

Privacy-Preserving Speech Processing

[Cryptographic and string-matching frameworks show promise]



Speech is one of the most private forms of communication. People do not like to be eavesdropped on. They will frequently even object to being recorded; in fact, in many places it is illegal to record people speaking in public, even when it is acceptable to capture their images on video [1]. Yet, when a person uses a speech-based service such as a voice authentication system or a speech recognition service, they must grant the service complete access to their voice recordings. This exposes the user to abuse, with security, privacy and economic implications. For instance, the service could extract information such as gender, ethnicity, and even the emotional state of the user from the recording—factors not intended to be exposed by the user—and use them for undesired purposes. The recordings may be edited to create fake recordings that the user never spoke, or to impersonate them for other services. Even derivatives from the voice are risky to expose. For example, a voice-authentication service could make unauthorized use of the models or voice prints it has for users to try to identify their presence in other media such as YouTube.

Privacy concerns also arise in other situations. For instance, a doctor cannot just transmit a dictated medical record to a

generic voice-recognition service for fear of violating the requirements of the Health Insurance Portability and Accountability Act of 1996; the service provider requires various clearances first. Surveillance agencies must have access to all recordings by all callers on a telephone line, just to determine if a specific person of interest has spoken over that line. Thus, in searching for “Jack Terrorist,” they also end up being able to listen to and thereby violating the privacy of “John and Jane Doe.”

The need to protect the privacy of users and their data is well recognized in other domains [2]. Any private information that can be gleaned by inspecting a user’s interaction with a system must be protected from prying eyes. To this end, techniques have been proposed in the literature for protecting user privacy in a variety of applications including e-voting, information retrieval, and biometrics. Yet, the privacy of voice has not been addressed until recently, and the issue of privacy of speech has been dealt with primarily as a policy problem [3], [4], and not as a technology challenge. In this article, we describe recent developments in privacy-preserving frameworks for voice processing. Here, we refer chiefly to secure pattern-matching applications of speech such as speech biometrics and speech recognition, rather than secure speech communication techniques, which have already been studied [5].

The goal of the described frameworks is to enable voice-processing tasks in a manner that ensures that no party, including the user, system, or a snooper, can derive undesired or unintended information from the transaction. This would imply, for instance, that a user may enroll in a voice-authentication system without fear that an intruder or even the system itself could capture and abuse his voice or statistical models derived from it. A surveillance agency could now determine if a crime suspect is on a telephone line, but would learn nothing if the speaker is an innocent person. Private speech data may be mined by third parties without exposing the recordings.

These frameworks follow two distinct paradigms. In the first paradigm [6]–[10], which we will refer to as the “cryptographic” paradigm, conventional voice-processing algorithms are rendered secure by computing them through secure operations. By recasting the computations as a pipeline of “primitives,” each of which can be computed securely through a combination of homomorphic encryption [11]–[14], secure multiparty computation (SMC) [15], and oblivious transfer [16], [17], we ensure that no undesired information is leaked by any party. In this paradigm, the accuracy of the basic voice-processing algorithm can remain essentially unchanged with respect to the original nonprivate version. The privacy requirements, however, introduce a computational and communication overhead as the computation requires user participation.

The second paradigm modifies voice-pattern classification tasks into a string-comparison operation [18], [10]. Using a combination of appropriate data representation followed by locality sensitive hashing (LSH) schemes both the data to be matched and the patterns they must match are converted to collections of bit strings, and pattern classification is performed by counting exact matches. The computational overhead of this string-comparison framework is minimal compared to the cryptographic framework. Moreover, the entire setup is noninteractive and secure, in the sense that no party learns anything undesired regardless of how they manipulate the data. This comes at the price of a slight loss of performance, since the modified classification mechanism is not as effective as conventional classification schemes.

A BRIEF PRIMER ON SPEECH PROCESSING

The speech processing applications we consider all deal with making inferences about the information in the recorded signal. Biometric applications attempt to determine or confirm the identity of the speaker of a recording. Recognition applications attempt to infer what was spoken in the recording. We present below a very brief description of the salient aspects of these applications as they pertain to this article. The description is not intended to be exhaustive; for more detailed information we refer readers to the various books and papers on the topics, e.g., [19].

In all cases, the problem of inference is treated as one of statistical pattern classification. Classification is usually performed

through a Bayes’ classifier. Let \mathcal{C} be a set of candidate classes to which a recording \mathbf{X} might belong. Let $P(\mathbf{X}|C)$ be the probability distribution of speech recordings \mathbf{X} from class C . $P(\mathbf{X}|C)$ is usually represented through a parametric model, i.e., $P(\mathbf{X}|C) \approx P(\mathbf{X}; \lambda_C)$ where λ_C are the parameters of the class C and are learned from data. Classification is performed as

$$\hat{C} = \arg \max_{C \in \mathcal{C}} \log P(\mathbf{X}; \lambda_C) + \log P(C), \quad (1)$$

where $P(C)$ represents the a priori bias for class C . Stated thus, the primary difference among the applications considered lies in the definition of the candidate classes in \mathcal{C} and the nature of the model $P(\mathbf{X}; \lambda_C)$.

Before proceeding, we note that the classifiers do not work directly from the speech signal. Instead, the signal is converted to a sequence of feature vectors, typically Mel-frequency cepstral coefficients (MFCC) [20]. To generate these, the signal is segmented into overlapping “frames,” each typically 25 ms wide, with an overlap of 15 ms between adjacent frames. From each frame, a vector of MFCC (or similar) features is derived, which may be further augmented with their temporal differences and double-differences. For our purposes, it suffices to know that when we refer to a speech recording, we actually refer to the sequence of feature vectors $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]$. For the privacy-preserving frameworks described later, we will assume that the user’s (client) device can compute these features. The conversion of signals to features by itself provides no privacy, since they can be used for pattern classification, and can even be inverted to generate intelligible speech signals [21].

BIOMETRIC APPLICATIONS:

IDENTIFICATION AND AUTHENTICATION

Biometric applications deal with determining the identity of the speaker. Here, the set \mathcal{C} in (1) is the set of candidate speakers who may have spoken in a recording. In addition, a “universal” speaker U , representing the aggregate of all speakers not otherwise in \mathcal{C} , is included in the set. U may be viewed as the “none-of the above” option—classifying a recording as U is equivalent to stating that the speaker is unknown. In speaker identification systems, \mathcal{C} comprises a collection of speakers S (possibly including U) for whom models λ_S are available. In speaker authentication, \mathcal{C} comprises only the speaker who has claimed to have spoken in the recording and the universal speaker U . $P(C)$ in (1) now becomes a tuning parameter to bias the classification towards specific speakers or U .

