A Framework for Secure Speech Recognition

Paris Smaragdis, Madhusudana Shashanka

TR2007-057 April 01, 2007

Abstract

In this paper, we present a process which enables privacy-preserving speech recognition transactions between two parties. We assume one party with private speech data and one party with private speech recognition models. Our goal is to enable these parties to perform a speech recognition task using their data, but without exposing their private information to each other. We will demonstrate how using secure multiparty computation principles we can construct a system where this transaction is possible, and how this system is computationally and securely correct. The protocols described herein can be used to construct a rudimentary speech recognition systm and can easily be extended for arbitrary audio and speech processing.

IEEE Transactions on Audio, Speech and Language Processing

A Framework for Secure Speech Recognition

Paris Smaragdis, *Senior Member, IEEE*, and Madhusudana Shashanka, *Student Member, IEEE*

***Abstract*—In this paper, we present a process which enables privacy-preserving speech recognition transactions between two parties. We assume one party with private speech data and one party with private speech recognition models. Our goal is to enable these parties to perform a speech recognition task using their data, but without exposing their private information to each other. We will demonstrate how using secure multiparty computation principles we can construct a system where this transaction is possible, and how this system is computationally and securely correct. The protocols described herein can be used to construct a rudimentary speech recognition system and can easily be extended for arbitrary audio and speech processing.**

***Index Terms*—Gaussian mixture models, hidden Markov model (HMM), secure multiparty computation (SMC), speech recognition.**

I. Introduction

THE WIDESPREAD use of networking technology today has spawned an industry of online services. Business models based on client–server interactions are common place and increasingly more prominent. Speech recognition could easily be a part of this trend where servers can provide speech recognition services for remote clients. The private nature of speech data however is a stumbling block for such a development. Individuals, corporations, and governments are understandably reluctant to send private speech data through a network to another party that cannot be trusted. In this paper, we address this issue and show how such a cooperative model can be realized with no privacy leaks from any involved party.

We will specifically focus on the realization of a hidden Markov model (HMM) in a secure framework that allows training and classification between multiple parties, some owning speech data, and some owning HMM models for speech recognition. Our formulation is shaped in such a way that the providers of the speech will not have to share any information about their data, and the providers of the HMM will not share any information on their model. After evaluation, the results will only be revealed to the parties that have provided the data, and not to the parties that provide the HMM models, thereby providing privacy at both the data and the semantic levels.

Manuscript received June 27, 2006; revised January 30, 2007. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Simon King.

P. Smaragdis is with the Mitsubishi Electric Research Laboratories, Cambridge, MA 02139 USA (e-mail: paris@merl.com).

M. Shashanka is with the Department of Cognitive and Neural Systems and the Hearing Research Center, Boston University, Boston, MA 02215 USA (e-mail: shashanka@cns.bu.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASL.2007.894526

We will demonstrate the use of this idea using two scenarios. One scenario will involve the training of HMMs from data provided by multiple parties, and the other will deal with evaluating already trained HMMs on data from another party. The utility of these scenarios in collaborative speech recognition projects is easy to see. The first scenario can enable model training on multiple speech databases without requiring the disclosure of actual speech data, whereas the other scenario can enable speech recognition as a service model to remote customers who wish to maintain privacy on their data and their transcription from both the service provider and malicious network intruders.

Strange as these constraints might seem, they can be satisfied using secure multiparty computation (SMC) protocols. SMC is a field of cryptography that provides means to perform arbitrary computations between multiple parties who are concerned with protecting their data. The field of SMC originated from the work of Yao [1] who gave a solution to the millionaire problem: two millionaires want to find which one has a larger fortune, without revealing any specific numbers to each other. Recently, this concept has been employed for simple machine learning tasks such as multiple parties performing k-means [2], computation of means and related statistics from distributed databases [3], and rudimentary computer vision applications [4]. See [5] for a detailed treatment of the topic. In this paper, we present an SMC formulation of training and evaluating HMMs as applied on speech data. To our knowledge, this is the first application of SMC concepts for privacy-constrained speech technology. We will consider HMMs where the observations are modeled by mixtures of Gaussians as is common in speech recognition applications. The main contributions in this paper are the creation of privacy-preserving protocols that support Gaussian mixture model and HMM learning and evaluation, as well as a secure method to combine these protocols so as to ensure data privacy.

The remainder of this paper is organized as follows. In Section II, we formally introduce the problem at hand and in Section III we introduce the secure computation primitives that are employed for this task. Using these primitives, we deal with the problem of secure classification using Gaussian mixtures in Section IV, and in Section V we extend that to present protocols for secure HMMs. We provide a brief discussion about security and efficiency of our protocols in Section VI. Finally, in Section VII, we provide conclusions and directions for future extensions.

II. Problem Formulation

Hidden Markov models find use in a wide range of applications and have been successfully used in speech recognition. There are three fundamental problems for HMM design, namely: the evaluation of the probability (or likelihood) of a sequence of observations given a specific HMM; the determination of a best sequence of model states; and the adjustment of model parameters so as to best account for the observed signal. The first problem is one of scoring how well a given model matches a given observation sequence. The second problem is one in which we attempt to uncover the hidden part of the model. The third problem is the problem of "training." Algorithms for the above three problems are well known and described in detail in [6].

We will consider these problems using a transaction between two parties named Alice and Bob. Suppose Bob has a trained HMM with all the model parameters learned. Let the HMM be characterized as follows.

- N states $\{S_1, \ldots, S_N\}$. Let the state at time t be q_t.
- The state transition probability distribution $\mathbf{A} = \{a_{ij}\}$ where

$$a_{ij} = P[q_{t+1} = S_j | q_t = S_i], \qquad 1 \leq i,j \leq N. \tag{1}$$

- The observation symbol probability distribution in state j given by a mixture of Gaussians

$$b_j(\mathbf{x}) = \sum_{m=1}^{M} c_{jm} \mathcal{N}(\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm}), \qquad 1 \leq j \leq N \tag{2}$$

  where $\mathbf{x}$ is the variable, c_{jm} is the mixture coefficient for the mth mixture in state j, and $\mathcal{N}(\boldsymbol{\mu}_{jm}, \boldsymbol{\Sigma}_{jm})$ is a Gaussian with mean vector $\boldsymbol{\mu}_{jm}$ and covariance matrix $\boldsymbol{\Sigma}_{jm}$.
- The initial state distribution $\pi = \{\pi_i\}$ where

$$\pi_i = P[q_1 = S_i] \qquad 1 \leq i \leq N. \tag{3}$$

We use λ to denote the entire parameter set of the model.

Consider the first two problems where Bob has a trained HMM with all the model parameters learned. Let Alice have an observation sequence $\mathbf{X} = \mathbf{x}_1\mathbf{x}_2, \ldots, \mathbf{x}_T$. We will show how Alice can securely compute $P(\mathbf{X}|\lambda)$, the probability of the observation sequence given the model, using the *forward–backward* procedure. We will also show how one can securely learn the best sequence of model states using the *viterbi* algorithm.

Once there is a secure way of computing likelihoods, it is easy to see how it can be extended to applications like speech recognition. Suppose Bob has trained several HMMs which characterize various speech sounds. Each HMM will correspond to a speech recognition unit. Let Alice's observation vector correspond to a small snippet of speech sound (we assume that Alice knows the features that Bob has used to train his HMMs on and has represented her sound sample in terms of those features). We then show how Alice and Bob can obtain additive shares of the likelihood of Alice's observation sequence for every speech recognition unit of Bob and use them to find out the unit that corresponds to Alice's sound snippet.

Now consider the third problem of training. The problem of security arises when Bob wants to train an HMM (or do data mining in general) on combined data from private databases owned by Alice and Charlie. We show how Bob can securely reestimate parameters of his HMM without gaining knowledge about the private data. The implicit assumption, of course, is that Alice and Charlie are willing to let Bob learn about distributions of their data.

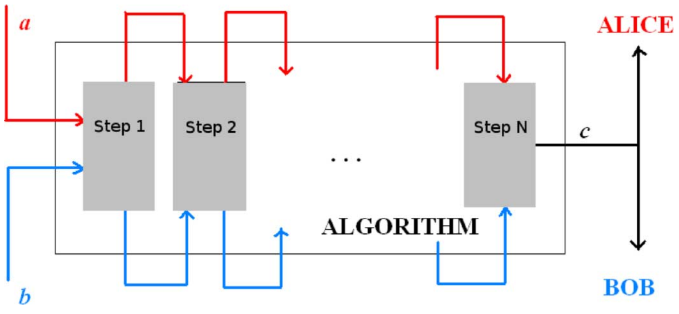

Fig. 1. Implementing an algorithm securely. The algorithm takes in private inputs $\mathbf{a}$ and $\mathbf{b}$. Algorithm is split into steps that can be implemented as secure primitives (shown as gray boxes). Intermediate results are distributed as random additive shares and feed into the following steps. Final result $\mathbf{c}$ is obtained by both parties (or the designated receiver).

