$ and sends $e_A = E(D_A(i-1),j-1))$ to Bob. Bob encrypts $e_B = E(D_B(i-1,j-1))$ and computes $f(e_A,e_B)$. Recall that f is a computable function such that $f(E(m_1),E(m_2))$ is equal to $E(m_1+m_2)$. Notice that because of the homomorphic property of the encryption we have the following:

  $$\begin{aligned} f(e_A,e_B) &= E(D_A(i-1,j-1)+D_B(i-1,j-1)) \\ &= E(D(i-1,j-1)) \end{aligned}$$

  Bob also generates an $(\log(n+m)+80)$-bit long random number r, and computes $E(1-r)$ and $E(-r)$. Computing $f(f(e_A,e_B),E(1-r))$, Bob obtains $E(D(i-1,j-1)+1-r)$. Similarly, by computing $f(f(e_A,e_B),E(-r))$, Bob obtains $E(D(i-1,j-1)-r)$.

  Recall that the strings are over an alphabet $\Sigma = \{1,\cdots,w\}$. Bob creates w messages $(m_1,\cdots,m_w)$ as follows:

  $$m_k \;=\; \begin{cases} E(D(i-1,j-1)+1-r) & \text{if } k \neq \beta(j) \\ E(D(i-1,j-1)-r) & \text{if } k = \beta(j) \end{cases}$$

  Bob's random share $z_B(i,j)$ for term $z(i,j)$ is r.

  Alice and Bob execute a 1-out-of-w oblivious transfer OT_1^w protocol (see section 3.1) with Bob acting as the sender on inputs $(m_1,\cdots,m_w)$ and Alice acting as the chooser on input $\alpha(i)$. As a result of the OT_1^w protocol, Alice obtains the message $m_{\alpha(i)}$. Observe that if $\alpha(i) = \beta(j)$, then Alice obtains $E(D(i-1,j-1)-r)$; otherwise, Alice obtains $E(D(i-1,j-1)+1-r)$. Alice decrypts m_{α_i}, and sets the plaintext as her share $z_A(i,j)$ of term $z(i,j)$.

- *Computing random shares of $D(i,j)$:*
  Now Alice and Bob execute the standard protocol for secure computation with shares (see section 3.1) to compute random shares of $\min[x(i,j),y(i,j),z(i,j)]$. The circuit in question is a simple "minimum-of-three" circuit implementing $\min[a,b,c]$ of three values a, b, c.

After the last iteration, Alice sends to Bob her random share $D_A(n,m)$ and Bob sends Alice his random share $D_B(n,m)$. This enables both Alice and Bob to reconstruct the edit distance as $D_A(n,m)+D_B(n,m)$.

Optimization. The above protocol requires an 1-out-of-w oblivious transfer in each iteration, which is expensive in terms of both computation and communications. Even using amortization techniques of [16], the entire protocol will require $w + nm$ modular exponentiations, and $(w-1)nm$ modular multiplications.

We now describe an optimized version of the protocol which only requires q instances of 1-out-of-2 oblivious transfers in each iteration, where $q = \lceil \log_2 w \rceil$.

1. Bob computes $E(D(i-1,j-1)+1-r)$ and $E(D(i-1,j-1)-r)$ as before. He also generates two random keys k_0 and k_1, and sends $E_{k0}(E(D(i-1,j-1)+1-r))$ and $E_{k1}(E(D(i-1,j-1)-r))$ to Alice in random order, *i.e.*, he uses k_0 to encrypt one of the ciphertexts and k_1 to encrypt the other, and shuffles the doubly encrypted ciphertexts randomly.

2. Bob creates the standard Yao garbled circuit (see section 3.1) for testing equality of $\alpha(i)$ and $\beta(j)$, using k_0 as the wire key encoding 0 on the output wire, and k_1 as the wire key encoding 1 on the output wire.

3. Bob sends the encrypted circuit to Alice, and they execute the standard Yao protocol. Transferring the encoding of Alice's input from Bob to Alice only requires q instances of 1-out-of-2 oblivious transfer, because the bit representation of $\alpha(i)$ is q bits long (recall that $q = \lceil \log_2 w \rceil$).

4. Alice evaluates the garbled circuit, obtaining either k_0, or k_1 - she does not know which. She uses this key to decrypt exactly one of the two ciphertexts sent by Bob, and obtains either $E(D(i-1,j-1)+1-r)$ or $E(D(i-1,j-1)-r)$. She decrypts the inner ciphertext with her private key, and sets the resulting plaintext to be her random share $z_A(i,j)$.

3.4 Protocol 3

Protocol 1 requires nq executions of OT_1^2, where $q = \lceil \log_2(\mid \Sigma \mid) \rceil$, but has to compute a large circuit $C_{D(n,m)}$. Protocol 2 requires nmq executions of OT_1^2, but only has to compute much smaller circuits for equality testing and returning the minimum of three values. Because oblivious transfers are likely to be the computation and communication bottleneck in practice, *Protocol 1 is time efficient, Protocol 2 is space efficient.*

In this section, we present a hybrid of protocols 1 and 2. In this protocol, we only compute the values $D(i,j)$ in the matrix D that lie on a grid.

