Multi-Observer Privacy-Preserving Hidden Markov Models

Hung X. Nguyen and Matthew Roughan
School of Mathematical Sciences, The University of Adelaide, Australia.
E-mail:hung.nguyen, matthew.roughan@adelaide.edu.au

***Abstract*— Detection of malicious traffic and network health problems would be much easier if ISPs shared their data. Unfortunately, they are reluctant to share because doing so would either violate privacy legislation or expose business secrets. However, secure distributed computation allows calculations to be made using private data, without leaking this data. This paper presents such a method, allowing multiple parties to jointly infer a Hidden Markov Model (HMM) for traffic and/or user behaviour in order to detect anomalies. We extend prior work on HMMs in network security to include observations from multiple ISPs and develop secure protocols to infer the model parameters without revealing the private data. We implement a prototype of the protocols, and our experiments with the prototype show its has a reasonable computational and communications overhead, making it practical for adoption by ISPs.**

I. Introduction

As the Internet grows, and becomes more and more a part of the modern world's critical infrastructure, the issue of maintaining cyber-security confronts ISPs. There are many aspects to this problem, broadly falling under the categories of prevention, detection and mitigation. It may be ideal to prevent attacks before they can cause damage, but it is currently impossible to anticipate all possible attacks, and so detection of novel, or unexpected problems is necessary.

There is a now large literature on network anomaly detection (for examples see [7], [22]) aimed at detecting network problems. The range of techniques is large, but the general approach is to estimate the characteristics of the network under "normal" conditions, and then to look for substantial deviations from those characteristics. The principle differentiator in methods is the type of model used for normal conditions, and the method used to test for uncharacteristic behavior.

A number of authors have applied Hidden Markov Models (HMMs) to the task [2], [3], [17]. A Markov model is a simple stochastic process based on random, memoryless transitions through some series of states. In HMMs, we assume that the states themselves are not directly observable, but that we can make some indirect observations, and from these estimate the underlying process.

HMMs are among the most popular approaches for modelling time series data [13]. They have widespread applications in areas such as speech recognition, bio-informatics, and Internet traffic modelling [10], [15], [18], and more pertinently to finding problems in networks [2], [3], [17]. Once we have trained such a model, we can then look for unlikely sequences of observations, and use these to signal anomalies.

There is no doubt that anomaly detection in general, and HMMs in particular, benefit from having as large a dataset as possible from as diverse a set of viewpoints as possible. In the Internet, these might take the form of observations from multiple ISPs who are all interested in detecting large-scale problems such Distributed Denial of Service (DDoS) attacks, worms, or address hijacking. However, such collaboration between ISPs is rare. The problem is that the type of data that must be shared is often considered sensitive; either because it contains business secrets, or customer or other data that has legal privacy requirements.

There are various approaches one could imagine to solve this problem. Here we apply an approach called variously Secure Multi-party Computation (SMC), or Privacy-Preserving Data Mining (PPDM). It has advantages over alternatives in that no-one (not even a "trusted" third party) ever learns the private data of another party. Moreover, there is no "anonymized" residue, i.e., unanonymised components of the data (needed in the analysis) that can be used to break the anonymization of the rest of the data. In effect, the entire computation is encrypted so that no participant learns anything except the desired answer.

In this work, we solve the problem of learning a HMM from observations made by multiple distributed and independent parties. The observations themselves are private; no-one can learn any one else's observations. However, there is almost no loss of fidelity as might be experienced in some anonymization schemes. The solution we obtain deviates only slightly from that if the data were completely public, provided some care is put into choice of key length and scaling coefficients.

Our solution performs computation in the encrypted domain to protect data privacy. Although data encryption and decryption introduce computation and communication overheads, we show in our experiments that even running on commodity hardware our prototype implementation can be used for realistic applications.

II. Background

We start by briefly introducing the elements of HMMs and SMC used in this paper.

A. HMM Theory

A Markov chain is a sequence of random variables $Q = q_1 \dots q_T$ with the Markov property: given the present state, the future and past states are independent. Consider a Markov

chain with N possible states $\mathcal{S} = \{s_1, \ldots, s_N\}$. The Markov property is formally defined as

$$\mathbb{P}(q_{t+1} = s_i | q_1, q_2, \ldots, q_T) = \mathbb{P}(q_{t+1} = s_i | q_t = s_j).$$

If the states of the Markov process are not directly observed, but rather we see some output sequence that is probabilistically associated with the Markov chain, the process is referred to as a Hidden Markov Model (HMM) [13]. A HMM is formally defined by the quintuple

- the set of N states $\mathcal{S} = \{s_1, \ldots, s_N\}$,
- the set of M observation symbols $\mathcal{V} = \{v_1, \ldots, v_M\}$,
- the initial probability

$$\boldsymbol{\pi} = (\pi_1, \ldots, \pi_N), \text{ where } \pi_i = \mathbb{P}(q_1 = s_i),$$

- the time-independent state transition probability

$$\boldsymbol{A} = (a_{ij})_{N \times N}, \text{ where } a_{ij} = \mathbb{P}(q_{t+1} = s_j | q_t = s_i),$$

- the time-independent observation probability

$$\boldsymbol{B} = (b_{ik})_{N \times M}, \text{ where } b_{ik} = \mathbb{P}(O_t = v_k | q_t = s_i).$$

HMMs have been used successfully in detecting network problems [2], [3], [17]. In these applications, a HMM is built for normal traffic conditions. For example, HMMs are built in [2] for application protocols and in [3] for SSH traffic. Using these models, we can quickly evaluate the probability of an observed traffic stream. When an anomaly occurs, the likelihood of the observations will drop, and the deviation can be detected.

