PRIVACY PRESERVING PROBABILISTIC INFERENCE WITH HIDDEN MARKOV MODELS

Manas Pathak, Shantanu Rane†, Wei Sun† and Bhiksha Raj**

* Carnegie Mellon University, Pittsburgh, PA 15213, USA
† Mitsubishi Electric Research Laboratories, Cambridge, MA 02139, USA

ABSTRACT

Alice possesses a sample of private data from which she wishes to obtain some probabilistic inference. Bob possesses Hidden Markov Models (HMMs) for this purpose, but he wants the model parameters to remain private. This paper develops a framework that enables Alice and Bob to collaboratively compute the so-called forward algorithm for HMMs while satisfying their privacy constraints. This is achieved using a public-key additively homomorphic cryptosystem. Our framework is asymmetric in the sense that a larger computational overhead is incurred by Bob who has higher computational resources at his disposal, compared with Alice who has limited computing resources. Practical issues such as the encryption of probabilities and the effect of finite precision on the accuracy of probabilistic inference are considered. The protocol is implemented in software and used for secure keyword recognition.

***Index Terms*—** Homomorphic Encryption, Hidden Markov Model, Forward Algorithm, Speech Recognition

1. INTRODUCTION

This paper presents a privacy-preserving framework for probabilistic inference based on Hidden Markov Models (HMMs). Classification based on HMMs is common in machine learning and is nearly ubiquitous in applications related to speech processing. We are particularly interested in a multi-party scenario in which the data and the HMMs belong to different individuals and cannot be shared. For example, Alice wants to analyze speech data from telephone calls. She outsources the speech recognition task to Bob, who possesses accurate HMMs obtained via extensive training. Alice cannot share the speech data with Bob owing to privacy concerns while Bob cannot disclose the HMM parameters because this might leak valuable information about his own training database.

Probabilistic inference with privacy constraints is a relatively unexplored area of research. To our knowledge, the only detailed treatment of privacy-preserving probabilistic classification appears in [1]. In that work, inference via HMMs is performed on speech signals using existing cryptographic primitives. Specifically, the protocols are based on repeated invocations of privacy-preserving two-party maximization algorithms, in which both parties incur exactly the same protocol overhead. In contrast, we present an asymmetric protocol that is more suitable for client-server interaction; the thin client encrypts the data and provides it to a server which performs most of the computationally intensive tasks. Further, HMM-based probabilistic inference involves probabilities and real numbers which must be approximated for encrypted-domain processing. Accordingly, we consider the effect of finite precision, underflow and exponentiation in the ciphertext domain.

*This work was carried out when Manas Pathak was an intern at Mitsubishi Electric Research Laboratories. Manas Pathak was partially supported by National Science Foundation Grant No. 1017256.

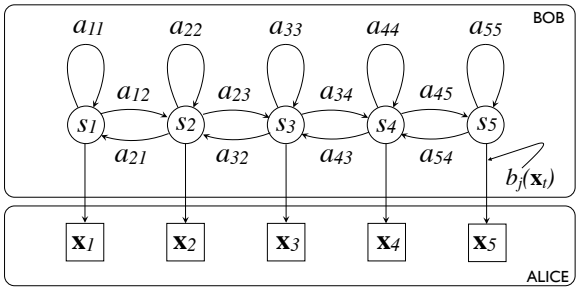

Fig. 1. An example of a 5-state HMM. In this work, Alice (client) possesses the data or features extracted from the data to be classified, while Bob (server) possesses a privately trained HMM.

The remainder of this paper is organized as follows: Section 2 reviews the tools that will be used for secure probabilistic inference in this paper, viz., Hidden Markov Models (HMMs), homomorphic encryption and oblivious transfer. Section 3 describes a privacy-preserving protocol to implement the probabilistic inference using the so-called forward algorithm of HMMs. An application to secure speech recognition is presented in Section 4.