Recall that the algorithm to compute the edit distance maintains a $(n+1) \times (m+1)$ matrix D. Let k be a number that divides both n and m, *i.e.*, $k \mid n$ and $k \mid m$.[3] The following set of values constitute a grid of granularity k.

$$\{D(i,j) \mid 0 \le i \le n \text{ and } j \in \{0, k, 2k, \cdots, \tfrac{m}{k}k\}\}$$
$$\{D(i,j) \mid i \in \{0, k, 2k, \cdots, \tfrac{n}{k}k\} \text{ and } 0 \le j \le m\}$$

Given an element $D(i,j)$, the *rectangle* of length l and width w with $D(i,j)$ at the top right corner (denoted

[3] Our protocol can be easily extended to remove the assumption that k divides both n and m.

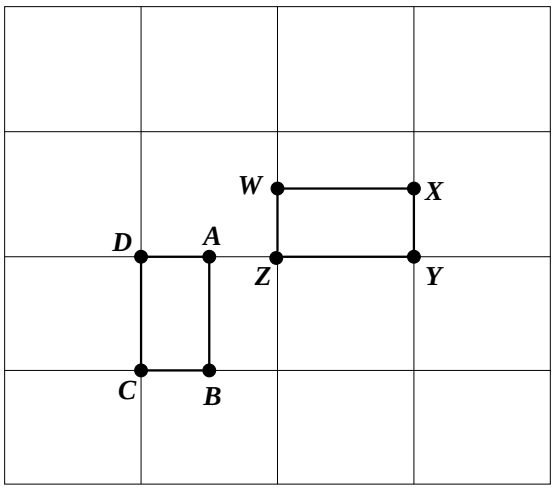

Figure 3: Example grid.

by $rect(D(i,j),l,w)$) is the union of the following four sets of points:

$$\{D(i,j-l), D(i,j-l+1), \cdots, D(i,j-1), D(i,j)\} \;=\; top(D(i,j),l,w)$$
$$\{D(i-w,j-l), D(i-w,j-l+1), \cdots, D(i-w,j-1), D(i-w,j)\} \;=\; bottom(D(i,j),l,w)$$
$$\{D(i-w,j), D(i-w+1,j), \cdots, D(i-1,j), D(i,j)\} \;=\; right(D(i,j),l,w)$$
$$\{D(i-w,j-l), D(i-w+1,j-l), \cdots, D(i-1,j-l), D(i,j-l)\} \;=\; left(D(i,j),l,w)$$

The above four sets of points correspond to the top, bottom, right, and left sides of the rectangle $rect(D(i,j),l,w)$. Therefore, we denote these set of points as $top(D(i,j),l,w)$, $bottom(D(i,j),l,w)$, $right(D(i,j),l,w)$, and $left(D(i,j),l,w)$, respectively. We have the following lemma:

Lemma 1 $D(i,j)$, can be expressed as a function of $bottom(D(i,j),l,w)$, $left(D(i,j),l,w)$, $\alpha[i-l+1\cdots i]$, and $\beta[j-w+1\cdots j]$.

Proof sketch: The proof is by simultaneous induction on l and w. For $l=1$ and $w=1$ the results follows using the following recursive relationship:

$$D(i,j) \quad = \quad \min[D(i-1,j)+1, D(i,j-1)+1, D(i-1,j-1)+t(i,j)]$$

The induction step is tedious but simple. The induction step can also be used to obtain the function referred to in the statement of the lemma. □

Protocol 3 is described in Figure 4 and proceeds in two steps. In the first step, we compute random shares of all values of the type $D(i,0)$ and $D(0,j)$ that correspond to the base case. In the second step, we compute all other values on the grid using the recurrence implicit in the proof of Lemma 1.

Consider the grid shown in Figure 3. First, the random shares of the values that correspond to left and bottom edge of the grid are computed. Now let us say we want to compute the random shares of the value corresponding to point A. Using lemma 1, the value corresponding to point A can be expressed as a function of values corresponding to sides CB and DC, $\alpha[CB]$, and $\beta[DC]$ (we are abusing the notation slightly by using CB to denote all indices that lie on the segment CB).

• *Compute the random shares for the initial values:*
Compute the random shares for the following values:

$$\begin{aligned} D(i,0) &= i\ ,\ 0 \le i \le n \\ D(0,j) &= j\ ,\ 0 \le j \le m \end{aligned}$$

• *Compute the random shares for values on the grid:*
We compute the random shares for all values on the grid in the row-major order. Consider a value $D(i,j)$ on the grid and the rectangle $rect(D(i,j,),l,w)$ with $l = i - k\lfloor\frac{i-1}{k}\rfloor$ and $w = i - k\lfloor\frac{i-1}{k}\rfloor$. The reader can check that all values in the grid $rect(D(i,j,),l,w)$ lie on the grid of granularity k. Let $C_{D(i,j)}$ be the circuit for computing $D(i,j)$ in terms of inputs $bottom(D(i,j),l,w)$, $left(D(i,j),l,w)$, $\alpha[i-l+1\cdots i]$, and $\beta[j-w+1\cdots j]$. Note that circuit $C_{D(i,j)}$ can be constructed by essentially mimicking the proof of lemma 1. Recall that we also have random shares for the values in the set $bottom(D(i,j),l,w)$ and $left(D(i,j),l,w)$. Now using the protocol for secure computation with shares we can compute the random shares for $D(i,j)$

Figure 4: Protocol 3.

4 Extensions

We describe how our protocol can be extended to yield a privacy-preserving version of the Smith-Waterman genome sequence algorithm [20]. We also describe how our protocol suggests a strategy for constructing privacy-preserving protocols for problems for which efficient dynamic-programming algorithms exist.

