Privacy Preserving Similarity Evaluation of Time Series Data

Haohan Zhu
Department of Computer Science
Boston University
zhu@cs.bu.edu

Xianrui Meng
Department of Computer Science
Boston University
xmeng@cs.bu.edu

George Kollios
Department of Computer Science
Boston University
gkollios@cs.bu.edu

ABSTRACT

Privacy preserving issues of time series databases in financial, medical and transportation applications have become more and more important recently. A key problem in time series databases is to compute the similarity between two different time series. Despite some recent work on time series security and privacy, there is very limited progress on securely computing the similarity between two time series. In this paper, we consider exactly this problem in a two-party setting (client and server). In particular, we want to compute the similarity between two time series, one from the client and the other from the server, without revealing the actual time series to the other party. Only the value of the similarity should be revealed to both parties at the end. At the same time, we want to do the computation as efficiently as possible. Therefore, we propose practical protocols for computing the similarity (or distance) for time series using two popular and well known functions: Dynamic Time Warping and Discrete Fréchet Distance. Since both of these functions require dynamic programming to be computed, our protocols not only encrypt the original time series data, but also try to hide intermediate results, including the matrix of the dynamic programming algorithm and the path of the optimal solution. The protocols combine partial homomorphic encryption and random offsets to protect intermediate information and at the same time provide efficient computation. Unlike previous approaches that are mostly theoretical, our protocols are scalable and easy to implement. We also provide an experimental evaluation where we assess the scalability and practicality of our schemes using both synthetic and real datasets.

Categories and Subject Descriptors

I.5.3 [**Clustering**]: Similarity measures

Keywords

Time Series Database, Similarity, Privacy Preserving

1. INTRODUCTION

Database security has received increased interest in recent years due to the increasing demand for efficient data management and the development and deployment of outsourced and cloud databases. Most of recent works in the database security community concentrates on traditional databases like relational, object relational, and XML storage systems. However, despite the fact that many important applications like financial, medical and transportation applications store data in the form of time series data and these applications store sensitive and valuable information, little work has been done on the security problems associated with these applications. In some very timely recent works [30, 32] related to time series data, the main problem is to answer aggregation and other statistical queries on time series data using differential privacy to quantify the security guarantees. However, there are a number of other important query types in time series databases that have not been addressed efficiently yet from a security or privacy point of view. Examples include time series similarity search and subsequence matching queries. In this paper, we initiate one of the first studies in developing *practical* algorithms and protocols to compute the similarity between time series data in a way that is *efficient* and at the same time protects the *privacy* of the individuals associated with these time series data. Indeed, there are a large number of applications that require such protocols.

For example, consider a hospital database which stores physiological data (e.g., ECG) in the form of time series associated with certain diseases. Also, consider a new patient, Alice, that has her own physiological data and need to search in the database to find whether there exists any patient with similar characteristics. The problem can be reduced to a similarity search of time series data. However, Alice does not want to reveal her own data, unless there is a good match. At the same time, the hospital does not want to provide the actual data of its database to Alice in order to protect the privacy of the patients associated with the data. Therefore, in that case, we need a protocol to exchange physiological data information and compute the similarity while preserving the privacy of both the hospital database and the new patient. Another example is the following: consider a user, Bob, that would like to verify if his signature, represented as time series, exists in a signature database. Again, both Bob and the database owner may not want to reveal the actual time series data. The only information that can be learned is the similarity between the Bob's signature and a database record. Similar problems

10.5441/002/edbt.2014.45

exist in many other time series applications including audio and video databases, financial databases, medical databases, and trajectory databases. Notice that in our examples we have two parties where one party is a user and the other party stores a database or large amounts of data. Following this paradigm, our protocols are asymmetric in the sense that we assume that one party can have more computational power than the other party. We call the weak party "client" and the other party "server" in the rest of the paper.

In our setting, we have exactly two parties that have their own time series and would like to compute the similarity between them without revealing the actual time series to the other party. Notice however, that the value of the similarity between two time series is revealed to both parties and therefore some additional information can be leaked because of that. In this paper, similar to previous related work [13, 18, 2], we assume that both parties agree that they are willing to accept this leakage. As it was pointed out in [13], understanding the amount of private information that the result itself leaks is an important and interesting problem but it is outside of the scope of this paper. Therefore, we consider only the information that is leaked during the execution of the protocol. More about our security model is discussed in Section 4.

Despite some recent works that investigate secure computation of some distance functions between strings or vectors, such as Hamming distance [17, 26], Edit distance [13, 18, 2] and Euclidean distance [14, 15, 8, 34, 16, 12], we are not aware of any work that computes *efficiently* and *securely* distance functions between time series data. In this paper we consider some specific distance functions, namely, Dynamic Time Warping and Discrete Fréchet Distance. Both Dynamic Time Warping and Discrete Fréchet Distances use dynamic programming to be computed and are very robust to small variations and noise. They have been shown to provide very good results in a number of different applications [35]. Existing approaches that can be used to compute these distance functions are either in-secure (computed on plaintexts) or very impractical (for example the approach in [2] will take many thousands of seconds to compute the distance between two time series of length 100.) In Section 2, we analyze why existing methods that we mention above cannot be easily extended to distance functions for time series data like Dynamic Time Warping and Discrete Fréchet Distance. On the other hand, our new proposed protocols decrease the computation of time series distance functions by at least three orders of magnitude compared to previous approaches and still provide considerable security. Furthermore, we provide the first real implementation of such protocols, as far as we know, for computing time series distance functions and provide a detailed experimental evaluation. We believe that in this work we show that computing on time series data with reasonable security and privacy guarantees can become both possible and practical. Finally, our techniques can be easily extended to other distance functions computed by dynamic programming algorithms as well.

Overall, this paper makes the following contributions:

- We propose novel privacy preserving protocols to compute Dynamic Time Warping and Discrete Fréchet Distances between two time series that belong to two different parties.
- Our protocols are scalable and practical. The performance is linear to the dimensionality of each element in the time series and quadratic to the length of the time series due to dynamic programming algorithm with no additional computation-overhead.
- The proposed protocols protect the original data and preserve at least *half* of the information entropy for the intermediate results generated during the execution of the protocols.
- Finally, we provide the implementation of our schemes and an experimental evaluation that validates the efficiency and practicality of the proposed protocols.

The paper is organized as follows: Section 2 introduces related work and Section 3 provides formal definitions of distance functions and existing protocols of secure Euclidean distance computation. Section 4 describes the threat model and potential risks in dynamic programming algorithms. Section 5 and Section 6 introduce privacy preserving protocols for computing Dynamic Time Warping and Discrete Fréchet Distance separately. Finally, Section 7 provides an experimental evaluation and Section 8 concludes the paper.

2. RELATED WORK

2.1 Preserving Privacy in Databases

Recently, issues related to database security and privacy have attracted more and more attention. Along with pervasive uses of differential privacy, many database applications can preserve privacy of both server and client. Differential privacy has already been applied in data publication [5, 33, 37], counting queries [23, 36], histogram queries [9], log queries [11], and spatial data queries [6]. Furthermore, there are some recent works on applying differential privacy to time series databases [32, 30]. However, the objective of both these two papers focus on answering aggregated information from time series databases. For exact match and nearest neighbor queries, differential privacy is not easy to apply since it is defined for statistical queries on databases. For those queries, recent work utilized applied cryptographic techniques such as private information retrieval [28] and oblivious transfer [29]. Multi-party computation and homomorphic encryption are also strong applied cryptographic techniques. Some works use multi-party computation protocol for table joins [19, 16, 1]. Murugesan et al [25] adopts multi-party computation to detect the similarity between documents. Yang et al [38] utilizes secure multi-party computation for secure summations in data mining. Moreover, many papers use homomorphic encryption to calculate Euclidean distances [15, 16, 12, 34].