Typically, the individual feature vectors in any recording \mathbf{X} are assumed to be independent and identically distributed (i.i.d.) according to a Gaussian mixture model (GMM) [22]. Thus, for any speaker S in \mathcal{C} , $P(\mathbf{X}; \lambda_S)$ is assumed to have the form $P(\mathbf{X}; \lambda_S) = \prod_t P(\mathbf{x}_t; \lambda_S)$, where

$$P(\mathbf{x}_t; \lambda_S) = \prod_{k=1}^K w_k^S(\mathbf{x}_t; \mu_k^S, \Sigma_k^S). \quad (2)$$

**THE GOAL OF THE DESCRIBED
FRAMEWORKS IS TO ENABLE VOICE-
PROCESSING TASKS IN A MANNER THAT
ENSURES THAT NO PARTY CAN DERIVE
UNINTENDED INFORMATION FROM
THE TRANSACTION.**

Here K is the total number of Gaussians in $P(\mathbf{x}_t; \lambda_S)$, $\mathcal{N}()$ represents a Gaussian density, and w_k^S , μ_k^S , and Σ_k^S are the mixture weight, mean vector, and covariance matrix of the k th Gaussian in the mixture. The parameters of the GMM are expressed as $\lambda_S = \{w_k^S, \mu_k^S, \Sigma_k^S \mid k = 1, \dots, K\}$.

λ_U , the parameters of the GMM for “universal speaker” U are learned from a large collection of speech recordings from many speakers. λ_U is often called a universal background model (UBM). The GMM parameters λ_S for any speaker S are learned from a collection of recordings for the speaker, using the expectation-maximization algorithm. When the amount of training data for the speaker are limited, e.g., from enrollment recordings in an authentication system, and insufficient to learn GMM parameters robustly, they are learned by adapting the parameters of the UBM to the data from the speaker using an MAP estimation algorithm [23], [24]. Classification of speakers with the GMMs now proceeds as

$$\hat{S} = \arg \max_{S \in \mathcal{C}} \sum_{t=1}^T \log P(\mathbf{x}_t; \lambda_S) + \theta_S, \quad (3)$$

where θ_S encodes our prior biases as mentioned earlier.

RECOGNITION APPLICATIONS

In speech recognition systems, \mathcal{C} in (1) represents the collection of all possible word sequences that a person may say in the recording. Phrase spotting systems only include the specific phrases of interest, along with a background model, usually called a “garbage model” in this context, which represents the “none of the above” option. Isolated word recognition systems assume that only a single word was spoken in the recorded segment being analyzed; \mathcal{C} here comprises the vocabulary of words to recognize. In continuous speech recognition (CSR) systems, \mathcal{C} represents the set of all possible sentences a person may speak. This set can be very large, and even infinite in size. To make classification manageable, the set of all possible sentences is represented as a compact, loopy word graph, which conforms to a grammar or n -gram language model that embodies the a priori probabilities $P(C)$ [19].

The probability distribution $P(\mathbf{X}; \lambda_C)$ for each class C is usually modeled by a hidden Markov model (HMM). The theory of HMMs is well known; we only reiterate the salient aspects of it here [25], [19]. An HMM is a model for time-varying processes and is characterized by a set of states $\{s_1, \dots, s_M\}$ and an associated set of probability distributions. According to the model, the process transitions through the states according to a Markov chain. After each transition, it draws an observation vector from a probability distribution associated with its current state. The parameters characterizing the HMM for any class C are 1) the initial state probabilities $\Pi^C = \{\pi_i^C, i = 1, \dots, M\}$, where π_i^C represent the probability that at the first instant the process will be in state s_i ; 2) transition probabilities $A^C = \{a_{ij}^C, i = 1, \dots, M, j = 1, \dots, M\}$, which represent the probability that, given the process is in state s_i at any time, it will jump to s_j at the next transition, and 3) the set of state output probability distributions $\{P^C(\mathbf{x}_t; \Lambda_i^C)\}$ associated with each state. In speech recognition systems the

$P^C(\mathbf{x}_t; \Lambda_i^C)$ are generally modeled as Gaussian mixture densities: $P^C(\mathbf{x}_t; \Lambda_i^C) = \sum_k w_{i,k}^C \mathcal{N}(\mathbf{x}_t; \mu_{i,k}^C, \Sigma_{i,k}^C)$. Thus, the parameters for class C are $\lambda_C = \{\Pi^C, A^C, \Lambda^C\}$, where $\Lambda^C = \{\Lambda_i^C \mid i = 1, \dots, M\}$ and $\Lambda_i^C = \{w_{i,k}^C, \mu_{i,k}^C, \Sigma_{i,k}^C \mid k\}$.

To compute $P(\mathbf{X}; \lambda_C)$ for any class C , we must employ the following recursion, commonly known as the forward recursion. Here, the term $\alpha^C(t, i)$ represents the total probability that the process arrives at state i after t transitions and generates the partial sequence $\mathbf{x}_1, \dots, \mathbf{x}_t$, i.e., $\alpha^C(t, i) = P(\mathbf{x}_1, \dots, \mathbf{x}_t, \text{state}(t) = i; \lambda_C)$.

$$\begin{aligned} \alpha^C(1, i) &= \pi_i^C P^C(\mathbf{x}_1 \mid i) \\ \alpha^C(t, i) &= P(\mathbf{x}_t \mid i, C) \sum_j \alpha^C(t-1, j) a_{j,i}^C \quad \forall t > 1 \\ P(\mathbf{X}; \lambda_C) &= \sum_i \alpha^C(T, i), \end{aligned} \quad (4)$$

where T is the total number of feature vectors in \mathbf{X} .

$P(\mathbf{X}; \lambda_C)$ computed in the above manner considers all possible state sequences that the process may have followed to generate \mathbf{X} . It can be used in (1) for phrase spotting and isolated-word recognizers. In CSR, however, the classes are word sequences, which are collapsed into a compact word graph [26], and computing $P(\mathbf{X} \mid C)$ through (4) is not feasible for individual word sequences. Here, $P(\mathbf{X}; \lambda_C)$ is replaced by the probability of the most likely state sequence through the HMM for C , and the classification is performed as

$$\hat{C} = \arg \max_C \log P(C) + \max_s \log P(\mathbf{X}, \mathbf{s}; \lambda_C), \quad (5)$$

where \mathbf{s} is the state sequence followed by the process. Unlike classification based on forward probabilities, this estimation can be performed over the set of word sequences represented by a word graph: the word graph is composed into a large HMM by replacing each edge in the graph by the HMM for the word it represents [26]. The word sequence corresponding to the most probable state sequence through the resulting HMM is guaranteed to be identical to the one obtained by (5). Even in contexts outside CSR, (5) is frequently used as a generic substitute for (1), since it is more efficient to compute.