4.1 The Smith-Waterman Algorithm

We first describe the genome sequence comparison algorithm by Smith and Waterman [20]. We then discuss how our protocols for privately computing edit distances can be adapted for the Smith-Waterman algorithm. As before, let α and β be two strings over the alphabet Σ. The Smith-Waterman algorithm uses two function: a cost function c and a gap function g. The cost function $c : \Sigma \times \Sigma \to \Re$ associates a cost $c(u,v)$ with each pair (u,v). Typically, $c(u,v)$ has the following form:

$$c(u,v) = \begin{cases} a & \text{if } u = v \\ -b & \text{if } u \neq v \end{cases}$$

If a symbol is deleted or inserted, a special symbol $-$ is inserted. For example, if the fourth symbol is deleted from CTGTTA it is written as CTG$-$TA. A sequence of $-$ is called a *gap*. Gaps are scored using a *gap function* g, which typically has an *affine* form:

$$g(k) = x + y(k-1)$$

In the equation given above k is the size of the gap (number of consecutive $-$ in a sequence) and $x > 0$ and $y > 0$ are two constants.

Define $H(i,j)$ as the following equation:

$$\max\{0, \Delta(\alpha[x\cdots i], \beta[y\cdots j]) \text{ for } 1 \le x \le i \text{ and } 1 \le y \le j\}$$

Recall that $\alpha[x \cdots i]$ is represents the string $\alpha[x]\alpha[x+1] \cdots \alpha[i]$. The distance between strings $\alpha[x \cdots i]$ and $\beta[y \cdots j]$ according the cost function c and gap function g is denoted by $\Delta(\alpha[x \cdots i], \beta[y \cdots j])$. The *Smith-Waterman* distance between the two strings α and β (denoted by $\delta_{SW}(\alpha, \beta)$) is simply $H(n, m)$, where n and m are lengths of the two strings α and β. Values $H(i, 0)$ and $H(0, j)$ are defined to be zero for $0 \leq i \leq n$ and $0 \leq j \leq m$. For $1 \leq i \leq n$ and $1 \leq j \leq m$, $H(i, j)$ is defined using the following recursive equation:

$$\begin{aligned} H(i,j) &= \max\Big[0, \max_{1 \leq o \leq i}\{H(i-o,j) - g(o)\}, \\ &\qquad \max_{1 \leq l \leq j}\{H(i,j-l) - g(l)\}, H(i-1,j-1) + c(\alpha[i], \beta[j])\Big] \end{aligned}$$

Next we discuss how the three privacy-preserving protocols for computing the edit distance between two strings can be adapted for computing the Smith-Waterman distance. Protocol 1 can be easily adapted for computing the Smith-Waterman distance. As before we compute a circuit $C_{H(i,j)}$ for computing $H(i, j)$ using the recursive equation. Protocol 3 can also be easily adapted for computing the Smith-Waterman distance. The key observation is that if $H(i, j)$ lies on the grid then the values used in the recursive equation

$$\begin{array}{l} \{H(i-o,j) \mid 1 \leq o \leq i\} \\ \{H(i,j-l) \mid 1 \leq l \leq j\} \end{array}$$

also lie on the grid. Protocol 2 uses homomorphic encryption in conjunction with SWCS and thus requires some substantial changes As before, Alice and Bob will each maintain a $(n+1) \times (m+1)$ matrix H_A and H_B, respectively, with the following invariant:

$$H(i,j) = H_A(i,j) + H_B(i,j)$$

Let $x(i, j)$, $y(i, j)$ and $z(i, j)$ be the last three terms in the recursive equation for $H(i, j)$. It is pretty clear how to split the first term (which is 0) in the equation. The random shares for $x(i, j)$ and $y(i, j)$ can be computed using the SWCS protocol and a circuit for $\max$. Once the random shares for $x(i, j)$, $y(i, j)$, and $z(i, j)$ are computed, the random shares for $H(i, j)$ can be computed using the "maximum-of-four" circuit and the SWCS protocol. We will next focus on computing the random shares of $z(i, j)$. For this computation, we will rely on additive homomorphism of the encryption scheme. Alice encrypts $H_A(i-1, j-1)$ and sends $e_A = E(H_A(i-1, j-1))$ to Bob. Bob encrypts $e_B = E(H_B(i-1, j-1))$ and computes $f(e_A, e_B)$. Notice that because of the homomorphic property of the encryption we have the following:

$$\begin{aligned} f(e_A, e_B) &= E(H_A(i-1,j-1) + H_B(i-1,j-1)) \\ &= E(H(i-1,j-1)) \end{aligned}$$

Bob also generates an $(\log(n+m)+80)$-bit long random number r, and computes $E(-r)$. Since the strings are over an alphabet $\Sigma = \{1, \cdots, w\}$, Bob creates w messages $(m_1, \cdots, m_w)$, where m_k is the following message

$$E(H(i-1, j-1) + c(k, \beta(j)) - r) \ .$$

Bob's random share $z_B(i, j)$ for term $z(i, j)$ is r. The reader can check that all the messages m_i can be computed using the homomorphic property of the encryption scheme.

Alice and Bob execute a 1-out-of-w oblivious transfer OT_1^w protocol (see section 3.1) with Bob acting as the sender on inputs $(m_1, \cdots, m_w)$ and Alice acting as the chooser on input $\alpha(i)$. As a result of the OT_1^w protocol, Alice obtains the message $m_{\alpha(i)}$. Observe that Alice obtains $E(H(i-1, j-1)+c(\alpha(i), \beta(j))-r)$, and sets the plaintext as her share $z_A(i, j)$ of term $z(i, j)$. We leave optimization of this protocol as future work.