2. REVIEW OF HMM AND SECURE PRIMITIVES

2.1. Hidden Markov Models

A HMM, depicted in Fig. 1, can be regarded as a generalization of a Markov chain in which the state is not directly visible but generates an output which can be observed. The outputs are also referred to as observations. Since observations depend on the hidden state, an observation reveals information about the underlying state.

By definition [2], a Hidden Markov model is a triple $\lambda = (A, B, \Pi)$, in which

- $A = (a_{ij})$ is the state transition matrix. Thus, $a_{ij} = \Pr\{q_{t+1} = S_j | q_t = S_i\}, 1 \leq i, j \leq N$, where $\{S_1, S_2, ..., S_N\}$ is the set of states and q_t is the state at time t;
- $B = (b_j(v_k))$ is the matrix containing the probabilities of the observations. Thus, $b_j(v_k) = \Pr\{\mathbf{x}_t = v_k | q_t = S_j\}, 1 \leq j \leq N, 1 \leq k \leq M$, where $\{v_1, v_2, ..., v_M\}$ is the alphabet of observation symbols, and $\mathbf{x}_t$ is the observation at time t;
- $\Pi = (\pi_1, \pi_2, ..., \pi_N)$ is the initial state probability vector, that is, $\pi_i = \Pr\{q_1 = S_i\}, i = 1, 2, ..., N$.

For a given sequence of observations $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$ and a HMM $\lambda = (A, B, \Pi)$, one problem of interest is to efficiently compute

the probability $\Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T|\lambda\}$. A well-known solution to this problem is the so-called forward algorithm.

2.2. Forward Algorithm of HMMs

Define $\alpha_t(j) = \Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, q_t = S_j|\lambda\}$, the joint probability of an observation sequence in state j at time t. The forward algorithm proceeds as follows:

1. Initialize $\alpha_1(j) = \pi_j b_j(\mathbf{x}_1)$, $1 \leq j \leq N$;
2. For each $1 \leq j \leq N$, compute

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(\mathbf{x}_{t+1})$$

for all $1 \leq t \leq T-1$; This is called the induction step.

3. Obtain $\Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T|\lambda\} = \sum_{j=1}^{N} \alpha_T(j)$.

In the problem considered here, Alice possesses the observation sequence $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$ and Bob has the HMM λ. They must execute the forward algorithm without revealing their information to each other, such that Alice can obtain $\Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T|\lambda\}$.

2.3. Homomorphic Cryptosytems

For any two messages m_1, m_2 and an additive homomorphic encryption function $\xi(\cdot)$, the additive homomorphic property ensures that $\xi(m_1 + m_2) = \xi(m_1)\xi(m_2)$. Owing to this property, simple operations can be performed directly on the ciphertext, enabling some desirable manipulations on the underlying plaintext messages. The protocols that we construct in this paper can be executed with any additively homomorphic cryptosystem. To give a concrete example and implementation, we use the Paillier cryptosystem [3], which is reviewed briefly below.

- **Configuration:** Choose two large prime numbers p, q, and let $N = pq$. Denote by $\mathbb{Z}^*_{N^2} \subset \mathbb{Z}_{N^2} = \{0, 1, ..., N^2 - 1\}$ the set of non-negative integers that have multiplicative inverses modulo N^2. Select $g \in \mathbb{Z}^*_{N^2}$ such that $\gcd(L(g^\lambda \mod N^2), N) = 1$, where $\lambda = \text{lcm}(p-1, q-1)$, and $L(x) = \frac{x-1}{N}$. Let (N, g) be the public key, and (p, q) be the private key.
- **Encryption:** Let $m \in \mathbb{Z}_N$ be a plaintext. Then, the ciphertext is given by

$$\xi_r(m) = g^m \cdot r^N \mod N^2 \tag{1}$$

where $r \in \mathbb{Z}^*_N$ is a number chosen at random.