The term $\max_s \log P(\mathbf{X}, \mathbf{s} \mid C)$ in turn can be computed very efficiently using a dynamic programming algorithm known as the Viterbi algorithm, which implements the following recursion. Here $\Gamma^C(t, i)$ represents the log of the joint probability of $\mathbf{x}_1 \dots \mathbf{x}_t$ and the most probable state sequence that arrives at state s_i at time t

$$\begin{aligned} \Gamma^C(1, i) &= \log \pi_i^C + \log P(\mathbf{x}_1; \Lambda_i^C) \\ \delta^C(t, i) &= \arg \max_j \Gamma^C(t-1, j) + \log a_{j,i}^C \quad \forall t > 1 \\ \Gamma^C(t, i) &= \log P(\mathbf{x}_t; \Lambda_i^C) + \Gamma^C(t-1, \delta^C(t, i)) + \log a_{\delta^C(t, i), i}^C \\ \max_s \log P(\mathbf{X}, \mathbf{s} \mid C) &= \max_i \Gamma^C(T, i). \end{aligned} \quad (6)$$

$\delta^C(t, i)$ is a table of “backpointers” from which the most likely state sequence $\arg \max_s \log P(\mathbf{X}, \mathbf{s}; \lambda_C)$ can be determined by tracing backwards recursively as $s_T = \arg \max_i \Gamma^C(T, i)$,

$s_t = \delta^C(t+1, s_{t+1})$, where s_t is the t th state in optimal state sequence. In CSR systems, in particular, the backtraced optimal state sequence traces a path through the word graph and is used to determine the spoken word sequence.

PRIVACY ISSUES IN SPEECH PROCESSING

From the discussion of the previous section, the key component of all the above applications is the computation of the class score $\log P(X; \lambda_C)$. In conventional (nonprivate) implementations of these applications, the system providing the application has access to data X , which represents the user's voice. This enables it to manipulate or misuse the data, raising privacy concerns. The alternative, i.e., permitting the user to access the models λ_C , is equally unacceptable. In many situations, the models are the system's intellectual property. Furthermore, in voice-data mining situations, the system's models also represent the patterns it is searching for; revealing these to external parties may not be acceptable. A voice-based authentication system where the user himself has access to the models cannot be considered an effective authenticator.

Thus, for maximally protecting the user and the system from one another, the system must not learn the user's data X , while the user must not be able to infer the system's models λ_C . For the speech processing applications discussed above, this means that log probability terms computed from GMMs, forward probabilities, and best-state-sequence probabilities (also called Viterbi scores) must all be computed without revealing X to the system or λ_C to the user. An additional twist arises in the case of authentication systems, where the system must be prevented from making unauthorized use of the models it has for the user. In this case, the model λ_S must itself be in a form that can only be useful when the system engages with the appropriate user.

The above deals with the computation of scores for any class. But what about the final outcome of the classification? This, too, has an intended recipient. For instance, in most biometric applications it is the system that must receive the outcome of the classification; however, for recognition systems the user obtains the result. Thus, we also stipulate that the outcome is only revealed to the intended recipient. We refer to any computational mechanisms that enable the above requirements as private computation, as opposed to conventional nonprivate methods that make no guarantees to privacy. The next two sections describe two frameworks that enable such private processing.

THE CRYPTOGRAPHIC APPROACH

The cryptographic approach treats private speech processing as an instance of secure two-party computation [15]. Consider the

case where two parties, Alice and Bob, have private data a and b , respectively, and they want to compute the result of a function $f(a, b)$. Any computational protocol to calculate $f(a, b)$ is said to be secure only if it leaks no more information about a and b than what either party can gain from learning the result c . We assume a semihonest model for the parties where each party follows the protocol but could save messages and intermediate results to learn more about other's private data. In other words, the parties are honest but curious and will follow the agreed-upon protocol but will try to learn as much as possible from the data flow between the parties. We return to the issue of honesty later in this section.

We recast the conventional speech processing algorithms described in the section "A Brief Primer on Speech Processing"

**WE RECAST THE CONVENTIONAL
SPEECH PROCESSING ALGORITHMS
AS A PIPELINE OF PRIMITIVE
COMPUTATIONS AND SHOW
HOW TO EXECUTE EACH
PRIMITIVE IN A COMPUTATIONALLY
SECURE MANNER.**

as a pipeline of primitive computations, and show how to execute each primitive in a computationally secure manner, i.e., a computationally bounded participant should not be able to derive private information possessed by the other participant from the computation. The output of each stage of the pipeline is either distributed across both participants in the

form of random additive shares, or arrives at one of the participants in a locked form, where the other participant holds the key. Thus, both participants must interact to perform the computations until the final outcome of the overall computation arrives at the correct participant.

SECURE PRIMITIVES FOR SPEECH PROCESSING

To keep the discussion simple, we will consider that there are only two parties, Alice and Bob, engaged in the computation. The reader may consider Alice as the client or end user who possesses a speech signal to be analyzed while keeping it private from Bob, the remote system that has the model parameters, some or all of which must be kept private from Alice.

To enable private computation, we will utilize public-key homomorphic encryption schemes, which enable operations to be performed on encrypted data [11]. We keep the development simple by considering an additively homomorphic cryptosystem [12]–[14] with encryption function $E[\cdot]$. Such a cryptosystem satisfies $E[x] \cdot E[y] = E[x+y]$ and $(E[x])^y = E[xy]$ for integer messages x, y . For a discussion of other flavors of homomorphic cryptosystems, specifically multiplicatively homomorphic, two-disjunctive normal form (DNF) homomorphic and fully homomorphic cryptosystems, the reader is referred to a companion article in this issue [27].

