Privacy Preserving Speaker Verification using Adapted GMMs

Manas A. Pathak and Bhiksha Raj

Carnegie Mellon University, Pittsburgh, PA, USA

{manasp,bhiksha}@cs.cmu.edu

Abstract

In this paper we present an adapted UBM-GMM based privacy preserving speaker verification (PPSV) system, where the system is not able to observe the speech data provided by the user and the user does not observe the models trained by the system. These privacy criteria are important in order to prevent an adversary having unauthorized access to the user's client device from impersonating a user and also from another adversary who can break into the verification system can learn about the user's speech patterns to impersonate the user in another system. We present protocols for speaker enrollment and verification which preserve privacy according to these requirements and report experiments with a prototype implementation on the YOHO dataset.

Index Terms: speaker verification, secure biometrics

1. Introduction

Speaker verification systems attempt to authenticate a person's identity based on their voice. A person's voice and manner of speaking are biometric signatures, and an authentication based on these factors is expected to be impervious to attacks that text-based forms of authentication are susceptible to.

Speech based authentication is however, not entirely secure. The most obvious way of tricking the authentication process is by imitation. Imposters may attempt to imitate a subject's voice, or produce speech similar to the user's voice using methods such as playing out recordings of the user's voice, or morphing their own voice into the user's voice [1, 2]. This form of attack on speaker verification systems has been an area of significant research.

However, there are also other mechanisms by which a speaker verification system can be misled, or may mislead a user. For instance, the verification system may itself be *phishing*, *i.e.*, merely acting as a front to capture users' voice patterns when they register with the system. These voice patterns could then be used to impersonate users in other voice authentication services. Alternately, a malicious agent may break into a system and gain access to stored voice patterns and use them to generate fake voice data that could be used to impersonate users.

In this paper we propose an algorithmic solution that addresses these problems. We propose a *privacy preserving speaker verification* (PPSV) system that performs authentication without revealing the actual voice patterns to the system, either during enrollment or during authentication phases.

Our solution is based on two main ideas. Firstly, through the use of *secure multiparty computation* protocols [3] that enable the user and system to interact only on *encrypted* speech data which the system cannot decrypt and observe in plaintext at any point, we eliminate the possibility that the system could phish for a user's voice. Secondly, by storing only *encrypted* models trained on users' speech in the system, we also ensure that an adversary who breaks into the system obtains no useful information. We assume that one adversary does not gain access to both the user's client device and the system at the same time.

We envision a client-server model, where a user executes a client program on a computer, or a smartphone, or any similar networked computation device coupled with a public key cryptosystem. The user retains his private key while the public key is shared with the system. As in all authentication systems, the user enrolls with the system by providing speech samples, which it uses to build a model for the user. For authentication, the user later transmits a new speech sample which the system evaluates against its model. In both cases, the speech transmitted to the system by user is encrypted with the user's private key and cannot be decrypted by the system. The models retained by the system are also encrypted similarly. Unlike the *open* framework where the system has access to all speech data and models, the PPSV framework requires the user to perform a part of the computation.

It should be noted that we do not aim to design *superior* algorithms for speaker verification towards achieving better accuracy. We rather aim to create a mechanism to ensure the privacy of user's speech data and the models learned by the system while implementing an existing verification technique. Although we do build on prior work by ourselves and other researchers, such work is currently limited. While there has been a substantial body of work on general techniques for data processing with privacy constraints, privacy preserving speech processing is sill a nascent area of research. Smaragdis and Shashanka [4] propose protocols for training and evaluating Gaussian mixtures and hidden Markov models on speech data, under privacy constraints. Pathak, et al. [5] develop and implement an efficient protocol for privacy preserving HMM inference applied to isolated word recognition. In this paper we apply some of these techniques in our protocols for speaker enrollment and verification.

In the remainder of the paper we describe the proposed protocols for private enrollment and verification in the PPSV system. We implement our protocols based on the commonly used UBM-GMM based speaker verification algorithm described in [6] which we outline in Section 2.1. We present a brief introduction to homomorphic encryption, which forms the basic building block of our PPSV system, in Section 2.2. We describe the proposed protocols for privacy preserving speaker enrollment and verification in Section 3.