- **Decryption:** Let $c \in \mathbb{Z}_{N^2}$ be a ciphertext. Then, the corresponding plaintext is given by

$$\psi(\xi_r(m)) = \frac{L(c^\lambda \mod N^2)}{L(g^\lambda \mod N^2)} = m \mod N \tag{2}$$

Decryption works irrespective of the value of r used during encryption. Since r is chosen at random for every encryption, the Paillier cryptosystem is probabilistic, and therefore *semantically secure*, *i.e.* repeated encryptions of the same number result in different ciphertexts. It can now be verified that the following homomorphic properties hold for the mapping (1) from the plaintext set $(\mathbb{Z}_N, +)$ to the ciphertext set $(\mathbb{Z}^*_{N^2}, \cdot)$,

$$\psi(\xi_{r_1}(m_1)\xi_{r_2}(m_2) \mod N^2) = m_1 + m_2 \mod N \tag{3}$$

$$\psi([\,\xi_r(m_1)\,]^{m_2} \mod N^2) = m_1 m_2 \mod N \tag{4}$$

where $r_1, r_2 \in \mathbb{Z}^*_N$ and $r_1 \neq r_2$ in general. For simplicity, we omit the subscripted random parameter in the following sections.

2.4. 1-of-n Oblivious Transfer

Suppose that Bob has n messages $m_1, m_2, ...m_n$ and Alice has the index $1 \leq i \leq n$. Oblivious Transfer (OT) is a protocol to accomplish the following (a) Alice obtains m_i but discovers nothing about the other messages (b) Bob does not discover i. There are many known ways to implement OT [4]. It has been shown that OT is a sufficient primitive, i.e., it can be used for secure evaluation of any function, provided that function can be represented as an algebraic circuit. However, evaluating general functions using only OT is extremely complex in terms of computation and data transfer costs, so OT is used sparingly in this work.

3. PROTOCOL FOR SECURE FORWARD ALGORITHM

Assume that Alice (a client) sets up a private and public key pair for the Paillier encryption function $\xi(\cdot)$, and sends only the public key to Bob (a server). Assume that, in all calculations involving logarithms, the base of the logarithm is $g \in \mathbb{Z}^*_{N^2}$, which is the parameter used for Paillier encryption in Section 2. To avoid notational clutter, the base of the logarithm is not explicitly shown in the sequel.

3.1. Secure Logarithm Protocol

Input: Bob has $\xi(\theta)$
Output: Bob obtains the encryption $\xi(\log\theta)$ of $\log\theta$, and Alice obtains no information about θ.

1. Bob randomly chooses an integer β, and sends $\xi(\theta)^\beta = \xi(\beta\theta)$ to Alice
2. Alice decrypts $\beta\theta$, and then sends $\xi(\log\beta\theta)$ to Bob.
3. Bob computes $\xi(\log\beta\theta) \cdot \xi(-\log\beta) = \xi(\log\theta + \log\beta) \cdot \xi(-\log\beta) = \xi(\log\theta)$

Note: In general $\log\beta$ and $\log\beta\theta$ are not integers, and we resort to integer approximations when we implement the protocol. Specifically, in Step 2, Alice actually sends $\xi(\lfloor L\log\beta\theta\rfloor)$ to Bob, where L is a large number. For example, with $L = 10^6$, our logarithms are accurate to 6 decimal places. Similarly, in Step 3, Bob actually computes $\xi(\lfloor L\log\beta\theta\rfloor)\xi(-\lfloor L\log\beta\rfloor) = \xi(\lfloor L\log\beta + L\log\theta\rfloor - \lfloor L\log\beta\rfloor)$. Every multiplication by L is compensated by a corresponding division at every decryption step that Alice performs. The effect of these finite precision approximations is further discussed in Section 4.

3.2. Secure Exponent Protocol

Input: Bob has $\xi(\log\theta)$
Output: Bob obtains the encryption $\xi(\theta)$, and Alice obtains no information about θ.