4.2 Dynamic Programming Algorithms

We claim that our algorithm for privacy-preserving protocol for computing edit distance between two strings suggests a strategy for designing privacy-preserving protocols for problems that have efficient dynamic-programming algorithms. Let $\mathcal{P}(x, y)$ be a problem with two inputs x and y, e.g., for the problem of computing edit-distance between two strings the inputs x and y are the strings. Typically, a dynamic-programming algorithm $\mathcal{A}_{\mathcal{P}}$ for problem $\mathcal{P}$ has the following components:

- A set S of sub-problems and a dependency relation $R \subseteq S \times S$ between the sub-problems. Intuitively, $(s, s') \in R$ means that the sub-problem s' depends on the sub-problem s. If there is a dependency between s and s', we write it as $s \rightarrow s'$. In the case of the problem of computing edit-distance between two strings α and β of length n and m, the set of sub-problems is $[0, \cdots, n] \times [0, \cdots, m]$. For all sub-problems (i, j) such that $i \neq 0$ and $j \neq 0$, we have the following dependencies: $(i-1, j) \rightarrow (i, j)$, $(i, j-1) \rightarrow (i, j)$, and $(i-1, j-1) \rightarrow (i, j)$. The *base sub-problems* are $s \in S$ such that they have no dependencies. For the edit-distance problem, the base sub-problems are:

  $$\{(i, 0) \mid 0 \leq i \leq n\}$$
  $$\{(0, j) \mid 0 \leq j \leq m\}$$

  We also assume that there is a unique root sub-problem $root \in S$ such that there does not exist a sub-problem that depends on $root$. For the edit-distance problem the unique root sub-problem is (n, m).

- Each sub-problem s is assigned a value $val(s)$. The goal is to compute $val(root)$. The function val from S to $\Re$ assigns values to sub-problems, such that it satisfies the following properties:
  - For all the base sub-problems $s \in S$, $val(s)$ is defined.
  - Let $s \in S$ be a non-base sub-problem. Define $pred(s)$ as all the predecessors of s, i.e. the set $pred(s)$ is defined as $\{s' \mid s' \rightarrow s\}$. Assume that $pred(s)$ is equal to $\{s_1, \cdots, s_k\}$. There is a recursive function f defining $val(s)$ in terms of $val(s_1), val(s_2), \cdots, val(s_k)$, $s(x)$, and $s(y)$, where $s(x)$ and $s(y)$ are parts of the input x and y that are relevant to the sub-problem s. In case of the edit-distance problem $val((i, j))$ is equal to $D(i, j)$. The value for the base and non-base sub problems for the edit-distance problems was defined in equations 1 and 3 in Section 2.

Consider a problem $\mathcal{P}(x, y)$ with two inputs x and y. Assume that problem $\mathcal{P}$ has a dynamic-programming algorithm $\mathcal{A}_{\mathcal{P}}$ with the space of sub-problems S. We describe of how we can design a privacy-preserving protocols for $\mathcal{P}(x, y)$, where Alice has input x and Bob has input y.

Protocol 1: Recall that $val : S \rightarrow \Re$ assigns a value to each sub-problem. Let s be a sub-problem and C_s be the circuit with inputs $s(x)$ and $s(y)$ that computes $val(s)$. The circuit C_s can be constructed using the recursive equation f for defining the value of non-base sub-problems and the circuits for sub-problems s' that are predecessors of s. Assume that we have constructed a circuit C_{root} for the root sub-problem. Using the circuit C_{root} and standard protocols we can privately compute the $val(root)$.

Protocol 2: In this protocol we randomly split $val(s)$ for all sub-problems. We denote the two shares of $val(s)$ by $val_A(s)$ and $val_B(s)$. Assume that we have randomly split $val(s)$ for all base sub-problems s. Consider a sub-problem s such that $pred(s) = \{s_1, \cdots, s_k\}$. Assume that we have computed random shares $val_A(s_i)$ and $val_B(s_i)$ for $val(s_i)$ (where $1 \leq i \leq k$). Recall that we have the following recursive equation describing $val(s)$:

$$val(s) = f(val(s_1), \cdots, val(s_k), s(x), s(y))$$

Since we have computed the random shares for $val(s_i)$ ($1 \leq i \leq k$), we can compute the random shares of $val(s)$. At the end of the protocol, $val_A(root) + val_B(root)$ gives the desired result.

Protocol 3: Recall that protocol 3 was a hybrid between protocol 1 and 2. However, protocol 3 heavily depends on the structure of the space S of sub-problems. For example, for the edit-distance problem, protocol 3 heavily used the matrix structure of space of sub-problems.

5 Security Proofs

Our protocols are secure in the so called *semi-honest* model of secure computation, *i.e.*, under the assumption that both participants faithfully follow the protocol specification. To achieve security in the *malicious* model, where participants may deviate arbitrarily from protocol specification, participants would need to commit to their respective inputs prior to protocol start and then prove in zero knowledge that they follow protocol specification. There exist generic techniques for "compiling" any protocol which is secure in the semi-honest model into one that is secure in the malicious model [8, section 7.4].

