Towards Practical Privacy for Genomic Computation

Somesh Jha* Louis Kruger* Vitaly Shmatikov†

Abstract

Many basic tasks in computational biology involve operations on individual DNA and protein sequences. These sequences, even when anonymized, are vulnerable to re-identification attacks and may reveal highly sensitive information about individuals.

We present a relatively efficient, privacy-preserving implementation of fundamental genomic computations such as calculating the edit distance and Smith-Waterman similarity scores between two sequences. Our techniques are cryptographically secure and significantly more practical than previous solutions. We evaluate our prototype implementation on sequences from the Pfam database of protein families, and demonstrate that its performance is adequate for solving real-world sequence-alignment and related problems in a privacy-preserving manner.

Furthermore, our techniques have applications beyond computational biology. They can be used to obtain efficient, privacy-preserving implementations for many dynamic programming algorithms over distributed datasets.

1 Introduction

Genomic data such as DNA and protein sequences are increasingly collected by government agencies for law enforcement and medical purposes, disseminated via public repositories for research and medical studies, and even stored in private databases of commercial enterprises. For example, deCODE Genetics aims to collect the complete genome sequences of the entire population of Iceland [8], while the non-profit HapMap Project is developing a public repository of representative genome sequences in order to help researchers to discover genes associated with specific diseases [16].

The underlying genome records are typically collected from specific individuals, and thus contain a lot of sensitive personal information, including genetic markers for diseases, information that can be be used to establish paternity and maternity, and so on. Therefore, genomic records are usually stored in an anonymized form, that is, without explicit references to the identities of people from whom they were collected.

Even if genome sequences are anonymized, *re-identification* is a major threat. In many cases, a malicious user can easily de-anonymize the sequence and link it to its human contributor simply by recognizing the presence of certain markers [11]. Furthermore, many genetic markers are expressible in the person's phenotype, which includes externally observable features [25]. In general, protecting privacy of individual DNA when the corresponding genome sequence is available to potential attackers does not appear realistic. Developing practical tools which can support collaborative analysis of genomic data without requiring the participants to release the underlying DNA and protein sequences is perhaps the most important privacy challenge in computational biology today.

In this paper, we design and implement cryptographically secure protocols for collaborative two-party computation on genomic data which are significantly more efficient than previously proposed solutions. Our main focus is on the dynamic programming algorithms such as the edit distance and the Smith-Waterman algorithm for sequence alignment, which are among the fundamental building blocks of computational biology [15, Chapter 11].

This paper makes the following contributions:

- We design and implement several privacy-preserving protocols for computing the *edit distance* between two strings α and β, *i.e.*, the minimum number of delete, insert, and replace operations needed to convert α into β.

[*] University of Wisconsin - Madison, WI 53706. Emails: {jha,kruger}@cs.wisc.edu

[†] The University of Texas at Austin, TX 78712. Email: shmat@cs.utexas.edu . Supported in part by the NSF grant IIS-0534198.

- We construct a protocol for computing the Smith-Waterman similarity score between two sequences [28] which is significantly more tractable than generic solutions. Smith-Waterman scores are used for sequence alignment (*e.g.*, pairwise Smith-Waterman distances are directly used in phylogenetic tree reconstruction) and also as a distance metric in clustering algorithms.
- We demonstrate that, in addition to privacy-preserving computation on genomic data, our techniques generalize to a wide variety of dynamic programming problems [4, Chapter 15].
- We evaluate our implementation on realistic case studies, including protein sequences from the Pfam database [2]. Our experimental results demonstrate that our methods are tractable on sequences of up to several hundred symbols in length.

  Note: Our implementation is available for download from the website `http://pages.cs.wisc.edu/~lpkruger/sfe/`.

Even though theoretical constructions for various secure multi-party computation (SMC) tasks have received much attention (see related work below), actual implementations and performance measurements are exceptionally rare. Asymptotic analysis can provide a rough intuition, but in the absence of concrete implementations and experimental evaluations it is hard to tell whether these theoretical designs are feasible for practical problems.

The protocols presented in this paper are accompanied by publicly available implementations. They have been evaluated on real protein sequences and analysis workloads, demonstrating that they can be applied in practice to problem instances of realistic size, while achieving the same level of cryptographic security as theoretical constructions.

Related work: Public availability of personal information due to the Internet has brought privacy concerns to the forefront [6, 30]. Therefore, there has been considerable interest in developing privacy protection technologies [5, 12, 27].

One of the fundamental cryptographic primitives for designing privacy-preserving protocols is *secure function evaluation (SFE)*. Generic protocols for SFE [14, 31] enable two parties A and B with respective inputs x and y to jointly compute any efficiently computable (*i.e.*, probabilistic polynomial-time) function $f(x, y)$ while preserving the privacy of their respective inputs: 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 symmetric condition holds for B. Our constructions employ Yao's "garbled circuits" method [21, 31] as a building block for several sub-protocols, including privacy-preserving equality testing. Some of our protocols use the garbled circuits construction in a non-black-box way, exploiting the specifics of circuit encoding.

Special-purpose privacy-preserving protocols have been developed for tasks such as auctions, surveys, remote diagnostics, and so on [23, 20, 9, 10, 3], but privacy-preserving genomic computation has received little attention. We are aware of only two papers devoted to this or similar problems: Atallah *et al.* [1] and Szaida *et al.* [29]. Neither paper provides a proof of security. The edit distance protocol of [1] is impractical even for very small problem instances due to its immense computational cost (see Section 6 and appendix C). The distributed Smith-Waterman algorithm of [29] involves decomposing the problem instance into sub-problems, which are passed out to several participants. It is presumed that because each participant sees only his sub-problem, he cannot infer the inputs for the original problem (this does not appear to imply standard cryptographic security). It is unclear how the protocol of [29] may be used in the two-party case, or whether it can be generalized to other dynamic programming algorithms. By contrast, our techniques are provably secure and substantially more scalable, as demonstrated by our evaluation on realistic instances of genomic analysis problems.

2 Cryptographic Toolkit

Oblivious transfer. *Oblivious transfer* was originally proposed by Rabin [26]. Informally, a 1-out-of-n oblivious transfer (denoted 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 construction 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 the Naor-Pinkas construction [22].

Oblivious circuit evaluation. We 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.