III. Secure Two-Party Computations—Background

The speech-recognition example that we will present is a specific example of a *secure two-party computation*. Consider the case where Alice and Bob have private data $\mathbf{a}$ and $\mathbf{b}$, respectively, and they want to compute the result of a function $f(\mathbf{a}, \mathbf{b})$. Consider a trusted third-party who can take the private data, compute the result $\mathbf{c} = f(\mathbf{a}, \mathbf{b})$, and intimate the result to the parties. Any protocol that implements an algorithm to calculate $f(\mathbf{a}, \mathbf{b})$ is said to be *secure* only if it leaks no more information about $\mathbf{a}$ and $\mathbf{b}$ than what one can gain from learning the result $\mathbf{c}$ from the trusted third-party. We assume a *semi-honest* model for the parties where they follow the protocol but could be saving messages and intermediate results to learn more about other's private data. In other words, the parties are *honest but curious* and will follow the agreed-upon protocol but will try to learn as much as possible from the data flow between the parties.[1]

To implement an algorithm securely, we will have to implement each step of the algorithm securely. If one of the steps is insecurely implemented, either party could utilize the information to work their way backwards to gain knowledge about the other's private data. In addition, one must also consider the results of intermediate steps. If such results of intermediate steps are available, there is a possibility that one could also get back to the original private inputs. To prevent this:

- we express every step of the algorithm in terms of a handful of basic operations (henceforth called as *primitives*) for which secure implementations are already known;
- we distribute intermediate results randomly between the two parties such that neither party has access to the entire result. For example, instead of obtaining the result z of a certain step, the parties receive *random additive shares* z_1 and z_2 ($z_1 + z_2 = z$). See Fig. 1 for a schematic illustration.

Secure protocols are often analyzed for correctness, security, and complexity. Correctness is measured by comparing the proposed protocol with the ideal protocol using a third party. If the results are indistinguishable, the protocol is correct (note that

[1]In a *malicious model*, no such assumptions are made about the parties' behavior. If both parties are malicious, security can be enforced by accompanying the protocols with zero-knowledge proofs that protocols are being followed. If only one of the parties is malicious, the other party can use *conditional disclosure of secrets* protocols [7] to make sure he/she receives valid inputs from the malicious party.

one can use secure approximation to an ideal algorithm). All the protocols we present are exact protocols. For security, one needs to show what can and cannot be learned from the data exchange. For complexity, one shows the computational and communication complexity of the secure algorithm. Based on the choice of primitives used and how they are implemented, one can achieve different levels of security and computational/communication efficiency. To evaluate the efficiency of protocols we propose in later sections, we provide measures in terms of efficiency of the primitives instead of absolute measures. Next, we describe the primitives that we use and briefly discuss about their implementations.

A. Secure Inner Products (SIP)

The primitive which we use most often is for computing secure inner products. If Alice has vector $\mathbf{x}$ and Bob has vector $\mathbf{y}$, a secure inner product protocol produces two numbers a and b such that $a+b=\mathbf{x}^T\mathbf{y}$. Alice will get the result a, and Bob will get the result b. To simplify notation, we shall denote a secure inner product computation $\mathbf{x}^T\mathbf{y}$ as $SIP(\mathbf{x},\mathbf{y})$.

Many protocols have been proposed and they can be categorized as cryptographic protocols (e.g., [8], [9]) and algebraic protocols (e.g., [10]–[14]). They provide different levels of security and efficiency. Most of the algebraic protocols leak some information but are more straightforward and efficient than their cryptographic counterparts. The properties and weaknesses of some of these algebraic protocols have been analyzed in detail ([3], [9]). In this paper, we will be using cryptographic protocols as they tend to be more secure. In Appendices I and II, we provide a description of two of the cryptographic protocols we have used. We have also included an algebraic protocol in Appendix III for comparison.

B. Secure Maximum Index (SMAX)

Let Alice have a vector $\mathbf{x}=[x_1,\ldots,x_d]$ and Bob have the vector $\mathbf{y}=[y_1,\ldots,y_d]$; they would like to compute the index of the maximum of $\mathbf{x}+\mathbf{y}=[(x_1+y_1),\ldots,(x_d+y_d)]$. At the end of the protocol, Alice (and/or Bob) will receive the result, but neither party will know the actual value of the maximum. Notice that the same protocol can be used to compute the index of the minimum. We denote this as $j=SMAX(\mathbf{x},\mathbf{y})$.

For this primitive, we use the permute protocol proposed by [15] (see Appendix IV). The protocol enables Alice and Bob to obtain additive shares, $\mathbf{q}$ and $\mathbf{s}$, of a permutation of the vector $\mathbf{x}+\mathbf{y}$, $\pi(\mathbf{x}+\mathbf{y})$, where π is chosen by Alice, and Bob has no knowledge of π. The idea is for Alice to send $\mathbf{q}-r$, where r is a random number chosen by her, to Bob. Bob sends back the index of the maximum element of $\mathbf{q}+\mathbf{s}-r$ to Alice who then computes the real index using the inverse of the permutation π. Neither party learns the value of the maximum element and Bob does not learn the index of the maximum element.

If the security requirements of Alice are more strict, she can encrypt elements of $\mathbf{q}$ using a homomorphic encryption scheme and send them to Bob along with the public key. Bob can make comparisons among the encrypted values of $\pi(\mathbf{x}+\mathbf{y})$ using protocols for Yao's millionaire problem (e.g., [16], [17]) and send back the index of the maximum element.

C. Secure Maximum Value (SVAL)

Let Alice have a vector $\mathbf{x}=[x_1,\ldots,x_d]$ and Bob have the vector $\mathbf{y}=[y_1,\ldots,y_d]$; they would like to compute the value of the maximum element in $\mathbf{z}=\mathbf{x}+\mathbf{y}$. After the protocol, Alice and Bob receive additive shares of the result, a and b, but neither party will know the index of the maximum element. Notice that the same protocol can be used to compute the value of the minimum. Let us denote this as $a+b=SVAL(\mathbf{x},\mathbf{y})$.

For this protocol, we can use the idea presented in [15]. Let us first consider a naive approach. Notice that $z_i \geq z_j \iff (x_i-x_j)\geq(y_j-y_i)$. Alice and Bob can do such pairwise comparisons and mimic any standard maximum finding algorithm to learn the value of the maximum. To perform the comparisons securely, they can use a protocol for Yao's millionaire problem [1].

However, if Alice and Bob follow the above naive approach, both will be able to also find the index of the maximum. Hence, the idea is for Alice and Bob to obtain two vectors whose sum is a random permutation of $\mathbf{z}$. Neither Alice nor Bob should know the permutation. They can then follow the above naive approach on their newly obtained vectors to compute additive shares of the maximum element. See Appendix IV for a description of the permutation protocol.

D. Secure Logsum (SLOG)

This primitive, unlike the other three which we have introduced above, is not a cryptographic primitive. The main reason we introduce it is because it simplifies the presentation of many of the protocols we propose in future sections.

Let Alice have a vector $\mathbf{x}=[x_1,\ldots,x_d]$ and Bob have the vector $\mathbf{y}=[y_1,\ldots,y_d]$ such that $\mathbf{x}+\mathbf{y}=\ln\mathbf{z}=[\ln z_1,\ldots,\ln z_d]$. They would like to compute additive shares q and s such that $q+s=\ln\left(\sum_{i=1}^d z_i\right)$. Let us denote this secure computation as $q+s=SLOG(\mathbf{x},\mathbf{y})$.

One can compute logarithm of a sum from the logarithms of individual terms as follows:

$$\ln\left(\sum_{i=1}^{d} z_i\right)=\ln\left(\sum_{i=1}^{d} e^{x_i+y_i}\right). \tag{4}$$

This suggests the following protocol.

1) Alice and Bob compute the dot product between vectors $e^{\mathbf{x}-q}$ and $e^{\mathbf{y}}$ using $SIP(e^{\mathbf{x}-q},e^{\mathbf{y}})$ where q is a random number chosen by Alice. Let Bob obtain ϕ, the result of the dot product.
2) Notice that Bob has $s=\ln\phi=-q+\ln\left(\sum_{j=1}^d e^{x_j+y_j}\right)$, and Alice has q.