To apply HMMs to network problems, we need to solve the following two basic problems:

- *The training problem*, which is the task of estimating the model parameters $\boldsymbol{w} = \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}\}$ from some sequence of observations $\mathcal{O}$. We could also estimate the set of states $\mathcal{S}$ and the set of observation symbols $\mathcal{V}$. This training problem is crucial for most applications of HMMs, and so has been extensively studied, and good algorithms to solve this problem are available.
- *The evaluation problem*, which is the task of taking an observation sequence $\mathcal{O} = O_1 O_2 \ldots O_T$, and a model $\boldsymbol{w} = \{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}\}$, and efficiently computing the probability $\mathbb{P}(\mathcal{O}|\boldsymbol{w})$. The probability reflects how well a given model matches the observations and is the key to using HMMs to detect network problems. For example, one can compute the likelihood of an event and compare it with a threshold. If the likelihood is lower than this threshold, it would trigger an alarm.

Solutions to these problems have long been known [13] for the single observer case. The solutions are, as a result, centralized.

The HMM we consider here differs in two ways. Firstly, we allow multiple observers: that is, we consider that there is a single set of states $\mathcal{S}$ and transitions of those states governed by $\boldsymbol{A}$, but there are multiple sequences of observations of these states, each potentially with its own observation matrix $\boldsymbol{B}^{(i)}$. This type of model could be subsumed in a standard HMM by increasing the size of $\mathcal{V}$. However, our second modification to the standard HMM is to assume that the parties are unwilling to share their individual observation sequences, so the solution must be (i) privacy preserving; and (ii) distributed.

Examples where data need to be combined from multiple ISPs have been studied in the past [8], [21]. Multiple observers can improve detection probabilities without a corresponding increase in false alarm rate as a tradeoff.

Our main interest in this paper is in the distributed solution to the HMM inference problems with multiple observers such that the data of one party is protected from other parties: the data could be pooled at a trusted third party; or it could be anonymized before sending to other parties. Neither of these solutions have been widely adopted in the Internet. Trusted third parties are not common in this domain, and anonymization (e.g., [6]) has technical limitations in that part of the data is lost (the part that is anonymized), but some residue remains (that part that is used in some computation). The problems in anonymization are not trivial [4], and the approach is rarely used except to allow scientific researchers access to certain datasets, often with legal machinery designed to prevent reverse engineering of the data.

The solution we are pursuing in this research is to allow the ISPs to jointly compute the HMMs without revealing the private data. This problem falls into the general class of secure multi-party computation problems, which we discuss next.

B. Introduction to Secure Multi-Party Computation

Secure Multi-party Computation (SMC) is a field of cryptography that provides means to perform arbitrary computations between multiple parties who are concerned with protecting their data. Mathematically, there are m parties $P_1, \ldots, P_m$. Each party has private data x_i. They want to compute a joint function $(y_1, \ldots, y_m) = f(x_1, \ldots, x_m)$. The goal is to compute f without P_i learning anything about x_j or y_j for all $j \neq i$, other than what can be inferred form their own data x_i and output y_i.

The field of SMC originated from the work of Yao [19] on the Millionaire Problem: two millionaires want to find which one has a larger fortune, without revealing their wealth to each other. Yao later showed that any polynomial time function could be computed in a secure distributed manner but the method is computationally expensive [20]. A great deal of following work has considered how to do so efficiently.

There is now a substantial literature on secure distributed computation and data mining. The parts most relevant to this work include work on applications to network management [5], [14], and anomaly detection using principle component analysis [9]; and application of SMC to HMMs [12], [16]. However our work is quite different from the last two cases, which consider the situation where one party holds the model, and the other the observations. That problem was relevant for a particular application, but has no obvious connection with network management. Instead, in our problem the observations are partitioned between the different parties.

We apply here the now standard *semi-honest* security model. In this model, the parties in the computation will follow the

protocol correctly, but may perform additional (polynomial time) computations to attempt to learn additional information. Many of the techniques we use here have been extended to deal with adversaries who are willing to corrupt the protocol itself leading to invalid results, however, the resulting approaches are then more complex, and have larger overhead that is usually unnecessary for data-mining applications where it is in everyone's interest to find the correct solution. There is a subsidiary issue of free-loading — participants who don't contribute to the calculation — but this is outside the scope of this paper, as free-loading is difficult to discover without recourse to the original measurements. We plan to solve this problem in future work using the same types of techniques espoused here.

The approach we take here is to use a number of well developed ideas from SMC as building blocks or *primitives* to create the algorithm we need. The challenge is to do so efficiently, and without intermediate data leaking useful information. The principle primitive used here is homomorphic encryption, which we describe in detail because the details matter. Homomorphic encryption, and related techniques operate on a field of integers, whereas most HMM estimation algorithms assume floating point numbers, and so we must perform scaling as part of the implementation. We also briefly describe the other techniques used in this paper (secure logsum and secure negation).

1) Homomorphic Encryption: A homomorphic encryption scheme is a special type of public-private crypto-system with the property that some operation in the plain-text are mirrored by operations in the cipher-text. In practice, that means we can perform computations on the encrypted text, e.g.,

$$x \oplus y = D\big[E[x] \otimes E[y]\big],$$

for some operators $\otimes$ and $\oplus$. The homomorphic encryption scheme we use here is the Paillier crypto-system [11], which we detail below.