Yao's "garbled circuits" method was originally proposed in [31] (a complete description and security proofs can be found in [21]). Informally, Alice securely transforms the circuit so that Bob can evaluate it without learning her inputs or the values on any internal circuit wire.

Alice does this by generating two random keys for each circuit wire, one representing 0 on that wire, the other representing 1. The keys representing Alice's own inputs into the circuit she simply sends to Bob. The keys representing Bob's inputs are transferred to Bob via the OT_1^2 protocol. For each of Bob's input wires, Bob acts as the chooser using his input bit on that wire 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. If Bob has a q-bit input into the circuit, then q instances of OT_1^2 are needed to transfer the wire keys representing his input, since each input bit is represented by a separate key.

Alice produces the "garbled" truth table for each circuit gate in such a way that Bob, if he knows the wire keys representing the values on the gate input wires, can decrypt exactly one row of the garbled truth table and obtain the key representing the value of the output wire. For example, consider an AND gate whose input wires are a and b, and whose output wire is c. Let $k_a^0, k_a^1, k_b^0, k_b^1, k_c^0, k_c^1$ be the random wire keys representing the bit values on these wires. The garbled truth table for the gate is a random permutation of the following four ciphertexts: $E_{k_a^1}(E_{k_b^0}(k_c^0))$, $E_{k_a^1}(E_{k_b^1}(k_c^1))$. $E_{k_a^0}(E_{k_b^1}(k_c^0))$, $E_{k_a^0}(E_{k_b^0}(k_c^0))$. 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 the actual wire values. The circuit can thus be evaluated "obliviously." For example, given the above table and the input wire keys k_a^0 and k_b^1 representing, respectively, 0 on input wire a and 1 on input wire b, Bob can decrypt exactly one row of the table, and learn the random key k_c^0 representing 0 (*i.e.*, the correct result of evaluating the gate) on the output wire c.

Observe that until Alice reveals the mapping, Bob does *not* know which bits are represented by the wire keys he holds. For the standard garbled circuit evaluation, Alice reveals the mapping only for the wires that represent the output of the entire circuit, but not for the internal wires.

Several of our protocols rely on the representation of bit values on circuit wires by random keys. These protocols use Yao's construction not as a "black box" implementation of secure circuit evaluation, but exploit its internal structure in a fundamental way.

The second standard method is *secure computation with shares* (SCWS) [13, Chapter 7]. This protocol maintains the invariant that, for every circuit wire w, Alice learns a random value s_A and Bob learns s_B, where $s_A \oplus s_B = b_w$, the actual bit value of the wire. In our protocols, we use exclusive-or, but they can work with any secret sharing scheme. Because the shares are random, neither party knows the actual wire value. For each output wire of the circuit, Alice and Bob can combine their shares to reconstruct the output bit.

3 Privacy-Preserving Edit Distance

3.1 Edit distance: definition

Let α and β be two strings over an alphabet Σ. Let the lengths of 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. The following dynamic programming algorithm computes $\delta(\alpha, \beta)$ in time $O(nm)$ [15].

Given a string α, let $\alpha[1 \cdots i]$ denote the first i characters of α, and $\alpha[i]$ denote the i-th character 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:

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

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

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:

$$\begin{aligned} D(i,j) \quad = \quad & \min[D(i-1,j)+1, D(i,j-1)+1, \\ & D(i-1,j-1)+t(i,j)] \qquad (3) \end{aligned}$$

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 entire algorithm for computing edit distance is shown in Figure 1.

- Compute $D(i, 0)$ and $D(0, j)$ for $1 \leq i \leq n$ and $1 \leq j \leq m$ using equations 1 and 2.
- 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.

3.2 Preserving privacy in edit distance computation

In the rest of this section, we will consider Alice (A) and Bob (B), who want to use the dynamic programming algorithm of Section 3.1 to compute the edit distance $\delta(\alpha, \beta)$ between their respective strings α (of size n) and β (of size m), but do not want to reveal the strings themselves. For example, "Alice" and "Bob" could be medical institutions participating in an NIH-sponsored collaborative study, while the strings in questions could be genome sequences with significant intellectual-property value.

We present three protocols. Protocol 1 is a straightforward application of Yao's method. For large problem instances, it requires generation of very large circuit representations. Protocol 2 splits the circuit for computing the edit distance into smaller component circuits and, furthermore, shares all intermediate values between protocol participants. While the sharing of intermediate values is not essential for the basic edit distance protocol, it is important for our efficient implementation of the Smith-Waterman protocol (see Section 4). Protocol 3 is a hybrid of Protocols 1 and 2. In order to keep circuit size manageable, it exploits the fundamental structure of the dynamic programming problem by dividing the matrix D into a grid and splitting each problem instance into sub-problems of size (k, k), where k divides both n and m. The differences between the protocols are summarized in Figure 2.

3.3 Protocol 1 (generic SMC)

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 $\beta[1 \cdots j]$. Strings 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 equations for computing $D(i, j)$ were given in equations 1, 2, and 3. 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 $C_{D(i-1,j-1)}$. The recursive computation given by equation 3 can be represented as a circuit $C_{D(i,j)}$, which 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 2).

3.4 Protocol 2

Protocol 1 represents the entire problem instance as a single circuit. The resulting representation, however, is impractically large for problems of realistic size (see Section 6). Protocol 2 splits the circuit corresponding to the problem instance into smaller sub-circuits and shares the result of evaluating each sub-circuit between the participants. Protocol 2 exploits the specific circuit representation in Yao's "garbled circuits" method instead of using it simply as an ideal functionality for secure circuit evaluation.

Let $w = \mid \Sigma \mid$ be the size of the alphabet from which the two strings are drawn, and recall that $q = \lceil \log_2(w) \rceil$. Let r be the length of wire keys in Yao's "garbled circuits" construction (see Section 2); r can be viewed as the security parameter for Yao's protocol.

3.4.1 Component circuits

Protocol 2 involves evaluation of multiple instances of the following two circuits.

Equality testing circuit. Circuit C_{eq} is the standard logic circuit for testing equality of two values. Its inputs are two q-bit values, x from Alice and y from Bob.