The operations on encrypted speech data motivated by our privacy requirements introduce a computational overhead as compared to the non-private system. In Section 4, we report the increase in execution time and the effect on accuracy in experiments with a prototype PPSV system over the YOHO

This work was supported by the NSF grant 1017256.

dataset [7]. Finally, in Section 5 we discuss the import of this work and future directions.

2. Preliminaries

2.1. Speaker Verification using GMMs

We use the text-independent speaker verification method based on adapted Gaussian mixture models (GMM) as the underlying technique. We outline the basic algorithm here; for further details please refer to [6].

In this procedure, speaker verification is performed using a likelihood ratio test. To authenticate a recording x given by a speaker, the system computes the probability of x using a model λ_s for the speaker and compares it to the probability computed from a *universal background model* (UBM) λ_U representing generic speech. Authentication uses the following rule:

$$\frac{P(\mathbf{x}|\lambda_s)}{P(\mathbf{x}|\lambda_U)} \begin{cases} \geq \theta & \text{accept speaker,} \\ < \theta & \text{reject speaker.} \end{cases} \quad (1)$$

Speech recordings are parameterized as sequences of Mel-frequency cepstral coefficients augmented by differences and double differences, *i.e.*, a recording $\mathbf{x}$ actually consists of a sequence of feature vectors $x_1, \ldots, x_T$. Both the speaker model λ_s and the UBM λ_{UBM} are assumed to be Gaussian mixture models that have the form:

$$P(x_t|\lambda_C) = \sum_j w_j^C \mathcal{N}\left(x_t; \mu_j^C, \Sigma_j^C\right),$$

where C is either s or U. $\mathcal{N}(x; \mu, \Sigma)$ is a Gaussian with mean μ and covariance Σ computed at x. μ_j^C and Σ_j^C are the mean and covariance of the j^{th} Gaussian.

The parameters of the UBM are learned from a collection of speech recordings from a large number of speakers, to represent the characteristics of a generic speaker. These parameters are learned using the expectation-maximization (EM) algorithm. The parameters of the model for a speaker are obtained by adapting the UBM to the speaker.

Model Adaptation

The UBM parameters are adapted to individual speakers using maximum *a posteriori* (MAP) estimation. It has been empirically established that speaker models obtained by MAP estimation significantly outperform the models trained directly on the enrollment data [6, 8].

The MAP estimation procedure comprises estimation of a *sample* estimate of the speaker's parameters, followed by interpolation with the UBM. Given set of enrollment speech samples $x_1, \ldots, x_T$, we first compute the *a posteriori* probabilities of the individual Gaussians in the UBM. For the i^{th} mixture component of the UBM,

$$P(i|x_t) = \frac{w_i^U \mathcal{N}(x_t; \mu_i^U, \Sigma_i^U)}{\sum_j w_j^U \mathcal{N}(x_t; \mu_j^U, \Sigma_j^U)}. \quad (2)$$

Similar to the M-step of EM, the *a posterior* probabilities are then used to compute the new weights, mean, and second moment parameters.

$$w_i' = \frac{1}{T}\sum_t P(i|x_t), \;\; \mu_i' = \frac{\sum_t P(i|x_t)x_t}{\sum_t P(i|x_t)},$$
$$\Sigma_i' = \frac{\sum_t P(i|x_t)x_t x_t^T}{\sum_t P(i|x_t)}. \quad (3)$$

Finally, the parameters of the adapted model $\lambda_s = \{\hat{w}_i^s, \hat{\mu}_i^s, \hat{\Sigma}_i^s\}$ are given by the convex combination of these new parameters and the UBM parameters as follows.

$$\hat{w}_i^s = \alpha_i w_i' + (1-\alpha_i) w_i^U, \;\; \hat{\mu}_i^s = \alpha_i \mu_i' + (1-\alpha_i)\mu_i^U,$$
$$\hat{\Sigma}_i^s = \alpha_i \Sigma_i' + (1-\alpha_i)\left[\Sigma_i^U + \mu_i^U \mu_i^{UT}\right] - \hat{\mu}_i^s \hat{\mu}_i^{sT}. \quad (4)$$

These adaptation coefficients α_i control the amount of contribution of the enrollment data relative to the UBM.