- *Key generation:* Choose two large prime numbers p and q randomly and independently such that $\gcd(pq, (p-1)(q-1)) = 1$. Let $n = pq$ and $\lambda = \text{lcm}(p-1, q-1)$. Let $\mathbb{Z}^*_{n^2} \subseteq \mathbb{Z}_{n^2} = \{0, 1, \ldots, n^2 - 1\}$ be the set of non-negative integers that have multiplicative inverse modulo n^2. Select a random integer $g \in \mathbb{Z}^*_{n^2}$ such that $\gcd(L(g^\lambda \bmod n^2), n) = 1$ where $L(u) = \frac{u-1}{n}$ with the division being the quotient of $u-1$ divided by n. The public key is (n, g) and the private key is (p, q) or equivalently λ.
- *Encryption:* For a plain-text $s \in \mathbb{Z}_n$, select a random nonce[1] $r \in \mathbb{Z}_n$. The cipher text is then

$$E[s; r] = g^s \cdot r^n \bmod n^2.$$

- *Decryption:* The decryption algorithm for cipher-text $c < n^2$, is

$$D[c] = \frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n,$$

such that

$$D\big[E[s; r]\big] = s.$$

The following homomorphic properties hold for the Paillier crypto-system

$$\begin{aligned} D\big[E[s_1; r_1] \times E[s_2; r_2] \bmod n^2\big] &= s_1 + s_2 \bmod n, \\ D\big[E[s_1; r_1]^{s_2} \bmod n^2\big] &= s_1 \times s_2 \bmod n, \end{aligned}$$

so we can add or multiply numbers (mod n) by multiplying or exponentiating their encrypted values, respectively. The latter operation is not perfectly privacy preserving, as s_2 is in the clear, however, using this in combination with other SMC techniques allows the flexibility we need for our application.

The other primitives we use are:

- *Secure logsum:* This is a simple protocol that uses the homomorphic properties of the Paillier crypto-system. Consider two parties A and B where A has a vector of encrypted values $(E[\log(x_1)], \ldots, E[\log(x_m)])$, and B holds the decryption key. A and B want to jointly compute $E[\log \sum_{i=1}^m a_i x_i]$ for a public vector $(a_1, \ldots, a_m)$. Algorithms for this protocol appear in [12], [16].
- *Secure negation:* We need to be able to allow two parties A and B to compute the encrypted negation $E[-a]$ of an encrypted value $E[a]$. Our protocol for doing so is a simple application of the homomorphic properties of the Paillier crypto-system.

  Input: Party A has encrypted value $E[a]$ and the public encryption key, and B has the private decryption key.

  Output: Party A obtains $E[-a]$ and B learns nothing.

  1: Party A generates a random number R, uniformly distributed in $\{0, \ldots, n-1\}$ and computes

$$E[a + R] = E[a] \cdot E[R].$$

  2: Party A sends $E[a + R]$ to B.

  3: Party B decrypts $E[a + R]$ to obtain $a + R$ and then sends back $E[-a - R]$.

  4: Party A computes $E[-a] = E[-a - R] \cdot E[R]$.

III. HMMs with Multiple Observers

A. Problem Statement

Assume that we have a HMM with parameters $\{\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}\}$ as described in section II-A. There are m parties denoted by $P_1, P_2, \ldots, P_m$ that make observations of the same underlying Markov process. Without loss of generality, assume that the observations are made between time 0 and T, and that the time interval is divided into T slots $[1, \ldots, T]$. Each party makes its own observations of the system and these observations are secret. At each time slot, we have m observations of the underlying HMM. The observation set now becomes

$$\mathcal{O} = \{\mathcal{O}_1, \ldots, \mathcal{O}_T\},$$

where each element $\mathcal{O}_t$ is a vector $\mathcal{O}_t = \{O_{1t}, \ldots, O_{mt}\}$ of observations from each party. The sequence of T observations that party P_j makes is denoted as $\mathcal{O}^{(j)} = \{O_{j1}, \ldots, O_{jT}\}$. An example of this model is given in Figure 1.

[1]A *nonce* is a number used once to prevent replay and similar attacks. Its value is inconsequential as long as it is chosen randomly.

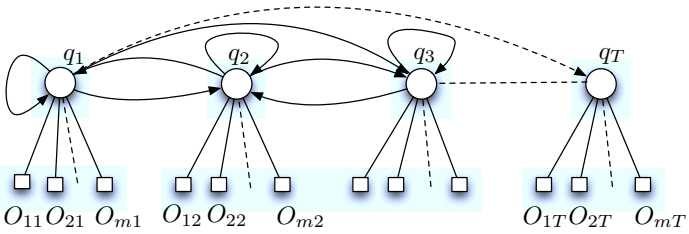

Fig. 1. A HMM with multiple observers.

Note that in real measurements, a party may not make a measurement on a particular time slot. The HMM can be easily extended to this case by including a $null$ state in $\mathcal{V}$ to represent "no observation" [21].

We shall assume that the observations of the different parties are independent. That is a natural assumption, as dependence between observations would weaken the need for privacy. Moreover, in this paper we assume all parties have the same observation probability given by the matrix $\boldsymbol{B} = \{b_{ik}\}$, though we plan to extend our model to the heterogeneous case in future work. Under these conditions the probability of a set of observations at time t, conditional on the state of the Markov process is given by

$$\mathbb{P}(\mathcal{O}_t|q_t = s_i) = \prod_{j=1}^{m} \mathbb{P}(O_t^{(j)} = v_{k_j}|q_t = s_i) = \prod_{j=1}^{m} b_{ik_j}.$$

We are interested in solving the training and evaluation problems for HMMs as defined in Section II-A in a privacy preserving manner. That is, the parties jointly compute the likelihood of the observations and the HMM parameters in such a way that at the end of the computation all parties learn the correct parameter value but do not learn anything more about the data of the other parties other than those that can be directly inferred from the output and their inputs. The solution requires two separate extensions to both the HMM algorithms and the privacy-preserving computation techniques.