Number of iterations	Round complexity	Optimized round complexity	Circuits used
Protocol 1 (generic)	1	1	Circuit for (n, m) instance
Protocol 2	$O(nm)$	$O(m+n)$	Circuits for equality testing and "minimum-of-three"
Protocol 3	$O(\frac{nm}{k^2})$	$O(\frac{m+n}{k})$	Circuit for (k, k) instance

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

The output for Alice is empty, and the output for Bob is supposed to be the outcome of the comparison, *i.e.*, 0 if $x = y$, and 1 if $x \neq y$. C_{eq} consists of $2q - 1$ gates.

Recall that in Yao's construction, the circuit creator generates two random r-bit "wire keys" for each circuit wire, including the output wire. Let k^0_{eq} (respectively, k^1_{eq}) be the wire key representing 0 (respectively, 1) on the output wire of circuit C_{eq}. In our protocol, we will assume that the output of C_{eq} is not the bit σ, which is the result of the comparison, but instead the r-bit random value k^σ_{eq}, which *represents* σ. Observe that this is *not* a black-box use of the "ideal two-party computation functionality," because it critically depends on the internal representation of circuit outputs by random wire keys. (In other words, an alternative implementation of the same functionality would not be sufficient for our purposes.)

Bob, acting as the circuit evaluator in Yao's protocol, learns k^σ_{eq}. He does not learn whether this value represents 0 or 1, since he does not know the mapping from random wire keys to the bit values they represent. This property, too, is essential in our construction.

Minimum-of-three circuit. Circuit C_{min3} computes the minimum of three values, each of which is randomly shared between Alice and Bob, and splits the result into random shares, too. Alice's inputs are four $\log(n + m)$-bit values x_1, x_2, x_3, and r. Bob's inputs are three $\log(n + m)$-bit values y_1, y_2, y_3, as well as $t \in \{0, 1\}$. The circuit's output for Bob is $z = \min(x_1 \oplus y_1 + 1, x_2 \oplus y_2 + 1, x_3 \oplus y_3 + t) \oplus r$, where $\oplus$ is bitwise exclusive-OR, while $+$ is addition modulo $(n + m)$. The output for Alice is empty.

Observe that the C_{min3} circuit takes a 1-bit value t as Bob's input. In Yao's construction, t is represented as a random r-bit wire key k^0_t or k^1_t. As mentioned above, we rely on this representation, and assume that Bob already has (from a previous evaluation of C_{eq}) some key k^σ_t representing the value of t. Bob holds this value *obliviously*. He knows that it is a valid wire key, *i.e.*, that it represents either 0, or 1 on the input wire of C_{min3} corresponding to t, but he does not know the value of $\sigma = t$ since he does not know the mapping from wire keys to the bit values they represent.

3.4.2 Computing edit distance

Alice and Bob each maintains an $(n + 1) \times (m + 1)$ matrix D_A and D_B, respectively. Each element of both matrices is a $\log(n + m)$-bit integer. All arithmetic is modulo $n+m$. The protocol maintains the invariant that every value in the edit distance 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 $D(i, j) = D_A(i, j) \oplus D_B(i, j)$

Additionally, Bob maintains an $n \times m$ matrix T, each element of which is an r-bit value.

Phase 0. Alice fills in $D_A(i, 0)$ and $D_A(0, j)$ with random $\log(n + m)$-bit values and sends them to Bob. Bob fills $D_B(i, 0)$ with $i \oplus D_A(i, 0)$ and $D_B(0, j)$ with $j \oplus D_A(0, j)$.

Phase 1. Alice and Bob perform $n \times m$ instances of Yao's secure circuit evaluation protocol on circuit C_{eq}. The inputs for the (i, j)-th instance are $\alpha[i]$ and $\beta[j]$, respectively. The output for Bob is a random r-bit value $k^\sigma_{eq}(i, j)$, where $\sigma = 0$ if $\alpha[i] = \beta[j]$, 1 otherwise. Bob sets $T(i, j) = k^\sigma_{eq}(i, j)$.

Observe that neither Alice, nor Bob learns the value of σ, *i.e.*, whether $\alpha[i]$ is equal to $\beta[j]$ or not. Bob obtains and stores a random key representing σ, but since he does not know the mapping from random keys to the bit values they represent, he cannot interpret this key. Alice knows the mappings because she created them herself when producing a garbled version of the C_{eq} circuit for each instance of the protocol, but she does not know which of the two output-wire keys Bob has obtained and thus does not learn the result of the equality test.

All $n \times m$ instances of C_{eq} can be evaluated in parallel. Each instance requires q 1-out-of-2 oblivious transfers in order to transfer the wire keys representing Bob's q-bit input into C_{eq} from Alice to Bob (see Section 2). These oblivious transfers can be parallelized. The total number of communication rounds is equal to those of a single oblivious transfer, *e.g.*, 2 in the case of Naor-Pinkas protocol [22]. Evaluation of all $n \times m$ garbled circuits is performed by Bob, without any interaction with Alice.

Phase 2. Recall that the recursive equation for computing $D(i,j)$ is

$$\begin{aligned} D(i,j) \quad &= \quad \min[D(i-1,j)+1, D(i,j-1)+1, \\ & \qquad D(i-1,j-1)+t(i,j)] \end{aligned}$$

where $t(i,j)$ is defined to have value 1 if $\alpha[i] \neq \beta[j]$, and 0 otherwise.

Phase 2 requires $n \times m$ iterations. Let (i,j) be the indices of the iterations. In the (i,j)-th iteration, Alice and Bob perform an instance of Yao's secure circuit evaluation protocol on circuit C_{min3}. Alice creates a garbled instance of C_{min3} in the usual way (see Section 2), generating two fresh random wire keys for each circuit wire *except* Bob's input wire corresponding to value t.

Instead of generating new wire keys for this wire, Alice re-uses the same wire keys $k^0_{eq}(i,j)$ and $k^1_{eq}(i,j)$ that she used when creating a garbled equality-testing circuit C_{eq} in the (i,j)-th instance of Phase 1. This re-use of random wire keys is an important technical device which exploits the internal circuit representation in Yao's protocol. It allows us to "connect up" the evaluations of circuits $C_{eq}(i,j)$ and $C_{min3}(i,j)$, even though these circuits are evaluated at different points in the protocol.

Bob obliviously stores the key $k^{\sigma(i,j)}_{eq}(i,j) = T(i,j)$. This key is the result of evaluating $C_{eq}(i,j)$ and represents $\sigma(i,j)$, which is equal to 0 if $\alpha[i] = \beta[j]$, and 1 otherwise. Observe that $\sigma(i,j) = t(i,j)$. Effectively, Bob stores the representation of $t(i,j)$, without knowing what he is storing, until this representation is used as an input into $C_{min3}(i,j)$.