2.2 Homomorphic Encryption

As described above, there are plenty of papers that introduce protocols to compute the square of Euclidean distance securely by using homomorphic encryption [14, 15, 8, 34, 16, 12]. In this paper, we adopt the same protocol to compute square of Euclidean distance securely which is part of our intermediate results. In Section 3.2, we discuss for completeness the protocol of computing the square of Euclidean distance. This is the same as the protocols in the literature above but differ a little bit at the information exchange settings. In the experiments, we use the Paillier cryptosystem [27] as a partial homomorphic encryption system. This encryption satisfies homomorphic addition of ciphertexts and homomorphic multiplication to plaintexts. This fact makes Paillier cryptosystem very well suited for computing the square of Euclidean distance efficiently.

2.3 Two-Party Computation

Since Yao proposed the millionaire problem [39], a number of protocols for secure multi-party computation have been developed. Many custom protocols have been devised for different problems. Furthermore, several systems such as Fairplay [24] and TASTY [10] based on garbled circuits have been developed recently and have been used for building secure protocols for securely computing functions in two-party settings.

To allow time shifting, dynamic programming algorithms are usually used for evaluating similarity between strings, sequences and time series. There are some recent papers that deal with the Edit Distance using dynamic programming algorithms. Huang et al [13] and Jha et al [18] use garbled circuits to compute the Edit Distance and Smith-Waterman distance for sequences of DNA and string databases. Although Edit Distance and Smith-Waterman are computed using dynamic programming, the distance between a pair of characters is quite different from the distance between two elements of time series sequences. Garbled circuits can be used to check the identity of two characters using very efficient and inexpensive XOR gates [22]. On the other hand, in time series, we have to compute a distance between real or integer values and sometimes, when the elements are vectors, we need to compute an L_p norm. For this case, it is known that garbled circuits approaches can be extremely expensive since we have to implement full adders and multipliers into the circuit that will increase its size and complexity [21]. Therefore, although in theory it is possible to use garbled circuits to compute the distance between time series data, it is expected to be very expensive in terms of space and communication overhead and computation time according to existing circuits. Hence, although Edit Distance, Dynamic Time Warping and Discrete Fréchet Distance all use dynamic programming algorithms to compute, the methods using garbled circuits to compute Edit Distance are not suitable for computing distance functions for time series data.

Atallah et al [2] proposes a protocol to compute edit distance rather than using garbled circuits. In their protocol, the matrix for dynamic programming is shared by two parties. To protect intermediate results, they develop a minimum finding protocol which runs many times Yao's protocol [39] which is expensive. In particular, if we have two time series X and Y with size m and n respectively, and each element of the time series is a d-dimensional vector, we need to run about $3 * m * n * d^2$ times the Yao's protocol. To compute the Yao's protocol a single time in one of the most efficient implementations right now it will take at least 1.25 seconds [24]. Notice that here we do not include other operations that can be also costly. Therefore, this protocol will be very inefficient in practice. We would like to point out that in our protocols we use a completely different approach to store the matrix and this allows our methods to be much more efficient as we discuss next.

Thus, to our best knowledge, there is no practical existing protocol which computes distance functions for time series data like Dynamic Time Warping and Discrete Fréchet Distance efficiently.

3. PRELIMINARIES

3.1 Distance Functions for Time Series Data

Let X and Y be two time series sequences with lengths $|X|$ and $|Y|$ respectively, where $X = \langle\, x_1, x_2, x_3, ..., x_{|X|} \,\rangle$ and $Y = \langle\, y_1, y_2, y_3, ..., y_{|Y|} \,\rangle$. Let the dimensionality of each element in each sequence be d, namely, x_i, $y_j \in \mathbb{R}^d$. Then, the two time series are comparable, only if the elements x_i and y_j have the same dimensionality. Notice that we allow $|X|$ to be different from $|Y|$.

There are a number of distance functions which can measure the dissimilarity (distance) between two time series sequences including Euclidean Distance, Dynamic Time Warping [20], Discrete Frechét Distance [7], ERP Distance [4], and etc. We denote by $\delta(X,Y)$ the distance between two time series sequences X and Y. In this paper, we concentrate on the Dynamic Time Warping δ_{DTW} and Discrete Frechét Distance δ_{DFD}. We use these distances because they are more robust to time shifting and small noise and have been shown to be practical and efficient for many time series data mining tasks [35].

Both Dynamic Time Warping δ_{DTW} and Discrete Frechét Distance δ_{DFD} are computed by using dynamic programming. Next, we discuss the definitions of these distances. First, we define a coupling C which is a sequence of pairs of elements from sequence X and sequence Y:

- $C = \langle C_1, \cdots, C_k\rangle$ where $\forall r, C_r = (x_i, y_j)$.
- $C_1 = (x_1, y_1)$ and $C_k = (x_{|X|}, y_{|Y|})$
- $C_r = (\, x_i \,,\, y_j \,) \Rightarrow C_{r+1} \in \{\, (\, x_{i+1} \,,\, y_j \,) \,,\, (\, x_i \,,\, y_{j+1} \,) \,,\, (\, x_{i+1} \,,\, y_{j+1} \,) \,\} \,;\, \forall\, r = 1 \,,\, \ldots \,,\, k-1$

Let X, Y be two time series sequences with elements in $\mathbb{R}^d$, and let $|\cdot|$ denote an underlying norm in $\mathbb{R}^d$. Then Dynamic Time Warping $\delta_{DTW}(X,Y)$ is defined as:

$$\delta_{DTW}(X,Y) = \min_{couplingC} \Sigma_{(x_i,y_j)\in C}|x_i - y_j| \tag{1}$$

while Discrete Fréchet Distance $\delta_{DFD}(X,Y)$ as:

$$\delta_{DFD}(X,Y) = \min_{couplingC} \max_{(x_i,y_j)\in C} |x_i - y_j| \tag{2}$$

Let $\delta(X_i, Y_j)$ be a cumulative distance, where X_i denotes a time series subsequence extracted from sequence X starting from the first element x_1 to the i-th element x_i. Let Y_j denote the same from y_1 to y_j. Then, Dynamic Time Warping can be represented as:

$$\delta_{DTW}(X_i,Y_j) = |x_i - y_j| + min\{\delta_{DTW}(X_{i-1},Y_j), \delta_{DTW}(X_i,Y_{j-1}), \delta_{DTW}(X_{i-1},Y_{j-1})\}$$

while Discrete Fréchet Distance can be represented as:

$$\delta_{DFD}(X_i,Y_j) = max\{|x_i - y_j|, min\{\delta_{DFD}(X_{i-1},Y_j), \delta_{DFD}(X_i,Y_{j-1}), \delta_{DFD}(X_{i-1},Y_{j-1})\}\}$$

Therefore, to compute the distance between X and Y, we have to fill an $|X| \times |Y|$ matrix $M_{|X||Y|}$ with each entry $m_{i,j} = \delta(X_i, Y_j)$. The entry $m_{|X|,|Y|} = \delta(X,Y)$ is the distance between the two sequences. Thus, the time complexity of computing both distance functions is $O(|X||Y|)$.

Now, we are ready to define the problem that we address in this paper:

Problem: Let P_1 and P_2 be two parties that own time series X and Y respectively. We call P_1 client and P_2 server.

The goal is to design a protocol to compute the distance $\delta(X, Y)$ without either party learning the actual time series of the other party. Furthermore, we would like to execute the computation as fast as possible and minimize any leakage of information during the execution of the protocol. The final result will be shared by both parties.

3.2 Protocol for Computing the Square of the Euclidean Distance Securely

In the process of computing either Dynamic Time Warping or Discrete Fréchet Distance X and Y, the norm of every pair elements x_i and y_j should be computed. The most common norm is the L_2 norm which corresponds to the Euclidean distance. In our schemes we use the square of the Euclidean distance because it is easier to compute securely and the results are similar as in the case of the exact Euclidean distance. We denote this distance as δ^2_{Eu}.