1. Bob randomly chooses $\beta \in \mathbb{Z}^*_{N^2}$ and sends $\xi(\log\theta)\xi(\log\beta) = \xi(\log\theta + \log\beta) = \xi(\log\beta\theta)$ to Alice.
2. Alice decrypts $\log\beta\theta$, and then sends $\xi(\beta\theta)$ to Bob
3. Bob computes $\xi(\beta\theta)^{\frac{1}{\beta}} = \xi(\theta)$ where $\frac{1}{\beta}$ is the multiplicative inverse of β in $\mathbb{Z}^*_{N^2}$.

Note: As before, multiplication by a large number L followed by truncation is used to generate an approximation wherever the logarithm is involved. Thus, in Step 1 of our implementation of this protocol, Bob actually sends $\xi(\lfloor L\log\theta\rfloor)\xi(\lfloor L\log\beta\rfloor)$ to Alice. To compensate for the multiplication by L, Alice decrypts the transmission from Bob and divides by L in Step 2.

3.3. Secure LOGSUM Protocol

Input: Bob has $(\xi(\log\theta_1), \xi(\log\theta_2), \ldots, \xi(\log\theta_n))$ and the constant vector $(a_1, a_2, ..., a_n)$;
Output: Bob obtains $\xi(\log\sum_{i=1}^{n} a_i\theta_i)$, and Alice discovers nothing about the θ_i and a_i.

1. With Bob's input $\xi(\log\theta_1), \xi(\log\theta_2), \ldots, \xi(\log\theta_n)$, Alice and Bob execute the Secure Exponent protocol repeatedly so that Bob obtains $\xi(\theta_1), \xi(\theta_2), \ldots, \xi(\theta_n)$.
2. Bob exploits the additive homomorphic property of Paillier encryption to obtain

$$\xi\left(\sum_{i=1}^{n} a_i\theta_i\right) = \prod_{i=1}^{n} \xi(a_i\theta_i) = \prod_{i=1}^{n} \xi(\theta_i)^{a_i}$$

3. Bob and Alice execute the Secure Logarithm protocol, at the end of which Bob obtains the encryption $\xi(\log\sum_{i=1}^{n} a_i\theta_i)$.

3.4. Secure Forward Algorithm Protocol

Input: Alice has an observation sequence $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$. Bob has the HMM $\lambda = (A, B, \Pi)$.
Output: Bob obtains $\xi(\log\Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T|\lambda)\}$.

Write the matrix B as $[\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_N]$, where for each $j = 1, 2, ..., N$, $\mathbf{b}_j$ is a column vector with component $b_j(v_k)$, $k = 1, 2, ..., M$. Now, a privacy-preserving version of the forward algorithm for HMMs proceeds as follows:

1. For each $t = 1, 2, ..., T$ and $j = 1, 2, ..., N$, Bob randomly chooses γ_{tj} and generates a column vector $\log\mathbf{b}_j + \gamma_{tj}$.
2. Based on the input $\mathbf{x}_t$, Alice uses 1-of-M OT to obtain $\log b_j(\mathbf{x}_t) + \gamma_{tj}$.
3. Alice sends $\xi(\log b_j(\mathbf{x}_t) + \gamma_{tj})$ to Bob
4. Using γ_{tj} and the homomorphic property, Bob computes $\xi(\log b_j(\mathbf{x}_t) + \gamma_{tj}) \cdot \xi(-\gamma_{tj}) = \xi(\log b_j(\mathbf{x}_t))$ for $j = 1, 2, ..., N$, $t = 1, 2, ..., T$.
5. Bob computes $\xi(\log\alpha_1(j)) = \xi(\log\pi_j) \cdot \xi(\log b_j(\mathbf{x}_1))$ for $j = 1, 2, ..., N$
6. Induction Step: For $j = 1, 2, ..., N$, with Bob's input $\xi(\log\alpha_t(j))$, $j = 1, 2, ..., N$ and the transition matrix $A = (a_{ij})$, Alice and Bob run the secure LOGSUM protocol, at the end of which Bob obtains $\xi(\log\sum_{l=1}^{N}\alpha_t(l)a_{lj})$.
7. For all $1 \le t \le T-1$, Bob computes