2.2. Homomorphic Encryption

Homomorphic encryption schemes allow for operations to be performed directly on encrypted data without requiring knowledge of their unencrypted values. If two data instances x and y are encrypted to $E[x]$ and $E[y]$ using a homomorphic encryption scheme, we can obtain the encryption of the result of an operation $\otimes$ performed on x and y by performing a different operation $\oplus$ directly on the encrypted versions of x and y, *i.e.* $E[x \otimes y] = E[x] \oplus E[y]$. This property is the foundation of our privacy preserving mechanisms.

In this work, we use the asymmetric key Paillier cryptosystem [9] satisfying additive homomorphism. Given two numbers x and y, the Paillier cryptosystem satisfies $E[x] \cdot E[y] = E[x+y]$ and as a corollary $E[x]^y = E[xy]$.

In our proposed protocols, the user retains the private and public key of the encryption and can both encrypt and decrypt data. The system, on the other hand, only receives the public key and can only encrypt data.

3. Speaker Verification Protocols

We now describe our privacy preserving protocols for enrolling and authenticating users. We use the following construction from [4]. The multivariate Gaussian $\mathcal{N}(x; \mu, \Sigma)$ computed on any d-dimensional vector x can be represented in terms of of a $(d+1) \times (d+1)$ matrix W.

$$\tilde{W} = \left[\begin{array}{c|c} -\frac{1}{2}\Sigma^{-1} & \Sigma^{-1}\mu \\ \hline 0 & w^* \end{array}\right],$$
$$\text{where } w^* = -\frac{1}{2}\mu^T\Sigma^{-1}\mu - \frac{1}{2}\log|\Sigma|. \quad (5)$$

This implies $\log \mathcal{N}(x; \mu, \Sigma) = \tilde{x}^T \tilde{W} \tilde{x}$, where $\tilde{x}$ is an extended vector obtained by concatenating 1 to x. As suggested by [5], we reduce this computation to a single inner product $\bar{x}^T W$, where the extended feature vector $\bar{x}$ consists of all pairwise product terms $\tilde{x}_i\tilde{x}_j \in \tilde{x}$ and W is obtained by unrolling $\tilde{W}$ into a vector. In this representation $\log \mathcal{N}(x; \mu, \Sigma) = \bar{x}^T \bar{W}$.

3.1. Private Enrollment Protocol

We assume that the system already has access to the UBM, λ_U trained on a collection of speech data. We require that the enrolling user should not gain access to the UBM as that would compromise system security. We assume that the computation of MFCC features from the speech is performed by the user's client program. In the discussion below and in the rest of the paper, by speech we refer to MFCC features computed on the user's speech input.

We propose the following protocol to adapt the system's UBM to the user's enrollment samples. We refer to individual Gaussian components in a mixture as $P(x|i)$ for conciseness.

Inputs:

(a) User has the MFCC feature vectors of the enrollment samples $x_1, \ldots, x_T$ and both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) System has the UBM $\lambda_U = W_i^U$ for $i = 1, \ldots, N$, mixing weight α, and the encryption key $E[\cdot]$.

Output: System has the encrypted adapted model $E[\lambda_s] = \{E[\hat{w}_i^s], E[\hat{\mu}_i^s], E[\hat{\Sigma}_i^s]\}$.

Computing the posterior probabilities:
For each t:

1. The user computes the extended vector $\bar{x}_t$ from x_t and sends the encrypted vectors $E[\bar{x}_t]$ to the system.
2. The system computes the encrypted log probabilities for each mixture component $E\left[\log w_i^U P(x_t|i)\right] = \sum_j E[\bar{x}_{t,j}]^{W_{i,j}^U} + E[\log w_i^U]$.
3. The *logsum* protocol [4] enables a party holding $E[\log x]$ and $E[\log y]$ to collaborate with another party who holds the private encryption key to obtain $E[\log(x+y)]$ without revealing x or y. The system participates with the user's client in the logsum protocol to obtain $E\left[\log \sum_i w_i^U P(x_t|i)\right]$.[1]
4. The system computes encrypted log posteriors: $E[P(i|x_t)]$ = $E\left[\log w_i^U P(x_t|i)\right]$ - $E\left[\log \sum_i w_i^U P(x_t|i)\right]$.
5. The *private exponentiation protocol* [10] enables a party holding $E[\log x]$ to collaborate with the party who holds the encryption key to obtain $E[x]$ without revealing x. The system then executes the private exponentiation protocol with the user to obtain $E\left[P(i|x_t)\right]$.