In our case, it is not clear whether security in the malicious model offers significant advantages over security in the semi-honest model. For example, there is no external validation of the parties' inputs. Even if the protocol forces the party to run the protocol on previously committed inputs (which requires zero-knowledge proofs and imposes a heavy overhead), this does not guarantee that the input was not maliciously chosen in the first place

Security of Protocols 1 and 3 follows directly from (i) security of subprotocols performed using standard methods for secure multi-party computation, and (ii) composition theorem for the semi-honest model [8, Theorem 7.3.3]. Proofs are standard and omitted for brevity.

Security of Protocol 2 is proved via a standard simulation proof in the semi-honest model. For each protocol participant, we demonstrate the existence of an efficient simulator algorithm which, with access to this participant's input and output, produces a simulation which is computationally indistinguishable from this participants's "view" of the protocol (informally, a "view" is a record of sent and received messages).

We give the proof for the unoptimized version of Protocol 2. The proof for the optimized version is essentially similar; the only substantial difference is that there are additional sub-simulators for simulating the parties' respective views of Yao's "garbled circuits" protocol for testing equality of two values.

Let $\mathsf{view}_A(\alpha)$ (respectively, $\mathsf{view}_B(\beta)$) be Alice's (respectively, Bob's) view of the protocol when executed on input string α (respectively, β). Each party's view consists of its respective input as well as all messages received by this party in the course of the protocol. The output of the protocol is the edit distance $\delta(\alpha, \beta)$. Because edit distance is a deterministic function of the parties' inputs, to prove security of the protocol it is sufficient to construct simulators S_A and S_B such that

$$\begin{aligned} \{S_A(\alpha, \delta(\alpha,\beta))\} &\stackrel{c}{\equiv} \{\mathsf{view}_A(\alpha)\} \\ \{S_B(\beta, \delta(\alpha,\beta))\} &\stackrel{c}{\equiv} \{\mathsf{view}_B(\beta)\} \end{aligned}$$

Here $\stackrel{c}{\equiv}$ stands for computational indistinguishability [7].

Our simulator will exploit semantic security of Alice's homomorphic public-key encryption scheme, defined by (G, E, D, M). As building blocks, it will also use the simulators for, respectively, the OT_1^w oblivious transfer and the secure protocol for computing the minimum of three values. Because the oblivious transfer protocol is assumed to be secure, there exist simulators $S_A^{\mathsf{ot}}, S_B^{\mathsf{ot}}$ such that:

$$\begin{aligned} \{S_A^{\mathsf{ot}}(i, x_i)\} &\stackrel{c}{\equiv} \{\mathsf{view}_A^{\mathsf{ot}}(i)\} \\ \{S_B^{\mathsf{ot}}(x_0, \ldots, x_w, \perp)\} &\stackrel{c}{\equiv} \{\mathsf{view}_B^{\mathsf{ot}}(x_0, \ldots, x_w)\} \end{aligned}$$

where $x_0, \ldots, x_w$ are Bob's inputs into the OT_1^w protocol, i is Alice's choice ($0 \leq i \leq w$), $\perp$ is "empty" output (it denotes that Bob does not receive any output from the protocol), and $\mathsf{view}_A^{\mathsf{ot}}$ and $\mathsf{view}_B^{\mathsf{ot}}$ are, respectively, Alice's and Bob's views of the OT_1^w protocol.

Similarly, security of the protocol for computing the minimum of three values implies that there exist simulators $S_A^{\mathsf{min}}, S_B^{\mathsf{min}}$ such that:

$$\begin{array}{rcl} \{S_A^{\mathsf{min}}(r_x^A, r_y^A, r_z^A, \min(x,y,z))\} & \stackrel{c}{\equiv} & \{\mathsf{view}_A^{\mathsf{min}}(r_x^A, r_y^A, r_z^A)\} \\ \{S_B^{\mathsf{min}}(r_x^B, r_y^B, r_z^B, \min(x,y,z))\} & \stackrel{c}{\equiv} & \{\mathsf{view}_B^{\mathsf{min}}(r_x^B, r_y^B, r_z^B)\} \end{array}$$

where r_x^A, r_y^A, r_z^A (respectively, r_x^B, r_y^B, r_z^B) are Alice's (respectively, Bob's) random shares of x, y, z.

Simulating Alice's view. Because Protocol 2 consists of $(n+1) \times (m+1)$ iterations, one for each value of the (i, j) pair, Alice's view_A of Protocol 2 is a composition of Alice's views of individual iterations $\mathsf{view}_A^{(i,j)}$. We give the simulator for $\mathsf{view}_A^{(i,j)}$ for all values of (i, j) such that $0 \leq i \leq n, 0 \leq j \leq m$. Recall that in addition to all of Alice's inputs, the simulator has access to the final result of the protocol, *i.e.*, the edit distance $\sigma(\alpha, \beta)$.

If $j = 0$ or $i = 0$, $\mathsf{view}_A^{(i,j)}$ consists simply of Alice's inputs $D_A(i, 0)$ and $D_A(0, j)$. Simulation is trivial.

For all (i, j) where $i \neq 0, j \neq 0$ and either $i \neq n$, or $j \neq m$, $\mathsf{view}_A^{(i,j)}$ consists of the following: (i) $D_A(i-1, j)$ (Alice's random share of $x(i, j)$), (ii) $D_A(i, j-1)$ (Alice's random share of $y(i, j)$), (iii) Alice's view of the oblivious transfer (OT) protocol, and (iv) Alice's view of the privacy-preserving $\min[x(i, j), y(i, j), z(i, j)]$ for computing the minimum of three numbers.