We assume that the cryptosystem is semantically secure [28], i.e., by using fresh random parameters during encryption but not during decryption, a given plain text may be mapped to different ciphertexts every time encryption is performed. This makes the cryptosystem, and hence the protocols discussed, resilient to a chosen plain text attack (CPA). We

further assume that while Alice and Bob share a public encryption key, Alice is the only person with a decryption key that reverses $E[\cdot]$. This means that the system may encrypt data if needed, but only the end user may decrypt it. However, if required the situation can be reversed by mirroring the protocols, with relatively minor additional changes. Assume that Alice and Bob own n -length integer vectors \mathbf{x} and \mathbf{y} , respectively. In what follows, $E[\mathbf{x}]$ denotes a vector containing encryptions of the individual elements of \mathbf{x} , i.e., $(E[x_1], E[x_2], \dots, E[x_n])$.

ADDITIVE SECRET SHARING

Bob has an encrypted value $E[x]$. He wishes to share it as random additive clear-text shares with Alice. He chooses an integer b at random, and sends $E[x] \cdot E[-b] = E[x - b]$ to Alice. Bob retains b as his additive share. Alice decrypts $x - b = a$, which is her additive share. We will represent this operation in shorthand by saying Alice and Bob receive a and b such that $a + b = \text{SHARE}(x)$.

SECURE INNER PRODUCT (SIP)

Alice and Bob want to compute uninformative additive shares a, b , respectively, of the inner product $\mathbf{x}^\top \mathbf{y}$. There are several ways to achieve this [29]–[31], but for expository purposes, we focus on a simple approach using additively homomorphic functions [32]. Alice sends elementwise encryptions $E[x_i]$ to Bob. Bob computes $\prod_{i=1}^n E[x_i]^{y_i} = E[\sum_{i=1}^n x_i y_i] = E[\mathbf{x}^\top \mathbf{y}]$. Alice and Bob can then receive additive shares a and b of $E[\mathbf{x}^\top \mathbf{y}]$ as $a + b = \text{SHARE}(\mathbf{x}^\top \mathbf{y})$. Observe that Bob operates in the encrypted domain and cannot discover \mathbf{x} . Similarly, Alice does not know b , and hence cannot discover \mathbf{y} .

When Bob possesses only $E[\mathbf{y}]$ rather than \mathbf{y} as assumed above, it is still possible—using a trick involving additive secret sharing—for Alice and Bob to obtain additive shares of $\mathbf{x}^\top \mathbf{y}$. As a notational shorthand, when we invoke any variant of this protocol, we will just state that Alice and Bob obtain additive shares a, b , such that $a + b = \text{SIP}(\mathbf{x}, \mathbf{y})$.

SECURE LOGSUM (SLOG)

Suppose that $\mathbf{x} + \mathbf{y} = (\ln z_1, \ln z_2, \dots, \ln z_n)$. By a slight abuse of notation, denote the vector of the elementwise logarithms and elementwise exponents of the elements of \mathbf{x} as $\ln \mathbf{x}$ and $e^{\mathbf{x}}$, respectively. Alice and Bob wish to obtain uninformative additive shares, a and b such that $a + b = \ln(\sum_{i=1}^n z_i)$, which is the “logsum” operation that gives the protocol its name. We note that $\sum_{i=1}^n \ln z_i = \sum_{i=1}^n e^{x_i + y_i}$ and achieve the desired secret sharing using the following protocol [7]:

- 1) Alice chooses a at random. Then Alice and Bob compute additive shares q, s such that $q + s = \text{SIP}(e^{x-a}, e^y)$ using the SIP protocol above. Bob combines these shares to obtain the inner product ϕ .
- 2) Bob computes $b = \ln \phi = -a + \ln(\sum_{i=1}^n e^{x_i + y_i}) = -a + \ln(\sum_{i=1}^n z_i)$, which gives the desired result.

In the first step above, Alice and Bob employ additive secret sharing in the exponent, which is equivalent to multiplicative secret sharing. The parameter a should be chosen large enough because multiplicative secret sharing is not as secure as standard additive secret sharing. We present this protocol to illuminate the fact that homomorphic functions can be manipulated to compute useful, nonobvious functions. The same functionality can be obtained in a secure manner using other cryptographic primitives, e.g., garbled circuits. To refer to this protocol henceforth, we will state that Alice and Bob obtain additive shares a, b , such that $a + b = \text{SLOG}(\ln \mathbf{z})$.

In an alternate scenario, Bob possesses $E[\ln z_1], E[\ln z_2], \dots$. He wishes to obtain $E[\log \sum_i z_i]$. He uses the SHARE protocol to share $E[\ln z_i] \forall i$ with Alice and proceeds as earlier. Finally, Alice sends $E[a]$ to Bob who computes $E[\log \sum_i z_i] = E[a]E[b]$. Note that in the process Alice and Bob also obtain additive shares of the outcome. We will denote this operation by stating that Bob receives c such that $c = \text{SLOG}(\ln \mathbf{z})$.

SECURE MAXIMUM INDEX (SMI)

Alice and Bob wish to compute additive shares, such that $a + b = \text{argmax}_i x_i + y_i$ without revealing their data to each other. This is achieved by exploiting homomorphic encryption to perform blind-and-permute operations [33]. Let $\mathbf{z} = \mathbf{x} + \mathbf{y}$. Then, the goal is to privately compute additive shares of the index of the largest element of \mathbf{z} . To accomplish this, Alice

chooses a secret permutation π_A on the index set $\{1, 2, \dots, n\}$. Similarly Bob chooses a secret permutation π_B on the same set. The outcome of the blind-and permute protocol is that Alice and Bob

each obtain additive shares of $\pi_A \pi_B \mathbf{z}$. Neither can reverse the other person’s permutations. However, given the permuted shares, Alice and Bob can use repeated instantiations of the millionaire protocol [17], obtain the index of the maximum element in the permutation of \mathbf{z} , from which, using their respective permutations, they can obtain shares of the index of the maximum element in \mathbf{z} . To see the steps involved in minimum finding based on homomorphic encryption, refer to a companion article in this special issue [34]. Alternatively, the entire minimum finding protocol may be executed using garbled circuits [35]. As a shorthand notation, we say that Alice and Bob obtain additive shares a, b , such that $a + b = \text{SMI}(\mathbf{x}, \mathbf{y})$.

In an alternate scenario, Bob has two encrypted numbers $E[x]$ and $E[y]$ and desires to find which of the two is larger, he can engage in protocols such as those described in [38], which employ threshold encryption or secret sharing schemes in which both parties must cooperate to perform decryption such that one of them can obtain the answer without exposing x or y to either. We denote this also as $\max(x, y) = \text{SMI}(x, y)$; this should not result in confusion since the actual operation used will be clear from the context.

HOMOMORPHIC FUNCTIONS CAN BE MANIPULATED TO COMPUTE USEFUL NONOBVIOUS FUNCTIONS.