Learning w_i^s.

1. The system computes the encrypted value of w_i', $E[w_i']$ as $E\left[\sum_t P(i|x_t)\right] = \prod_t E\left[P(i|x_t)\right]$.
2. The system then computes the encrypted updated mixture weights as

$$E[\hat{w}_i^s] = E[w_i']^{\alpha/T} E[w_i^U]^{1-\alpha}.$$

Learning $\hat{\mu}_i^s$.

1. The system generates a random number r and homomorphically computes $E[P(i|x_t) - r]$. The system sends this quantity to the user.
2. The user decrypts this quantity and multiplies it by individual feature vectors x_t to obtain the vector $P(i|x_t)x_t - rx_t$. The user encrypts this and sends $E[P(i|x_t)x_t - rx_t]$ to the system along with encrypted feature vectors $E[x_t]$.
3. The system computes $E[P(i|x_t)x_t] = E[P(i|x_t)x_t] = E[x_t]^r + E[P(i|x_t)x_t - rx_t]$. It then computes $E[\sum_t P(i|x_t)x_t] = \prod_t E[P(i|x_t)x_t]$.
4. The *private division protocol* [10] enables a party holding $E[x]$ and $E[y]$ to collaborate with the party holding the encryption key to obtain $E[x/y]$ without revealing x or y.The system engages the user in a private division protocol with $E[\sum_t P(i|x_t)x_t]$ and $E[w']$ as inputs to obtain $E[\mu']$.
5. The system then computes the encrypted adapted mean as.

$$E[\hat{\mu}_i^s] = E[\mu_i']^{\alpha} E[\mu_i^U]^{1-\alpha}.$$

Learning $\hat{\Sigma}_i^s$.
This is similar to learning $\hat{\mu}_i^s$ and continues from Step 2 of that protocol.

1. The user multiplies $P(i|x_t) - r$ by the product of the feature vectors $x_t x_t^T$ to obtain $P(i|x_t)x_t x_t^T - r x_t x_t^T$.[2] The user encrypts this and sends $E[P(i|x_t)x_t x_t^T - r x_t x_t^T]$ to the system along with encrypted feature vectors $E[x_t x_t^T]$.
2. The system computes $E[r x_t x_t^T] = E[x_t x_t^T]^r$ and multiplies it to $E[P(i|x_t)x_t x_t^T - r x_t x_t^T]$ to obtain $E[P(i|x_t)x_t x_t^T]$. The system multiplies these terms for all values of t to obtain $E[\sum_t P(i|x_t)x_t x_t^T]$.
3. The system engages the user in the private division protocol with this encrypted sum and $E[w']$ as inputs to obtain $E[\Sigma_i']$.
4. The system and the user participate in a *private vector product protocol* [10] with input $E[\hat{\mu}_i^s]$ to obtain $E[\hat{\mu}_i^s \hat{\mu}_i^{sT}]$. The system then computes the encrypted updated covariance as

$$E[\hat{\Sigma}_i^s] = E[\Sigma_i']^{\alpha} E[\Sigma_i^U + \mu_i^U \mu_i^{UT}]^{1-\alpha} - E[\hat{\mu}_i^s \hat{\mu}_i^{sT}].$$

The encrypted model parameters obtained by the system cannot be directly used in the verification protocol. We can then compute the encrypted model matrix $E[\hat{W}_i^s]$ from the encrypted model parameters. We describe this procedure in the Appendix [10], for lack of space. At no point in the enrollment protocol, the system observes feature vectors x_t in plaintext and the adapted models obtained by the system are also encrypted.

3.2. Private Verification Protocol

Inputs:

(a) User has feature vectors $x_1, \ldots, x_T$ for a test sample and both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) System has $E[\lambda_s] = E[\hat{W}_i^s]$, for $i = 1, \ldots, N$, and the encryption key $E[\cdot]$.

Output: System obtains the score $E[\log P(x_1, \ldots, x_T|\lambda_s)]$.