$$\xi(\log\alpha_{t+1}(j)) = \xi(\log\sum_{l=1}^{N}\alpha_t(l)a_{lj}) \cdot \xi(\log b_j(\mathbf{x}_{t+1}))$$

8. Alice and Bob again run a secure LOGSUM protocol, so Bob obtains $\xi(\log\sum_{j=1}^{N}\alpha_T(j)) = \xi(\log P(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T|\lambda))$

3.5. Security Analysis

The Secure Logarithm, Secure Exponent and Secure LOGSUM protocols rely on multiplicative secret sharing to prevent Alice from discovering θ. Bob cannot discover θ because he does not possess the decryption key for the Paillier cryptosystem. The security of the Forward Algorithm protocol derives from the security of the previous three primitive protocols and the security of 1-of-M OT.

4. SECURE KEYWORD RECOGNITION

Consider the following scenario for privacy-preserving keyword recognition: Alice has a sampled speech signal, which she converts into T frames, where each frame is represented by a d-dimensional vector of Mel Frequency Cepstral Coefficients (MFCCs). Thus Alice possesses $\mathbf{x}_i$, $i = 1, 2, ..., T$ where each $\mathbf{x}_i \in \mathbb{R}^d$. Typically, $d = 39$ is used in practice. Derivation of MFCCs from speech signals is an established practice in the speech processing literature [5]. Bob possesses Δ different HMMs, each trained for a single keyword. Alice and Bob will execute a secure protocol, at the end of which, Alice will discover the keyword that is most likely to be contained in her speech sample.

Let a d-dimensional vector $\mathbf{y}$ of MFCCs have a multivariate Gaussian distribution, i.e., $b_j(\mathbf{y}) = \mathcal{N}(\mu_j, \mathbf{C}_j)$, where $j = 1, 2, ..., N$ indexes the state of a HMM λ. Let $\mathbf{z} = [\mathbf{y}^T, 1]^T$. Then, $\log b_j(\mathbf{y}) = \mathbf{z}^T\mathbf{W}_j\mathbf{z}$, where

$$\mathbf{W}_j = \left[\begin{array}{c|c} -\frac{1}{2}\mathbf{C}_j^{-1} & \mathbf{C}_j^{-1}\mu_j \\ \hline 0 & w_j \end{array}\right] \in \mathbb{R}^{(d+1)\times(d+1)}$$

and $w_j = \frac{1}{2}\mu_j^T\mathbf{C}_j^{-1}\mu_j - \frac{1}{2}\log|\mathbf{C}_j^{-1}| - \frac{d}{2}\log 2\pi$. The above treatment considers $\mathbf{y}$ to be a single multivariate Gaussian random variable, though an extension to mixture of multivariate Gaussians is also possible. Note that the matrix $\mathbf{W}_j$ is available to Bob. Further, note that $\log b_j(\mathbf{y})$ is a linear function of products z_iz_j where $i, j \in \{1, 2, ..., d+1\}$. This allows us to simplify the Secure Forward Algorithm Protocol of Section 3.4 as follows:

4.1. Simplified Secure Forward Algorithm

1. For each $t = 1, 2, ..., T$, Alice sets $\mathbf{z} = [\mathbf{x}_t^T, 1]^T$. Alice sends to Bob the encryptions of all z_iz_j with $i, j \in \{1, 2, ..., d+1\}$.
2. For each HMM state $j = 1, 2, ..., N$, Bob obtains the encryption $\xi(\log b_j(\mathbf{x}_t)) = \xi(\mathbf{z}^T\mathbf{W}_j\mathbf{z})$ using the additive homomorphic property.
3. Bob executes steps 5–8 from Section 3.4.