In step 1), Bob receives the entire result of a dot product. However, this does not reveal any information to him about $\mathbf{x}$ due to the presence of the random number q. In no other step does either party receive the complete result of an operation. Thus, the protocol is secure. In terms of efficiency, this primitive is equivalent to using the *SIP* primitive once.

IV. Secure Classification: Gaussian Mixture Models

Alice has a d-component data vector $\mathbf{x}$ and Bob knows multivariate Gaussian distributions of N classes $\omega_i, i=\{1,\ldots,,N\}$

that the vector could belong to. They would like to engage in a protocol that lets Bob classify Alice's data, but neither of them wants to disclose data to the other person. We propose protocols which enable such computations.

The idea is to evaluate the value of the *discriminant function*

$$g_i(\mathbf{x}) = \ln p(\mathbf{x}|\omega_i) + \ln P(\omega_i) \tag{5}$$

for all classes ω_i and assign $\mathbf{x}$ to class ω_i if $g_i(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq i$. Here, $p(\mathbf{x}|\omega_i)$ is the class-conditional probability density function, and $P(\omega_i)$ is the *a priori* probability of class ω_i. We consider two cases where 1) each class is modeled as a single multivariate Gaussian, and 2) each class is modeled as a mixture of Gaussians.

A. Case 1: Single Multivariate Gaussian

We assume that the distribution of data is multivariate Gaussian i.e., $p(\mathbf{x}|\omega_i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$, where $\boldsymbol{\mu}_i$ is the mean vector, and $\boldsymbol{\Sigma}_i$ is the covariance matrix of class ω_i. Hence, the log-likelihood is given by

$$\ln p(\mathbf{x}|\omega_i) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{d}{2}\ln 2\pi - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i|. \tag{6}$$

Ignoring the constant term $(d/2)\ln 2\pi$, we can write (5) as

$$g_i(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_i)^t\boldsymbol{\Sigma}_i^{-1}(\mathbf{x}-\boldsymbol{\mu}_i) - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i). \tag{7}$$

Simplifying, we have

$$g_i(\mathbf{x}) = \mathbf{x}^T\bar{\mathbf{W}}_i\mathbf{x} + \bar{\mathbf{w}}_i^T\mathbf{x} + w_{i0} \tag{8}$$

where

$$\bar{\mathbf{W}}_i = -\frac{1}{2}\boldsymbol{\Sigma}_i^{-1}, \quad \bar{\mathbf{w}}_i = \boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i$$

and

$$w_{i0} = -\frac{1}{2}\boldsymbol{\mu}_i^T\boldsymbol{\Sigma}_i^{-1}\boldsymbol{\mu}_i - \frac{1}{2}\ln|\boldsymbol{\Sigma}_i| + \ln P(\omega_i). \tag{9}$$

Let us create the $(d+1)$-dimensional vectors $\bar{\mathbf{x}}$ and $\mathbf{w}_i$ by appending the value 1 to $\mathbf{x}$ and appending w_{i0} to $\bar{\mathbf{w}}_i$. By changing $\bar{\mathbf{W}}_i$ into a $(d+1)\times(d+1)$ matrix $\mathbf{W}_i$ where the first d components of the last row are zeros and the last column is equal to $\mathbf{w}_i^T$, we can express (8) in a simplified form

$$g_i(\mathbf{x}) = \bar{\mathbf{x}}^T\mathbf{W}_i\bar{\mathbf{x}}.$$

Expressing $\bar{\mathbf{x}}$ as $\mathbf{x}$ for simplicity, we can write the above equation as

$$g_i(\mathbf{x}) = \mathbf{x}^T\mathbf{W}_i\mathbf{x}. \tag{10}$$

Henceforth, we shall use $\mathbf{x}$ to denote a $(d+1)$-dimensional vector with the last component equal to 1 unless otherwise mentioned.

1) Protocol SMG: Single Multivariate Gaussian:

Input: Alice has vector $\mathbf{x}$, Bob has $\mathbf{W}_i$ for $i = 1, 2, \ldots, N$. We express the matrix $\mathbf{W}_i$ as $[\mathbf{W}_i^1\mathbf{W}_i^2, \ldots, \mathbf{W}_i^{d+1}]$, where $\mathbf{W}_i^j$ is the jth column of $\mathbf{W}_i$.

Output: Alice learns I such that $g_I(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq I$. Bob learns nothing about $\mathbf{x}$.

1) For $i = 1, 2, \ldots, N$
   a) For $j = 1, \ldots, d+1$, Alice and Bob perform $SIP(\mathbf{x}, \mathbf{W}_i^j)$ to obtain the vectors $\mathbf{a}_i = [a_i^1, \ldots, a_i^{d+1}]$ and $\mathbf{b}_i = [b_i^1, \ldots, b_i^{d+1}]$, respectively. Alice then computes $\mathbf{a}_i\mathbf{x}$.
   b) Alice and Bob perform $SIP(\mathbf{b}_i, \mathbf{x})$ to obtain q_i and r_i, respectively.
2) Alice has vector $\mathbf{A} = [(\mathbf{a}_1\mathbf{x} + q_1), \ldots, (\mathbf{a}_N\mathbf{x} + q_N)]$, and Bob has vector $\mathbf{B} = [r_1, \ldots, r_N]$.
3) Alice and Bob perform the secure maximum index protocol between the vectors $\mathbf{A}$, and $\mathbf{B}$ and Alice obtains $I = SMAX(\mathbf{A}, \mathbf{B})$.

Correctness: In step 1), $\mathbf{a}_i$ and $\mathbf{b}_i$ are vectors such that $\mathbf{a}_i + \mathbf{b}_i = \mathbf{x}^T\mathbf{W}_i$. Also, $\mathbf{b}_i\mathbf{x} = q_i + r_i$. Hence, $\mathbf{x}^T\mathbf{W}_i\mathbf{x}$ is given by $\mathbf{a}_i\mathbf{x} + q_i + r_i$. I is the value of i for which $\mathbf{x}^T\mathbf{W}_i\mathbf{x}$ is maximum.

Efficiency: For a given $i = I$, the above protocol has $(d+2)$ SIP calls. Hence, it would take $N(d+2)$ SIP calls and one call of *SMAX*.

Security: If Bob gets to know the dot products of d different vectors with $\mathbf{x}$, he can learn $\mathbf{x}$ completely. However, we see that neither Bob nor Alice ever learn the complete result of any dot product. Hence, if the protocols for *SIP* and *SMAX* are secure, the above protocol is secure.

B. Case 2: Mixture of Gaussians

Let us now consider the case where each class is modeled as a mixture of Gaussians. Let the mean vector and covariance matrix of the jth Gaussian in class ω_i be $\boldsymbol{\mu}_{ij}$ and $\boldsymbol{\Sigma}_{ij}$, respectively. Hence, we have $p(\mathbf{x}|\omega_i) = \sum_{j=1}^{J_i}\alpha_{ij}\mathcal{N}(\boldsymbol{\mu}_{ij}, \boldsymbol{\Sigma}_{ij})$, where J_i is the number of Gaussians describing class ω_i, and α_{ij} are the mixture coefficients. The log likelihood for the jth Gaussian in the ith class is given by

$$l_{ij}(\mathbf{x}) = \mathbf{x}^T\bar{\mathbf{W}}_{ij}\mathbf{x} + \bar{\mathbf{w}}_{ij}^T\mathbf{x} + w_{ij} \tag{11}$$

where

$$\bar{\mathbf{W}}_{ij} = -\frac{1}{2}\boldsymbol{\Sigma}_{ij}^{-1}, \qquad \bar{\mathbf{w}}_{ij} = \boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{\mu}_{ij}$$

and

$$w_{ij} = -\frac{1}{2}\boldsymbol{\mu}_{ij}^T\boldsymbol{\Sigma}_{ij}^{-1}\boldsymbol{\mu}_{ij} - \frac{1}{2}\ln|\boldsymbol{\Sigma}_{ij}| + \ln\alpha_{ij}.$$

Expressing $\mathbf{x}$ as a $(d+1)$-dimensional vector and $\bar{\mathbf{W}}_{ij}$, $\bar{\mathbf{w}}_{ij}$, w_{ij} together as the $(d+1)\times(d+1)$ matrix $\mathbf{W}_{ij}$ as done in the previous case, we can simplify (11) as

$$l_{ij}(\mathbf{x}) = \mathbf{x}^T\mathbf{W}_{ij}\mathbf{x}. \tag{12}$$

Hence, the discriminant function for the ith class can be written as

$$g_i(\mathbf{x}) = \text{logsum}\big(l_{i1}(\mathbf{x}), \ldots, l_{iJ_i}(\mathbf{x})\big) + \ln P(\omega_i) = \ln\left(\sum_{j=1}^{J_i} e^{l_{ij}(\mathbf{x})}\right) + \ln P(\omega_i). \tag{13}$$

1) Protocol MOG: Mixture of Gaussians:

Input: Alice has vector $\mathbf{x}$, Bob has $\mathbf{W}_{ij}$ and $P(\omega_i)$ for $i = 1, 2, \ldots, N$, and $j = 1, 2, \ldots, J_i$.