1. The user computes the extended vectors $\bar{x}_t$ for all the original feature vectors x_t.
2. The system generates a random vector p and computes $E[\hat{W}_i^s] \cdot E[-p]$ to obtain $E[\hat{W}_i^s - p]$. The system sends this quantity to the user.
3. The user decrypts it and obtains $\hat{W}_i^s - p$. The user computes the inner product $\bar{x}_t^T(\hat{W}_i^s - p)\ \forall t$, and sends the encryption $E[\bar{x}_t^T(\hat{W}_i^s - p)]\ \forall t$ to the system along with encrypted extended vector $E[\bar{x}_t]$.
4. The system homomorphically computes the inner product $E[\bar{x}_t^T p] = \prod_k E[x_{t,k}]^{p_k}\ \forall t$. It then computes $E[\bar{x}_t^T \hat{W}_i^s] = E[\bar{x}_t^T(\hat{W}_i^s - p)] \cdot E[\bar{x}_t^T p]\ \forall i$. The system and the user then participate in the *logsum* protocol to obtain $E[\log P(x_t|\lambda_s)]$. The system finally computes $E[\log P(x_1, \ldots, x_T|\lambda_s)] = \prod_t E[\log P(x_t|\lambda_s)]$.

[1] An extended version of the paper with an Appendix containing all the supplementary protocols can be downloaded from `http://mlsp.cs.cmu.edu/publications/pdfs/ppsv.pdf`

[2] For efficiency purposes, only the diagonal terms of the product $x_t x_t^T$ can be included without having a significant impact on accuracy.

As the system has access to the UBM in plaintext, the user and the system can execute the private mixture of Gaussians evaluation protocol (MOG) given by [4]. We however observe paradoxically that the above protocol is substantially faster than MOG using unencrypted models. This is because in the above protocol, the user computes part of the inner products $\bar{x}_t^T \hat{W}_i^s$ in plaintext. We therefore repeat the above protocol with the encrypted UBM $E[W_i^U]$ to obtain the encrypted probability $E[\log P(x_1, \ldots, x_T|\lambda_U)]$. The system and the user finally execute the *compare protocol* [10], to privately compute if $\log P(x_1, \ldots, x_T|\lambda_U) > \log P(x_1, \ldots, x_T|\lambda_s) + \theta$ and the system uses this as the decision to authenticate the user.

Throughout this protocol, the system never observes x_t in plaintext. The various supplementary protocols require the system to transmit encrypted partial results to the user [10]. While doing so the system either adds or multiplies the values it sends to the user by a random number. This also prevents the user from learning anything about the partial results obtained by the system. Even after satisfying these privacy constraints, the system is able to make the decision on authenticating the user.

4. Experiments

We present the results of experiments on the privacy preserving speaker verification protocols described above. Since our basic adaptation algorithm itself is the same as that in [6] and can be expected to be more or less as accurate as it, the key aspect we need to evaluate is the computational overhead of the proposed protocols. We created a prototype implementation of the verification protocol in C++ and used the variable precision arithmetic libraries provided by OpenSSL [11] to implement the Paillier cryptosystem. We performed the experiments on a 2 GHz Intel Core 2 Duo machine with 3 GB RAM running 64-bit Ubuntu.

We trained a UBM with 32 Gaussian mixture components on the enrollment data in the YOHO dataset [7]. We adapted the UBM to individual speakers and performed the verification of a test speech sample containing 418 frames. We present the execution times for the verification protocol in Table 1 for Paillier encryption keys of sizes 256-bits and 1024-bits. In both the cases, the protocol resulted in the same final probability scores as the non-private verification algorithm up to 5 digits of precision. The non-private verification algorithm required 13.79 s on the same input.

Table 1: Execution time for the verification protocol.

Steps	**Time (256-bit)**	**Time (1024-bit)**
Encrypting $\bar{x}_t \ \forall t$	576.81 s	35747.82 s
Evaluating Adapted	407.21 s	7597.57 s
Evaluating UBM	same as adapted	same as adapted
Comparison	0.3 s	16.88 s
Total =$E[\bar{x}_t]$ + adapted + UBM + compare	1391.53 s $\sim$ 23 min	50959.84 s $\sim$ 14 hr, 9 min

Evaluation of the UBM using the MOG protocol [4] mentioned in Section 3.2 required 528.45 s and 8207.85 s using 256-bit and 1024-bit encryption, respectively which is slower than evaluation of the adapted model using the private verification protocol. We therefore use the same protocol to evaluate the UBM.