Because the OT protocol is assumed to be secure, there exists a simulator S_A^{ot} which, when executed on Alice's input $\alpha[i]$ (recall that $1 \leq \alpha[i] \leq w$) and message m produces a computationally indistinguishable simulation of Alice's view of the OT protocol whose output is message m. Our simulator runs S_A^{ot} as a sub-simulator, giving it as inputs $\alpha[i]$ and m', where m' is a "fake" ciphertext $E(r')$ created by the simulator. It encrypts, under Alice's public key, a random $(\log(n+m)+80)$-bit integer r', which is as long as r used by Bob in the real protocol. Observe that in a real protocol execution, Alice would have obtained either $E(D(i-1, j-1) + 1 - r)$, or $E(D(i-1, j-1) - r)$, where r is the random integer generated by Bob. Both in the real protocol and in the simulation, Alice is able to successfully decrypt the value obtained as a result of the OT protocol. Because adding a random r to any value x statistically hides x as long as r at least 80 bits longer than x, Alice cannot tell the difference between $D(i-1, j-1) + 1 - r$, $D(i-1, j-1) - r$ and r'. Therefore, the substitution performed by the simulator is undetectable.

To simulate Alice's view of the secure minimum protocol $\min[x(i, j), y(i, j), z(i, j)]$, the simulator invokes the sub-simulator S_A^{min} for this protocol. This sub-simulator requires the actual minimum value as one of its inputs. The simulator substitutes a random value r'' for the actual minimum. As in any protocol for securely evaluating function $f(x, y)$ where Alice and Bob hold random shares of x and y [8], Alice's share of the result is random and independent of $f(x, y)$. Therefore, the substitution is undetectable.

Finally, $\mathsf{view}_A^{(n,m)}$ contains an additional message m_B from Bob at the very end of the protocol, which in the real execution enables Alice to reconstruct the output of the entire protocol, *i.e.*, $\delta(\alpha, \beta)$. Because the simulator has access to $\delta(\alpha, \beta)$, he simulates m_B as $\delta(\alpha, \beta) - r_A$, where r_A is the output simulated for Alice by S_A^{min} in the last iteration of the secure minimum protocol $\min[x(n, m), y(n, m), z(n, m)]$. Observe that in both the simulation and the real execution, the sum of Alice's share of $\min[x(n, m), y(n, m), z(n, m)]$ and m_B is equal to $\delta(\alpha, \beta)$. This completes the simulation of Alice's view.

Simulating Bob's view. Simulating Bob's view $\mathsf{view}_B(i,j)$ of each iteration of the protocol is very similar to simulating Alice's view. The only difference is that the simulator for Bob's view must simulate the ciphertext $E(D_A(i-1,j-1))$ received by Bob prior to the oblivious transfer protocol. The simulator simulates this ciphertext by generating a random number r^*, encrypting it under Alice's public key, and sending the resulting ciphertext $E(r^*)$. This substitution is not detectable by Bob due to semantic security of the encryption scheme E.

To simulate Bob's view of the OT protocol, the simulator invokes S_B^{ot} as a sub-simulator. Note that Bob does not receive any output from the OT protocol, so the simulator simply runs S_B^{ot} on Bob's inputs. To simulate Bob's view of the secure minimum protocol, the simulator invokes S_B^{min} as a sub-simulator, again substituting a random value for the actual minimum. Note that Alice's and Bob's roles in the secure minimum protocol are symmetric, so the same argument applies for Bob.

Finally, in $\mathsf{view}_B^{(n,m)}$, the simulator substitutes Alice's message m_A with $\delta(\alpha,\beta) - r_B$, where r_B is the output simulated for Bob by S_B^{min} when simulating the final instance of the secure minimum protocol. Observe that in both the simulation and the real execution, the sum of Bob's share of $\min[x(n,m), y(n,m), z(n,m)]$ and m_A is equal to $\delta(\alpha,\beta)$. This completes the simulation of Bob's view.

6 Implementation and Experimental Results

In this section, we describe our implementation of the three protocols for privately computing edit distances between two strings and present experimental results for network bandwidth and execution times.

We implemented the two standard methods for secure circuit evaluation, i.e., the Yao's "garbled circuits" method and secure computation with shares. We used the oblivious transfer protocol presented by Noar and Pinkas [16]. For protocol 2, which uses homomorphic encryption, we use the Pallier encryption scheme [17]. Using these primitives, we implemented the three protocols. Circuits for various functions implicit in the protocols were obtained using the Fairplay compiler [15]. Fairplay converts a function represented in a simple language into Boolean circuits.

The experiments were executed on two 3-GHz Pentium 4 machines, with two gigabytes of memory, and connected by a local LAN. Using this setup, we obtained measurements (network bandwidth and execution times) for the three protocols on various problem sizes. The reason for performing the experiment on a local LAN is to provide a "best case" result for execution times in an environment where network bandwidth is not a bottleneck. Because the bandwidth numbers presented do not depend on the experimental setup, execution times for bandwidth-limited networks can be estimated from the numbers presented here.

The size of the problem instance is (n,m), where n and m are the sizes of the two strings. The main conclusions that can be drawn from our measurements are:

- *Protocol 1 is not suitable for large problems.* We observed that the Fairplay compiler exhausted the two gigabytes of available memory on our test machine while compiling a circuit for a problem instance of size $(26,26)$. However, protocol 1 is ideal for small strings because the entire computation is performed in one round.