B. Solution of HMMs with multiple observers

We present first our modification to the centralized algorithms for HMMs in [13] to handle multiple observers.

The forward procedure: We are interested in computing $P(\mathcal{O}|\boldsymbol{w})$, the likelihood of the observations $\mathcal{O}$ given the parameters $\boldsymbol{w} = (\boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$. Define the forward variables

$$\alpha_t(i) = \mathbb{P}(\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_t, q_t = s_i|\boldsymbol{w}),$$

i.e., $\alpha_t(i)$ is the probability of partial observation sequence from 1 to t and the state of the process at time t is $q_t = s_i$, given the model parameters $\boldsymbol{w}$.

The following forward algorithm computes the $\alpha_t(\cdot)$ in linear time:

1) Initialization:
$$\alpha_1(i) = \pi_i \prod_{j=1}^{m} b_{ik_j},$$
where $k_j \in [1, \ldots, N]$, $O_{j1} = v_{k_j}$ and b_{ik_j} is the probability that P_j observes v_{k_j} given that the state of the HMM at time 1 is s_i.

2) Induction: for $t \in [2, T]$ compute
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} \prod_{l=1}^{m} b_{jk_l}, \tag{1}$$
where $O_{jt} = v_{k_l}$ and b_{jk_l} is the probability that P_l observes v_{k_l} given the state of the HMM at time t is s_j.

Here, as elsewhere, the only difference from the standard estimation algorithm for HMMs is the inclusion of multiple observers. Once we have the $\alpha_t(i)$ we can compute the likelihood $P(\mathcal{O}|\boldsymbol{w}) = \sum_{i=1}^{N} \alpha_T(i)$.

The Baum-Welch algorithm: There is no known method to effectively select parameters $\boldsymbol{w}$ to globally maximize the probability of the observed sequence. However, the Baum-Welch algorithm provides a local maximum that is often sufficient. It proceeds as follows.

We first define a backward variables $\beta_t(i)$, analogously to the forward variable $\alpha_t(i)$, as

$$\beta_t(i) = \mathbb{P}(\mathcal{O}_{t+1}\mathcal{O}_{t+2}\ldots\mathcal{O}_T|q_t = s_i, \boldsymbol{w}), \tag{2}$$

i.e., $\beta_t(i)$ is the probability of partial observation sequence from $t+1$ to the end, given that the state of the process is $q_t = s_i$ at time t and the model parameters are $\boldsymbol{w}$.

We can write $\beta_t(i)$ for $1 \le t < T$ inductively as

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} \prod_{l=1}^{m} b_{jk_l} \beta_{t+1}(j), \tag{3}$$

where $O_{jt} = v_{k_l}$ and b_{jk_l} is the probability that P_l observes v_{k_l} given the state of the HMM at time $t+1$ is s_j.

We also need the following variables $\xi_t(i,j)$ – the probability of being in state s_i at time t and state s_j at time $t+1$ given the observations, i.e.,

$$\xi_t(i,j) = \mathbb{P}(q_t = s_i, q_{t+1} = s_j|\mathcal{O}, \boldsymbol{w}). \tag{4}$$

The $\xi_t(i,j)$ can be computed from the forward and backward variables $\alpha_t(\cdot)$ and $\beta_t(\cdot)$ as

$$\xi_t(i,j) \quad = \quad \frac{\alpha_t(i) a_{ij} \prod_{l=1}^{m} b_{ik_l} \beta_{t+1}(j)}{\mathbb{P}(\mathcal{O}|\boldsymbol{w})}, \tag{5}$$

where k_l is the index of the tth observation by Party P_l, i.e., $O_{lt} = v_{k_j}$.

Then $\gamma_t(i)$, the probability of being in state s_i at time t given the observation sequence and the model parameters, is simply the sum of $\xi_t(i,j)$ over all j:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{6}$$

Summation of $\gamma_t(i)$ over time can be interpreted as the expected number of transitions from state s_i, and summation of $\xi_t(i,j)$ over all time slots can be interpreted as the expected number of transitions from state i to state j:

$$\sum_{t=1}^{T-1} \gamma_t(i) \quad = \quad \text{expected number of transitions from } s_i,$$

$$\sum_{t=1}^{T-1} \xi_t(i,j) \quad = \quad \text{expected number of transitions from } s_i \text{ to } s_j.$$

Once we have calculated $\gamma_t(i)$ and $\xi_t(i,j)$, we can estimate the initial state and state transition probabilities $\boldsymbol{\pi}, \boldsymbol{A}$ as:

$$\overline{\pi}_i = \gamma_1(i), \quad \text{and} \quad \overline{a}_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}. \tag{7}$$

Estimation of the observation probabilities $\boldsymbol{B}$ is based on the the ratio between the expected number of times the process is in state s_i and the expected number of times v_k is observed, however, our formula is slightly different from the standard HMM in that we must take into account observations from multiple observers:

$$\overline{b}_{ik} = \frac{\sum_{t=1}^{T} \gamma_t(i) \sum_{j}^{m} I\{O_{jt} = v_k\}}{m \sum_{t=1}^{T} \gamma_t(i)}, \tag{8}$$

where $I\{O_{jt} = v_k\}$ is the indicator function:

$$I\{O_{jt} = v_k\} = \begin{cases} 1, & \text{if } O_{jt} = v_k, \\ 0, & \text{otherwise,} \end{cases}$$

i.e., the function is 1 when the observation made by party P_j at time t is v_k.

IV. Privacy Preserving Protocols

A. Protocol for Secure Forward Algorithm

The first step we consider is deriving a secure version of the forward algorithm. Our protocol encrypts data and performs computation in the encrypted domain using homomorphic encryption to protect privacy of the parties.