5. Discussion and Future Work

Both the enrollment and verification protocols can be shown to be secure. The system observes only encrypted speech data and hence cannot learn anything about the user's speech. During the exchanges required by the protocols, the user only observes additively or multiplicatively transformed data, and also cannot learn anything from it. Due to length restrictions of this paper, we defer both the descriptions of the protocols and the explanations relating to their security to the Appendix [10], which we encourage the reader to read. The proposed protocols are also found to give results which are same up to a high degree of precision compared to a non-private GMM adaptation based scheme.

Throughout this work, we assumed the user to be *semi-honest* in performing his/her part of the computation. In a realistic setting, an adversary impersonating a user can also send fake inputs in the intermediate steps of the protocol in order to trick the system into acceptance. Such a scenario needs to be analyzed in the malicious model [12], where the security of the protocol is ensured using zero-knowledge proofs. Privacy preserving computation is eventually limited by the computation requirements, the semi-honest privacy model alone causes a large computational overhead. Overlaying the protocols with zero knowledge proofs and other mechanisms to protect against malicious behavior further increase the overhead by many factors. One of the main directions of future work include reducing the overhead using techniques such as hardware accelerated encryption, which will result in a system that can be deployed and used feasibly in practice.

6. References

[1] B.L. Pellom and J.H.L. Hansen, "An experimental study of speaker verification sensitivity to computer voice-altered imposters," in *ICASSP*, 1999.

[2] David Sundermann, Harald Hoge, Antonio Bonaforte, Hermann Ney, Alan Black, and Shri Narayanan, "Text-independent voice conversion based on unit selection," in *ICASSP*, 2006.

[3] Andrew Yao, "Protocols for secure computations," in *FOCS*, 1982.

[4] Paris Smaragdis and Madhusudana Shashanka, "A framework for secure speech recognition," *IEEE Trans. on Audio, Speech & Language Processing*, vol. 15, no. 4, pp. 1404–1413, 2007.

[5] Manas Pathak, Shantanu Rane, Wei Sun, and Bhiksha Raj, "Privacy preserving probabilistic inference with hidden Markov models," in *ICASSP*, 2011.

[6] Frederic Bimbot *et. al.*, "A tutorial on text-independent speaker verification," *EURASIP Journal on Applied Signal Processing*, vol. 4, pp. 430–451, 2004.

[7] Joseph P. Campbell, "Testing with the YOHO CD-ROM voice verification corpus," in *ICASSP*, 1995, pp. 341–344.

[8] Douglas A. Reynolds, "Comparison of background normalization methods for text-independent speaker verification," in *Eurospeech*, 1997, vol. 2, pp. 963–966.

[9] Pascal Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *EUROCRYPT*, 1999.

[10] "Appendix," `http://mlsp.cs.cmu.edu/publications/pdfs/ppsv.pdf`.

[11] "Openssl," `http://www.openssl.org/docs/crypto/bn.html`.

[12] Murat Kantarcioglu and Onur Kardes, "Privacy-preserving data mining in the malicious model," *J. Information and Computer Security*, vol. 2, no. 4, pp. 353–375, 2008.

7. Appendix: Supplementary Protocols

Model Construction Protocol.
Inputs:

(a) User has encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$

(b) System has encrypted adapted model parameters $E[\lambda_s] = \{E[\hat{w}_i], E[\hat{\mu}_i], E[\hat{\Sigma}_i]\}$, for $i = 1, \ldots, N$, and the encryption key $E[\cdot]$.

Output: System has the encrypted adapted model matrices $E[\hat{W}_i]$, for $i = 1, \ldots, N$.