Output: Alice learns I such that $g_I(\mathbf{x}) > g_j(\mathbf{x})$ for all $j \neq I$. Bob learns nothing about $\mathbf{x}$.

1) For $i = 1, 2, \ldots, N$.
   a) Alice and Bob engage in steps 1) and 2) of Protocol SMG for the J_i Gaussians in the ith mixture to obtain vectors $\mathbf{A}_i = [A_{i1}, \ldots, A_{iJ_i}]$ and $\mathbf{B}_i = [B_{i1}, \ldots, B_{iJ_i}]$. Notice that $A_{ij} + B_{ij} = l_{ij}(\mathbf{x})$.
   b) Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{A}_i$ and $\mathbf{B}_i$ to obtain u_i and z_i, i.e., $u_i + z_i = SLOG(\mathbf{A}_i, \mathbf{B}_i)$.
2) Bob computes the vector $\mathbf{v} = [v_1, \ldots, v_N]$ where $v_i = z_i + \ln P(\omega_i)$. Alice forms the vector $\mathbf{u} = [u_1, \ldots, u_N]$.
3) Alice and Bob perform the secure maximum index protocol between vectors $\mathbf{u}$, and $\mathbf{v}$ and Alice obtains $I = SMAX(\mathbf{u}, \mathbf{v})$.

Correctness: If one follows the protocol carefully, it is easy to see that $u_i + v_i$ is equal to $g_i(\mathbf{x})$.

Efficiency: For a given i, there are $(J_i(d+2)+1)$ SIP calls. Hence, in all, there are $(d+2)\sum_{i=1}^{N} J_i + N$ SIP calls and 1 *SMAX* call.

Security: If Protocol SMG and the protocols for *SIP*, *SVAL* and *SMAX* are secure, the above protocol is secure.

In case Alice and Bob want to compute additive shares of the likelihood instead of the class label, they can use the *SVAL* protocol instead of *SMAX* in the last step.

C. Training Gaussian Mixture From Data

We now focus on a related problem. Let us suppose Alice has K d-component vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$. And she wants to learn a mixture of c Gaussians from the data. She can use the iterative *expectation-maximization* (EM) algorithm to estimate the parameters of the c Gaussians and the mixture weights. Now, consider the scenario when Bob wants to learn the parameters but Alice does not want to disclose her data to Bob. One solution is for Alice to estimate the parameters herself and then give them to Bob. Another approach is for Alice and Bob to engage in a secure protocol which lets Bob learn the parameters while Alice's data remain private. The latter approach becomes necessary in a scenario where Bob wants to do data mining on combined data from private databases owned by Alice and Charlie. Below, we describe a secure protocol which lets two parties perform such computations.

1) EM Algorithm: We denote the estimate of a particular parameter after the rth iteration by using a superscript. Thus, $\boldsymbol{\mu}_i^r$, $\boldsymbol{\Sigma}_i^r$ and $P^r(\omega_i)$ denote the mean vector, covariance matrix, and the mixture weight for the ith Gaussian after the rth iteration. For convenience, let us denote the entire parameter set after the rth iteration by λ^r. At any given iteration, Alice has access to her data and Bob has access to the parameter $\boldsymbol{\Sigma}_i^r$. Alice and Bob have additive shares $\boldsymbol{\mu}_{iA}^r$, $\boldsymbol{\mu}_{iB}^r$, and ℓ_{iA}, ℓ_{iB} such that $\boldsymbol{\mu}_{iA}^r + \boldsymbol{\mu}_{iB}^r = \boldsymbol{\mu}_i^r$ and $\ell_{iA} + \ell_{iB} = \ln P^r(\omega_i)$. We can write the steps of the EM algorithm as follows.

E Step:

$$P(\omega_i|\mathbf{x}_k, \lambda^r) = \frac{p(\mathbf{x}_k|\omega_i, \boldsymbol{\mu}_i^r, \boldsymbol{\Sigma}_i^r)P^r(\omega_i)}{\sum_{j=1}^{c} p(\mathbf{x}_k|\omega_j, \boldsymbol{\mu}_j^r, \boldsymbol{\Sigma}_j^r)P^r(\omega_j)}. \tag{14}$$

Input: Alice has $\mathbf{x}_k$, $\boldsymbol{\mu}_{iA}^r$ and ℓ_{iA}; Bob has $\boldsymbol{\mu}_{iB}^r$, $\boldsymbol{\Sigma}_i^r$ and ℓ_{iB}, $i = 1, 2, \ldots, c$.

Output: Alice and Bob obtain u_{ik} and v_{ik} such that $u_{ik} + v_{ik} = \ln P(\omega_i|\mathbf{x}_k, \lambda^r)$.

1) Bob forms matrices $\mathbf{W}_i$ for $i = 1, \ldots, c$ with $\boldsymbol{\mu}_{iB}^r$, $\boldsymbol{\Sigma}_i^r$ as described in Section IV-A and **(9)** (using $(d/2)\ln 2\pi$ instead of $\ln P(\omega_i)$ to compute w_{i0}). With $(\mathbf{x}_k - \boldsymbol{\mu}_{iA}^r)$ as Alice's input and $\mathbf{W}_i$ for $i = 1, \ldots, c$ as Bob's input and $N = c$, Alice and Bob engage in steps 1) and 2) of Protocol SMG (Section IV-A.I) to obtain vectors $\mathbf{A}'_k$ and $\mathbf{B}'_k$.
   - Log-likelihood $\ln p(\mathbf{x}_k|\omega_i, \boldsymbol{\mu}_i^r, \boldsymbol{\Sigma}_i^r)$ is given by **(6)**. Notice that using $(\mathbf{x}_k - \boldsymbol{\mu}_{iA}^r)$ in place of $\mathbf{x}_k$ and $\boldsymbol{\mu}_{iB}^r$ in place of $\boldsymbol{\mu}_i^r$ in **(6)** yields the same result as using $\mathbf{x}_k$ and $\boldsymbol{\mu}_i^r$.
   - The sum of the ith elements, $A'_{ik} + B'_{ik}$, is equal to $\ln p(\mathbf{x}_k|\omega_i, \boldsymbol{\mu}_i^r, \boldsymbol{\Sigma}_i^r)$.
2) Alice and Bob obtain vectors $\mathbf{A}_k$ and $\mathbf{B}_k$, where for each i, $A_{ik} = A'_{ik} + \ell_{iA}$ and $B_{ik} = B'_{ik} + \ell_{iB}$.
   - Notice that $A_{ik} + B_{ik}$ is the logarithm of the numerator in **(14)**.
3) Alice and Bob engage in the secure logsum protocol with the vectors $\mathbf{A}_k$ and $\mathbf{B}_k$ to obtain y_k and z_k i.e., $y_k + z_k = SLOG(\mathbf{A}_k, \mathbf{B}_k)$.
   - Notice that $y_k + z_k$ is the logarithm of the denominator of **(14)** [follows from **(4)**].
4) Alice forms vector $\mathbf{u}_k$, where $u_{ik} = (A_{ik} - y_k)$. Bob forms the vector $\mathbf{v}_k$, where $v_{ik} = (B_{ik} - z_k)$.
   - $u_{ik} + v_{ik} = \ln P(\omega_i|\mathbf{x}_k, \lambda^r)$.

M Step:

$$\boldsymbol{\mu}_i^{r+1} = \frac{\sum_{k=1}^{K} P(\omega_i|\mathbf{x}_k, \lambda^r)\mathbf{x}_k}{\sum_{k=1}^{K} P(\omega_i|\mathbf{x}_k, \lambda^r)}$$

$$P^{r+1}(\omega_i) = \frac{\sum_{k=1}^{K} P(\omega_i|\mathbf{x}_k, \lambda^r)}{K}$$

$$\boldsymbol{\Sigma}_i^{r+1} = \frac{\sum_{k=1}^{K} P(\omega_i|\mathbf{x}_k, \lambda^r)(\mathbf{x}_k - \boldsymbol{\mu}_i^{r+1})(\mathbf{x}_k - \boldsymbol{\mu}_i^{r+1})^T}{\sum_{k=1}^{K} P(\omega_i|\mathbf{x}_k, \lambda^r)}. \tag{15}$$

Input: Alice has $\mathbf{x}_k, k = 1, \dots, K$. Alice and Bob have K-vectors $\mathbf{E}$ and $\mathbf{F}$ such that $E_k + F_k = \ln P(\omega_i|\mathbf{x}_k, \lambda^r)$.