We present the protocol for the two party case ($m = 2$). It could obviously be extended to the multi-party $m > 2$ case, but we may also be able to substantially reduce the overhead in that case by using, for instance, secure summation protocols in place of homomorphic encryption. Hence we describe the two party case here, and leave the multi-party case for future development.

Denote the two parties P_1 and P_2 and assume that P_2 sets up a private and public key pair for the Paillier crypto-system and sends the public key to P_1. The secure two-party forward protocol is presented in Protocol 1.

Party P_2 sends only encrypted data to P_1, and P_1 only sends P_2 the data in the secure logsum protocol at steps 12 and 17, which is known to be secure [12], [16], so the overall protocol is secure. The protocol's performance is dominated by the logsum protocol, which is called $NT+1$ times. Each execution of the logsum protocol requires P_1 to send $2N$ integers to P_2, where there are N states in the Markov chain. The total communication cost is therefore $O\left(N^2 T \log n\right)$, where n is the largest possible integer given the key-size.

The computational cost depends on the cost for encryption/decryption operations, which is implementation dependent. We explore this cost in our tests of the implementation of the protocol in Section V.

Protocol 1 Secure Two-Party Forward Protocol

Input: Both parties P_1 and P_2 know the model $\{\mathcal{S}, \mathcal{V}, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{\pi}\}$. Each party has a set of private observations $\mathcal{O}^{(i)} = \{O_{i1}, \ldots, O_{iT}\}$. P_2 holds the public and private keys of a Paillier crypto-system on the field $\{0, 1, \ldots, n-1\}$, while P_1 knows only the public key.

Output: $P(\mathcal{O}|\boldsymbol{w}) = \sum_{i=1}^{N} \alpha_T(i)$.

1: **Initialization ($t = 1$):**
2: k_j is the index of the tth observation by Party j, i.e., $O_{jt} = v_{k_j}$ for $j = \{1, 2\}$.
3: **for** $i = 1, \ldots, N$ **do**
4: P_2 sends $E\left[\log(b_{ik_2})\right]$ to P_1,
5: P_1 computes

$$E[\log(b_{ik_1} b_{ik_2})] = E[\log(b_{ik_2})] \cdot E[\log(b_{ik_1})].$$

6: P_1 computes

$$E[\log(\alpha_1(i))] = E\left[\log(\pi_i)\right] \cdot E\left[\log(b_{ik_1} b_{ik_2})\right].$$

7: **end for**
8: **Induction:**
9: **for** $t = 2, \ldots T$ **do**
10: Repeat steps 2-5, replacing the time index with t, so that P_1 obtains $E[\log(b_{ik_1} b_{ik_2})]$ for $i = 1, 2, \ldots, N$.
11: **for** $j = 1, \ldots, N$ **do**
12: P_1 and P_2 use the secure logsum protocol to compute

$$E\left[\log \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right].$$

13: P_1 computes

$$E\left[\log \alpha_t(j)\right] = E\left[\log \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij}\right] \cdot E[\log b_{jk_1} b_{jk_2}],$$

14: **end for**
15: **end for**
16: **Termination:**
17: P_1 and P_2 use the secure logsum protocol to compute

$$E\big[\log \mathbb{P}(\mathcal{O}|\boldsymbol{w})\big] = E\left[\log\left(\sum_{i=1}^{N} \alpha_T(i)\right)\right].$$

18: P_2 decrypts the result, and transmit the value to P_1.

B. Protocol for Secure Baum-Welch Algorithm

We first need to compute the backward variable $\beta(\cdot)$ securely. The secure backward algorithm is directly analogous to the secure forward algorithms. Due to space constraints, we will omit the detail.

Once we know $E[\log \alpha_t(\cdot)]$ and $E[\log \beta_t(\cdot)]$, we can update the model parameters using the *secure Baum-Welch protocol* given in Protocol 2. The protocol uses the Paillier homomorphic encryption to protect data privacy and perform computation in the encrypted domain. As with the forward protocol, we present the case of two parties, for clarity, and omit the obvious generalization to $m > 2$.

The security of the protocol is guaranteed in the same way as the forward protocol. The protocol uses $N^2T + 4$ calls to the logsum protocol in Step 6, 9 and 10, so the total communications overhead is $O(N^3T \log n)$.

The protocol describes one iteration of the Baum-Welch algorithm. In practice, this is iterated in combination with the forward-backward algorithms until the log-likelihood $\log \mathbb{P}(\mathcal{O}|\boldsymbol{w})$ converges. In our case, all of the variables output from the algorithms are output/input as encrypted logs, and so iteration is straight forward, except we must also create a stopping condition for the algorithm. The easiest approach is to stop the algorithm after a fixed number of iterations (we found that 10 was satisfactory in our test cases), but a more general approach is to test whether the log-likelihood has converged by looking at ratios of values, which can be easily done by having P_1 compute the log-ratio (using secure negation), and giving this value to P_2 for decryption. Party P_2 can then terminate the algorithm when the procedure has converged. The advantage is that the algorithm may converge more quickly; the disadvantage is that the number of iterations itself may reveal some detail of the private data to one party or the other. The choice is a tradeoff of computation cost against security — in our experiments we use the most secure, fixed iteration approach.

V. Implementation and Results

In this section we describe our implementation of the secure protocols described in the preceding section. The code for our implementation is available at *www.hxnguyen.net*. It is written in Python, using the Paillier encryption scheme by Ivanov[2], which we have extended by adding the secure logsum and secure negation protocols. The HMM code was implemented on top of the standard HMM implementation by Hamiltom[3].

