A FRAMEWORK FOR SECURE SPEECH RECOGNITION

Paris Smaragdis

Mitsubishi Electric Research Labs (MERL)
201 Broadway, Cambridge MA 02139, USA

Madhusudana V. S. Shashanka

Boston University Hearing Research Center
677 Beacon St, Boston MA 02215, USA

ABSTRACT

We present an algorithm that enables privacy-preserving speech recognition transactions between multiple parties. We assume two commonplace scenarios. One being the case where one of two parties has private speech data to be transcribed and the other party has private models for speech recognition. And the other being that of one party having a speech model to be trained using private data of multiple other parties. In both of the above cases data privacy is desired from both the data and the model owners. In this paper we will show how such collaborations can be performed while ensuring no private data leaks using secure multiparty computations. In neither case will any party obtain information on other parties data. The protocols described herein can be used to construct rudimentary speech recognition systems and can be easily extended for arbitrary audio and speech processing.

Index Terms— Cryptography, Data Security, Speech Recognition

1. INTRODUCTION

Today's networked world presents a variety of service models in terms of client-server services. Speech recognition could surely be a part of this, however privacy concerns have impeded the development of this model. Individuals, and more so corporations and governments, are understandably reluctant to share private speech data with other parties providing speech recognition services. In this paper, we address this issue and provide a framework which can allow this cooperation by guaranteeing data privacy for both data and speech recognition model providers.

To be more specific, we will present a formulation for securely training and evaluating Hidden Markov Models (HMMs) between multiple parties. We formulate it in such a way so that the owners of the data will not have to share the data, the owner of the HMM will not have to share the HMM parameters and at the end of the transaction, the owner of the HMM will not know what the final computation result is. These results will only be revealed to the providers of the speech data.

We will show how this type of secure multiparty computation can be achieved using two scenarios. One scenario will involve an HMM party training from private data of multiple other parties, the other scenario will deal with the case where already trained HMMs will be applied on private data from other parties. The utility of these scenarios is easy to see in collaborative speech recognition projects. In the first case we can enable the consolidation of private speech databases to train large speech recognition models while ensuring data privacy. In the other case we can enable speech recognition as a service model to off-site customers who need to maintain privacy of their speech data and their transcriptions from both the service provider and malicious network intruders.

Although these seem like impossible constraints to deal with, they can be achieved using protocols for secure multiparty computations (SMC). Such protocols have been around since the early 80s [8] and provide the tools by which to perform arbitrary computations between multiple parties concerned with data privacy. Recently this concept has been employed for simple machine learning tasks such as k-means and rudimentary computer vision applications [7, 2]; in this paper we present a SMC formulation of training and evaluating HMMs as applied on speech data.

2. PRELIMINARIES

2.1. Secure Two-party Computations

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. [1]

To implement an algorithm securely, we will have to implement each step of the algorithm securely and make sure the intermediate results of these steps are also secure. 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. Also, if the 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, and
- 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 figure 1 for a schematic illustration.

We now present primitives that will be used in the rest of the paper. Based on how they are implemented, one can achieve different levels of security and computational/communication efficiency.

[1] In a *malicious model*, no assumptions are made about the parties' behavior. Enforcing security is harder in such a case but can be done by accompanying the protocols with zero-knowledge proofs that protocols are being followed. A detailed discussion is out of scope of this paper.

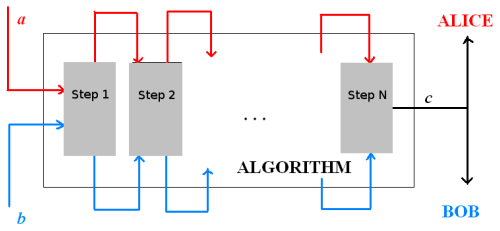

Fig. 1. Implementing an algorithm securely. The algorithm takes in private inputs $\mathbf{a}$ and $\mathbf{b}$. Algorithm is split into steps that can be implemented as secure primitives (shown as grey boxes). Intermediate results are distributed as random additive shares and feed into the following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

In general, there is a trade-off between security and efficiency. In the following subsections, let $\mathbf{x} = [x_1 \dots x_d]$ and $\mathbf{y} = [y_1 \dots y_d]$ denote d-dimensional vectors owned by Alice and Bob respectively.