Alice and Bob execute standard Yao's protocol to evaluate the (i,j)-th instance of C_{min3}. Alice's inputs are three $\log(n+m)$-bit values $D_A(i-1,j)$, $D_A(i,j-1)$, and $D_A(i-1,j-1)$. Alice's fourth input is a fresh random $\log(n+m)$-bit value r. Bob's first three inputs are $\log(n+m)$-bit values $D_B(i-1,j)$, $D_B(i,j-1)$, and $D_B(i-1,j-1)$, and his fourth input is $T(i,j)$, *i.e.*, the result of evaluating the equality-testing circuit $C_{eq}(i,j)$ on $\alpha[i]$ and $\beta[j]$.

Alice sets $D_A(i,j) = r$. Bob obtains output z from the protocol, and sets $D_B(i,j) = z$. Observe that $D_A(i,j) \oplus D_B(i,j)$ is equal to

$$\begin{aligned} &\min(D_A(i-1,j) \oplus D_B(i-1,j)+1, \\ &\qquad D_A(i,j-1) \oplus D_B(i,j-1)+1, \\ &\qquad D_A(i-1,j-1) \oplus D_B(i-1,j-1)+t(i,j)) \\ &\qquad \oplus r \oplus r \\ &= \min(D(i-1,j)+1, D(i,j-1)+1, \\ &\qquad\quad D(i-1,j-1)+t(i,j)) \\ &= D(i,j) \end{aligned}$$

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 $D_A(n,m) \oplus D_B(n,m)$.

Each iteration of Phase 2 requires $3\log(n+m)$ instances of OT^2_1 in order to transfer the wire keys representing Bob's inputs into $C_{min3}(i,j)$ from Alice to Bob (see Section 2). These oblivious transfers can be parallelized. The total number of iterations is equal to $2nm$, assuming a 2-round oblivious transfer protocol.

Pre-computation and online complexity. All garbled circuits for both phases of Protocol 2 can be pre-computed by Alice since circuit representation in Yao's protocol is independent of the actual inputs. The only online cost is that of qnm 1-out-of-2 oblivious transfers in Phase 1 (a total of 2 iterations), and $3nm\log(n+m)$ oblivious transfers in Phase 2 (a total of $2nm$ iterations).

Optimization. The matrix D has $n+m-1$ diagonals, where the k-th diagonal ($0 \leq k \leq m+n$) $Diag_k$ of the matrix D is the set of elements $\{D(i,j) \mid i+j = k\}$. Since there is no dependency between the elements of the diagonal $Diag_k$, they can computed in parallel. Hence the round complexity of Protocol 2 can be brought down to $O(m+n)$ by evaluating all elements of $Diag_k$ in parallel.

3.5 Protocol 3

Protocol 1 is fairly efficient, requiring only nq executions of OT^2_1, where $q = \lceil \log_2(\mid \Sigma \mid) \rceil$, but it has to compute a large circuit $C_{D(n,m)}$. As we show in Section 6, for large problem instances the circuit representation requires several Gigabytes of memory.

Protocol 2 splits this circuit into very small component circuits for equality testing and computing the minimum of three values. Each of the small circuits is evaluated separately. Furthermore, Protocol 2 shares each intermediate value (the result of evaluating a component circuit) between the two participants. This sharing—which will be essential in the Smith-Waterman protocol of Section 4—comes at a significant cost, because Protocol 2 requires $qnm + 3nm\log(n+m)$ executions of OT_1^2.

In this section, we present Protocol 3, which exploits the geometric structure of the dynamic programming problem to split the single circuit of Protocol 1 into smaller sub-circuits. Recall that the edit distance algorithm 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 $k \mid m$.[1] The union of the following sets constitutes 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 by $rect(D(i,j), l, w)$) is the union of the following four sets of points :

$$\begin{aligned}&\{D(i,j-l), D(i,j-l+1), \cdots,\\ &\quad D(i,j-1), D(i,j)\}\\ &\{D(i-w,j-l), D(i-w,j-l+1), \cdots,\\ &\quad D(i-w,j-1), D(i-w,j)\}\\ &\{D(i-w,j), D(i-w+1,j), \cdots,\\ &\quad D(i-1,j), D(i,j)\}\\ &\{D(i-w,j-l), D(i-w+1,j-l), \cdots,\\ &\quad D(i-1,j-l), D(i,j-l)\}\end{aligned}$$

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 $bottom(D(i,j), l, w)$, $left(D(i,j), l, w)$, $\alpha[i-w+1 \cdots i]$, and $\beta[j-l+1 \cdots j]$.

The proof for this lemma is straightforward but tedious, and is provided in appendix A.

Protocol 3 is described in Figure 4 and proceeds in three phases similar to Protocol 2. Phase 0 and 1 of Protocol 3 are exactly the same as Protocol 2. In Phase 2, 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 assume that 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 the 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). A straightforward implementation of Protocol 3 takes $O(\frac{mn}{k^2})$ iterations. However, using the optimization discussed in the previous subsection the number of rounds can be brought down to $O(\frac{m+n}{k})$.

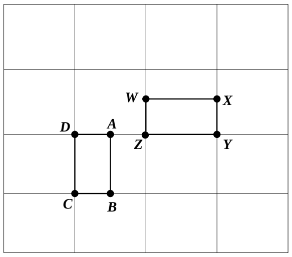

Figure 3. Example grid.

4 Privacy-Preserving Smith-Waterman

We now give a privacy-preserving version of the Smith-Waterman algorithm for comparing genome sequences [28]. This algorithm is more sophisticated than the edit distance algorithm, because the cost of delete, insert, and replace operations may no longer be equal to 1, but determined by special functions.

As before, let α and β be two strings over the alphabet Σ. The Smith-Waterman algorithm uses a cost function c and a gap function g. The cost function $c : \Sigma \times \Sigma \rightarrow \Re$ 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 above equation k is the size of the gap (number of consecutive "$-$" in a sequence), while $x > 0$ and $y > 0$ are constants.