However, the code is not a trivial extension of these packages. The crucial implementation issue is that the encryption scheme applies to a finite field of non-negative integers, whereas the HMM algorithms were designed to work with potentially negative floating-point numbers. There is significant room for problems if the translation between these two domains is not performed carefully. We first describe the implementation issues, and then present experiment results for the performance of the algorithm.

A. Scaling for HMMs

The first implementation issue in HMM algorithms is the under-flow problem that arise when multiplying hundreds of small probabilities. These probabilities are very small for large number of samples (500 or more) and will cause under-flow problems when multiplied.

Our secure protocols work in the logarithmic domain and would therefore avoid the problem but for the secure logsum algorithm, which converts the logs back to the actual values. To avoid the issue, we apply the scaling procedure in [13], however, here it must work in the encrypted domain.

[2]*https://github.com/mikeivanov/paillier*

[3]*http://www.cs.colostate.edu/hamiltom/code.html*

Protocol 2 Secure Two-Party Baum-Welch Protocol

Input: The encrypted logarithmic of the forward and backward variables $E[\log \alpha_t(\cdot)]$, $E[\log \beta_t(\cdot)]$, and $E[\log \mathbb{P}(\mathcal{O}|\boldsymbol{w})]$. Each party has a set of private observations $\mathcal{O}^{(i)}$. P_2 holds the public and private keys of a Paillier crypto-system on the field $\{0, 1, \ldots, n-1\}$, while P_1 knows only the public key.

Output: The updated model parameters $\boldsymbol{A}, \boldsymbol{B}$, and $\boldsymbol{\pi}$

1: **for** $t \in [1, T]$ **do**
2: **for** $i = 1, \ldots, N$ **do**
3: **for** $j = 1, \ldots, N$ **do**
4: P_1 computes $E[\log \xi_t(i,j)]$ by taking the log of (5)

$$E[\log \xi_t(i,j)] = E[\log \alpha_t(i)] \cdot E[\log \beta_{t+1}(j)] \cdot E[\log a_{ij}] \cdot E[\log b_{ik_1} b_{ik_2}] \cdot E[-\log \mathbb{P}(\mathcal{O}|\boldsymbol{w})].$$

5: **end for**
6: P_1 and P_2 use the secure logsum to compute $E[\log \gamma_t(i)]$ from (6)

$$E[\log \gamma_t(i)] = E\left[\log \sum_{j=1}^{N} \xi_t(i,j)\right].$$

7: **end for**
8: **end for**
9: P_1 uses (7) to update the model parameters $E[\log \bar{\pi}_i] = E[\log \gamma_1(i)]$ and

$$E[\log \bar{a}_{ij}] = E\left[\log \sum_{t=1}^{T-1} \xi_t(i,j)\right] \cdot E\left[-\log \sum_{t=1}^{T-1} \gamma_t(i)\right].$$

10: P_1 then updates $E[\log \bar{b}_{ik}]$ as in (8), with $m = 2$

$$E[\log \bar{b}_{ik}] = E\big[\log \textstyle\sum_{j=1}^{2} \sum_{t=1}^{T} \gamma_t(i) I\{O_{jt} = v_k\}\big] \cdot E\big[-\log(2 \textstyle\sum_{t=1}^{T} \gamma_t(i))\big],$$

using the secure logsum and negation protocols.

For each $1 \leq t \leq T$, we first compute the forward variable $\alpha_t(i)$ as in the secure forward protocol, and then we multiply it by a scaling coefficient c_t in the encrypted domain, where

$$E[\log c_t] = E\left[-\log \sum_{i=1}^{N} \alpha_t(i)\right].$$

The forward and backward variables are then updated as:

$$E[\log \hat{\alpha}_t(i)] = E[\log \alpha_t(i)] \cdot E[\log c_t],$$

$$E[\log \hat{\beta}_t(i)] = E[\log \beta_t(i)] \cdot E[\log c_t].$$

The likelihood $\mathbb{P}(\mathcal{O}|\boldsymbol{w})$ is computed from c_t

$$E\big[\log \mathbb{P}(\mathcal{O}|\boldsymbol{w})\big] = E\left[-\sum_{t=1}^{T} \log c_t\right].$$

The Baum-Welch protocol remains unchanged under scaling.

B. Fixed point computation and negative numbers

The Paillier crypto-system is defined over a finite field $\mathbb{Z}_n = \{0, \ldots, n-1\}$. In our solutions for HMM we need to encrypt the HMM parameters, which are real numbers. We translate between floating-point numbers and non-negative integers by scaling and rounding off the values: a real number x is translated to integer $\bar{x} = \lfloor Lx \rfloor$, where $\lfloor x \rfloor$ is the largest integer $\leq x$. In our implementation, we incorporate this operation into the encryption and decryption. For instance, for $x \in I\!R$, we take

$$E'[x] = E\big[\bar{x}\big] = E\big[\lfloor Lx \rfloor\big], \text{ and } D'\big[E'[x]\big] = \bar{x}/L \simeq x.$$

Obviously this approximation will be more accurate for larger values of L. However, there is a problem in that the largest value of x that can be represented will be $(n-1)/L$, and in fact, to avoid overflow we must ensure that all values in calculations lie below this threshold. For large key-sizes this is not a significant issue, but could be for smaller keys.

The other important issue is that of encrypting negative numbers (remember we frequently use the log of probabilities, which will be negative). We use modulo n arithmetic here, and so negative numbers are represented by their modular additive inverse, i.e., $x < 0$ then

$$E'[\bar{x}] = E'[\bar{x} + n].$$

This changes the domain of the possible values we can work with to $x \in [-\frac{n}{2L}, \frac{n-1}{2L}]$. We will test various combinations of n and L in the following sections.