Secure Inner Products (SIP): Given $\mathbf{x}$ and $\mathbf{y}$, the secure inner product protocol produces scalars a and b such that $a+b = \mathbf{x}^T\mathbf{y}$. We shall denote this computation by $SIP(\mathbf{x}, \mathbf{y})$. We use cryptographic [3, 4] protocols for this primitive.

Secure Maximum Index ($SMAX$): Given $\mathbf{x}$ and $\mathbf{y}$, Alice would like to know the index of the maximum element in the vector sum $\mathbf{x} + \mathbf{y}$. Neither party should get to know the actual value of the maximum. We denote this computation as $j = SMAX(\mathbf{x}, \mathbf{y})$.

For this primitive, we use the permute protocol proposed by [1]. Given $\mathbf{x}$ and $\mathbf{y}$ and a permutation π chosen by Alice (Bob should not learn π), the protocol enables Alice and Bob to obtain additive shares of the permuted sum. In other words, they obtain vectors $\mathbf{q}$ and $\mathbf{s}$ such that $\mathbf{q} + \mathbf{s} = \pi(\mathbf{x} + \mathbf{y})$. Alice chooses a random number r and sends $\mathbf{q} - r$ to Bob. Bob sends the index of the maximum element in $\mathbf{q}+\mathbf{s}-r$ 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.

Secure Maximum Value ($SVAL$): Given $\mathbf{x}$ and $\mathbf{y}$, the secure maximum value protocol produces a and b such that their sum is equal to the maximum element in $\mathbf{x} + \mathbf{y}$. Let us denote this as $a + b = SMAX(\mathbf{x}, \mathbf{y})$. For this primitive, we use the minimum finding protocol presented in [1].

Secure Logsum ($SLOG$): This primitive unlike the others above, is introduced here only because it simplifies the presentation of protocols later. It is not a cryptographic primitive. Given $\mathbf{x}$ and $\mathbf{y}$ such that $\mathbf{x} + \mathbf{y} = \ln \mathbf{z} = [\ln z_1 \dots \ln z_d]$, the protocol produces two scalars q and s such that $q + s = \ln(\sum_{i=1}^{d} z_i)$. Let us denote this by $q + s = SLOG(\mathbf{x}, \mathbf{y})$.

One can obtain $q + s$ as the logarithm of the inner product of exponentiated vectors $e^{\mathbf{x}}$ and $e^{\mathbf{y}}$ as follows: Alice chooses a random q and computes $SIP(e^{\mathbf{x}-q}, e^{\mathbf{y}})$ with Bob. Bob obtains the result ϕ. Bob has $s = \ln \phi = -q + \ln(\sum_{j=1}^{d} e^{x_j+y_j})$ and Alice has q.

Gaussian Mixture Likelihood (GML): In [6], we proposed protocols for classification using gaussian mixture models. One of the protocols presented can be used to find the log-likelihood of data given a gaussian mixture model. If Bob has a gaussian mixture model $b(\mathbf{x})$ and Alice has data vector $\mathbf{x}_1$, the protocol generates additive shares of the log-likelihood $\log b(\mathbf{x}_1)$. See [6] for a detailed description.

2.2. Hidden Markov Models

Hidden Markov Models find use in a wide range of applications, and have successfully been used in speech recognition. There are there fundamental problems for HMM design, namely: the evaluation of the probability of (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 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 [5].

2.3. Problem Formulation

Suppose Bob has a trained HMM with all the model parameters learned. Let the HMM be characterized as follows:

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

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

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

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

  where $\mathbf{x}$ is the variable, c_{jm} is the mixture coefficient for the m-th 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

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

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

Let Alice have an observation sequence $\mathbf{X} = \mathbf{x}_1\mathbf{x}_2 \dots \mathbf{x}_T$. 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. 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 - otherwise these features can be computed securely as well). Alice and Bob can obtain additive shares of the likelihood of Alice's observation sequence for every speech recognition unit of Bob. They can then engage in the $SMAX$ protocol and find out the unit that corresponds to Alice's sound snippet. We will also show how one can securely learn the best sequence of model states using the *viterbi* algorithm. And finally, we will show how one can train HMM parameters using data from a private database.