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

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

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

- Phases 0 and 1 are the same as in Protocol 2.
- Phase 2: *Compute 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 $w = j - k\lfloor\frac{j-1}{k}\rfloor$. 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 $D(i,j)$ from 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 mimicking the proof of Lemma 1. Recall that we also have random shares for the values in the sets $bottom(D(i,j),l,w)$ and $left(D(i,j),l,w)$. We can now compute the random shares for $D(i,j)$. Essentially this is similar to Phase 2 of Protocol 2, but using $C_{D(i,j)}$ instead of C_{min3}.

Figure 4. Protocol 3.

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

$$H(i,j) = \max\,[\;\begin{array}{l} 0 \\ \max_{1\le o\le i}\{H(i-o,j)-g(o)\}, \\ \max_{1\le l\le j}\{H(i,j-l)-g(l)\}, \\ H(i-1,j-1)+c(\alpha[i],\beta[j])\;\;] \end{array}$$

We now adapt the privacy-preserving protocols for computing the edit distance to compute the Smith-Waterman distance. Note that the size of each element of the matrix is no longer $\log(n+m)$ bits, but depends on the specific cost function used.

As before, Protocol 1 requires a single circuit $C_{H(i,j)}$ for computing $H(i,j)$ using the recursive equation. This circuit, however, is significantly more complex. Unlike in the edit distance protocol, where $\sigma = 0$ if $\alpha[i] = \beta[j]$ and 1 otherwise, σ now is an arbitrary cost function $c(\alpha[i],\beta[j])$. Therefore, the circuit has to perform a table lookup on $c(\alpha[i],\beta[j])$ to determine the lowest cost alignment. Likewise, the gap function, which is a constant 1 for edit distance, is replaced by the gap value of the scoring function for Smith-Waterman. By convention, lower numbers represent higher costs (higher numbers represent a similarity score), so a maximum-of-three circuit is used instead of minimum-of-three.

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

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

also lie on the grid.

Protocol 2 can be adapted to compute the Smith-Waterman distance with significant *space savings* vs. Protocols 1 and 3. Unlike the edit distance protocol, where the cumulative size of circuits used by Protocol 2 is the same as the size of the single circuit used by Protocol 1, Protocol 2 for Smith-Waterman can avoid "embedding" the values of the cost and gap functions in the circuit.

As in the case of edit distance, Alice and Bob must maintain a $(n+1)\times(m+1)$ matrix H_A and H_B, respectively, with the following invariant:

$$H(i,j) \;=\; H_A(i,j)\oplus H_B(i,j)$$

In Phase 0, Alice fills in $H_A(i,0)$ and $H_A(0,j)$ with random values and sends them to Bob. Bob fills $H_B(i,0)$ with $H_A(i,0)$ and $H_B(0,j)$ with $H_A(0,j)$.

Recall that during Phase 1 of Protocol 2 for computing the edit distance (see Section 3.4), a circuit to test equality of Alice's and Bob's respective characters is evaluated. In Protocol 2 for Smith-Waterman, $OT^1_{|\Sigma|}$ is performed instead. Alice, acting as the sender, sends values $v_1,\ldots,v_{|\Sigma|}$, where $v_l = r - c(\alpha[i],\beta[l])$ and r is Alice's random share of the current matrix element. Bob, acting as the chooser, selects the element with index $\beta[j]$, thus obtaining $r - c(\alpha[i],\beta[j])$. In Phase 2, Alice and Bob's shares are input into a maximum-of-three circuit which computes and shares the next value of the dynamic programming matrix. The remaining details of the protocol are the same as for edit distance.

5 Generalization to Privacy-Preserving Dynamic Programming

We now generalize the protocols of Section 3 to arbitrary dynamic programming problems. Let $\mathcal{P}(x,y)$ be a problem with two inputs x and y, *e.g.*, in the edit distance case, x and y are the two 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 the 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:

$$\begin{array}{c} \{(i,0) \mid 0 \leq i \leq n\} \\ \{(0,j) \mid 0 \leq j \leq m\} \end{array}$$

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 so that it satisfies the following properties:

- For all 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 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 the case of the edit distance problem, $val((i, j))$ is equal to $D(i, j)$. The values for the base and non-base sub-problems for the edit distance problem are defined in equations 1 and 3 in Section 3.1.

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 now design a privacy-preserving protocol 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, $val_A(root) \oplus val_B(root)$ gives the desired result.

Protocol 3: Protocol 3 depends heavily on the structure of the space S of sub-problems. For example, for the edit distance problem, Protocol 3 fundamentally relies on the matrix structure of S.

6 Experimental Evaluation

In this section we present experimental results for our protocols for computing the edit distance and the Smith-Waterman distance between two strings. For edit distance, our tests were performed on random strings. For the Smith-Waterman distance, we aligned representative protein sequences from the Pfam database of protein sequences [2] in protein family QH-AmDH_gamma (PF08992), which is a crystalline quinohemoprotein amine dehydrogenase from Pseudomonas putida. The average length of these proteins is 78 amino acids. In order to demonstrate the scalability of the algorithm, we truncate the proteins to various lengths as shown in figure 6. For the cost function, we used the BLOSUM62 matrix [7] which is a $(20, 20)$ substitution matrix based on log-odds statistics derived from experimental protein data which approximates the probability of substitution of amino acids in homologous proteins. It is a commonly used metric in genomic research.

6.1 Edit distance

We implemented the standard methods for secure circuit evaluation, *i.e.*, the Yao's "garbled circuits" method and secure computation with shares (see Section 2). We used the oblivious transfer protocol due to Naor and Pinkas [22]. For the minimum-of-three computation, we used the lowest-price auction circuit of Kurosawa and Ogata [18]. Using these primitives, we implemented the three protocols of Section 3. For comparison purposes, we also implemented the edit distance protocol of Atallah *et al.* [1], using the Lin-Tzeng construction for the millionaires' protocol [19] and Paillier homomorphic encryption [24] (see appendix C). All of the code was written in Java.

The experiments were executed on two 3-GHz Pentium 4 machines, with 2 Gigabytes of memory, and connected via 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. For simplicity, all experiments were performed on problems where $m = n$. We used the alphabet size of 256 in our experiments. The main conclusions that can be drawn from our measurements are:

- *Protocol 1 (generic SMC) is very fast.* Protocol 1 is ideal for small strings because the entire computation is performed in one round, but the circuit size is extremely large for longer strings. Our prototype circuit compiler can compile circuits for problems of size $(200, 200)$ but uses almost 2 GB of memory to do so. Significantly larger circuits would be constrained by available memory for constructing their garbled versions.