Output: Alice obtains $\boldsymbol{\mu}_{iA}^{r+1}, \ell_{iA}$; Bob obtains $\boldsymbol{\mu}_{iB}^{r+1}, \boldsymbol{\Sigma}_i^{r+1}$ and ℓ_{iB}. ($\boldsymbol{\mu}_{iA}^{r+1} + \boldsymbol{\mu}_{iB}^{r+1} = \boldsymbol{\mu}_i^{r+1}$ and $\ell_{iA} + \ell_{iB} = \ln P^{r+1}(\omega_i)$).

1) Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{E}$ and $\mathbf{F}$ to obtain e and f i.e., $e + f = SLOG(\mathbf{E}, \mathbf{F})$.
2) Alice computes $\ell_{iA} = e - \ln K$, and Bob computes $\ell_{iB} = f$.
3) For $j = 1, 2, \dots, d$:
   Let $\mathbf{h}_j$ be the K-vector formed by the jth elements of $\mathbf{x}_1, \dots, \mathbf{x}_K$. Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{E} + \ln \mathbf{h}_j$ and $\mathbf{F}$ to obtain e' and f' i.e., $e' + f' = SLOG(\mathbf{E} + \ln \mathbf{h}_j, \mathbf{F})$.
   - Notice that $(e' - e) + (f' - f) = \ln \mu_{ij}^{r+1}$, the jth element of $\boldsymbol{\mu}_i^{r+1}$.

   Alice and Bob obtain the jth elements of $\boldsymbol{\mu}_{iA}^{r+1}$ and $\boldsymbol{\mu}_{iB}^{r+1}$, respectively, as a result of $SIP(exp(e' - e), exp(f' - f))$.
4) Consider the evaluation of σ_{mn}, the mnth element of the matrix $\boldsymbol{\Sigma}_i^{r+1}$. We first consider evaluating the mnth element of $(\mathbf{x}_k - \boldsymbol{\mu}_i^{r+1})(\mathbf{x}_k - \boldsymbol{\mu}_i^{r+1})^T$. As mentioned earlier, this is equivalent to evaluating the mnth term of $(\bar{\mathbf{x}}_k - \bar{\boldsymbol{\mu}}_i)(\bar{\mathbf{x}}_k - \bar{\boldsymbol{\mu}}_i)^T$, where $\bar{\mathbf{x}}_k = (\mathbf{x}_k - \boldsymbol{\mu}_{iA}^{r+1})$ and $\bar{\boldsymbol{\mu}}_i = \boldsymbol{\mu}_{iB}^{r+1}$. Let the jth elements of $\bar{\mathbf{x}}_k$ and $\bar{\boldsymbol{\mu}}_i$ be $\bar{x}_{kj}$ and $\bar{\mu}_{ij}$, respectively. Notice that Alice has access to $\bar{\mathbf{x}}_k$ and Bob had access to $\bar{\boldsymbol{\mu}}_i$.
   - For $k = 1, \dots, K$, Alice and Bob engage in the secure inner product protocol with vectors $exp(\gamma_k)[\bar{x}_{km}\bar{x}_{kn}, -\bar{x}_{km}, \bar{x}_{kn}, 1]$ and $[1, \bar{m}u_{in}, -\bar{\mu}_{im}, \bar{\mu}_{im}\bar{\mu_{in}}]$, where γ_k is a random scalar chosen by Alice. Let Bob obtain the result ϕ_k.
   - Alice forms the K-vector $\boldsymbol{\gamma} = [\gamma_1, \dots, \gamma_K]$ and Bob forms the vector $\boldsymbol{\phi} = [\phi_1, \dots, \phi_K]$.

   Alice and Bob engage in the secure logsum protocol with vectors $(\mathbf{E} - \boldsymbol{\gamma})$ and $(\mathbf{F} + \ln \boldsymbol{\phi})$ to obtain $\bar{e}$ and $\bar{f}$, i.e., $\bar{e} + \bar{f} = SLOG((\mathbf{E} - \boldsymbol{\gamma}), (\mathbf{F} + \ln \boldsymbol{\phi}))$.
   - Notice that $(\bar{e} - e) + (\bar{f} - f) = \ln \sigma_{mn}$, the mnth element of $\boldsymbol{\Sigma}_i^{r+1}$.

   Alice sends $(\bar{e} - e)$ to Bob so that he can calculate σ_{mn}.

At the end of all iterations, Alice sends her shares $\boldsymbol{\mu}_{iA}$ and ℓ_{iA} to Bob so that he can calculate the mean $\boldsymbol{\mu}_i$ and the mixture weight $P(\omega_i)$ for $i = 1, 2, \dots, c$.

Efficiency: We only consider the cost of computations that occur between Alice and Bob. In the E-step, for a given $\mathbf{x}_k$ and for all classes ω_i, there are $c(d+2)$ SIP calls with $(d+1)$-dimensional vectors and one *SIP* call involving a c-vector. In the M-step, to compute a mixture weight, there is a *SIP* call involving a K-vector. To calculate a single mean vector, there are d SIP calls involving K-vectors and d SIP calls with scalars. To calculate each element of the covariance matrix for a given class, there are K SIP calls involving 4-dimensional vectors and one *SIP* call with a K-vector.

Security: We assume that $k \gg d$ and $d > c$. Until the end of the last iteration, Bob does not learn values of the means or the mixture weights. He does not learn the values of likelihoods or posterior probabilities during the iterations. He does learn the value of the covariance matrix with every iteration. This does leak some information about the distribution of Alice's data vectors but Bob's aim is to learn the distributions. The goal of Alice is to prevent Bob from knowing her individual data vectors and without the mean, Bob cannot gain any knowledge about the data vectors. Another important constraint is that Alice does not learn the values of the parameters, and following the protocol closely shows that this holds true.

V. Hidden Markov Modelss

A. Forward–Backward Procedure

Consider the forward variable $\alpha_t(i)$ defined as

$$\alpha_t(i) = P(\mathbf{x}_1\mathbf{x}_2, \dots, \mathbf{x}_t, q_t = S_i|\lambda). \tag{16}$$

We can solve for $\alpha_t(i)$ inductively and calculate $P(\mathbf{X}|\lambda)$ as follows.

1) Initialization:

$$\alpha_1(i) = \pi_i b_i(\mathbf{x}_1), \qquad 1 \le i \le N$$

Input: Bob has the Gaussian mixture distribution that defines $b_i(\mathbf{x})$ and the initial state distribution $\pi = \{\pi_i\}$; Alice has an observation $\mathbf{x}_1$.

Output: Alice and Bob obtain vectors $\mathbf{Q}$ and $\mathbf{R}$ such that $Q_i + R_i = \ln \alpha_1(i)$.

a) Bob forms the matrices $\mathbf{W}_{ij}$ as mentioned in Section IV-B. With matrices $\mathbf{W}_{ij}$ and mixture weights c_{jm} as Bob's inputs and $\mathbf{x}_1$ as Alice's input, they perform steps 1) and 2) of the protocol MoG of Section IV-B. Alice and Bob obtain vectors $\mathbf{U}$ and $\mathbf{V}$. Notice that $U_i + V_i = \ln b_i(\mathbf{x}_1)$.

b) Alice forms the vector $\mathbf{Q} = \mathbf{U}$. Bob forms vector $\mathbf{R}$, where for each i, $R_i = V_i + \ln \pi_i$. Thus, $Q_i + R_i = \ln b_i(\mathbf{x}_1) + \ln \pi_i = \ln \alpha_1(i)$.

2) Induction:

$$\alpha_{t+1}(j) = \Big(\sum_{i=1}^{N} \alpha_t(i) a_{ij}\Big) b_j(\mathbf{x}_{t+1}) \quad \text{where} \quad 1 \le t \le T-1, 1 \le j \le N.$$

Input: Alice and Bob have vectors $\mathbf{Q}$ and $\mathbf{R}$ such that $Q_i + R_i = \ln \alpha_t(i)$. Alice and Bob have U_j and V_j such that $U_j + V_j = \ln b_j(\mathbf{x}_{t+1})$. Bob has the vector $\mathbf{a}_j = [a_{1j}, a_{2j}, \dots, a_{Nj}]$.