- *Protocol 2 can execute for problem of any size.* However, since protocol 2 must perform a round of the SWCS protocol for each element, it is prohibitively slow. For example in our experimental setup, a problem of size $(25,25)$ takes over 9 minutes, compared to just 6 seconds for protocol 1.

- *Protocol 3 is most suitable for large problems.* Protocol 3 used the grid structure of the problem space, which makes it most suitable for scaling up to larger problems. For example, a problem instance of

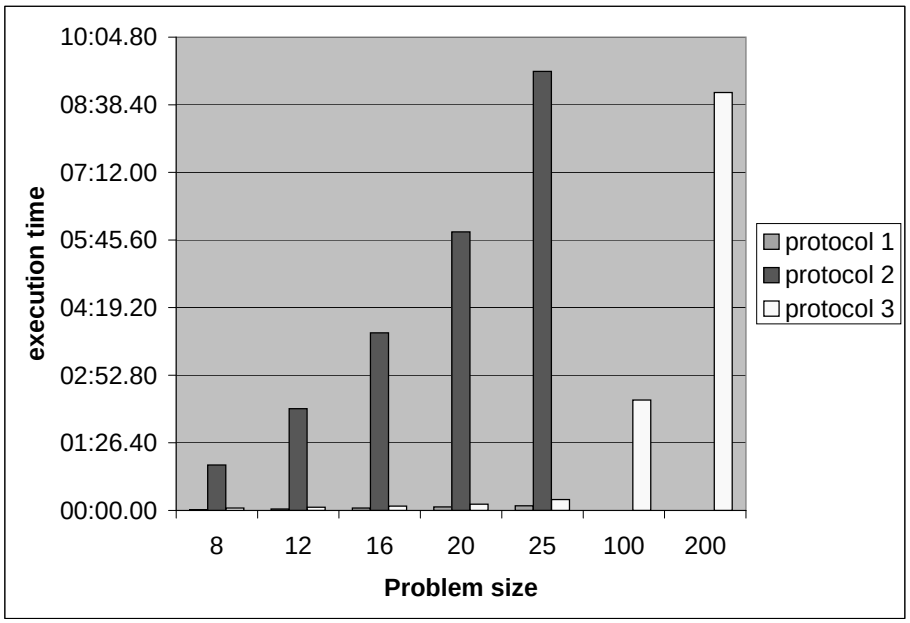

Figure 5: Timing measurements (in minutes and seconds) comparing protocols 1, 2, and 3. For problem sizes (100,100) and (200,200), protocol 1 could not compile the circuit, and protocol 2 was aborted after one hour.

size $(200, 200)$ takes under 10 minutes. Asymptotically, protocol 3 has the same performance as protocol 2, but since protocol 3 exploits the grid structure of the problem space it is hundreds of times faster.

Figure 5 shows the execution times for the three protocols. Clearly, protocol 3 scales the best as the problem size increases. Protocol 1 is suitable for small problems. Protocol 2 has a large execution time, but only takes limited bandwidth per round. Our experimental results confirm the protocol characteristics discussed in Section 3 (see Figure 2).

We present detailed results for protocols 1 and 2 in the appendix. We discuss results for protocol 3 in detail. Recall that in this protocol a grid is used (see the description in Section 3.4). Using protocol 3 we were able to solve problems instances of considerable size. For protocol 3 we present measurements for a problem instance of size (200,200). Table 1 shows the results using various grid sizes. The performance steadily improves upto a grid size of 20, but the performance begins to slightly decrease after that. However, continuing to increase the grid size slightly decreases the network bandwidth requirement, which would result in fewer round trips, so it would still be a consideration environments with limited network bandwidth. With a grid size of 20, protocol 3 completes the problem of size $(200, 200)$ in just under the same amount

Grid size	Bandwidth (Alice)	Bandwidth (Bob)	CPU (Alice)	CPU (Bob)	wall clock
25	362.2 M	2.1 M	518	84	658
20	368.5 M	2.6 M	385	90	534
10	397.4 M	5.4 M	476	123	655
8	412.0 M	5.8 M	520	145	729
4	485.3 M	14.4 M	784	234	1095
2	635.2 M	32.0 M	1296	408	1804
1	948.0 M	76.7 M	2480	780	4883

Table 1: Network bandwidth (in bytes) and timing measurements (in seconds) for protocol 3 with a problem of size $(200, 200)$. (M refers to Megabytes)

of time as protocol 2 takes for a problem of size $(25, 25)$. As with protocol 1, the circuits operate on 8-bit integers.

7 Conclusion

We presented three privacy-preserving protocols for computing edit-distance between two strings. The two key ideas in the protocols were as follows: randomly split the table of values maintained by the dynamic programming algorithm between the two parties and exploit the structure of the table. We also discussed how these key ideas can be applied to other dynamic-programming algorithms. Currently, we have only implemented the protocol for computing the edit distance. In the future, we plan to implement privacy-preserving versions of other dynamic-programming algorithms. There are several applications for the algorithm for computing the edit-distance. For example, hierarchical clustering algorithms use the edit-distance as a distance metric. We will investigate privacy in the context of these applications.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 2001.

[2] Lorrie Cranor, Marc Langheinrich, Massimo Marchiori, Martin Presler-Marshall, and Joseph Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, 16 April 2002.

[3] Lorrie Faith Cranor. Internet privacy. *Communications of the ACM*, 42(2):28–38, 1999.