1. For each covariance matrix $\hat{\Sigma}_i$, the system generates a random number q and homomorphically multiplies it with $E[\hat{\Sigma}_i]$ and sends the result $E[q\hat{\Sigma}_i]$ to the user.
2. The user decrypts this quantity and obtains the matrix $q\hat{\Sigma}_i$. The user then computes the reciprocal matrix $\frac{1}{q}\hat{\Sigma}_i^{-1}$. The user encrypts this matrix and sends $E[\frac{1}{q}\hat{\Sigma}_i^{-1}]$ to the system.
3. The system homomorphically multiplies the encrypted matrix by $-\frac{q}{2}$ to get the encrypted matrix $E[-\frac{1}{2}\hat{\Sigma}_i^{-1}]$.
4. The user also computes the log determinant of the matrix $\frac{1}{q}\hat{\Sigma}_i^{-1}$ to obtain $-\frac{1}{q}\log|\hat{\Sigma}_i|$, which the user encrypts and sends to the system.
5. The system homomorphically multiplies the encrypted log determinant by $\frac{q}{2}$ to obtain $E[-\frac{1}{2}\log|\hat{\Sigma}_i|]$.
6. The system generates a random $d \times 1$ matrix r and homomorphically computes $E[\hat{\mu}_i - r]$ and sends it to the user.
7. The user decrypts this vector to obtain $\hat{\mu}_i - r$ and multiplies it with $\frac{1}{q}\hat{\Sigma}_i^{-1}$ computed in Step 3 to obtain $\frac{1}{q}\hat{\Sigma}_i^{-1}\hat{\mu}_i - \frac{1}{q}\hat{\Sigma}_i^{-1}r$. The user encrypts this and sends it to the system.
8. Using the matrix r and $E[\frac{1}{q}\hat{\Sigma}_i^{-1}]$ which was obtained in Step 4, the system homomorphically computes $E[\frac{1}{q}\hat{\Sigma}_i^{-1}r]$ and adds it homomorphically to the encrypted vector obtained from the user to get $E[\frac{1}{q}\hat{\Sigma}_i^{-1}\hat{\mu}_i]$. The system then homomorphically multiplies this by q to get $E[\hat{\Sigma}_i^{-1}\hat{\mu}_i]$.
9. The system executes the encrypted product protocol using $E[\hat{\Sigma}_i^{-1}\hat{\mu}_i]$ and $E[\hat{\mu}_i]$ as inputs to obtain $E[\hat{\mu}_i^T\hat{\Sigma}_i^{-1}\hat{\mu}_i]$ and multiplying it homomorphically with $-\frac{1}{2}$ to obtain $E[-\frac{1}{2}\hat{\mu}_i^T\hat{\Sigma}_i^{-1}\hat{\mu}_i]$.
10. Finally, the system homomorphically adds the encrypted scalars obtained above along with $E[\hat{w}_i]$ to get $E[w^*]$. The system thus obtains all the components necessary to construct $E[\hat{W}_i]$.

Logsum Protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has $E[\log z_i]$ for $i = 1, \ldots, N$, and the encryption key $E[\cdot]$.

Output: Bob has $E\left[\log \sum_i z_i\right]$

1. Bob generates a random number r and homomorphically computes
$$E[\log z_i - r] = E[\log z_i] - E[r].$$
He sends this to Alice.
2. Alice decrypts this quantity and exponentiates it to obtain $z_i e^{-r}$.
3. Alice adds these quantities to compute $e^{-r}\sum_i z_i$ and then computes the log to obtain $\log \sum_i z_i - r$.
4. Alice encrypts $E\left[\log \sum_i z_i - r\right]$ and sends it to Bob.
5. Bob homomorphically adds r to obtain the desired output.

$$E\left[\log \sum_i z_i\right] = E\left[\log \sum_i z_i - r\right] + E[r].$$

Private multiplication protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has two encrypted numbers $E[a]$ and $E[b]$ and the encryption key $E[\cdot]$.

Output: Bob has the encrypted product $E[ab]$.

1. Bob generates a random number r and homomorphically computes $E[a+r]$ which he sends to Alice. He similarly generates a random vector q and homomorphically computes $E[qb]$ which he also sends to Alice.
2. Alice decrypts both of these vectors and obtains $a + r$ and qb. She computes the product
$$qb(a + r) = qab + qar.$$
She encrypts this product to get $E[qab + qar]$ and sends it to Bob.
3. Using the encrypted number $E[a]$, Bob homomorphically computes $E[qar]$ as he already knows q and r. He homomorphically subtracts this from the encrypted product obtained from Alice to get $E[qab]$.
4. Finally, Bob divides $E[qab]$ homomorphically by $\frac{1}{q}$ to obtain the result $E[ab]$.