3. SECURE FRAMEWORK

Consider the computation of a function

$$z = f(x_1y_1, x_2y_2, \dots, x_ny_n) = \Phi_{1 \le i \le N} x_iy_i \tag{4}$$

where Φ is a generic operator. For every i, let Alice and Bob have additive shares of $\log x_i$ and $\log y_i$. They can receive additive shares of z by using $SLOG$, $SVAL$ or $SMAX$ when the operator is a summing operator Σ, a maximum operator max, or a maximum index operator $argmax$ respectively. With the above basic operations, one can implement protocols for all problems of HMM.

3.1. The Forward-Backward Procedure

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

$$\alpha_t(i) = P(\mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_t, q_t = S_i|\lambda) \tag{5}$$

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 mixture distribution that defines $b_i(\mathbf{x})$ and the initial state distribution $\pi = \{\pi_i\}$; Alice has $\mathbf{x}_1$.
Output: Alice and Bob obtain vectors $\mathbf{Q}$ and $\mathbf{R}$ such that $Q_i + R_i = \ln \alpha_1(i)$.

   (a) Alice and Bob perform GML on their data to 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:

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

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 $\mathbf{a}_j = [a_{1j}, a_{2j}, \ldots, a_{Nj}]$.
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 $\bar{Q} = y' + U_j$, 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 $SLOG$ 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.

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

$$\beta_t(i) = P(\mathbf{x}_{t+1}\mathbf{x}_{t+2} \ldots \mathbf{x}_T|q_t = S_i, \lambda) \tag{6}$$

We initialize $\beta_T(i) = 1, \forall 1 \le i \le N$ and solve for $\beta_t(i)$ as:

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

Notice that the above equation is a sum of products. Alice and Bob have additive shares of the logarithm of each product term. They can compute additive shares of the final result using $SLOG$.

3.2. Viterbi Algorithm

Consider the quantity

$$\delta_t(i) = \max_{q_1, q_2 \cdots q_{t-1}} P[q_1 q_2 \ldots q_t = S_i, \mathbf{x}_1\mathbf{x}_2 \ldots \mathbf{x}_t|\lambda] \tag{7}$$

$\delta_t(i)$ is the best score along a single path, at time t, which accounts for the first t observations and ends in state S_i.

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.

2. Recursion:

$$\begin{aligned} \delta_t(j) &= \Big(\max_{1 \le i \le N} [\delta_{t-1}(i)a_{ij}]\Big) b_j(\mathbf{x}_t) \\ \psi_t(j) &= \text{argmax}_{1 \le i \le N} [\delta_{t-1}(i)a_{ij}] \\ \text{where} & \qquad 2 \le t \le T, 1 \le j \le N \end{aligned}$$

For $\delta_t(j)$, Alice and Bob use $SVAL$ and obtain additive shares of the log of the maximum. They'll already have additive shares of $\log b_j(\mathbf{x}_t)$ and hence they'll have additive shares of $\log \delta_t(j)$. Similarly, they can obtain $\psi_t(j)$ by using $SMAX$.

3. Termination and Path backtracking:

$$P^* = \max_{1 \le i \le N} [\delta_T(i)] \qquad q_T^* = \text{argmax}_{1 \le i \le N} \delta_T(i).$$
$$q_t^* = \psi_{t+1}(q_{t+1}^*) \qquad t = T-1, T-2, \ldots, 1.$$

Alice and Bob will have additive shares of $\log \delta_T(i)$ from the previous step. They can use $SVAL$ and $SMAX$ to obtain P^* and q_T^*. Alice, who has access to q_t and ψ_t, can evaluate the path sequence.

Security and efficiency considerations for this protocol are similar to what was discussed with regard to the forward-backward procedure.