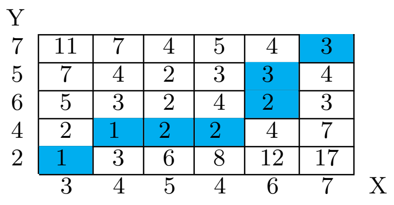
There are many approaches using partial homomorphic encryption systems to compute the square of Euclidean distance between two vectors securely [14, 15, 8, 34, 16, 12]. For completeness, we briefly introduce the protocol of computing square of Euclidean distance in this Section. The only slight difference compared to the previous protocols is the setting for information exchange. In our protocol, we require one party to send all the encrypted information to the other party only once and the other party calculates the encrypted square of the Euclidean distance that is used during the calculations of the dynamic programming algorithm. In other words, in our protocol only one way communication is enough and we do not require any party to decrypt or retrieve the real value of square of Euclidean distance. The encrypted values are the intermediate results in our protocol.

Let the client hold a d-dimensional vector $\widehat{p} = \langle p_1, ...p_d \rangle$ and let the server hold another d-dimensional vector $\widehat{q} = \langle q_1, ...q_d \rangle$. The client would like to evaluate the *encryption* of square of Euclidean distance $Enc(\delta^2_{Eu}(\widehat{p}, \widehat{q}))$. Namely, the client should compute the following function:

$$Enc(\delta^2_{Eu}(\widehat{p}, \widehat{q})) = Enc(\sum_{i=1}^{d} p_i^2 + \sum_{i=1}^{d} q_i^2 - 2\sum_{i=1}^{d} p_i q_i) \quad (3)$$

To protect the square of Euclidean distance to be revealed, the public key is known by both the client and the server, however only the server has the secret key, since the client is the evaluator in this protocol. Therefore, if the client receives only encrypted values generated from $\widehat{q}$, it cannot learn anything about $\widehat{q}$. Also, since the server does not receive anything from the client, the server cannot learn anything about $\widehat{p}$ either. If the encryption system satisfies homomorphic addition, the client can evaluate $Enc(\delta^2_{Eu})$ by using the ciphertexts from $\widehat{q}$ and the plaintexts from $\widehat{p}$.

For the Paillier cryptosystem, we have that $Enc(m_1 + m_2 \bmod n)$ is equal to $Enc(m_1) \cdot Enc(m_2) \bmod n^2$. Also, we have that $Enc(m_1 * k \bmod n)$ is equal to $Enc(m_1)^k \bmod n^2$. Then the Paillier cryptosystem satisfies all the requirements to compute the encrypted square of the Euclidean distance. Thus, Equation (3) can be decomposed to:



$$\begin{array}{c|cccccc} Y & & & & & & \\ 7 & 11 & 7 & 4 & 5 & 4 & 3 \\ 5 & 7 & 4 & 2 & 3 & 3 & 4 \\ 6 & 5 & 3 & 2 & 4 & 2 & 3 \\ 4 & 2 & 1 & 2 & 2 & 4 & 7 \\ 2 & 1 & 3 & 6 & 8 & 12 & 17 \\ \hline & 3 & 4 & 5 & 4 & 6 & 7 \quad X \end{array}$$

Figure 1: Matrix Computed during the Execution of Dynamic Programming

$$\underbrace{Enc(\sum_{i=1}^{d} p_i^2)}_{\text{(I)}} \cdot \underbrace{Enc(\sum_{i=1}^{d} q_i^2)}_{\text{(II)}} \cdot \underbrace{\prod_{i=1}^{d} Enc(q_i)^{-2p_i}}_{\text{(III)}} \quad (4)$$

Now, the three parts can be calculated separately. Indeed, the server (the owner of $\widehat{q}$) can compute "part (II)" locally and send ciphertexts to the client (the evaluator and the owner of $\widehat{p}$). Also the client can calculate "part (I)" locally. Then the server sends all encrypted q_i to the client. The client can compute "part (III)" by using $Enc(q_i)$ and p_i. Finally, the client can evaluate $Enc(\delta^2_{Eu})$ without knowing any information about $\widehat{q}$.

The one-way protocol for computing the encrypted square of Euclidean distance is:

1. The Server
   (a) Encrypt $\sum_{i=1}^{d} q_i^2$ and each q_i
   (b) Send $Enc(\sum_{i=1}^{d} q_i^2)$ and $Enc(q_i)$ to the client
2. The Client
   (a) Encrypt $\sum_{i=1}^{d} p_i^2 \Rightarrow Enc(\sum_{i=1}^{d} p_i^2)$
   (b) Calculate $Enc(q_i)^{-2p_i}$ by using $Enc(q_i)$.
   (c) Calculate $Enc(\delta^2_{Eu})$ using Equation (4)

The above protocol is used to compute the pairwise distances between every pair of elements of X and Y. In Sections 5 and 6, we present the overall protocols which will use the above protocol as a part and show how to use encrypted squares of Euclidean distances in order to fill the matrix of the dynamic programming algorithm. Before we go there, we discuss first our security model and settings.

4. SECURITY MODEL AND SETTINGS

In this paper, we adopt the semi-honest threat model which is also called honest-but-curious (HBC) threat model. In this threat model, we assume that both parties (client and server) follow the protocols exactly. The semi-honest threat model is a commonly used model in two-party computation.

Even though both the client and the server follow exactly each protocol, they may try to learn additional information about the time series of the other party during the execution of the protocol. The basic objective in our scheme is to limit the information revealed to each other during the computation. As we discussed in the introduction, the final result of the computation may reveal some information that depends on the distance function and the data. We assume that both parties are willing to take that risk. On the other hand, the information that is leaked during the execution of the protocol should be quantified and minimized as much as

Y					
$Enc(7)$	$Enc(11)$	$Enc(7)$	$Enc(4)$	$Enc(5)$	...
$Enc(5)$	$Enc(7)$	$Enc(4)$	$Enc(2)$	$Enc(3)$	...
$Enc(6)$	$Enc(5)$	$Enc(3)$	$Enc(2)$	$Enc(4)$	...
$Enc(4)$	$Enc(2)$	$Enc(1)$	$Enc(2)$	$Enc(2)$	...
$Enc(2)$	$Enc(1)$	$Enc(3)$	$Enc(6)$	$Enc(8)$	...
	3	4	5	4	X

Figure 2: Cipher Matrix Owned by Evaluator X

y_j	$Enc(a)$	$Enc(\delta^2_{Eu}(x_i, y_j)) \cdot Enc(\min\{a, b, c\})$
y_{j-1}	$Enc(b)$	$Enc(c)$
	x_{i-1}	x_i

Figure 3: Filling One Cell for DTW

possible. In addition, we consider the leakage for a single query. The information that is leaked if a client runs many queries one after the other against the same time series is beyond the scope of the current paper.

There are two kinds of intermediate results during the computation of the distance functions. One is the underlying norm distance between a pair of single elements in the sequences. Our method is secure by using protocol in Section 3.2. The other are the entries of the $m \times n$ matrix generated during dynamic programming. For example, assume $X = (3, 4, 5, 4, 6, 7)$ and $Y = (2, 4, 6, 5, 7)$ are two time series sequences. When computing the Dynamic Time Warping distance $\delta_{DTW}(X, Y)$, we need to maintain a 6×5 matrix M. Figure 1 shows the matrix M for computing Dynamic Time Warping distance $\delta_{DTW}(X, Y)$.

Clearly, if the matrix is maintained by one party in plaintext, this party can infer the other party's data step by step. For example, assume the matrix owner is also the owner of X. Since this party knows $x_1 = 3$ and $m_{11} = 1 = \delta_{Eu}(x_1, y_1)$, then it can infer that y_1 is either 2 or 4. Since this party knows $x_2 = 4$ and $m_{21} = 3 = m_{11} + \delta_{Eu}(x_2, y_1)$. Then it knows $\delta_{Eu}(x_2, y_1) = 2$ and it can infer that y_1 is either 2 or 6. Thus, this party knows $y_1 = 2$. Similarly, this party can infer the whole sequence of Y. Therefore, the matrix of the dynamic programming algorithm should not be revealed.