Private Division protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has two encrypted numbers $E[a]$ and $E[b]$ and the encryption key $E[\cdot]$.

Output: Bob has the encrypted quotient $E[\frac{a}{b}]$.

1. Bob generates a random number r and homomorphically computes $E[a+r]$ which he sends to Alice. He similarly generates a random vector q and homomorphically computes $E[qb]$ which he also sends to Alice.
2. Alice decrypts both of these numbers and obtains $a + r$ and qb. She divides $a + r$ by qb to get
$$\frac{a+r}{qb} = \frac{a}{qb} + \frac{r}{qb}.$$
She encrypts this quantity and sends it to Bob.
She also computes the reciprocal $\frac{1}{qb}$, encrypts it and sends it to Bob.

3. Using the number r, Bob homomorphically computes $E[\frac{r}{qb}]$. He homomorphically subtracts this from the first encrypted quantity obtained from Alice to get $E[\frac{a}{qb}]$.
4. Finally, Bob multiplies $E[\frac{a}{qb}]$ homomorphically by q to obtain the result $E[\frac{a}{b}]$.

Private Exponentiation protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has two encrypted numbers $E[x]$ and the encryption key $E[\cdot]$.

Output: Bob has the encrypted exponent $E[e^x]$.

1. Bob generates a random number r and homomorphically computes $E[x + \log r]$ which he sends to Alice.
2. Alice decrypts this number and obtains $x + \log r$, which she exponentiates to get re^x. She encrypts this quantity and sends it to Bob.
3. Bob homomorphically divides $E[re^x]$ he obtains from Alice to get $E[e^x]$.

Private Logarithm protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has two encrypted numbers $E[x]$ and the encryption key $E[\cdot]$.

Output: Bob has the encrypted log $E[\log x]$.

1. Bob generates a random number q and homomorphically computes $E[qx]$ which he sends to Alice.
2. Alice decrypts this number and obtains qx, which she uses to compute $\log x + \log q$. She encrypts this quantity and sends it to Bob.
3. Bob homomorphically subtracts $\log q$ from $E[\log x + \log q]$ to obtain $E[\log x]$.

Private vector product protocol.
Inputs:

(a) Alice has both encryption key $E[\cdot]$ and decryption key $E^{-1}[\cdot]$.

(b) Bob has two encrypted vectors $E[x]$ and $E[y]$ of the same length and the encryption key $E[\cdot]$.

Output: Bob has the encrypted product $E[x^T y]$

1. Bob generates a random vector q of the same length as x and homomorphically computes $E[x - q]$ which he sends to Alice. He similarly generates a random vector r and homomorphically computes $E[y - r]$ which he also sends to Alice.
2. Alice decrypts both of these vectors and obtains $x - q$ and $y - r$. She computes the product

$$(x - q)^T (y - r) = x^T y - x^T r - q^T y + q^T r.$$

She encrypts this product and sends it to Bob.

3. Using the vectors $E[x]$ and r, Bob homomorphically computes $E[x^T r]$ and using the vectors $E[y]$ and q, he computes $E[q^T y]$. He also computes $-q^T r$ and encrypts it to obtain $E[-q^T r]$.
4. By adding these products homomorphically to the encrypted product he obtained from Alice, Bob obtains $E[x^T y]$.

Compare protocol.
Inputs:

(a) Alice has the encryption keys $E_A[\cdot]$, $E_B[\cdot]$ and decryption key $E_A^{-1}[\cdot]$.

(b) Bob has $E_A[x]$ and $E_A[y]$ for and the encryption keys $E_A[\cdot]$, $E_B[\cdot]$ and decryption key $E_B^{-1}[\cdot]$.

Output: Bob knows if $x > y$.

1. Bob generates a random positive or negative number q and homomorphically computes $E_A[q(x-y)]$. He sends this quantity to Alice.
2. Alice decrypts this quantity to obtain $q(x - y)$. She generates a positive random number r and computes $qr(x - y)$ and encrypts this using Bob's key to obtain $E_B[qr(x - y)]$.
3. Bob decrypts this quantity and divides it by q. Bob checks for $r(x - y) > 0$ and uses that to conclude whether $x > y$.