3.3. HMM training

Formulae for re-estimation of HMM parameters are given below.

$$\gamma_t(j,k) = \left(\frac{\alpha_t(j)\beta_t(j)}{P(\mathbf{X}|\lambda)}\right)\left(\frac{c_{jk}\mathcal{N}(\mathbf{x}_t, \boldsymbol{\mu}_{jk}, \boldsymbol{\Sigma}_{jk})}{b_j(\mathbf{x}_t)}\right)$$

$$\xi_t(i,j) = \frac{\alpha_t(i)a_{ij}b_j(\mathbf{x}_{t+1})\beta_{t+1}(j)}{P(\mathbf{X}|\lambda)}, \quad a_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$

$$c_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)}{\sum_{t=1}^{T}\sum_{k=1}^{M} \gamma_t(j,k)}, \quad \boldsymbol{\mu}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)\mathbf{x}_t}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

$$\pi_i = \gamma_1(i), \quad \boldsymbol{\Sigma}_{jk} = \frac{\sum_{t=1}^{T} \gamma_t(j,k)(\mathbf{x}_t - \boldsymbol{\mu}_{jk})(\mathbf{x}_t - \boldsymbol{\mu}_{jk})'}{\sum_{t=1}^{T} \gamma_t(j,k)}$$

Notice that all the above formulae are made of products or sums of products. It is easy to show that Alice and Bob, using SIP and $SLOG$ protocols, can compute $\gamma_t(j,k)$, $\xi_t(i,j)$, π_i, a_{ij} and c_{jk}. We do not show the details due to lack of space.

Now consider the estimation of the mean and covariance of the gaussian mixture components. The requirements are that Alice should not have access to the learned parameters and Bob should not have access to the data in any iteration. In addition, we have to be careful that Bob does not learn the parameters at every iteration. This is required because one can deduce information about the data by knowing the results of repeated operations with the data vector. For example, knowledge of d inner products with a d dimensional vector $\mathbf{x}$ is enough to learn $\mathbf{x}$. To enforce this, we let Alice and Bob have additive shares of the mean at every iteration. Only after all the iterations are complete does Bob receive values of the mean.

Input: Alice has $\mathbf{x}_t, t = 1, \ldots, T$. Alice and Bob have T-vectors $\mathbf{E}$ and $\mathbf{F}$ such that $E_t + F_t = \ln \gamma_t(j,k)$.
Output: Alice obtains $\boldsymbol{\mu}_{jk_A}$; Bob obtains $\boldsymbol{\mu}_{jk_B}$ and $\boldsymbol{\Sigma}_{jk}$. ($\boldsymbol{\mu}_{jk_A} + \boldsymbol{\mu}_{jk_B} = \boldsymbol{\mu}_{jk}$).

1. Alice and Bob obtain e and f where $e + f = SLOG(\mathbf{E}, \mathbf{F})$.
2. For $i = 1, 2, \ldots, d$:
   Let $\mathbf{h}_i$ be the T-vector formed by i-th elements of $\mathbf{x}_1, \ldots, \mathbf{x}_T$. Alice and Bob obtain e' and f' where $e' + f' = SLOG(\mathbf{E} + \ln \mathbf{h}_i, \mathbf{F})$.
   - Notice that $(e' - e) + (f' - f) = \ln \mu^i_{jk}$.

   Alice and Bob obtain the i-th elements of $\boldsymbol{\mu}_{jk_A}$ and $\boldsymbol{\mu}_{jk_B}$ respectively as a result of $SIP(exp(e' - e), exp(f' - f))$.
3. Consider the evaluation of σ_{mn}, the mn-th element of the matrix $\boldsymbol{\Sigma}_{jk}$. We first consider evaluating the mn-th element of $(\mathbf{x}_t - \boldsymbol{\mu}_{jk})(\mathbf{x}_t - \boldsymbol{\mu}_{jk})'$. This is equivalent to evaluating the mn-th term of $(\bar{\mathbf{x}}_t - \bar{\boldsymbol{\mu}}_{jk})(\bar{\mathbf{x}}_t - \bar{\boldsymbol{\mu}}_{jk})'$, where $\bar{\mathbf{x}}_t = (\mathbf{x}_t - \boldsymbol{\mu}_{jk_A})$ and $\bar{\boldsymbol{\mu}}_{jk} = \boldsymbol{\mu}_{jk_B}$. Let the i-th elements of $\bar{\mathbf{x}}_t$ and $\bar{\boldsymbol{\mu}}_{jk}$ be $\bar{x}^i_t$ and $\bar{\mu}^i_{jk}$ respectively. Notice that Alice has access to $\bar{\mathbf{x}}_t$ and Bob has access to $\bar{\boldsymbol{\mu}}_{jk}$.
   - For $t = 1, \ldots, T$, Alice and Bob engage in SIP protocol with vector $exp(r_t)[\bar{x}^m_t \bar{x}^n_t, -\bar{x}^m_t, \bar{x}^n_t, 1]$ and vector $[1, \bar{\mu}^n_{jk}, -\bar{\mu}^m_{jk}, \bar{\mu}^m_{jk}\bar{\mu}^n_{jk}]$, where r_t is a random scalar chosen by Alice. Let Bob obtain the result ϕ_t.
   - Alice forms the T-vector $\mathbf{r} = [r_1, \ldots, r_T]$ and Bob forms the vector $\boldsymbol{\phi} = [\phi_1, \ldots, \phi_T]$.