[4] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean. Secure computation of surveys. In *2004 EU Workshop on Secure Multiparty Protocols (SMP)*, 2004.

[5] Michael Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In Christian Cachin and Jan Camenisch, editors, *Proceedings of Eurocrypt 2004*, volume 3027 of *LNCS*, pages 1–19. Springer-Verlag, May 2004.

[6] Ian Goldberg, David Wagner, and Eric Brewer. Privacy-enhancing technologies for the internet. In *Proc. of 42nd IEEE Spring COMPCON*. IEEE Computer Society Press, February 1997.

[7] O. Goldreich. *Foundations of Cryptography: Volume 1, Basic Tools*. Cambridge University Press, May 2001.

[8] O. Goldreich. *The Foundations of Cryptography — Volume 2*. Cambridge University Press, 2004.

[9] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game – a completeness theorem for protocols with honest majority. In *19th STOC*, pages 218–229, 1987.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and Systems Science*, 28:270–299, 1984.

[11] D. Gusfield. *Algorithms on Strings, Trees, and Sequences*. Cambridge University Press, 1997.

[12] Y. Lindell and B. Pinkas. Privacy preserving data mining. *Journal of Cryptology*, 15(3), 2002.

[13] Yehuda Lindell and Benny Pinkas. A proof of Yao's protocol for secure two-party computation. Cryptology ePrint Archive, Report 2004/175, 2004. `http://eprint.iacr.org/2004/175`.

[14] B. Pinkas M. Naor and R. Sumner. Privacy preserving auctions and mechanism design. In *Proceedings of the 1st ACM conf. on Electronic Commerce*, 1999.

[15] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay — a secure two-party computation system. In *Proceedings of the 13th Usenix Security Symposium*, San Diego, CA, USA, August 2004.

[16] Moni Naor and Benny Pinkas. Efficient oblivious transfer protocols. In *Proceedings of the Twelfth Annual Symposium on Discrete Algorithms (SODA)*, pages 448–457, 2001.

[17] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Proceedings of Advances in Cryptology (EUROCRYPT'99)*, 1999.

[18] M. Rabin. How to exchange secrets by oblivious transfer. Technical Report TR-81, Aiken Computation Laboratory, Harvard University, 1981.

[19] D. M. Rind, I. S. Kohane, P. Szolovits, C. Safran, H. C. Chueh, and G. O. Barnett. Maintaining the confidentiality of medical records shared over the internet and the world wide web. *Annals of Internal Medicine*, 127(2), July 1997.

[20] T. Smith and M. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147, 1981.

[21] E. Szajda, M. Pohl, J. Owen, and B. Lawson. Toward a practical data privacy scheme for a distributed implementation of the smith-waterman genome sequence comparison algrotihm. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2006.

[22] Joseph Turow. Americans and online privacy: The system is broken. Technical report, Annenberg Public Policy Center, June 2003.

[23] A.C. Yao. How to generate and exchange secrets. In *Proceedings of the 27th IEEE Symposium on Foundations of Computer Science*, 1986.

Problem size	Bandwidth (Alice)	Bandwidth (Bob)	CPU (Alice)	CPU (Bob)	wall clock
(25,25)	4.38M	10472	4.26	1.17	5.94
(20,20)	2.97 M	8764	3.10	0.88	4.46
(16,16)	1.83 M	7057	2.12	0.68	3.02
(12,12)	0.96 M	5348	1.30	0.54	1.92
(8,8)	0.37 M	3633	0.74	0.39	1.12

Table 2: Network bandwidth (in bytes) and timing measurements (in seconds) for protocol 1. (M refers to Megabytes)

Problem size	Bandwidth (Alice)	Bandwidth (Bob)	CPU (Alice)	CPU (Bob)	wall clock
(32,32)	216.6 M	48.1 M	515	173	915
(25,25)	132.2 M	29.3 M	312	108	561
(20,20)	84.6 M	18.8 M	199	69	356
(16,16)	54.2 M	12.0 M	127	44	227
(12,12)	30.5 M	6.8 M	72	25	130
(8,8)	13.5 M	3.0 M	32	11	58

Table 3: Network bandwidth (in bytes) and timing measurements (in seconds) for protocol 2. (M refers to Megabytes)

A Detailed Results for Protocols 1 and 2

Protocol 1

As a preparation step, a specific circuit must be constructed for each problem instance. We implemented a program which takes as its input a description of the problem instance and outputs a description in the Fairplay input language SHDL. Fairplay is able to take the SHDL description and output a circuit. Table 2 shows the network bandwidth (in bytes) and execution times (in seconds) for various problem instances. For problems of size greater than (25,25), the Fairplay compiler exhausted available memory and could not create the circuit. In this experiment, the circuits operate using 8-bit integers bits, which allows for a maximum edit distance of 255.

Protocol 2

As for protocol 1, we present network bandwidth and execution times for several problem instances. Recall that protocol 2 uses a round of SWCS for computing the shares for each value in the matrix D. For each matrix value, protocol 2 requires approximately 250000 bytes and 0.9 seconds in our test setup. The circuits used for protocol 2 use 80-bit integers.[4] Table 3 shows the bandwidth and time required for various problem sizes.

[4]This large word size is necessary to allow for sufficient statistical hiding of the split matrix elements, because the arithmetic using the homomorphic encryption properties cannot perform modular arithmetic, which is necessary for perfect information hiding.