- *Protocol 3 is most suitable for large problems.* Protocol 3 uses the grid structure of the problem space, which makes it most suitable for large instances. For example, a problem instance of size $(500, 500)$ takes under an hour. Asymptotically, Protocol 3 has the same performance as Protocol 2, but in practice it is substantially faster.

- *Bandwidth requirements are asymmetrical.* Because Alice sends the majority of data in the Naor-Pinkas oblivious transfer [22], bandwidth requirements are asymmetrical. Specifically, Alice sends far more data than she receives, while the opposite is the case for Bob. This fact can be exploited if the communication channel is asymmetric, such as with ADSL or cable lines, which typically offer a greater bandwidth for transmitting data in one direction than in the other. In this case, Alice's role can be assigned to the party that has higher upload bandwidth.

- *The edit distance protocol by Atallah et al. [1] is not practical.* In our edit-distance experiments, the protocol of [1] performed at least an order of magnitude worse than our protocols. This is because many large numbers (Paillier ciphertexts) are computed and sent multiple times by both Alice and Bob at each step. For example, on a problem instance of size $(25, 25)$ the protocol by Atallah *et al.* took 5 and half minutes. Our Protocol 3 took 14 seconds on the same problem instance.

Figure 5 shows the execution times for our three protocols. Clearly, Protocol 3 scales the best as the problem size increases. Protocol 1 is suitable for small problems. Protocol 2 has a larger execution time, but only requires limited bandwidth per round. Our experimental results confirm the protocol characteristics shown in Figure 2.

Detailed results for Protocols 1 and 2 are presented in the appendix. We discuss results for Protocol 3 in detail. Recall that in this protocol a grid structure is used (see Section 3.5). Using Protocol 3, we were able to solve problem instances of considerable size; here we present measurements for a problem instance of size (200,200). Table 1 shows the results using various grid sizes. Performance steadily improves up to the grid size of 20, but begins to decrease slightly after that. In spite of decreased overall performance, further increases in the grid size slightly decrease network bandwidth requirements, which results in fewer round trips, so even larger grid sizes may be suitable for environments with limited network bandwidth. With a grid size of 20, Protocol 3 requires about as much time for an instance of size $(200, 200)$ as Protocol 2 requires for an instance of size $(25, 25)$.

6.2 Smith-Waterman

Figure 6 shows the timing measurements for the three protocols. For Protocols 1 and 3, the computation time scales with the size of the score matrix, which is $|\Sigma|^2$. For example, the bytes transferred over the network aligning protein sequences using BLOSUM62 are approximately 40 times that of for simple edit distance of the same size problem. This is caused by the use of extra gates in the Yao circuit which encode each value of the score function.

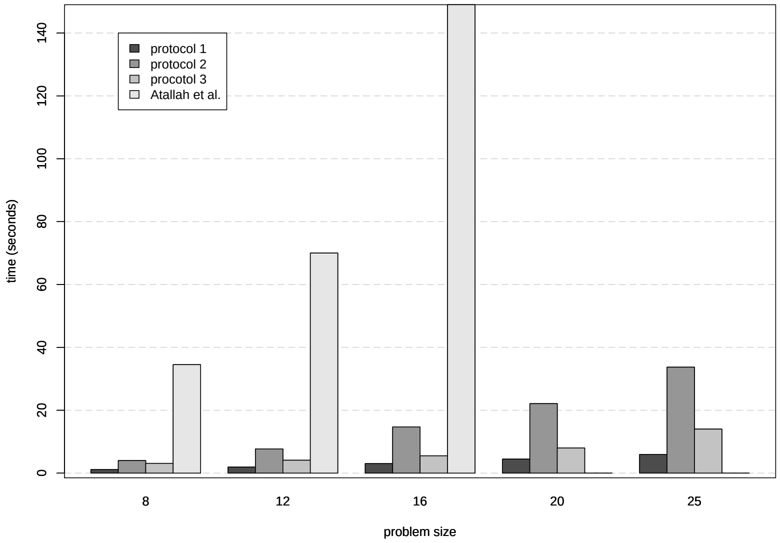

Figure 5. Timing measurements (in minutes and seconds) comparing edit distance Protocols 1, 2, and 3. The protocol by Atallah et al. [1] could not complete problems of size (20, 20) and (25, 25) within the 3-minute cutoff for this experiment.

Protocol 2 scales with the alphabet size $| \Sigma |$. For a very large alphabet with hundreds of symbols, Protocol 2 is the best choice because the cost of embedding the entire matrix into a Yao circuit becomes prohibitive.

7 Conclusion

We presented several privacy-preserving protocols for computing on genomic data, including calculating the edit distance and Smith-Waterman similarity scores between genome sequences. We evaluated our implementation on real problem instances involving alignment of protein sequences, and demonstrated that its performance is tractable even for instances of substantial size. Our techniques generalize to other dynamic programming algorithms, and we expect that they will find application in other contexts, *e.g.*, in hierarchical clustering algorithms which use edit distance as the metric.

Our protocols are but a first step towards practical methods for achieving privacy in genomic computation. In our solutions, cryptographic security comes at a substantial cost in computation and communication. Development of more efficient, yet provably secure techniques for collaborative computation on genome sequences is an important challenge for future research.

References

[1] M. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES*, 2003.

[2] A. Bateman, E. Birney, L. Cerruti, R. Durbin, L. Etwiller, S. R. Eddy, S. Griffiths-Jones, K. L. Howe, M. Marshall, and E. L. Sonnhammer. The Pfam protein families database. *Nucleic Acids Res*, 30(1):276–280, January 2002.

[3] J. Brickell, D. Porter, V. Shmatikov, and E. Witchel. Privacy-preserving remote diagnostics. In *CCS*, 2007.

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

[5] L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification*. W3C Recommendation, 16 April 2002.

[6] L. F. Cranor. Internet privacy. *Communications of the ACM*, 42(2):28–38, 1999.

[7] S. R. Eddy. Where did the blosum62 alignment score matrix come from? *Nat Biotechnol*, 22(8):1035–1036, August 2004.

[8] M. Fedoruk. Mapping the Icelandic genome. `http://www.scq.ubc.ca/?p=381`, 2003.