SECURE MAXIMUM VALUE (SMV)

Alice and Bob wish to compute uninformative additive shares a, b such that $a + b = \max_i x_i + y_i$. This is achievable using a protocol similar to the one described above, with the only difference being in the last step, which—instead of revealing the index of the maximum—reveals additive shares of the maximum value. To invoke this protocol, we will state that Alice and Bob obtain additive shares a, b , such that $a + b = \text{SMV}(\mathbf{x}, \mathbf{y})$ [7].

Other similar protocols may be defined. In particular, it is often essential to compute distance measures, such as the Hamming or Euclidean or absolute distance between vectors \mathbf{x} and \mathbf{y} held privately by Alice and Bob, respectively. Protocols for these computations can be efficiently designed using homomorphic functions and are an integral part of privacy-preserving nearest neighbor methods, which are covered in a separate article in this issue [34].

PRACTICAL CONSIDERATIONS

An important practical consideration is that we cannot always assume that \mathbf{x} and \mathbf{y} are integer vectors. Indeed, speech processing routinely uses probabilistic models, such as GMMs and HMMs in which the model parameters are floating-point values between zero and one. A particular problem that is nearly ubiquitous in the iterative algorithms used to perform modern speech processing is that multiplication of probability values result in extremely small numbers. One way to mitigate this issue of floating-point precision is to take the logarithm of the probability values and operate exclusively in the logarithmic domain. In fact, the need to perform operations on the logarithms of the probability values is what makes the SLOG protocol so useful. A second way to mitigate the issue of extremely small probability values is to appropriately scale the probability values by a large constant, e.g., 10^6 prior to all encryptions and compensate for the scaling after decryption.

COMPUTING SCORES

PRIVATELY FOR SPEECH PROCESSING

The computations involved in speech processing tasks can now be cast in terms of the above primitives. We first consider how the various required scores can be computed privately.

THE GAUSSIAN AS A DOT PRODUCT

The fundamental component of speech processing models is the Gaussian, since observation distributions are generally modeled as Gaussian mixtures. The form of the log of a multivariate Gaussian is well known

$$\log P(\mathbf{x}) = -0.5(\mathbf{x} - \mu)^\top \Sigma^{-1}(\mathbf{x} - \mu) - 0.5 \log(2\pi)^D |\Sigma|,$$

where D is the dimensionality of the data, and μ and Σ are the mean vector and covariance matrix of the Gaussian. It is fairly simple to show that this can be manipulated into the form $\tilde{\mathbf{x}}^\top \tilde{W} \tilde{\mathbf{x}}$ [7] where $\tilde{\mathbf{x}}$ is obtained by extending \mathbf{x} as $\tilde{\mathbf{x}} = [\mathbf{x}^\top 1]^\top$, and

$$\tilde{W} = \begin{bmatrix} -0.5\Sigma^{-1} & \Sigma^{-1}\mu \\ 0 & w^* \end{bmatrix}, \quad (7)$$

where $w^* = -0.5\mu^\top \Sigma^{-1}\mu - 0.5(2\pi)^D \log |\Sigma| + \log \text{prior}$, and prior captures an a priori probability associated with the Gaussian. In the case of a solitary Gaussian, prior = 1; however, if the Gaussian is one from a mixture, prior represents the mixture weight for the Gaussian. We can reduce the above computation further to a single inner product $\tilde{\mathbf{x}}^\top W$, where $\tilde{\mathbf{x}}$ is a quadratically extended feature vector derived from $\tilde{\mathbf{x}}$, which consists of all pairwise product terms $\tilde{x}_i \tilde{x}_j$ of all components $\tilde{x}_i, \tilde{x}_j \in \tilde{\mathbf{x}}$, and W is obtained by unrolling \tilde{W} into a vector. In this representation $\log \mathcal{N}(\mathbf{x}; \mu, \Sigma) = \tilde{\mathbf{x}}^\top W$.

PRIVATE COMPUTATION OF LOG GAUSSIAN

Input: Alice possesses a vector \mathbf{x} . Bob possesses a Gaussian parameterized by $\lambda = \{\mu, \Sigma\}$.

Output: Alice and Bob obtain random additive shares a and b such that $a + b = \log P(\mathbf{x}; \lambda)$.

- 1) Alice computes the extended vector $\tilde{\mathbf{x}}$ from \mathbf{x} . Bob arranges his model into a vector W as explained above.
- 2) Alice and Bob engage in the SIP protocol to obtain additive shares a and b : $a + b = \text{SIP}(\tilde{\mathbf{x}}, W)$.

Note that this is possible even if Bob only possesses an encrypted version of W . Note also that given only $E[\mu]$, $E[\Sigma]$ and $E[\text{prior}]$ Bob can obtain $E[W]$ using the protocol given in [9]. In either case, Alice and Bob obtain no information about each other's data.

PRIVATE COMPUTATION OF LOGARITHM OF GAUSSIAN MIXTURE

Input: Alice possesses \mathbf{x} . Bob possesses $\Lambda = \{\lambda_1, \lambda_2, \dots, \lambda_K\}$, the parameters of a Gaussian mixture with K Gaussians, each with parameter λ_i .

Output: Alice and Bob receive a and b such that $a + b = P(\mathbf{x}; \Lambda)$.

- 1) Bob arranges the parameters λ_i of each Gaussian into a vector W_i , $i = 1, \dots, K$.
- 2) For each $i = 1, \dots, K$, Alice and Bob engage in the SIP protocol to obtain additive shares c_i and d_i such that $c_i + d_i = \text{SIP}(\tilde{\mathbf{x}}, W_i)$. Note that if W_i is available in unencrypted form, Alice only needs to transmit $E[\tilde{\mathbf{x}}]$ once for the entire Gaussian mixture.
- 3) Alice and Bob apply the SLOG protocol to obtain $a + b = \text{SLOG}(\ln \mathbf{g})$, where $\mathbf{g} = [c_1 + d_1, \dots, c_K + d_K]$, where $c_i + d_i$ is the log of the i th Gaussian.

As before, Alice and Bob do not learn about each other's data. We will refer to this operation as $a + b = \text{SMOG}(\mathbf{x}, \Lambda)$.

COMPUTING THE LOGARITHM OF AN HMM FORWARD SCORE

Input: Alice possesses $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T]$. Bob possesses an M -state HMM $\Gamma = \{\Pi, \mathbf{A}, \Lambda\}$, with initial state probabilities $\Pi = [\pi_1, \dots, \pi_M]$, a transition matrix \mathbf{A} comprising vectors $\mathbf{a}_i = [a_{i,1}, \dots, a_{i,M}]$ and a set of state output Gaussian mixture densities $\Lambda = \{\Lambda_1, \Lambda_2, \dots, \Lambda_M\}$.