Output: Alice and Bob obtain $\bar{Q}$ and $\bar{R}$ such that $\bar{Q} + \bar{R} = \ln \alpha_{t+1}(j)$.

a) Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{Q}$ and $(\mathbf{R} + \ln \mathbf{a}_j)$ to obtain y' and z' i.e., $y' + z' = SLOG(\mathbf{Q}, \mathbf{R} + \ln \mathbf{a}_j)$.

b) Alice obtains $\bar{Q} = y' + U_j$ and Bob obtains $\bar{R} = z' + V_j$.

3) Termination:

$$P(\mathbf{X}|\lambda) = \sum_{i=1}^{N} \alpha_T(i).$$

Input: Alice and Bob have vectors $\mathbf{Q}$ and $\mathbf{R}$ such that $Q_i + R_i = \ln \alpha_T(i)$.
Output: Alice and Bob obtain y and z such that $y + z = \ln P(\mathbf{X}|\lambda)$.

a) Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{Q}$ and $\mathbf{R}$ to obtain y and z i.e., $y + z = SLOG(\mathbf{Q}, \mathbf{R})$.

Efficiency: In the initialization step, there are $(d+2)MN + N\ SIP$ calls and $N\ SMAX/SVAL$ calls involving d-dimensional vectors. In the induction step, for every j and for every t, there is one *SIP* call with an N-vector. In the termination step, there is one *SIP* call with an N-vector.

Security: Bob does not learn any $\mathbf{x}_k$ and Alice does not learn any of Bob's parameters. Hence, if the primitives *SIP*, *SMAX*, and *SVAL* are secure, the protocol is secure.

We can obtain a similar procedure for a backward variable $\beta_t(i)$ defined as

$$\beta_t(i) = P(\mathbf{x}_{t+1}\mathbf{x}_{t+2}, \ldots, \mathbf{x}_T | q_t = S_i, \lambda). \tag{17}$$

We can solve for $\beta_t(i)$ inductively as follows.

1) Initialization:

$$\beta_T(i) = 1, \qquad 1 \le i \le N.$$

2) Induction:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j),$$
$$\text{where} \qquad t = T-1, T-2, \ldots, 1, \quad 1 \le j \le N.$$

Input: Alice and Bob have vectors $\mathbf{Y}$ and $\mathbf{Z}$ such that $Y_j + Z_j = \ln \beta_{t+1}(j)$. Alice and Bob have $\mathbf{U}$ and $\mathbf{V}$ such that $U_j + V_j = \ln b_j(\mathbf{x}_{t+1})$. Bob has the vector $\mathbf{a}'_i = [a_{i1}, a_{i2}, \ldots, a_{iN}]$.
Output: Alice and Bob obtain $\bar{Y}$ and $\bar{Z}$ such that $\bar{Y} + \bar{Z} = \ln \beta_t(i)$.

a) Alice and Bob engage in the secure logsum protocol with vectors $\mathbf{Y} + \mathbf{U}$ and $(\mathbf{Z} + \mathbf{V} + \ln \mathbf{a}'_i)$ to obtain $\bar{Y}$ and $\bar{Z}$, i.e., $\bar{Y} + \bar{Z} = SLOG(\mathbf{Y} + \mathbf{U}, \mathbf{Z} + \mathbf{V} + \ln \mathbf{a}'_i)$.

B. Viterbi Algorithm

Consider the quantity

$$\delta_t(i) = \max_{q_1, q_2, \ldots, q_{t-1}} P[q_1 q_2, \ldots, q_t = S_i, \mathbf{x}_1 \mathbf{x}_2, \ldots, \mathbf{x}_t | \lambda]. \tag{18}$$

$\delta_t(i)$ is the best score (highest probability) along a single path, at time t, which accounts for the first t observations and ends in state S_i. The procedure for finding the best state sequence can be stated as follows.

1) Initialization:

$$\delta_1(i) = \pi_i b_i(\mathbf{x}_1), \qquad \psi_1(i) = 0 \qquad 1 \le i \le N.$$

The procedure is evaluating $\delta_1(i)$ is analogous to the initialization step of the forward backward procedure. After this step, Alice and Bob will have additive shares of $\ln \delta_1(i)$.

2) Recursion:

$$\delta_t(j) = \Big(\max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}] \Big) b_j(\mathbf{x}_t)$$
$$\psi_t(j) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i) a_{ij}]$$

where

$$2 \le t \le T, 1 \le j \le N.$$

Input: Alice and Bob have vectors $\mathbf{Q}$ and $\mathbf{R}$ such that $Q_i + R_i = \ln \delta_{t-1}(i)$. Alice and Bob have U and V such that $U + V = \ln b_j(\mathbf{x}_t)$. Bob has the vector $\mathbf{a}_j = [a_{1j}, a_{2j}, \ldots, a_{Nj}]$.
Output: Alice and Bob obtain $\bar{Q}$ and $\bar{R}$ such that $\bar{Q} + \bar{R} = \ln \delta_t(j)$. Alice obtains $\psi_t(j)$.

a) Alice and Bob engage in the secure maximum value protocol with vectors $\mathbf{Q}$ and $(\mathbf{R} + \ln \mathbf{a}_j)$ to obtain y and z. They also perform $SMAX$ on the same vectors and Alice obtains the result which is equal to $\psi_t(j)$.
b) Alice computes $\bar{Q} = y + U$ and Bob computes $\bar{R} = z + V$.

3) Termination:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \qquad\qquad q_T^* = \arg\max_{1 \le i \le N} \delta_T(i).$$

Alice and Bob will use $SVAL$ on their additive shares of $\ln \delta_T(i)$ for all i to evaluate $\ln P^*$. Similarly, they engage in $SMAX$ on their shares and Alice obtains the result q_T^*.

4) Path backtracking:

$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, \ldots, 1.$$

Alice, who has access to q_t and ψ_t, can evaluate the path sequence. Notice that Bob could be made to get this result instead of Alice if we let Bob learn the values of ψ_t and q_t in steps 2) and 3) instead of Alice.

Security and efficiency considerations for this protocol are similar to what was discussed with regard to the forward–backward procedure (Section V-A).

C. HMM Training

In the above formulation, we assumed that Bob had already trained his HMMs. Let us consider the case when Alice has all the training data, and Bob wants to train an HMM using her data. Below, we show how Bob can securely reestimate parameters of his HMM.

Consider the variables

$$\gamma_t(i) = P(q_t = S_i | \mathbf{X}, \lambda) = \frac{(\alpha_t(i)\beta_t(i))}{P(\mathbf{X}|\lambda)}$$
$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j | \mathbf{X}, \lambda) = \frac{(\alpha_t(i) a_{ij} b_j(\mathbf{x}_{t+1}) \beta_{t+1}(j))}{P(\mathbf{X}|\lambda)}.$$

In the previous subsections, we have shown how Alice and Bob can obtain additive shares of $\ln \alpha_t(i)$ (Q_i and R_i), $\ln \beta_t(i)$ ($\bar{Y}$ and $\bar{Z}$), $\ln b_j(\mathbf{x}_{t+1})$ (U_j and V_j), $\ln \beta_{t+1}(j)$ (Y_j, and Z_j)

and $\ln P(\mathbf{X}|\lambda)$ (y and z). It is easy to see that using these shares, Alice and Bob can compute additive shares e_t, g_t and f_t, h_t such that $e_t + f_t = \ln \xi_t(i,j)$ and $g_t + h_t = \ln \gamma_t(i)$. Alice computes $g_t = Q_i + \bar{Y} - y$ and $e_t = Q_i + U_j + Y_j - y$. Bob computes $h_t = R_i + \bar{Z} - z$ and $f_t = R_i + \ln a_{ij} + V_j + Z_j - z$.

The variables π_i and a_{ij} can then be reestimated as follows:

$$\bar{\pi}_i = \gamma_1(i)$$
$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}.$$

Input: Alice and Bob have $(T-1)$-vectors $\mathbf{e}$ and $\mathbf{f}$ such that $e_t + f_t = \ln \xi_t(i,j)$. They also have vectors $\mathbf{g}$ and $\mathbf{h}$ such that $g_t + h_t = \ln \gamma_t(i)$.

Output: Bob obtains $\ln \bar{a}_{ij}$.