Secret sharing is an important method to protect the matrix. However, as discussed in Section 2, splitting the matrix data [18, 2] is not efficient, because retrieving the original values needs extra two-party computation protocols. In our protocols, we adopt the schema which splits the encrypted data and the key between the parties. The encrypted values of the matrix are stored in one party (the client) and the secret key of the encryptions is kept in the other party (the server). Since the encryption is partially homomorphic, the client can still perform the necessary computations to create and update the matrix without learning any real value in the matrix. An example of the ciphertexts matrix is shown in Figure 2. We assume that the owner of sequence X maintains the ciphertext matrix. Then, it can access the plaintexts of X and the ciphertexts of Y in order to compute the ciphertexts of the entries $m_{i,j}$ in the matrix.

There is another important information included in the matrix: the path of the optimal couplings. If the matrix owner knows $m_{2,2}$ is equal to $m_{1,1}$, then the matrix owner learns that the coupling path is from $\{X_1, Y_1\}$ to $\{X_2, Y_2\}$. We claim that the ciphertext matrix will not expose such information, if the ciphertexts of the same plaintext are different. For example, in Figure 2, we require that $m_{1,1}$ and $m_{2,2}$ are different and independent ciphertexts although the decryptions are both 1. By using Paillier cryptosystem, we can achieve this since Paillier cryptosystem is probabilistic encryption system. The detail of analysis is in Section 5.5.

5. PRIVACY PRESERVING PROTOCOL FOR DTW

5.1 Privacy Preserving Protocol

Here, we present our privacy preserving protocol for computing the matrix for the dynamic programming computation of DTW. We assume that to compute the distance between two elements of the time series we use the square of the Euclidean distance.

Algorithm 1: Dynamic Time Warping

```
Input: Sequence X[m], Sequence Y[n]
Data: Matrix M[m][n]
Output: Value M[m][n]
M[i][1] = $\delta^2_{Eu}$(X[1], Y[1]);
for $i = 2 : m$ do  M[i][1] = $\delta^2_{Eu}$(X[i], Y[1]) + M[i-1][1];
for $j = 2 : n$ do  M[1][j] = $\delta^2_{Eu}$(X[1], Y[j]) + M[1][j-1];
for $i = 2 : m$ do
    for $j = 2 : n$ do
        cost = $\delta^2_{Eu}$(X[i], Y[j]);
        M[i][j] = cost +
            min(M[i-1][j-1], M[i-1][j], M[i][j-1]);
```

The main component of the dynamic programming algorithm is filling the matrix. This algorithm is the same as in the non-secure case with the main difference that the values in the matrix are actually encrypted using the Paillier encryption scheme. There are two parts in this component: 1) computing the square of the Euclidean distance and 2) comparing three entries that have already been filled in the matrix to find the minimum among them and then calculate the next entry. The process of filling one entry of the matrix is shown in Figure 3.

Since the two parts are independent, they can be executed in two different phases. First, we compute the distance between the ith element of X and the j element of Y for every i and j and store the results. Then, we perform the second phase where for each cell $M(i, j)$ in the matrix, we try to find the minimal value of the three previous ciphertexts $M(i-1, j), M(i-1, j-1), M(i, j-1)$ without revealing the real values to either party.

In the first phase, we reuse the protocol discussed in Section 3.2 to compute the square of the Euclidean distances for all pairs of elements from X and Y. As we discussed already this phase is fully secure and at the end of this phase the client will have all the values in encrypted form. The server holds the secret key of this encryption. In the second phase we need to compute the minimum among three ciphertexts and get a new ciphertext of the minimum. If we have both the square of the Euclidean distance and the minimum in encrypted form, then the encryption of new entry can be calculated since the encryption scheme is additively homomorphic.

In this section, we mainly discuss how to compute the encryption of the minimum value. Notice that, computing the encrypted minimum is another two-party computation problem. First we introduce the settings in our model. Here we have one party, the client, who has three encrypted values $e_1 = Enc(a)$, $e_2 = Enc(b)$, and $e_3 = Enc(c)$ and needs to acquire the encryption of the minimum $e' = Enc(min\{a, b, c\})$ with the help of the server. In order to hide the path, the new encryption e' should be different from all previous three encryptions e_1, e_2 and e_3. Neither of the client and the server should learn the plaintext values a, b, and c.

Before we present our solution, we discuss why existing approaches cannot be used to solve this problem efficiently. Kolesnikov et al. [21] proposed a simple theoretical solution to solve this problem securely using Yao's garbled circuits. However, since the circuits take homomorphically encrypted inputs and return homomorphically encrypted outputs, the cost of their solution will be very expensive in practice and therefore not practical, as they mention in their paper. Huang et al.[14] proposed a garbled circuit to return the index of the minimum value. However, due to the nature of our computation, if the client knows the index of the minimum value, he will learn the path of the optimal couplings which can be serious leakage. Finally, Atallah et al [2] proposed a minimum finding protocol rather than garbled circuits which retrieves the minimum from two split matrices. The protocol runs the Yao's protocols a few times to compute the minimum which is not practical either as we discussed before.

To solve the problem efficiently, we propose another solution. Our protocol guarantees that the client will learn nothing about the original values or the optimal path. On the other hand, some limited information may be leaked to the server. Details of the security analysis and information preservation are discussed in 5.3 and 5.4.

Since only the server has the secret key of the encryption scheme, the client can ask the server to decrypt the three values and find the minimum. However, this will leak the actual values to the server. In order to address this problem, we add a set of random values to the original values in a way that the server will still be able to compute the representative of the minimum value without knowing the actual values. Also, the server will send back an encrypted value, $Enc(min\{a, b, c\})$, to the client who can fill in the encrypted matrix.

Assume the client has three ciphertexts $Enc(a)$, $Enc(b)$ and $Enc(c)$. The client can add randomness to the ciphertexts by using homomorphic addition:

$$Enc(a + r) = Enc(a) \cdot Enc(r) \tag{5}$$

Let the client generate a set of random values R uniformly from some certain range, i.e. $R = \{ r_1 , \cdots , r_k \mid r_i > r_1 , \forall\ i \neq 1 \}$. W.l.o.g. we also denote r_1 as r_{min}. The client can generate a set of encrypted values that will be sent to the server. The set of new encrypted values is $C = \{ Enc(a+r_{min}), Enc(b+r_{min}), Enc(c+r_{min}), Enc(x_2+r_2), Enc(x_3+r_3) \ldots Enc(x_k+r_k) \}$, where x_i is chosen from a, b and c randomly. The size of the random set k is a parameter of our protocols and we will analyze its effect on the security of the protocol in Section 5.3 and effect on performance in Section 7.

Next, the server will decrypt all the ciphertexts and will return the encryption of the minimal one $Enc(m)$ to the client. The minimal value m from the server must be one of the $a + r_{min}$, $b + r_{min}$ and $c + r_{min}$, since all of r_i where $i \neq 1$, are larger than r_{min}. Then the client can retrieve the encryption of the real minimal value as $Enc(m - r_{min})$ using homomorphic addition.

The full protocol is as follow:

1. The Client
   (a) Has three ciphertexts: $Enc(a)$, $Enc(b)$ and $Enc(c)$.
   (b) Generate a set of random values $R = \{ r_1 , r_2 , r_3 \cdots , r_k \mid r_i > r_1 , \forall\ i > 1 \}$.
   (c) Generate 3 minimal candidates $Enc(a + r_{min})$, $Enc(b + r_{min})$ and $Enc(c + r_{min})$.
   (d) Generate other $k - 1$ candidates:
   For $i \in [2, k]$, generate $Enc(x_i + r_i)$, where each $Enc(x_i)$ is chosen randomly from $Enc(a)$, $Enc(b)$, and $Enc(c)$.
   (e) Send all $k + 2$ candidates to the server.
2. The Server
   (a) Decrypt all inputs.
   (b) Get the minimal plaintext as m.
   (c) Encrypt m as $Enc(m)$.
   (d) Send $Enc(m)$ to the client.
3. The Client
   (a) Calculate $Enc(m - r_{min})$.
   (b) The encryption of minimum of a, b, c is $Enc(m - r_{min})$.