Output: Alice and Bob obtain a and b such that $a + b = \log P(\mathbf{X}; \Gamma)$.

Step 1) State Output Density Computation

- 1) For all $t = 1 \dots T$, $i = 1 \dots M$
 - a) Alice and Bob engage in the SMOG protocol to obtain additive shares $g_{t,i} + h_{t,i} = \text{SMOG}(x_t, \Lambda_i)$. Alice sends $E[g_{t,i}]$ to Bob.
 - b) Bob computes $q_{t,i} = E[g_{t,i}]E[h_{t,i}] = E[g_{t,i} + h_{t,i}] = E[\log P(x_t; \Lambda_i)]$ to obtain the encrypted value of the logarithm of the state output distribution for state i on x_t .

In the protocol below we use the notation $\alpha_i(t, i) = \log \alpha(t, i)$ and $\alpha_i(t) = [\log \alpha(t, 1), \dots, \log \alpha(t, M)]$. We represent $\{q_{t,i} \forall i\}$ obtained by Bob in the above computation as q_t . Now Bob precomputes $E[\log a_i] \forall i$. The recursions of the forward probability computation in HMMs given by (4) can now be computed in the log domain securely as follows.

THE VITERBI ALGORITHM IS VERY SIMILAR TO THE FORWARD ALGORITHM, EXCEPT FOR TWO KEY DIFFERENCES.

Step 2) Forward Algorithm

- 1) Bob computes $E[\alpha_i(1)] = q_t \cdot E[\log \pi]$.
- 2) For $t = 2 \dots T$, $i = 1 \dots M$
 - a) Bob engages with Alice in the SLOG protocol to obtain $c = \text{SLOG}(\log a_i + \alpha_i(t-1))$. Note that $c = E[\log \sum_j \alpha(t-1, j) a_{j,i}]$.
 - b) Bob computes $E[\alpha_i(t, i)] = c \cdot q_{t,i}$.
- 3) Alice and Bob engage in the SLOG protocol to obtain additive shares $a + b = \text{SLOG}(\alpha_i(T))$. Note that a and b are additive shares of $\log P(x_1, x_2, \dots, x_T; \Gamma)$.

We will refer to this protocol of computing shares of the HMM forward scores as $a + b = \text{SFWD}(X, \Gamma)$.

SECURE VITERBI ALGORITHM

The Viterbi algorithm is very similar to the forward algorithm, except for two key differences. First, instead of adding all incoming probabilities into any state in (4), we choose the maximum in (6). Additionally, Alice must also receive the optimal state sequence. Ideally, state indices would be permuted in the received sequence; however, for simplicity, we omit the additional complexity involved with permuting the indices. Using the notation of (6): $\delta(t, i)$ refers to the “best” predecessor for state s_i at time t . $\Gamma(t, i)$ is the joint log likelihood of the most probable state sequence arriving at state s_i at time t and the observation sequence x_1, \dots, x_t . We will use the notation that a bold character represents a vector that aggregates the scalar values represented by the corresponding unbolded symbol for all states. For instance $\mathbf{\Gamma}(t) = [\Gamma(t, 1), \Gamma(t, 2), \dots, \Gamma(t, M)]$.

Input: Alice possesses $X = [x_1, x_2, \dots, x_T]$. Bob possesses an M -state HMM $\Gamma = \{\Pi, A, \Lambda\}$.

Output: Alice and Bob obtain additive shares a and b such that $a + b = \log \max_s P(X, s; \Gamma)$. Alice receives the optimal state sequence $\bar{s} = s_1, \dots, s_T$.

The protocol uses operations similar to the ones described in the previous section and in the secure forward algorithm described above. For the detailed steps of the protocol, the reader is referred to [7].

IMPLEMENTING SECURE SPEECH TECHNIQUES

Having set up the operations as described above, we now consider how they may be applied to our speech problems. We will generally assume that all models have already been learned by the system, and, where required (e.g., speaker authentication), the models are encrypted. To avoid making the development unnecessarily complex, we do not describe how to privately learn GMM/HMM parameters [7], or how to privately adapt an existing background GMM or UBM to a user's enrollment data [9].

SPEAKER AUTHENTICATION

Input: The system possesses encrypted models Λ_S for the speaker and clear-text models Λ_U as the UBM. The user possesses vectors x_1, x_2, \dots, x_T .

Output: The system authenticates the user.

- 1) For each $t = 1 \dots T$
 - a) The user and the system engage in the SMOG operation with x_t and Λ_S to obtain additive shares a_t^S and b_t^S .
 - b) The user and the system engage in the SMOG operation with x_t and Λ_U to obtain additive shares a_t^U and b_t^U .
- 2) The user computes $A^S = \sum_t a_t^S$ and $A^U = \sum_t a_t^U$. The system computes $B^S = \sum_t b_t^S$ and $B^U = \sum_t b_t^U$.
- 3) The user and system engage in an SMI protocol to determine $\max(A^S + B^S, A^U + B^U)$. The system gets the result.

SPEAKER IDENTIFICATION

Input: The system possesses a set of models $\Lambda_1, \Lambda_2, \dots, \Lambda_N$ corresponding to speakers $1 \dots N$ (we consider the “background” model to be one of the speakers). The user has $X = [x_1, \dots, x_T]$.

Output: The system learns $\arg \max_i P(X; \Lambda_i)$.

- 1) For each speaker $s = 1 \dots N$
 - a) For each time $t = 1 \dots T$, the user and the system engage in the SMOG operation with x_t and Λ_s to obtain additive shares a_t^s and b_t^s .
 - b) The user computes $A^s = \sum_t a_t^s$. The system computes $B^s = \sum_t b_t^s$.
- 2) The user and system engage in an SMI protocol to determine $\arg \max_s A^s + B^s$. The system gets the result.

SPEECH RECOGNITION

Input: The system possesses a set of models $\Gamma_1, \Gamma_2, \dots, \Gamma_N$ corresponding to words or word sequences $1 \dots N$ (we consider the “background” model to be one of the phrases). The user has $X = [x_1, \dots, x_T]$.