1) Alice and Bob engage in secure logsum protocol with vectors $\mathbf{e}$ and $\mathbf{f}$ to obtain $\bar{e}$ and $\bar{f}$. They also engage in the secure logsum protocol with vectors $\mathbf{g}$ and $\mathbf{h}$ to obtain $\bar{g}$ and $\bar{h}$, respectively.
2) Alice sends $(\bar{e} - \bar{g})$ to Bob. Bob computes $(\bar{e} - \bar{g}) + (\bar{f} - \bar{h})$ to obtain $\ln \bar{a}_{ij}$.

Notice that instead of Bob obtaining the final result, Alice and Bob can have additive shares of $\ln \bar{a}_{ij}$. Protocols for forward–backward and viterbi algorithms will then have to modified so that Alice and Bob have additive shares of the vector $\ln \mathbf{a}_j$.

As for the Gaussian mixture distributions $b_i(\mathbf{x})$, Bob can learn them from Alice's data securely as we have shown in Section IV-C. We emphasize here that Bob does not learn all the parameters in every iteration. He learns the mean vector for every component Gaussian only after the last iteration. He does learn the covariance matrix in every iteration but quantities used to calculate the covariance matrix are additive shares which does not help him in inferring Alice's data. The example shown in Section IV-C uses two parties but it can be generalized to the case where Bob learns from multiple parties. In that case, learned statistics are averaged and provide an additional layer of security for the data providers.

VI. Discussion

In this section, we discuss the computational efficiency considerations of protocols presented above. As mentioned earlier, efficiency of the protocols was evaluated in terms of primitives and absolute measures were not provided. This is due to the fact that efficiency of the primitives themselves varies widely and depends on how the primitives are implemented.

If one follows all the protocols carefully, efficiency mainly depends on the computational complexity of the *SIP* primitive. We shall focus on one particular implementation of this primitive: secure inner product using homomorphic encryption proposed by [9] (see Appendix I, the reference provides proof that the protocol is correct and secure).

To validate the secure model, we ran experiments performing learning and classification. The experiments were run using a MATLAB implementation and tested both the Gaussian mixture models and the hidden Markov models. Simulations were performed twice using the secure and the nonsecure (traditional) methods. In all cases, the results from both the secure and nonsecure simulations were numerically identical as we have predicted. The secure versions were obviously less efficient due to the increased computational cost of the cryptographic operations and the increased network traffic. We did not study the communications complexity in these experiments and rather focused on the computational load, which is the primary bottleneck [18]. One simulation used a generalized version [19] of the Paillier public-key scheme [20]. We used cryptographic keys of 1024 bits and the cryptosystem was implemented in Java. The computational load of this algorithm coupled with a nonoptimal implementation resulted into a processing time per input vector in the order of a few seconds. An alternative implementation using algebraic primitives, which leak some information but are more computationally efficient, resulted into a significant speedup of less than a second's time processing per input vector. As shown by the last experiment, the choice of implementation for a primitive (for example, *SIP* using algorithm in Appendix I instead of *SIP* using algorithm in Appendix II) significantly impacts performance, communication complexity and security. A wise choice will have to balance tradeoffs such as computational efficiency and network bandwidth as opposed to security/privacy. A discussion of these issues is out of the scope of this paper since it is a lengthy research project of its own and a moving target given the continuous discoveries of increasingly efficient protocols by the cryptography community.

The figures we have obtained were using nonoptimized implementations, as noted by [18] careful implementation can produce significant speedups in computation. For practical implementations it is also possible (and recommended) that specialized hardware is used for the cryptography layer which can result into dramatic performance improvements.

VII. Conclusion and Future Work

In this paper, we have presented an implementation of privacy preserving hidden Markov model and Gaussian mixtures computations. We first proposed a simple privacy-preserving protocol for computing logsums. Using primitives for computing scalar products and maxima, we proposed secure protocols for classification using Gaussian mixture models. We then proposed secure protocols for the *forward–backward* algorithm, the *viterbi* algorithm, and HMM training. The protocols are defined modularly in terms of primitives so that future advances in cryptography, which will hopefully provide more robust and efficient protocols, can be readily employed in our framework by straightforward replacement. The approach we have taken also illustrates the process required to transform a signal processing algorithm to its privacy preserving version. Other data processing and classification algorithms can also be described in terms of secure primitives and easily reformulated for secure multiparty computations.

This paper is intended to be a starting point for secure audio frameworks, and because of that it exposes a lot of new research directions which warrant more attention. One of these directions includes the design of alternative classifiers and algorithms using this process, and there is still ongoing work on the building block primitives (*SIP*, *SMAX*, *SVAL*, etc) themselves. These are all topics that present plenty of opportunities to explore efficiency and security and their tradeoffs. We expect these to be fruitful areas of research in the near future. It is our hope that a migration towards secure algorithms can help promote a more open collaboration setting where parties can freely exchange data and algorithms without legal and privacy issues.

Appendix I
Secure Inner Product Using Homomorphic Encryption

The following protocol is based on homomorphic encryption and was proposed by [9]. Let the triple $(\mathtt{Ge}, \mathtt{En}, \mathtt{De})$ denote a public-key homomorphic cryptosystem (probabilistic polynomial time algorithms for key-generation, encryption, and decryption). The key generation algorithm generates a valid pair $(\mathtt{sk}, \mathtt{pk})$ of private and public keys for a security parameter k. The encryption algorithm $\mathtt{En}$ takes as an input a plaintext m, a random value r, and a public key $\mathtt{pk}$ and outputs the corresponding ciphertext $\mathtt{En}(\mathtt{pk}; m, r)$. The decryption algorithm $\mathtt{De}$ takes as an input a ciphertext c and a private key $\mathtt{sk}$ (corresponding to the public key $\mathtt{pk}$) and outputs a plaintext $\mathtt{De}(\mathtt{sk}; c)$. It is required that $\mathtt{De}(\mathtt{sk}; \mathtt{En}(\mathtt{pk}; m, r)) = m$. A public-key cryptosystem is *homomorphic* if $\mathtt{En}(\mathtt{pk}; m_1, r_1) \cdot \mathtt{En}(\mathtt{pk}; m_2, r_2) = \mathtt{En}(\mathtt{pk}; m_1 + m_2, r_1 + r_2)$, where $+$ is a group operation and $\cdot$ is a groupoid operation.

Inputs: Private vectors $\mathbf{x}$ and $\mathbf{y}$ with Bob and Alice, respectively.

Outputs: Shares a and b such that $a + b = \mathbf{x}^T\mathbf{y}$.

1) Setup phase. Bob:
   - generates a private and public key pair $(\mathtt{sk}, \mathtt{pk})$.
   - sends pk to Alice.
2) For $i \in \{1, \ldots, d\}$, Bob:
   - generates a random new string r_i.
   - sends $c_i = \mathtt{En}(\mathtt{pk}; x_i, r_i)$ to Alice.
3) Alice:
   - sets $z \leftarrow \prod_{i=1}^{d} c_i^{y_i}$.
   - generates a random plaintext b and a random nonce r'.
   - sends $z' = z \cdot \mathtt{En}(\mathtt{pk}; -b, r')$ to Bob.
4) Bob computes $a = \mathtt{De}(\mathtt{sk}; z') = \mathbf{x}^T\mathbf{y} - b$.

See [9] for a proof that the protocol is correct and secure.

Appendix II
Secure Inner Product From Oblivious Polynomial Evaluation

[8] proposes an elegant protocol for oblivious evaluation of multivariate polynomials using oblivious transfer [21] as a cryptographic primitive. It can be easily modified to securely evaluate dot products. Let Alice represent each x_i as $x_i = \sum_j a_{ij} 2^{j-1}$ with $a_{ij} \in \{0, 1\}$. Let $v_{ij} = 2^{j-1} y_i$. Notice that for each i, $1 \le i \le d$, $\sum_j a_{ij} v_{ij} = x_i y_i$. The idea is to have Bob prepare v_{ij} and have Alice get those v_{ij} with $a_{ij} = 1$ in some secret way. This is achieved as follows: Bob prepares the pair $(r_{ij}, v_{ij} + r_{ij})$ for randomly chosen r_{ij} and Alice runs independent oblivious transfer with Bob to get r_{ij} if $a_{ij} = 0$ and $v_{ij} + r_{ij}$ otherwise. At the end of the protocol, Alice will obtain $\sum_i \sum_j (a_{ij} v_{ij} + r_{ij}) = \sum_i x_i y_i + \sum_{i,j} r_{ij}$. Bob will have $-\sum_{i,j} r_{ij}$. Thus, Alice and Bob will have additive shares of the desired dot product.

[8] proves that this protocol is secure when the parties are semi-honest. The efficiency of the protocol depends on the implementation of oblivious transfer.