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

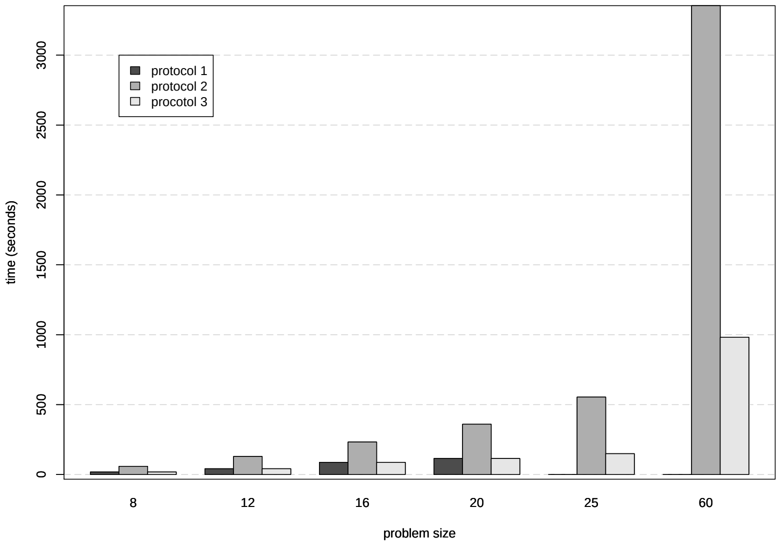

Figure 6. Timing measurements (in minutes and seconds) comparing Smith-Waterman Protocols 1, 2, and 3. For problem sizes (25,25) and (60,60), Protocol 1 could not evaluate the circuit.

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 the edit distance Protocol 3 with a problem of size $(200, 200)$ (M refers to Megabytes).

[10] M. Freedman, K. Nissim, and B. Pinkas. Efficient private matching and set intersection. In *EUROCRYPT*, 2004.

[11] Genomic Privacy Project. `http://privacy.cs.cmu.edu/dataprivacy/projects/genetic/`, 2004.

[12] I. Goldberg, D. Wagner, and E. Brewer. Privacy-enhancing technologies for the Internet. In *COMPCON*, 1997.

[13] O. Goldreich. *Foundations of Cryptography: Volume II Basic Applications*. Cambridge University Press, 2004.

[14] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *STOC*, 1987.

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

[16] HapMap. International HapMap project. `http://www.hapmap.org/`, 2007.

[17] S. Jarecki and V. Shmatikov. Efficient two-party secure computation on committed inputs. In *EUROCRYPT*, 2007.

[18] K. Kurosawa and W. Ogata. Bit-slice auction circuit. In *ESORICS*, 2002.

[19] H.-Y. Lin and W.-G. Tzeng. An efficient solution to the millionaires' problem based on homomorphic encryption. In *ACNS*, 2005.

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

[21] Y. Lindell and B. Pinkas. A proof of Yao's protocol for secure two-party computation. `http://eprint.iacr.org/2004/175`, 2004.

[22] M. Naor and B. Pinkas. Efficient oblivious transfer protocols. In *SODA*, 2001.

[23] M. Naor, B. Pinkas, and R. Sumner. Privacy preserving auctions and mechanism design. In *EC*, 1999.

[24] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, 1999.

[25] Personal Genome Project. Are guarantees of genome anonymity realistic? `http://arep.med.harvard.edu/PGP/Anon.htm`, 2006.

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

[27] 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.

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

[29] 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 *NDSS*, 2006.

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

[31] A. Yao. How to generate and exchange secrets. In *FOCS*, 1986.

A Proof of Lemma 1

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

$$\begin{aligned} D(i,j) &= \min[D(i-1,j)+1, D(i,j-1)+1, \\ &\quad D(i-1,j-1)+t(i,j)] \end{aligned}$$

The induction step is tedious but simple. Assume that the result is true for all l' and w' such that $l' \leq l$ and $w' \leq w$. We will prove the result that for $l+1$ and w. Recall that we have to prove that $D(i,j)$, can be expressed as a function of $bottom(D(i,j), l+1, w)$, $left(D(i,j), l+1, w)$, $\alpha[i-w+1 \cdots i]$, and $\beta[j-l \cdots j]$. The following three statements are true by induction hypothesis:

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

Notice that $D(i,j)$ can be expressed in terms of $D(i-1,j)$, $D(i,j-1)$, $D(i-1,j-1)$, $\alpha[i]$, and $\beta[j]$. Now the result is clear by combining the the four statements mentioned above. Similarly, we can prove that $D(i,j)$, can be expressed as a function of $bottom(D(i,j), l, w+1)$, $left(D(i,j), l, w+1)$, $\alpha[i-w \cdots i]$, and $\beta[j-l+1 \cdots j]$. $\Box$

B Detailed Results for Protocols 1 and 2

As a preparation step, a specific garbled circuit must be constructed for each problem instance. To create circuits, we used a prototype circuit compiler we are developing, which can be downloaded from the website at `http://www.cs.wisc.edu/~lpkruger/sfe/` . Table 2 shows the network bandwidth (in bytes) and execution times (in seconds) for various problem instances. In this experiment, the circuits operate on 8-bit integers, which allows for a maximum edit distance of 255. Results for Protocol 2 can be found in Table 3.

C Comparison with the Edit Distance Protocol of Atallah et al. [1]

In [1], Atallah *et al.* presented a privacy-preserving edit distance protocol, which is superficially similar to our Protocol 2 in that the intermediate values $D(i,j)$ are additively shared between Alice and Bob. The protocol of [1] relies on different cryptographic techniques, including special-purpose solutions to the so called "millionaires' problem" (a two-party protocol, in which the parties determine whose input is bigger without revealing the actual input values) and additively homomorphic encryption.

In this section, we present a detailed comparison of the online computational cost of our protocol vs. that of [1]. Let $q = \lceil \log w \rceil$ be the length of each alphabet symbol, and let $s = \log(n+m)$ be the length of random shares used to mask $D(i,j)$ in our protocol. In the protocol of [1], masking is done by adding random values under homomorphic encryption, in a group of very larger order (*e.g.*, RSA modulus N in the case of Paillier encryption) which is much bigger than $n+m$. Modular addition requires random shares to be N bits long and will make the computations unrealistically expensive. The alternative is to perform addition of random shares to encrypted values over integers. To achieve standard cryptographic security in this case, the length of random shares in bits must be at least $s' = \log(n+m) + 80 = s + 80$.