Now, the protocol for secure keyword recognition is as follows.

4.2. Protocol for Secure Keyword Recognition

Input: Alice has the MFCC feature vectors $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T$ corresponding to her privately owned speech sample. Bob has the database of HMMs $\{\lambda_1, \lambda_2, ..., \lambda_\Delta\}$ where each HMM λ_δ corresponds to a single keyword, which is denoted by τ_δ;
Output: Alice obtains $\delta^* = \arg\max_\delta \Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x_T}|\lambda_\delta\}$.

1. Alice and Bob execute the protocol of Section 4.1 for each HMM λ_δ, at the end of which Bob obtains $\xi(p_\delta) = \xi(\log\Pr\{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x_T}|\lambda_\delta\})$, $\delta = 1, 2, ..., \Delta$.
2. Bob chooses an order-preserving matrix[1] $R = (r_{ij})_{\Delta\times\Delta}$ with random coefficients. Using the additively homomorphic property of Paillier encryption, he computes the element-wise encryption given by $(\xi(p'_1), ..., \xi(p'_\Delta)) = (\xi(p_1), ..., \xi(p_\Delta)) \cdot R$. Bob sends the result to Alice.
3. Alice decrypts and obtains $\delta^* = \max_\delta p'_\delta = \max_\delta p_\delta$. This is true because R is order-preserving mapping.
4. Alice and Bob perform a 1-of-Δ OT, and Alice obtains the word τ_{δ^*}.

[1] See [6] for more details

4.3. Computational Complexity

Let us consider the computational overhead of the protocol in terms of the number of keywords (Δ), the number of HMM states (N), the time duration of the observation (T) and the size of each observation (d). Alice has an overhead of $O(d^2T)$ encryptions at the beginning of the protocol. This initial computational overhead is independent of the number of keywords and the size of the HMM. Bob has an overhead of $O(d^2N\Delta T)$ ciphertext multiplications and exponentiations in order to determine the $\xi(\log b_j(\mathbf{x}_t))$ terms. Finally, to determine the $\xi(\alpha_T(j))$ terms, Alice and Bob both have an overhead of $O(N\Delta T)$. Thus Alice's total protocol overhead is $O((d^2+N\Delta)T)$ while Bob's total overhead is $O(d^2N\Delta T)$, which means that Bob's overhead grows faster than Alice's.

4.4. Practical Issues: Fixed Precision and Numerical Underflow

The Paillier cryptosystem is defined over an integer ring but the speech data consists of real numbers. In our implementation, we used numbers accurate to 6 decimal places, which means that all floating point values were multiplied by a constant value 10^6 before encryption and divided by the same value after decryption. Furthermore, whenever there was a floating point number in the ciphertext exponent – of the form $\xi(a)^b$ – then, the fractional part was truncated, and only the integer portion of b was used in the calculation. These approximations introduce errors in the protocol which propagate during the iterations of the forward algorithm.

When implementing the protocol in software, another problem is that the value of $\alpha_t(j)$ gets progressively smaller after each iteration, and causes an underflow after a only a few iterations. This problem would not be encountered if the probabilities were always expressed in the form $\log \alpha_t(j)$. However, it is necessary to consider the actual value of $\alpha_t(j)$ as seen in Section 3.4. The underflow problem is resolved by normalizing $\alpha_t(1), \ldots, \alpha_t(N)$ in each iteration, such that the relative values are preserved. This necessitates a small change in the protocol, but owing to space constraints, these details are deferred to a future publication.

4.5. Experimental Analysis

We performed speech recognition experiments on a 3.2 GHz Intel Pentium 4 machine with 2 GB RAM and running 64-bit GNU/Linux. We created an efficient C++ implementation of the Paillier cryptosystem using the variable precision integer arithmetic library provided by OpenSSL. The encryption and decryption functions were used in a software implementation of the protocol of Section 3.4.