Output: The system learns $\arg \max_i P(X; \Gamma_i)$. For isolated word recognition or phrase spotting, the speech recognition processes can generally be summarized into the following procedure:

- 1) For each word sequence $s = 1 \dots N$, the user and the system engage in the SFWD operation with X and Γ_s to obtain additive shares A^s and B^s .
- 2) The user and system engage in an SMV protocol to determine $\arg \max_s A^s + B^s$. The system gets the result.

The above operation is entirely performed in the log domain—all probabilities are log probabilities—and is hence robust to underflow. In practice, the secure Viterbi algorithm is a better choice than the forward algorithm to compute class scores, since its secure implementation has fewer expensive SMOG operations, which are replaced by SMV operations. Thus, $P(X; \Gamma_s)$ can be computed using the secure Viterbi algorithm instead of SFWD as described earlier. For CSR, the optimal state sequence (and word sequence corresponding to it) can both be obtained using the secure Viterbi algorithm.

ANALYZING THE PROTOCOLS

CORRECTNESS

All of the presented protocols, including the primitives, the score computation and the actual classification procedures above are easily shown to be correct—the final result obtained with the private protocols is exactly what would have been obtained had the operations been performed in a nonprivate manner. It can also be ascertained that in practical implementations the insecure and secure versions of the computations are virtually indistinguishable, thus the accuracy tradeoff owing to encryption is negligible [7], [8].

SECURITY

Each of the operations above is also computationally secure against honest but curious participants. The individual primitives in the section “Secure Primitives for Speech Processing” are easily shown to be secure (e.g., [37])—Alice and Bob do not learn about each others’ data at all, since they only see ciphertexts, or data masked by additive noise [38]. Consequently, the secure Gaussian operation, the SMOG operation, the SFWD, and the secure Viterbi protocol are all guaranteed not to reveal the user and system’s data to one another if the protocols are correctly followed. In general, none of the protocols reveal more than what the outcome of the computation itself reveals.

One might also consider a malicious model, where one or more parties may manipulate the data or send bogus data in an attempt to disrupt the protocol or learn about the other party’s data. If both parties are malicious, security can be enforced by accompanying the protocols with zero-knowledge proofs [39]. If only one of the parties is malicious, the other party can use conditional disclosure of secrets [40] to make sure he/she receives valid inputs from the malicious party. Both these methods, however, greatly increase the computation and communication overhead of the protocols.

Finally we note that although, technically, the proposed protocols also permit fully CSR, they assume that the entire HMM representing the complete word graph to be searched is evaluated for each analysis frame. In practice, CSR is never performed in this manner, even in conventional nonprivate

implementations. CSR has a high memory footprint and high computational complexity, and the word graphs must be pruned heavily based on partial scores, to restrict the computation. The act of pruning restricts the hypothesis set considered and reveals information about the recognition output. Techniques to hide this information are not within the scope of this article, although they are topics of active research.

PERFORMANCE

The private computation techniques must in principle result in identical classification outcomes to their conventional nonprivate counterparts. Although there is a minor loss of resolution resulting from the fact that all computation must now be performed with fixed-point arithmetic to accommodate encryption over integer fields, this has little effect on accuracy—speech processing applications have historically achieved reasonable performance with fixed-point implementations with as little as 16 bits of resolution.

Perhaps the most important performance consideration is the additional computational complexity imposed by the privacy primitives. In particular, computing a single Gaussian securely takes a significant fraction of a second on a desktop computer [7]. Table 1 reports computation times for 1 s of audio [8] on an isolated word recognition experiment for a vocabulary of the ten

digits zero to ten, each of which was modeled by a five-state HMM with a single Gaussian. Results were obtained on a 3.2 GHz Pentium 4. The Paillier encryption scheme was used [12]. The difference between the forward scores computed using secure computation those computed in the con-

ventional manner was less than 0.52%, and the classification accuracy for the secure and conventional versions was nearly the same, being 99%. Results with different key sizes are given primarily to show the trends in the computation time. It should be noted that Paillier encryption based on 256-bit and 512-bit keys is no longer considered sufficiently secure in the cryptography community.

Table 2 shows a similar computational expense table containing the average time required to perform privacy-preserving speaker authentication on a Core 2 Duo 2 GHz Linux machine per second of input audio. In this case, Boneh-Goh-Nissim

PERHAPS THE MOST IMPORTANT PERFORMANCE CONSIDERATION IS THE ADDITIONAL COMPUTATIONAL COMPLEXITY IMPOSED BY THE PRIVACY PRIMITIVES.

[TABLE 1] EXECUTION TIME FOR ISOLATED WORD RECOGNITION PROTOCOL (PAILLIER ENCRYPTION).

ACTIVITY	256-BIT KEYS	512-BIT KEYS	1,024-BIT KEYS
ALICE ENCRYPTS INPUT DATA (ONLY ONCE)	205.23 S	1,944.27 S	1,1045 S
BOB COMPUTES $\xi(\log b_j(x_i))$ (PER HMM)	79.47 S	230.30 S	461 S
BOTH COMPUTE $\xi(\alpha_r(j))$ (PER HMM)	16.28 S	107.35 S	785 S

[TABLE 2] EXECUTION TIME FOR THE VERIFICATION PROTOCOL (PAILLIER ENCRYPTION).

STEPS	TIME (256-BIT)	TIME (1,024-BIT)
ENCRYPTING $\bar{x}_t \forall t$	138 S	8,511 S
EVALUATING ADAPTED	97 S	1,809 S
EVALUATING UBM	\approx SAME AS ADAPTED	\approx SAME AS ADAPTED
COMPARISON	0.07 S	4.01 S
TOTAL = $E[\bar{x}_t] + \text{ADAPTED} + \text{UBM} + \text{COMPARE}$	331 S ~5.47 MIN	12,133 S ~3 H, 32 MIN

encryption [41] is employed, rather than the significantly less expensive Paillier encryption. The data are from the YOHO data set [42]. Both the speaker and the subject were modeled by mixtures of 32 Gaussians, and the audio was represented by a sequence of 39-dimensional feature vectors computed at the rate of 100 times/s. By comparison, the “insecure” computation on the same experiment took only 3.2 s per second of audio. The scores computed by the secure computation were identical to within five decimal places to those obtained in the insecure conventional classifier. Classification accuracies on this data set with this setup achieves an equal error rate (EER) of about 7%. Clearly then, computation is a serious bottleneck. Encryption and decryption take the most time. Additional operations take relatively small time in comparison to the overhead of ciphertext processing; nevertheless, even they are considerably more expensive than insecure computation.

The times shown do not include communication overhead; however, for these tasks the communication overhead is trivial compared to the computational expenses.