We have to be careful when we add the random values in order to avoid wrap-around. The idea is to control the range of the random values and the range of the plaintexts in the matrix and ensure that there are enough bits to accommodate both in the additions.

5.2 Performance Analysis

In this section, we discuss first the communication cost and then the computational cost of our protocol. To compute an entry in the matrix, one and half rounds communication between the server and the client are needed. In phase 1 of the protocol one way communication from the server to the client is used. Since every element of the time series is in $\mathbb{R}^d$, $d + 1$ values need to be transferred for each entry in that phase. Phase 2 of the protocol uses one full round of communication from the client to the sever. The number of values transferred from the client to the sever depends on the size of the random set. If the size of this set is k, then $k + 2$ ciphertexts need to be transferred. On the other hand, the server returns only one value to the client. So, in phase 2, total of $k + 3$ values need to be transferred. Therefore, $d + k + 4$ total values will be transferred for each entry of the matrix. Assuming that $|X| = m$ and $|Y| = n$, the total number of transferred values is: $mn(d + k + 4)$.

Next, we consider the computational cost. The most expensive parts are the encryptions and decryptions of plaintexts and ciphertexts respectively when the dimension of time series is not very high. Since only the server has the secret key, it is the only party that performs decryptions.

In phase 1, the server needs to do $d+1$ encryptions. The client needs to do only one encryption. In phase 2, the sever performs $k+2$ decryptions and one encryption. The client needs to do k encryptions. Thus, $d+2$ encryptions and $k+2$ decryptions are required to be done by the server and $k+1$ encryptions for the client, for each entry of the matrix. Since the decryption takes more time than the encryption in the Paillier cryptosystem, the server will perform more of the computational work in the protocol. The experimental evaluation in Section 7 validates the analysis that we have here.

5.3 Security Analysis

Assume an $m \times n$ matrix M is used for computing the distance in dynamic programming. When computing one entry of the matrix, the client sends a set of values in encryption: $a + r_{min}$, $b + r_{min}$, $c + r_{min}$, $x_2 + r_2$, ..., $x_k + r_k$, where x_i is randomly chosen from a, b, or c, and let $k = 2^\alpha$. Then the client generates $2^\alpha + 2$ ciphertexts as new candidates.

During the execution of our protocol, the client receives all the values encrypted and the secret key is kept at the server side. Furthermore, no information about the optimal path is leaked to the client. Therefore, the client can learn nothing about the time series Y of the server.

On the other hand, the server needs to decrypt some values and this will leak some information about the matrix and therefore the time series X. Next, we discuss some simple attacks. For filling each entry in the matrix, the server receives $2^\alpha + 2$ values. Thus, after completing the whole matrix, the server has received $(2^\alpha + 2) * mn$ values. The server knows that each value it received is a summation of one entry from the matrix and one random value. There are total mn entries in the matrix and $2^\alpha * mn$ random values introduced by the client. In order for the server to reconstruct the plaintexts of M, it can create a system of linear equations. The server can create $(2^\alpha + 2) * mn$ linear equations with $(2^\alpha + 1) * mn$ total variables. However, since the values are sent to the server permuted, to find the solution set for this linear equations, the server needs to find out the correspondence between the received values and the equations. More specifically, the server needs to find out some permutation on $(2^\alpha + 2) * mn$ values and a solution set for the linear equations system. It has been shown that a general problem, solving permuted linear equation systems is NP-hard [3].

Even if it is hard for the server to learn exactly all the values generated from the client, the server can still learn something. Consider how much information the server learns during each round. If the gaps between random values are too large or the range of random values are too small, the original values may not be hidden properly. If the gaps between random values are too large, the 3 smallest values maybe the $a + r_{min}$, $b + r_{min}$, $c + r_{min}$. Even if the server cannot guess the random value, the difference among a, b and c can help server reconstruct part of the matrix. That's also why we have to choose a set of random numbers rather than only one random number. If the range of random values are too small, there will be big gaps between $a + r_{min}$, $b + r_{min}$, $c + r_{min}$. Then it is easy to infer which values are $a + r_{min}$, $b + r_{min}$, $c + r_{min}$ too.

To prevent these problems we do the following. Suppose $a, b, c \in (2^\beta, 2^{\beta+1}]$ and the randomness $r_i \in (2^\gamma, 2^{\gamma+1}]$, where $\beta + \gamma < |P|$ and $|P|$ is the number of bits for plaintexts to avoid overflow. Assume that the random values are generated uniformly from the specified range. There are 2^α random values from the random set. The average length of gaps between random values is $2^\gamma/2^\alpha$. If $2^\gamma/2^\alpha < 2^\beta$, the gaps between random values are smaller than the original values. Also, if $2^\gamma > 2^\beta$, the range of random values are larger than the original values. Hence, we require $0 < \gamma - \beta < \alpha$ to allow the decrypted ciphertexts to be hidden in an appropriate range. Even if $k = 64$, the performance is still acceptable. But in this case, $\alpha = 6$, which means that the random values can have 6 more bits than the original values and the original values can still be hidden.

Furthermore, the server can retrieve useful information if it can get the minimal random value or get the relationship between original values. When the random values hide original values successfully, the probability for the server to guess the minimal random value correctly is $1/2^\gamma$ which can be negligible. In addition, the probability that the server can pick up $a + r_{min}$, $b + r_{min}$ and $c + r_{min}$ from $k+2$ values correctly is $\frac{2}{k(k+1)}$ which is also negligible. Hence, the probability for the server to retrieve information successfully during a single round is negligible.

In our discussion so far, we present some specific attacks and how to address them. Although with careful implementation of the protocol and tuning of the parameters we may achieve provable limited leakage, the protocol is not completely secure since it may leak information to the server. Then, in the next section we provide a quantifying method to characterize the amount of information leaked to the server using the notion of information entropy.

5.4 Information Entropy Preservation

In this section, we use entropy to evaluate how much information is exposed to the server.

Let the density function for $x_i \in \{a, b, c\}$ be $f_X(p)$ and let the density function for the random value r_i be $f_R(p)$. The value that is computed and sent to the server from the client is the summation $s_i = x_i + r_i$. Then, the density function for s_i is $f_S(q)$:

$$f_S(q) = \int_{-\infty}^{\infty} f_X(q-p) f_R(p) dp \qquad (6)$$

According to equation (6), the distribution of summation s_i depends on the distributions of original value x_i and random value r_i. Assume that the random values r_i and $\{a, b, c\}$ are in the same range $[\Gamma, 2\Gamma - 1]$. Then the sum $s_i = x_i + r_i$ is in the range $[2\Gamma, 4\Gamma - 2]$. Also assume that r_i and $\{a, b, c\}$ are discrete and uniformly distributed in the range. Then, the density function of summation s_i is:

when $q \in [2\Gamma, 3\Gamma - 1]$,

$$f_S(q) = (q - 2\Gamma + 1)/\Gamma^2 \qquad (7)$$

when $q \in [3\Gamma, 4\Gamma - 2]$,

$$f_S(q) = (4\Gamma - 1 - q)/\Gamma^2 \qquad (8)$$

For the protocol to be completely secure and leak no information to the server, the s_i should be uniformly distributed in $[2\Gamma, 4\Gamma - 2]$. In that case, the entropy is $\log(2\Gamma - 1)$. While in our protocol, s_i is distributed as in formulas (7) and (8), and the Shannon entropy of the summation is:

$$H(S) = -\sum p(q) \log p(q) > \log(2\Gamma - 1)/2 \qquad (9)$$

Y_j	$Enc(a)$	$Enc(\ max\{\ \delta^2_{Eu}(X_i, Y_j)\ ,\ \min\{a, b, c\}\ \}\ )$
Y_{j-1}	$Enc(b)$	$Enc(c)$
	X_{i-1}	X_i

Figure 4: Filling One Cell for DFD

Hence our protocol can preserve more than half (1/2) of the information entropy. For example, if there exists a secure protocol that has 256 bits of entropy, our protocol keeps at least 128 bits of them. By using min-entropy [31] to evaluate information entropy, exact 1/2 information entropy is being preserved in that case. Notice also that the formulas can be used for any data distribution $f_X(p)$, if this is known.