The experiment was conducted on audio samples of length 0.98 seconds each. The samples consisted of spoken words which were analyzed by ten HMMs each having 5 states. These 10 HMMs were assumed to be trained offline (by Bob) on ten different words, viz., utterances of "one", "two", "three", and so on up to "ten". Each test speech sample was converted into 98 overlapping vectors (called frames), a 39-dimensional feature vector composed of MFCCs were extracted from each frame. These MFCC vectors serve as the input observations $\mathbf{x}_i$ owned by Alice.

Times needed to process audio samples of length 0.98 seconds using the 10 HMMs were measured. Table 1 gives the processing times for different key lengths with Paillier homomorphic encryption. It is evident that stronger keys incur significant cost in terms of processing time. Note that, once Bob receives the encrypted input from Alice, the calculation of $\xi(\log b_j(\mathbf{x}_t))$ for all 10 HMMs may proceed in parallel. This is possible, for instance, in a cloud computing framework, where Bob has access to several server nodes.

Activity	256-bit keys	512-bit keys	1024-bit keys
Alice encrypts input data (Done only once)	205.23 s	1944.27 s	11045.2 s
Bob computes $\xi(\log b_j(\mathbf{x}_t))$ (Time per HMM)	79.47 s	230.30 s	460.56 s
Both compute $\xi(\alpha_T(j))$ (Time per HMM)	16.28 s	107.35 s	784.58 s

Table 1. Protocol execution times in seconds.

In order to observe the effect of errors due to finite precision, we compared the values of $\alpha_T(N)$ for a given 0.98-second speech sample derived using the privacy preserving speech recognition protocol as well as a conventional non-secure implementation which uses no encryption. This comparison of $\alpha_T(N)$ with and without encryption was performed for each of the 10 HMMs corresponding to the spoken words "one" through "ten". The relative error in $\alpha_T(N)$ with respect to the true values was found to be 0.52%.

5. CONCLUSION AND OUTLOOK

We described protocols that enable both practical privacy preserving inference and classification with HMMs. To our knowledge, our experiments constitute the first reported results on this topic. A natural progression is to extend the protocols to more complete inference, including decoding from large HMMs with privacy preservation. Other extensions include privacy preserving inference from generalizations of HMMs such as dynamic Bayes networks. We are currently investigating these possibilities. Applications for such technologies are myriad. One area where HMM technology is the basis for most applications, is the processing of voice, which is arguably the most personal and private medium of communication. Yet, most current voice processing systems require full, unobscured access to the voice of the user. The protocols proposed in this paper represent the first step towards developing a suite of privacy preserving technologies for voice processing, including applications ranging from keyword spotting, to biometrics and full speech recognition.

6. REFERENCES

[1] P. Smaragdis and M. Shashanka, "A Framework for Secure Speech Recognition," *IEEE Transactions on Audio, Speech and Language Processing*, vol. 15, no. 4, pp. 1404–1413, May 2007.

[2] L. Rabiner, "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, Feb. 1989.

[3] P. Paillier, "Public-Key Cryptosystems Based on Composite Degree Residuosity Classes," in *Advances in Cryptology, EUROCRYPT 99*. 1999, vol. 1592, pp. 233–238, Springer-Verlag, Lecture Notes in Computer Science.

[4] M. Naor and B. Pinkas, "Oblivious transfer with adaptive queries," in *Advances in Cryptology: CRYPTO'99*, Seattle, USA, 1999, vol. LNCS 1666, pp. 573–590, Springer.

[5] L. Rabiner and B. H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[6] S. Rane and W. Sun, "Privacy Preserving String Comparisons Based on Levenshtein Distance," in *Proc. IEEE international Workshop on Information Forensics and Security (WIFS)*, Seattle, USA, Dec. 2010.