Appendix III
Secure Inner Product Using Linear Transformation

[11] proposes an algebraic approach which assumes that the dimensionality is even. Let us define $\mathbf{x}_1$ as the $d/2$ dimensional vector consisting of the first $d/2$ elements of $\mathbf{x}$ and $\mathbf{x}_2$ as the vector consisting of the last $d/2$ elements of $\mathbf{x}$. We observe that $\mathbf{x}^T\mathbf{y} = \mathbf{x}_1^T\mathbf{y}_1 + \mathbf{x}_2^T\mathbf{y}_2$. Alice and Bob jointly generate a random invertible $d \times d$ matrix M. Alice computes $\mathbf{x}' = \mathbf{x}^T M$, splits it as $\mathbf{x}_1'$ and $\mathbf{x}_2'$, and sends $\mathbf{x}_2'$ to Bob. Bob computes $\mathbf{y}' = M^{-1}\mathbf{y}$, splits it as $\mathbf{y}_1'$ and $\mathbf{y}_2'$, and sends $\mathbf{y}_1'$ to Alice. Alice computes $\mathbf{x}_1'\mathbf{y}_1'$, and Bob computes $\mathbf{x}_2'\mathbf{y}_2'$ so that their sum is equal to the desired result.

This protocol has little communication and computational overhead compared to the cryptographic protocols, but it comes at the cost of security. Alice and Bob learn $d/2$ linear equations for the d unknowns that constitute the other party's vector which leaks a lot of information. Hence, it is important that the same matrix M should be used when this protocol is used multiple times with the same vector $\mathbf{x}$ (or $\mathbf{y}$). [3] has analyzed this protocol which showed serious security flaws, and hence this is not practical when security is crucially important.

Appendix IV
Permute Protocol

This protocol was proposed in [15].

Input: Alice and Bob have d-component vectors $\mathbf{x}$ and $\mathbf{y}$. Bob has a random permutation π.

Output: Alice and Bob obtain $\mathbf{q}$ and $\mathbf{s}$ such that $\mathbf{q} + \mathbf{s} = \pi(\mathbf{x}) + \pi(\mathbf{y})$.

1) Alice generates public and private keys for a homomorphic cryptosystem and sends the public key to Bob. Let $E()$ denote encryption with Alice's public key.
2) Alice encrypts each element of $\mathbf{x}$ and sends the resulting vector $\bar{\mathbf{x}}$ to Bob.
3) Bob generates a random vector $\mathbf{r}$ and computes a new vector $\boldsymbol{\theta}$ where $\theta_i = \bar{x}_i E(r_i) = E(x_i + r_i)$, for $i = 1, \ldots, d$.
4) Bob permutes $\boldsymbol{\theta}$ and sends $\pi(\boldsymbol{\theta})$ to Alice. Alice decrypts the vector to obtain $\mathbf{q}$.
5) Bob computes $\mathbf{y} - \mathbf{r}$ and then permutes it using π to obtain $\mathbf{s} = \pi(\mathbf{y} - \mathbf{r})$.

Alice and Bob engage in the above permute protocol twice, the second time with their roles interchanged. After this is done, Alice and Bob will have two vectors whose sum will be a random permutation of the original sum but neither of them will know what the permutation is.

Acknowledgment

The authors would like to thank S. Avidan for his help and influence in the making of this work. The authors also would also like to thank the anonymous reviewers for their comments and suggestions.

References

[1] A. C.-C Yeo, "Protocols for secure computation," in *Proc. 23rd IEEE Symp. Foundations Comput. Sci.*, 1982, pp. 160–164.

[2] J. Vaidya and C. Clifton, "Privacy preserving k-means clustering over vertically partitioned data," in *Proc. ACM SIGKDD Conf. Knowledge Discovery Data Mining*, 2003, pp. 206–215.

[3] E. Kiltz, G. Leander, and J. Malone-Lee, "Secure computation of the mean and related statistics," in *Proc. Theory Cryptography Conf.*, 2005, vol. 3378, Lecture Notes in Computer Science, pp. 283–302.

[4] S. Avidan and M. Butman, "Blind Vision," in *Proc. ECCV*, 2006, vol. 3, pp. 1–13.

[5] O. Goldreich, "Secure multi-party computation," *Working Draft*, 2000 [Online]. Available: http://citeseer.ist.psu.edu/goldreich98secure.html

[6] L. R. Rabiner, "A tutorial on hidden Markov models and selected applications in speech recognition," *Proc. IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[7] S. Laur and H. Lipmaa, "Additive conditional disclosure of secrets and applications," Cryptology ePrint Archive, 2005, Rep. 2005/378.

[8] Y.-C. Chang and C.-J. Lu, "Oblivious polynomial evaluation and oblivious neural learning," in *Proc. Adv. Cryptol–Asiacrypt'01*, 2001, vol. 2248, Lecture Notes in Computer Science, pp. 369–384.

[9] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikainen, C. Park and S. Chee, Eds., "On private scalar product computation for privacy-preserving data mining," in *Proc. Int. Conf. Inf. Security Cryptol.*, 2004, vol. 2506, Lecture Notes in Computer Science, pp. 104–120.

[10] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in *Proc. 17th Annu. Comput. Security Applicat. Conf.*, New Orleans, LA, Dec. 2001, pp. 102–110.

[11] W. Du and Z. Zhan, "A practical approach to solve secure multi-party computation problems," in *Proc. New Security Paradigms Workshop*, Virginia Beach, VA, Sep. 23–26, 2002, pp. 127–135.

[12] I. Ioannidis, A. Grama, and M. Atallah, "A secure protocol for computing dot-products in clustered and distributed environments," in *Proc. Int. Conf. Parallel Process.*, Vancouver, BC, Canada, 2002, pp. 379–384.

[13] P. Ravikumar, W. W. Cohen, and S. E. Fienberg, "A secure protocol for computing string distance metrics," in *Proc. Workshop Privacy and Security Aspects of Data Mining*, Brighton, U.K., 2004, pp. 40–46.

[14] J. Vaidya and C. Clifton, "Privacy preserving association rule mining in vertically partitioned data," in *Proc. Int. Conf. Knowledge Discovery and Data Mining*, Edmonton, AB, Canada, 2002, ACM SIGKDD.

[15] M. J. Atallah, F. Kerschbaum, and W. Du, "Secure and private sequence comparisons," in *Proc. Workshop Privacy Electron. Soc.*, Washington, DC, Oct. 2003, pp. 39–44.

[16] I. F. Blake and V. Kolesnikov, "Strong conditional oblivious transfer and computing on intervals," in *Proc. Adv. Cryptol.–Asiacrypt'04*, P. J. Lee, Ed. New York: Springer-Verlag, 2004, vol. 3329, LNCS, pp. 515–529.

[17] H.-Y. Lin and W.-G. Tzeng, "An efficient solution to the millionaires' problem based on homomorphic encryption," in *Proc Int. Conf. Appl. Cryptography Network Security*, 2005, vol. 3531, LNCS, pp. 456–466.

[18] Z. Yang, R. Wright, and H. Subramaniam, "Experimental analysis of a privacy-preserving scalar product protocol," *Int. J. Comput. Syst. Sci. Eng.*, vol. 21, no. 1, pp. 47–52, 2006.

[19] I. Damgard and M. Jurik, "A generalisation, simplification and some applications of Paillier's probabilistic public-key system," in *Proc. Int. Workshop Practice Theory Public Key Cryptography*, 2001, vol. 1992, Lecture Notes in Computer Science, pp. 119–136.

[20] P. Paillier, J. Stern, Ed., "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Adv. Cryptol.–Eurocrypt'99*, 1999, vol. 1592, Lecture Notes in Computer Science, pp. 223–238.

[21] M. Naor and B. Pinkas, "Oblivious transfer and polynomial evaluation," in *Proc. 31st Annu. ACM Symp. Theory Comput.*, 1999, pp. 245–254.

Paris Smaragdis (SM'06) is a member of the research staff at Mitsubishi Electric Research Laboratories, Cambridge, MA. Prior to that position, he was at the Massachusetts Institute of Technology, Cambridge, where he completed his graduate and postdoctoral training. His interests are computational audition, scene analysis, and the intersection of machine learning with signal processing.

Madhusudana Shashanka (S'06) received the B.E. degree (hons) in computer science from the Birla Institute of Technology and Science, Pilani, India, in 2003. He is currently pursuing the Ph.D. degree at the Department of Cognitive and Neural Systems and the Hearing Research Center, Boston University, Boston, MA.