5.5 Hiding the Optimal Path

In this section we analyze how our protocol hides the path of optimal couplings of the matrix.

Recall that in phase 2 we need to do a privacy preserving comparison of three ciphertexts. The client sends a set of encryption of values to the server and the server returns the encryption of the minimal value. Let the minimal plaintext from the set of encryption of values be m and ciphertexts of m encrypted by the client is denoted as $Enc(m)_C$. Let the encryption of m from the server side be $Enc(m)_S$. If $Enc(m)_S = Enc(m)_C$, then the client can learn which value is the minimal one (index of the set of values). However this is not acceptable, because if the client knows the index of the minimal value, the client learns the optimal path of couplings which is also an important part of private information and can be used to retrieve the other party's original data.

To hide the optimal path, the server should re-encrypt the minimal value rather than reusing the values from the client side. Namely, after the server receives and decrypts all ciphertexts generated by the client, the server can get the minimal plaintext m, and the server should re-encrypt this minimal plaintext and send the new encryption back to the client. In this case, the probability of $Enc(m)_S = Enc(m)_C$ is negligible due to the probabilistic encryption. Then, although the minimal value m from the server must be one of $a + r_{min}$, $b + r_{min}$ and $c + r_{min}$ in phase 2 of our protocol, $Enc(m - r_{min})$ is different from any of $Enc(a)$, $Enc(b)$ or $Enc(c)$. Therefore the client acquire an encryption of the minimal value from the server which is different from any of ciphertexts that the client generates. Thus the computation can continue while the optimal path is hidden.

6. PRIVACY PRESERVING PROTOCOL FOR DFD

Here we present a privacy preserving protocol for computing Discrete Fréchet Distance. The difference between the Discrete Fréchet Distance and Dynamic Time Warping is that we need to calculate the maximum rather than the summation. However this increases the cost of our protocol as we will see soon. The process of filling one entry for Discrete Fréchet Distance is shown as Figure 4.

Since the entry cannot be computed with an addition to other ciphertexts, the client cannot use homomorphic addition to calculate the ciphertext for this entry. In the protocol for Dynamic Time Warping, if the client has the encryption of the square of Euclidean distance $Enc(\delta^2_{Eu})$ and the encryption of the minimum $Enc(min)$, then the client can calculate $Enc(\delta^2_{Eu} + min)$ using homomorphic addition. However, for the Discrete Fréchet Distance, the client needs to communicate with the server again to get the larger value between δ^2_{Eu} and min which will definitely increase the communication and computation costs. Thus, computing the Discrete Fréchet Distance needs 3 phases instead of 2.

Algorithm 2: Discrete Fréchet Distance

```
Input: Sequence X[m], Sequence Y[n]
Data: Matrix M[m][n]
Output: Value M[m][n]
M[i][1] = $\delta^2_{Eu}$(X[1], Y[1]);
for $i = 2 : m$ do
    M[i][1] = max ($\delta^2_{Eu}$(X[i], Y[1]), M[i-1][1]);
for $j = 2 : n$ do
    M[1][j] = max ($\delta^2_{Eu}$(X[1], Y[j]), M[1][j-1]);
for $i = 2 : m$ do
    for $j = 2 : n$ do
        cost = $\delta^2_{Eu}$(X[i], Y[j]);
        M[i][j] = max (cost,
                min(M[i-1][j-1], M[i-1][j], M[i][j-1]) );
```

The phase 3 of the protocol for the Discrete Fréchet Distance is also an one round communication protocol which is similar to the phase 2 of the protocol for getting the minimal values. To prevent the server from learning the square of the Euclidean distance and the minimal value, the client still needs to add random values to generate new candidates for this phase. However, in this case, the server should return the maximal one. Then, the client should generate a maximal random value as r_{max} and many other random values which are all smaller than r_{max}. Assume the client generates a family of random values $R' = \{\ r_1\ , r_2\ , r_3 \cdots\ , r_k \mid r_i < r_1\ , \forall\ i > 1\ \}$. The client can generate a set of new inputs including $Enc(\delta^2_{Eu} + r_{max})$, $Enc(min + r_{max})$,$Enc(x_2 + r_2)$ $\ldots$ $Enc(x_k + r_k)$, where x_i is either δ^2_{Eu} or min. The setting is similar to the protocol of comparison in Section 5.

The complementary phase 3 protocol for Discrete Fréchet Distance is described next:

1. The Client
   - (a) Has two ciphertexts: $Enc(\delta^2_{Eu})$ and $Enc(min)$
   - (b) Generate a set of random values $R' = \{\ r_1\ , r_2\ , r_3 \cdots\ , r_k \mid r_i < r_1\ , \forall\ i > 1\ \}$
   - (c) Generate minimal candidates $Enc(\delta^2_{Eu} + r_{max})$ and $Enc(min + r_{max})$
   - (d) Generate other $k - 1$ candidates: For $i \in [2, k]$, generate $Enc(x_i + r_i)$, where $Enc(x_i)$ is randomly chosen from $Enc(\delta^2_{Eu})$ and $Enc(min)$.
   - (e) Send all $k + 1$ candidates to the server
2. The Server
   - (a) Decrypt all inputs
   - (b) Get the maximal plaintext as M
   - (c) Send $Enc(M)$ to the client
3. The Client

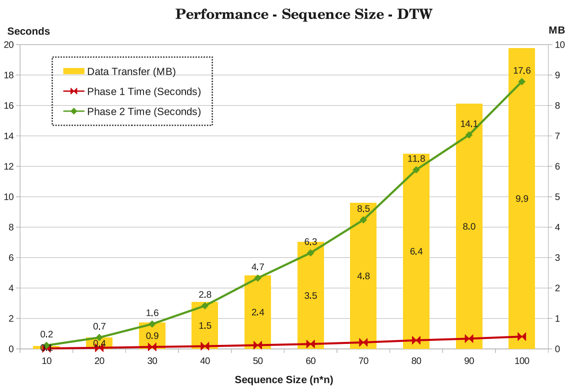


Figure 5: Performance vs Sequence Size

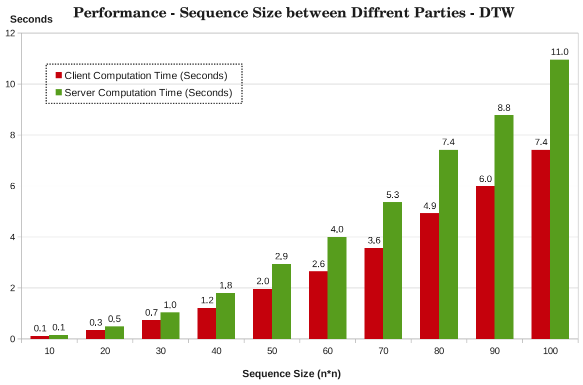


Figure 6: Performance vs Sequence Size for the Different Parties (Client and Server)

(a) Calculate $Enc(M - r_{max})$ as the encryption of maximal one.

The cost of phase 3 in the protocol for the Discrete Fréchet Distance is comparable to phase 2, the protocol of comparison in Section 5. So the protocol to compute Discrete Fréchet Distance is less efficient than the protocol to compute Dynamic Time Warping. As shown in the experiments, phase 2 costs much more time than phase 1. Since phase 3 is comparable to phase 2, the protocol for Discrete Fréchet Distance takes almost twice the time than the protocol for Dynamic Time Warping. Similarly, the protocol for Discrete Fréchet Distance transfers almost two times more data than the protocol for Dynamic Time Warping.