Problem size	Bandwidth (Alice)	Bandwidth) (Bob)	CPU (Alice)	CPU (Bob)	wall clock
(8,8)	0.37 M	3633	0.74	0.39	1.12
(12,12)	0.96 M	5348	1.30	0.54	1.92
(16,16)	1.83 M	7057	2.12	0.68	3.02
(20,20)	2.97 M	8764	3.10	0.88	4.46
(25,25)	4.38M	10472	4.26	1.17	5.94
(100,100)	86.7M	43029	71.5	14.1	92.4

Table 2. Network bandwidth (in bytes) and timing measurements (in seconds) for the edit distance Protocol 1 (M refers to Megabytes).

Problem size	Bandwidth (Alice)	Bandwidth) (Bob)	CPU (Alice)	CPU (Bob)	wall clock
(8,8)	717 k	68 k	3.03	1.29	4.05
(12,12)	1.60 M	154 k	5.96	2.37	7.68
(16,16)	3.36 M	315 k	11.5	4.38	14.7
(20,20)	5.26 M	492 k	17.5	6.54	22.1
(25,25)	8.21 M	769 k	26.8	9.7	33.7
(100,100)	171.1 M	32.0 M	519	177	649

Table 3. Network bandwidth (in bytes) and timing measurements (in seconds) for the edit distance Protocol 2 with various problem sizes (k and M refer to Kilobytes and Megabytes, respectively).

Below, we compare the cost for a single iteration, since the number of iterations is equal to nm in both protocols.

Online computational cost of [1]. Each iteration has to compute a minimum or maximum three times: twice in step 1 on q-bit values, and once in step 5 on s'-bit values [1, section 4.1]. Each minimum/maximum finding protocol requires two instances of the millionaires' subprotocol, and six re-randomizations of Paillier ciphertexts. The latter involves exponentiation modulo N^2, where N^2 is the modulus of an instance of Paillier cryptosystem. N itself is an RSA modulus and must be at least 1024 bits; therefore, N^2 is at least 2048 bits.

The implementations of the millionaires' protocol suggested in [1] are relatively inefficient. For fair comparison, we will assume that the construction of [1] is instantiated with a state-of-the-art sub-protocol for the millionaires' problem, *e.g.*, the Lin-Tzeng protocol [19]. This protocol requires $(1540s'-6)$ online modular multiplications per instance if s'-bit values are being compared ($1540q-6$ if q-bit values are being compared), assuming the standard size of 512 bits for the prime moduli in ElGamal encryption.

Assuming that the permutations required by [1] are free, the online cost of each iteration is thus equivalent to $2\times(2\times(1540q-6)+6\times2048)+(2\times(1540s'-6)+6\times2048)$ = $2\times(3080q-12+12288)+(3080s'-12+12288)$ = $3080s'+6160q+36828$ = $3080s+6160q+283228$ modular multiplications.

Online computational cost of our Protocol 2. Each iteration of our Protocol 2 involves evaluation of several "garbled circuits." Each C_{eq} circuit has $2q$ gates of arity 2, and each C_{min3} circuit has $10s$ gates of arity 2, and $5s-6$ gates of arity 3. In each iteration, a single instance of C_{eq} and a single instance of C_{min3} must be evaluated (in our presentation, evaluation of circuits C_{eq} and C_{min3} is split between two phases, but there is a 1:1 correspondence between the iterations of each phase).

All garbled circuits can be pre-computed in advance, because the representation of the circuit in Yao's protocol is independent of the actual input values. Each row of the truth table of each gate becomes a double-encrypted symmetric ciphertext (see Section 2), for a total of $4\times2q+(4\times10s+8\times(5s-6))$ = $8q+80s-48$ ciphertexts. Decrypting each double-encrypted ciphertext requires two online symmetric decryptions, but, on average, the evaluator of a garbled gate will only need to try decrypting half the ciphertexts before decryption succeeds and he obtains the wire key representing the bit value of the gate's output wire.

Transferring the wire-key representation of Bob's q-bit input into C_{eq} requires q instances of OT_1^2. The online cost of each instance is 2 modular exponentiations and 1 modular multiplication. Therefore, assuming 512-bit moduli, the total online cost of obliviously transferring the inputs to C_{eq} is equivalent to $1025q$ modular multiplications.

In the same iteration, a single instance of C_{min3} must be evaluated. Bob has three s-bit inputs (after evaluating C_{eq}, he already has the representation for his fourth input). Obliviously transferring the wire-key representation of these inputs requires $3s$ instances of OT_1^2, for a total cost of $3075s$ modular multiplications.

Therefore, the total online cost of each iteration of our Protocol 2 is $(3075s + 1025q)$ modular multiplications and $8q + 80s - 48$ symmetric decryptions vs. $(3080s + 6160q + 283228)$ modular multiplications in each iteration of [1]. Since symmetric decryption is much cheaper than modular multiplication, we conclude that our Protocol 2 offers significantly better efficiency than the protocol of [1]. In general, the protocol of [1] requires at least 300,000 modular multiplications per iteration, rendering it unrealistically expensive for practical applications.

D Security Proofs

Our protocols are secure in the so called the *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 the protocol specification.

Since we use Yao's "garbled circuits" method as the underlying primitive, security in the malicious model, if needed, can be achieved at a constant cost [17]. For practical usage scenarios, however, 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 each party to run the protocol on previously committed inputs, this does not guarantee that the inputs are not maliciously chosen in the first place. In other words, a malicious party may simply commit to a "bad" input (deliberately chosen so that the result of the edit distance computation reveals some information about the other party's input) and pass all proofs.

In general, we expect that our protocols will be used for tasks such as collaborative analysis of genome sequence in joint medical studies, where it is reasonable to assume that participants provide actual sequences as inputs into the protocol, and are not deliberately supplying fake sequences in an attempt to learn something about the other participant's data.

Security of our protocols 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 [13, Theorem 7.3.3]. Proofs are standard and omitted from the conference version of the paper. Note that it is not possible to completely hide the length of the sequences, as an upper bound can be inferred from the size of the computation. Random padding of the sequences can be used to mitigate this with a corresponding performance penalty.