   Alice and Bob obtain $\bar{e}$ and $\bar{f}$ where $\bar{e} + \bar{f} = SLOG((\mathbf{E} - \mathbf{r}), (\mathbf{F} + \ln \boldsymbol{\phi}))$.
   - Notice that $(\bar{e} - e) + (\bar{f} - f) = \ln \sigma_{mn}$.

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

At the end of all iterations, Alice sends her shares $\boldsymbol{\mu}_{jk_A}$ to Bob so that he can calculate $\boldsymbol{\mu}_{jk}$. Notice that Bob 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 about Alice's data. This particular example of HMM training using only two parties was used for derivation purposes. This procedure can be easily generalized to the case where Bob learns parameters using data from multiple parties instead of one, in which case the learned statistics are averaged and provide an additional layer of security for the data providers.

4. CONCLUSIONS AND FUTURE WORK

In this paper we have presented a secure multiparty computation formulation of hidden Markov models. We have shown how we can employ simpler SMC primitives to perform training and evaluation of HMMs for security sensitive speech applications. The implementation that we have presented is based on the concept of additive shares to enforce data privacy. Although this is a viable implementation it is only one of many possible ways such a secure system can be implemented. Future advances in the field of cryptography, which will hopefully provide more robust and efficient protocols, can be readily employed in our framework by straightforward replacement of the basic primitives. A wise choice of underlying primitives will have to balance tradeoffs such as computational efficiency and network bandwidth as opposed to security/privacy and is a research project in its own right. Because of the breadth of options and the limited space in this paper we defer a performance evaluation discussion to a future publication. At the moment, and depending on the primitives used, performance can range from a few times the computational/network overhead of a straightforward implementation, to many orders of magnitude. The emergence of specialized hardware that perform cryptographical operations is also a factor that can drastically reduce computational complexity.

In this paper we only presented a formulation to perform HMM computations which are central for speech applications. It is straightforward however to employ this methodology to construct various secure applications using arbitrary signal processing operations. It is our hope that such privacy conserving methodologies can help foster computational collaborations for either research or commercial purposes while ensuring the data privacy of all participants.

5. REFERENCES

[1] MJ Atallah, F Kerschbaum, and W Du. Secure and private sequence comparisons. In *Workshop on Privacy in the Electronic Society*, 2003.

[2] S Avidan and M Butman. Blind vision. In *ECCV*, 2006.

[3] Y-C Chang and C-J Lu. Oblivious polynomial evaluation and oblivious neural learning. In *Advances in Cryptology*, 2001.

[4] B Goethals, S Laur, H Lipmaa, and T Mielikainen. On private scalar product computation for privacy-preserving data mining. In *ICISC 2004*.

[5] LR Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proc of the IEEE*, 77(2), 1989.

[6] MVS Shashanka and P Smaragdis. Secure sound classification: Gaussian mixture models. In *Proc. of ICASSP*, 2006.

[7] J Vaidya and C Clifton. Privacy preserving k-means clustering over vertically partitioned data. In *KDD 2003: 206-215*.

[8] A. C-C. Yao. Protocols for secure computation. In *IEEE Symposium on Foundations of Computer Science*, 1982.