The security attacks and analysis of the protocol for Discrete Fréchet Distance are similar to the security analysis of the protocol for Dynamic Time Warping.

7. EXPERIMENTS

In this section we present an experimental evaluation of our protocols. The goal of this evaluation is to show that our approach can be efficient and practical. We have implemented our protocols in Java and run our experiments on a machine with 2.53GHz CPU and 3GB of RAM.

The server and the client communicate using a socket, and there is no shared information nor third party in our experiments. The security parameter for the Paillier cryptosystem is set to 64 bits which is enough for encrypting the values that we have in our experiments. In the experiments, we use both real world data and synthetic data. The real world time series are derived from the UCR ECG data [1]. In the following experiments we normalized ECG data to positive integer values.

We are not aware of any other implementations of protocols that compute securely time series distance functions. As we mentioned before, methods that are based on garbled circuits can be used to compute the Edit Distance [13, 18], but will be very inefficient to be used for computing time series distances, like DTW. On the other hand, Atallah et al's paper [2] could be used to compute the DTW distance. However, as discussed in Section 2, the protocol proposed by Atallah et al needs to call many times Yao's protocols. According to Fairplay [24], which is one of the best two-party computation practical systems, it takes 1.25 seconds to run Yao's protocol using fast communication, and 4 seconds when communication is slow. Therefore, when the size of sequences is 100 and the dimensionality of each element is 1, Atallah et al's protocol needs at least 37000 seconds for executing Yao's protocols alone. This is just an estimate based on the number of times the protocol calls Yao's protocol. We are not aware of any implementation of Atallah et al's protocol, but the estimated time that we report here is extremely optimistic since we do not account for other costs of the protocol (for example, computation of Euclidean distances.) On the other hand, our protocols take tens of seconds as we will show next. Therefore, we only evaluate the implementation of our protocols next.

[1] `http://www.cs.ucr.edu/~eamonn/discords/`

Our protocols include only two parameters: the dimension d of single element in each sequence and the size of random set k. We perform a number of experiments to validate the efficiency and potential practicality of our protocols.

7.1 Performance vs Sequence Size

To investigate the relationship between performance and sequence size, we segment the ECG sequences with different lengths that range from 10 to 100. The distance computation is between two time series with the same length. The dimensionality of a single element d is 1 and the size of random set k is set to 10 in these experiments.

As shown in Figure 5, since the time complexity of dynamic programming is $O(n^2)$, the total time and data transferred are both quadratic to the sequence sizes. The phase 2 of the protocol for Dynamic Time Warping costs much more running time than the phase 1 when the dimension is low.

The phase 2 of the protocols include encryptions by the client and decryptions by the server which are the most expensive parts in the protocol. The client encrypts at least $k + 2$ random values in phase 2 and the server decrypts at least $k+2$ ciphertexts. Figure 6 shows the computation time for the client and server respectively. Since decryption costs more time than encryption, the server has higher computational cost.

The protocol for Discrete Fréchet Distance (DFD) has similar relationships between running time and sequence sizes. Since the protocol for Discrete Fréchet Distance needs 3 phases rather than 2 for Dynamic Time Warping, the time of computing Discrete Fréchet Distance is higher than Dynamic Time Warping as shown in Figure 7. From Figure 8,

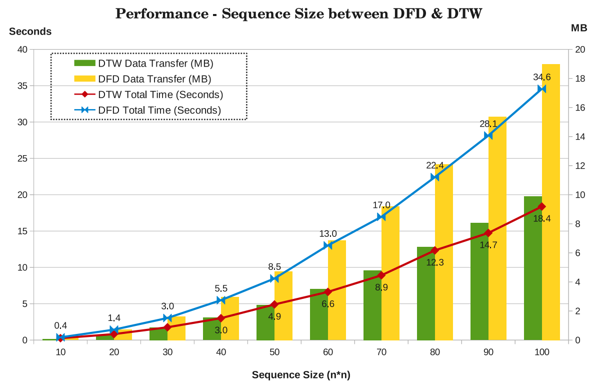


Figure 7: Performance between DTW and DFD

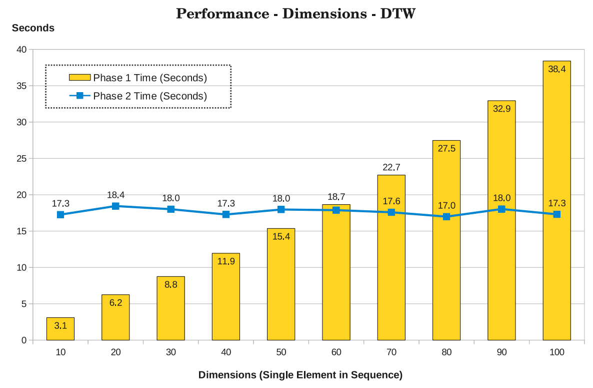


Figure 9: Performance vs Dimensionality

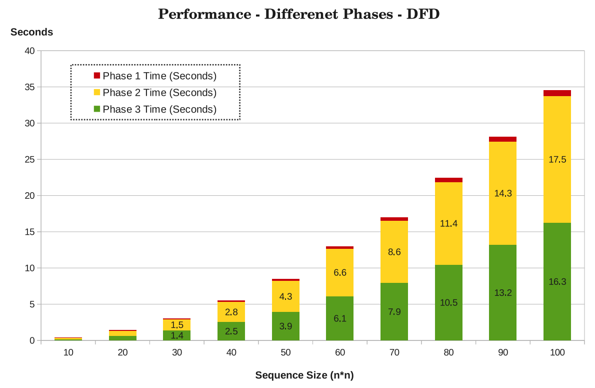


Figure 8: Performance by Phases for DFD

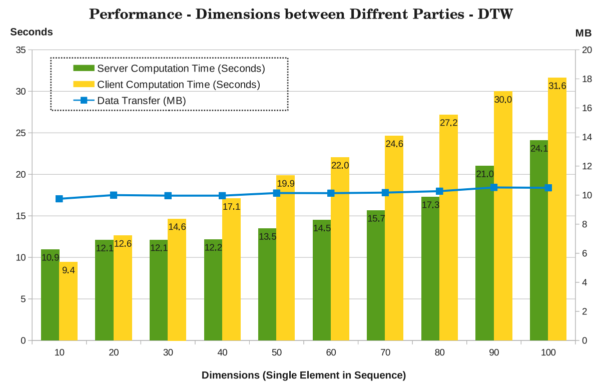


Figure 10: Performance vs Dimensionality between different parties

we can see that the phase 3 costs almost the same as the phase 2. And both phase 3 and phase 2 are much more expensive than phase 1. So the time for computing the Discrete Fréchet Distance is nearly twice of the time for Dynamic Time Warping.

7.2 Performance vs Element Dimensionality

To investigate the relationship between the performance and the dimensionality d of the elements in each sequence, we generated synthetic sequences with dimensionalities from 10 to 100 because when the dimensionality d is low, the time of phase 1 is negligible. The values of each vector are random values between 1 and 100. However, as the dimensionality increases, the cost of phase 1 also increases as expected. The size of sequences is set as 100 and the size of random sets k is 10. Since phase 2 uses entries from the matrix only and the square of Euclidean distances are computed in phase 1, the dimensionality of single element in sequences affects phase 1 only.

In Figure 9 we show the results of the experiments where we compare the running time of phase 1 and phase 2 for computing DTW. It is clear that even if the dimensionality is 10, phase 2 still dominates the whole computing time. When the dimensionality is increased to 60, the time for phase 1 is comparable to phase 2. The time for phase 1 is linear to the dimensionality as analyzed in Section 5. Also, even if the dimension is up to 100, phase 1 costs only 2 times more of phase 2.