C. Experiment results

We construct a simplified HMM for the detection of SSH brute-force attacks in [3]. In this model, the HMM has two states where the attackers alternate between "attack" and "inactive", i.e. $\mathcal{S} = \{Attack, Inactive\}$, and there are 6 observation outputs representing possible the traffic counts $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, v_6\}$.

The following parameters are used for initial, transition and observation probabilities

$$\boldsymbol{\pi} = (0.01, 0.99),$$

$$\boldsymbol{A} = \begin{array}{c} \\ attack \\ inactive \end{array} \begin{array}{c} \begin{array}{cc} attack & inactive \end{array} \\ \begin{pmatrix} 0.95 & 0.05 \\ 0.05 & 0.95 \end{pmatrix} \end{array},$$

$$\boldsymbol{B} = \begin{array}{c} \\ attack \\ inactive \end{array} \begin{array}{c} \begin{array}{cccccc} v_1 & v_2 & v_3 & v_4 & v_5 & v_6 \end{array} \\ \begin{pmatrix} 0.1 & 0.1 & 0.1 & 0.1 & 0.1 & 0.5 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{pmatrix} \end{array}.$$

We simulate a set of 10 realizations for the HMM, for each set of parameter values, and run both the centralized (insecure) and secure distributed version of the estimation algorithms to compare results.

Key length (bits)	L	$\mathbb{P}(\mathcal{O}\|\boldsymbol{w})$	$\bar{\boldsymbol{\pi}}$	$\bar{\boldsymbol{A}}$	$\bar{\boldsymbol{B}}$
64	10^3	6.15 %	7.91 %	8.39 %	15.70 %
128	10^3	4.98 %	5.42 %	7.32 %	12.57 %
256	10^3	3.57 %	3.92 %	6.27 %	9.09 %
512	10^3	1.83 %	3.03 %	5.17 %	6.92 %
64	10^6	0.16 %	1.53 %	3.08 %	7.17 %
128	10^6	0.16 %	1.28 %	2.97 %	5.85 %
256	10^6	0.10 %	0.65 %	2.17 %	4.43 %
512	10^6	0.10 %	0.6 %	1.09 %	2.03 %

TABLE I

WORST-CASE ERRORS OVER 10 RUNS, GIVEN AS PERCENTAGES.

1) Accuracy of the secure protocols: The approximation of real numbers by integers in the Paillier crypto-system (see Section V-B) introduces errors. Here, we compare the accuracy of the secure protocols against results provided by the ideal result produced by a centralized algorithm. We compare the ideal result with the secure result using the relative error $\epsilon = |\hat{p} - p|/p$, where $\hat{p}$ is the result of the secure protocol, and p the ideal estimate. We evaluate the errors by varying the values of the scaling parameter L and the key length.

Space constraints prevent us presenting results for each parameter $\bar{\pi}_i, \bar{a}_{ij}$ and $\bar{b}_{ik}$ so we present, in Table I, the *worst case* errors over each of the estimated components. The results (given for $T = 1000$ samples) show that the further through the estimation process we go, the larger the errors, but that for large keys, and reasonable scaling parameters, the errors introduced by integer approximation and consequent over- or underflow are insignificant (2% in the worst case over multiple simulations, and parameter estimates). Larger keys also provide better security, so best performance occurs in the most secure case.

2) Runtime analysis: Another important consideration is the runtime of the protocol. We evaluate the runtime of our protocols by varying the key length and the number of samples used to evaluate and train the HMM, with $L = 10^6$. The results, generated on a laptop with a duo core 2.8 Ghz processor with 4GB of RAM are shown in Figure 2. The results show that the algorithm is approximately linear in the number of samples T, and quadratic as a function of key length. This quadratic growth is due to the runtime of the Paillier encryption and decryption functions, which could be more efficiently implemented. There is a clear trade-off between security and runtime as the longer the key length the more secure the protocol but also longer computation time.

The Baum-Welch protocol takes on average 5 times longer than the secure forward protocol for each iteration, so estimation/training component of running these algorithms represents a significant workload. However, detection of anomalies requires only the forward algorithm.

3) Collaboration benefits: The other obvious question to ask is whether there is an advantage in multiple parties collaborating. It is intuitive that a larger set of data is beneficial, but it may not be obvious that these benefits outweigh the costs involved in participating in such a protocol. Here we study these to allow potential collaborators to determine the cost/benefits.

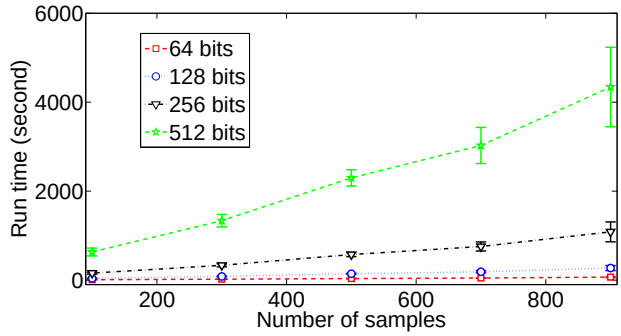

Fig. 2. Runtime for the secure forward protocol under different key-length.

Using the same model, we compare the errors as we increase the number of participants in the protocol. We apply the Baum-Welch algorithm to $T = 100$ samples and compare errors in the estimates. In particular, due to space restrictions, we focus on the estimates of $\boldsymbol{B}$, which we can see from Table I are the hardest to estimate accurately, and we calculate the Mean Squared Error (MSE) over the matrix.

The resultant MSE is shown in Figure 3. The plot shows that there is a substantial increase in accuracy when two parties collaborate, but that the marginal improvement lessens with increasing numbers of participants in the protocol.