Figure 10 shows the time consumed by server and client. The computation time for the client increases faster than the computation time for the server as expected. Since the communication cost of phase 2 dominates the whole communication cost and is independent of the dimensionality of the time series elements, we can see the communication cost does not change much when dimensionality increases.

Therefore, we could claim that the performance of our protocol is nearly linear to the dimensionalities of time series. Only if the dimensionalities of time series are very high, the client may have more computation workloads than the server.

7.3 Performance vs Size of Random Set

To investigate the relationship between performance and the size of the random set, we use time series from the ECG datasets of size 100, where the dimensionality d is 1. We change the size of the random sets k from 10 to 50. Since phase 1 does not use any random set, the sizes of random sets affect only phase 2 and 3. Here we report the results for phase 2.

As shown in Figure 11, since phase 2 dominates running time of the protocol, increasing the size of the random set makes the cost of phase 2 to increase. Notice that the time for phase 1 is negligible in this experiment. Both the running

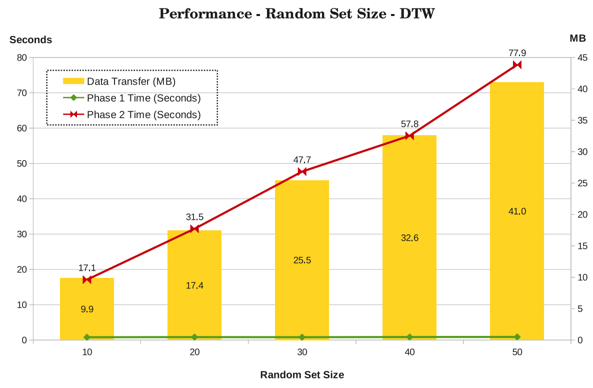


Figure 11: Performance vs Random Set

time performance as well as the communication cost grow linearly with k as expected.

Thus, the experiments show that the performance is dominated by phase 2 and phase 3 of the protocols since encryption and decryption are time consuming parts. The performance is linear to the parameters of the protocols, i.e. the dimensionality of each element in sequences d and the size of random set k. Also, the protocols are not very sensitive to dimensionality, which means that our protocols are suitable for high dimensional vector sequences. Finally, notice that our protocols are at least three orders of magnitude faster than the protocols proposed in [2].

8. CONCLUSIONS

This paper proposes privacy preserving protocols to compute two distance functions: Dynamic Time Warping and Discrete Fréchet Distances for time series data. To our best knowledge, this is the first practical work that addresses this problem explicitly. The major objective of the protocols is to protect the privacy of input data from the client and the server. The two parties can calculate Dynamic Time Warping and Discrete Fréchet Distances together without revealing their own data to each other.

The protocols can be used for many time series database applications such as similarity sequences retrieval and nearest sequences query. The protocols combine partial homomorphic encryption and random offsets which make the protocols practical and preserve enough information entropy. Our protocols can be easily extended to any privacy preserving distance computation using dynamic programming when data are kept by two different parties. A detailed analysis of the protocols is provided and an experimental evaluation validates that the protocols are scalable and potentially practical.

9. ACKNOWLEDGMENTS

This work has been partially supported by NSF grants CNS-1012910 and III-1320542.

10. REFERENCES

[1] R. Agrawal, A. V. Evfimievski, and R. Srikant. Information sharing across private databases. In *SIGMOD Conference*, pages 86–97, 2003.

[2] M. J. Atallah, F. Kerschbaum, and W. Du. Secure and private sequence comparisons. In *WPES*, pages 39–44, 2003.

[3] P. Butkovič. Permuted linear system problem and permuted eigenvector problem are np-complete. Technical Report preprint 2008/28, University of Birmingham, 2008.

[4] L. Chen and R. T. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[5] R. Chen, N. Mohammed, B. C. M. Fung, B. C. Desai, and L. Xiong. Publishing set-valued data via differential privacy. *PVLDB*, 4(11), 2011.

[6] G. Cormode, C. M. Procopiuc, D. Srivastava, E. Shen, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012.

[7] T. Eiter and H. Mannila. Computing discrete fréchet distance. Technical Report CD-TR 94/64, Technische UniversitÃd'Wien, 1994.

[8] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *Privacy Enhancing Technologies*, pages 235–253, 2009.

[9] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 3(1):1021–1032, 2010.

[10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider, and I. Wehrenberg. Tasty: tool for automating secure two-party computations. In *CCS*, pages 451–462, 2010.

[11] Y. Hong, J. Vaidya, H. Lu, and M. Wu. Differentially private search log sanitization with optimal output utility. In *EDBT*, pages 50–61, 2012.

[12] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *ICDE*, pages 601–612, 2011.

[13] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security Symposium*, 2011.

[14] Y. Huang, L. Malka, D. Evans, and J. Katz. Efficient privacy-preserving biometric identification. In *NDSS*, 2011.

[15] A. Inan, M. Kantarcioglu, E. Bertino, and M. Scannapieco. A hybrid approach to private record linkage. In *ICDE*, pages 496–505, 2008.

[16] A. Inan, M. Kantarcioglu, G. Ghinita, and E. Bertino. Private record matching using differential privacy. In *EDBT*, pages 123–134, 2010.

[17] A. Jarrous and B. Pinkas. Secure hamming distance based computation and its applications. In *ACNS*, pages 107–124, 2009.

[18] S. Jha, L. Kruger, and V. Shmatikov. Towards practical privacy for genomic computation. In *IEEE Symposium on Security and Privacy*, pages 216–230, 2008.

[19] M. Kantarcioglu, A. Inan, W. Jiang, and B. Malin. Formal anonymity models for efficient privacy-preserving joins. *Data Knowl. Eng.*, 68(11):1206–1223, 2009.

[20] E. J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[21] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider.

Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, pages 1–20, 2009.

[22] V. Kolesnikov and T. Schneider. Improved garbled circuit: Free xor gates and applications. In *ICALP (2)*, pages 486–498, 2008.

[23] C. Li and G. Miklau. An adaptive mechanism for accurate query answering under differential privacy. *PVLDB*, 5(6):514–525, 2012.

[24] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In *USENIX Security Symposium*, 2004.

[25] M. Murugesan, W. Jiang, C. Clifton, L. Si, and J. Vaidya. Efficient privacy-preserving similar document detection. *VLDB J.*, 19(4):457–475, 2010.

[26] M. Osadchy, B. Pinkas, A. Jarrous, and B. Moskovich. Scifi - a system for secure face identification. In *IEEE Symposium on Security and Privacy*, pages 239–254, 2010.

[27] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[28] S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest neighbor search with strong location privacy. *PVLDB*, 3(1):619–629, 2010.

[29] R. Paulet, M. G. Kaosar, X. Yi, and E. Bertino. Privacy-preserving and content-protecting location based queries. In *ICDE*, pages 44–53, 2012.

[30] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD Conference*, pages 735–746, 2010.

[31] L. Reyzin. Some notions of entropy for cryptography - (invited talk). In *ICITS*, pages 138–142, 2011.

[32] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-preserving aggregation of time-series data. In *NDSS*, 2011.

[33] M. Terrovitis, J. Liagouris, N. Mamoulis, and S. Skiadopoulos. Privacy preservation by disassociation. *PVLDB*, 5(10):944–955, 2012.

[34] J. Vaidya and C. Clifton. Privacy-preserving outlier detection. In *ICDM*, pages 233–240, 2004.

[35] X. Wang, A. Mueen, H. Ding, G. Trajcevski, P. Scheuermann, and E. J. Keogh. Experimental comparison of representation methods and distance measures for time series data. *Data Min. Knowl. Discov.*, 26(2):275–309, 2013.

[36] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[37] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu. Differentially private histogram publication. In *ICDE*, pages 32–43, 2012.

[38] B. Yang, H. Nakagawa, I. Sato, and J. Sakuma. Collusion-resistant privacy-preserving data mining. In *KDD*, pages 483–492, 2010.

[39] A. C.-C. Yao. Protocols for secure computations. In *FOCS*, pages 160–164, 1982.