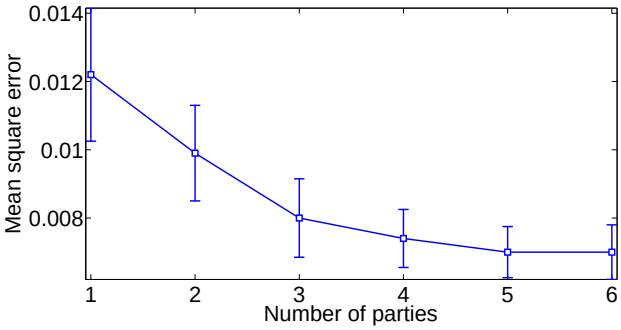

Fig. 3. MSE of the observation probabilities $\boldsymbol{B}$.

VI. Conclusion and Future Work

In this paper we have shown that collaboration between multiple parties can improve the quality of estimates provided by HMMs. More importantly, we have shown how the parties can collaborate without revealing private data to each other. In the context of ISPs, this would mean that multiple ISPs can help each other detect network problems without running the risk of exposing critical data to competitors.

We have implemented the protocol using Paillier's homomorphic encryption, and shown how several details such as the conversion between real and integer arithmetic can be handled. The implemented protocols are accurate and secure, and reasonably fast. However, the protocol comes at a cost. As with all security there is a computational and communications overhead in encryption. In the future we plan to study approaches to allow multiple parties to reduce this cost using more efficient algorithms.

There are other ways in which the protocol can be enhanced, for instance, we would use secure distributed protocols to prevent free-riding, which is hard to detect in the context of private data, and we aim to tackle this problem in future work.

References

[1] RIPE Network Coordination Center. `http://www.ripe.net/`.

[2] D. Ariu, G. Giacinto, and R. Perdisci. Sensing attacks in computers networks with Hidden Markov Models. In *Proc. of the Machine Learning and Data Mining in Pattern Recognition*, pages 449–463, 2007.

[3] C. Bartolini, L. Gaspary, A. Sperotto, R. Sadre, P.-T. de Boer, and A. Pras. Hidden Markov Model modeling of SSH brute-force attacks. In *Integrated Management of Systems, Services, Processes and People in IT*, pages 164–176. Springer Berlin / Heidelberg, 2009.

[4] M. Burkhart, D. Schatzmann, B. Trammell, E. Boschi, and B. Plattner. The role of network trace anonymization under attack. *SIGCOMM Comput. Commun. Rev.*, 40:5–11, January 2010.

[5] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos. SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics. In *USENIX Security Symposium*, Washington, DC, USA, August, 2010.

[6] J. Fan, J. Xu, M. H. Ammar, and S. B. Moon. Prefix-preserving IP address anonymization: measurement-based security evaluation and a new cryptography-based scheme. *Comput. Netw.*, 46:253–272, Oct. 2004.

[7] A. Lakhina, K. Papagiannaki, M. Crovella, C. Diot, E. D. Kolaczyk, and N. Taft. Structural analysis of network traffic flows. In *ACM SIGMETRICS / Performance*, 2004.

[8] X. Li, M. Parizeau, and R. Plamondon. Training Hidden Markov Models with multiple observations-a combinatorial method. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:371–377, April 2000.

[9] S. Nagaraja, V. Jalaparti, M. Caesar, and N. Borisov. P3CA: Private anomaly detection across ISP networks. In S. Fischer-Hübner and N. Hopper, editors, *Privacy Enhancing Technologies*, volume 6794 of *Lecture Notes in Computer Science*, pages 38–56. Springer Berlin / Heidelberg, 2011.

[10] H. X. Nguyen and M. Roughan. SAIL: Statistically Accurate Internet Loss Measurement. In *Proceedings of ACM Sigmetrics 2010 Conference*, New York, NY, June, 2010.

[11] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *ADVANCES IN CRYPTOLOGY - EUROCRYPT 1999*, pages 223–238. Springer-Verlag, 1999.

[12] M. Pathak, S. Rane, W. Sun, and B. Raj. Privacy preserving probabilistic inference with Hidden Markov Models. In *Proc. of ICASSP 2011*, 2011.

[13] L. R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. *Proc. of the IEEE*, 77(2):257–286, February 1989.

[14] M. Roughan and Y. Zhang. Secure distributed data-mining and its application to large-scale network measurements. *ACM SIGCOMM Computer Communication Review*, 36(1):7–14, January 2006.

[15] K. Salamatian and S. Vaton. Hidden Markov Model for network communcation channels. *Proceedings of ACM SIGMETRICS*, 2001.

[16] P. Smaragdis and M. Shashanka. A framework for secure speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 15(4):1404 –1413, May 2007.

[17] Y. Song, S. Stolfo, and T. Jebara. Markov models for network-behavior modeling and anonymization. In *Technical reports-Columbia University, http://hdl.handle.net/10022/AC:P:10682*, 2011.

[18] C. V. Wright, F. Monrose, and G. M. Masson. On inferring application protocol behaviors in encrypted network traffic. *J. Mach. Learn. Res.*, 7:2745–2769, December 2006.

[19] A. C. Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, SFCS '82, pages 160–164, Washington, DC, USA, 1982. IEEE Computer Society.

[20] A. C. Yao. How to generate and exchange secrets. In *Proceedings of the 27rd Annual Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[21] S.-Z. Yu and H. Kobayashi. A Hidden Semi-Markov Model with missing data and mulitple observation sequences for mobility tracking. *Elsevier Transations on Signal Processing*, 83:235–250, December 2003.

[22] Y. Zhang, Z. Ge, M. Roughan, and A. Greenberg. Network anomography. In *Proceedings of the Internet Measurement Conference (IMC '05)*, Berkeley, CA, USA, October 2005.