In the experiments presented above, the implementations were far from optimal. Moreover, the primitives described herein can themselves be optimized; it is not clear that the particular structure chosen to decompose the computations is optimal from the perspective of computational complexity. Performing computations on parallel processors, more efficient implementations of encryption etc. all may improve the speeds by some orders of magnitude; regardless, the final outcome is currently far slower than the speed performance of insecure computations.

The methods described so far provide privacy to conventional state-of-art mechanisms for performing pattern recognition applications on speech. An alternate mechanism may be to modify the matching algorithms themselves to make them more amenable to efficient secure implementations. In the next section, one such approach is described. These techniques are not as generic in their scope as SMC-based methods described above, since the right form of classifier must be found for the task. Specifically, the technique we describe now applies only to speaker authentication and congruent tasks.

STRING COMPARISON IS NONINTERACTIVE, AND ALSO MUCH FASTER THAN ENCRYPTION, WHICH ENABLES US TO PERFORM THE PRIVACY-PRESERVING PROCESSING VERY EFFICIENTLY.

SPEECH PROCESSING AS PRIVACY-PRESERVING STRING MATCHING

We now discuss an alternative framework for privacy-preserving speech processing based on private string comparison. The main idea is to convert a speech sample into a fingerprint, i.e., a fixed-length bit string. This representation allows us to compare two speech samples by checking if their respective fingerprints match exactly. Unlike the cryptographic framework described earlier, string comparison is noninteractive, and also much faster than encryption, which enables us to perform the privacy-preserving processing very efficiently.

The fingerprint representation is similar to a text-password system. The privacy issues are also similar; the user requires the system to store and compare the passwords in an obfuscated form, so that an adversary cannot observe the original passwords. Furthermore, we require the system to be accurate; just as the password system is able to reject users entering incorrect passwords, the privacy-preserving speech processor should be able to classify speech samples accurately. Finally, the performance of this string comparison-based speech processing approach should be competitive with conventional speech processing methods. Initial software implementations reveal a tradeoff between a significant increase

in speed and a small degradation in the accuracy with respect to classical methods based on, for example, HMMs.

The twin objectives of accuracy and privacy are achieved as follows: A data-length independent feature vector (string) is derived from the speech signal

such that string comparison implies a nearest-neighbor classification. Second, these vectors are converted into password-like bit strings that are not invertible. We describe these briefly below.

FEATURE REPRESENTATION: SUPERVECTORS

An obvious solution to create a fingerprint is to apply a cryptographic hash function $H[\cdot]$, e.g., SHA-256 [43], to the speech input itself. However, direct conversion of audio signals into fingerprint-like patterns is difficult, due the inherent variation in cadence and length of audio recordings. A more robust length- and cadence-invariant representation of the audio is first required.

Campbell et al. [44] extend the maximum a posteriori (MAP) adaptation procedures for GMMs mentioned in the section “Biometric Applications: Identification and Authentication” to construct a supervector to represent each speech sample. A supervector is a characterization of an estimate of the distribution of feature vectors derived from the speech recording. It is obtained by performing MAP adaptation of the UBM over the recording and concatenating the parameters, typically just the means, or means and mixture weights of the adapted model. For instance, given the adapted model

$\lambda_s = \{\hat{w}_i^s, \hat{\mu}_i^s, \hat{\Sigma}_i^s\}$ with M Gaussian components, the supervector sv is given by $(\hat{\mu}_1^s \| \hat{\mu}_2^s \| \dots \| \hat{\mu}_M^s)$.

The supervector is then used as a feature vector instead of the original feature vectors derived from the speech sample. In the speaker authentication task addressed by Campbell et al., authentication verification is performed using a binary support vector machine (SVM) classifier for each user. The SVM is trained on supervectors obtained from enrollment utterances from the user, as instances of one class, and from a collection of impostor recordings, as instances of the opposite class. As the classes are usually not separable in the original space, [44] also use a kernel mapping that is shown to achieve higher accuracy.

A related approach is to use k -nearest neighbors trained on supervectors as our classification algorithm. The rationale for this approach is twofold: first, k -nearest neighbors allow classification with nonlinear decision boundaries with accuracy comparable to SVMs with kernels [45]. Second, using the LSH transformations discussed below, private k -nearest neighbors computation can be reduced to private string comparison, which can be easily accomplished without an interactive protocol.

LOCALITY SENSITIVE HASHING

LSH [46] is a widely used technique for performing efficient approximate nearest-neighbor search. An LSH function $L(\cdot)$ proceeds by applying a random transformation to a data vector x and projecting it to a vector $L(x)$ in a lower-dimensional space, which we refer to as the LSH key or bucket. A set of data points that map to the same key are considered as approximate nearest neighbors (Figure 1).

A single LSH function does not group the data points into fine-grained clusters; one must use a hash key obtained by concatenating the output of k LSH functions. This k -bit LSH function $L(x) = L_1(x) \dots L_k(x)$ maps a d -dimensional vector into a k -bit string. Two data vectors may be deemed to be highly similar if the k -bit hashes derived from them are identical. Such fine selectivity may, however, have an adverse effect on the recall in identifying neighbors when the data from a class have significant inherent variations. To address this, m -different LSH keys are computed over the same input to achieve better recall. Two data vectors x and y are said to be neighbors if at least one of their keys, each of length k , matches exactly. LSH provides major efficiency advantages: By precomputing the keys, the approximate nearest neighbor search can be done in time sublinear in the size of the data set.

A family of LSH functions is defined for a particular distance metric. A hash function from this family has the property that data points that are close to each other as defined by the distance metric are mapped to the same key with high probability. There exist LSH constructions for a variety of distance

metrics, including arbitrary kernels [47], but we mainly consider LSH for Euclidean distance (E2LSH) [48] and cosine distance [49] as the LSH functions for these constructions are simply random vectors. As the LSH functions are data independent, it may be possible to distribute them to multiple parties without privacy loss.

The LSH construction for Euclidean distance transforms a d -dimensional vector into a vector of k integers. The i th entry of the hash is as follows:

$$L_i(x) = \left\lfloor \frac{\mathbf{r}_i^T \mathbf{x} + b}{w} \right\rfloor, \quad (8)$$

where \mathbf{r}_i is a d -dimensional vector with each component drawn i.i.d. from $\mathcal{N}(0,1)$, w is the width of the bin, (e.g., 255), and $b \in [0, w]$. Similarly, the construction of the i th bit of the LSH for cosine distance, using \mathbf{r}_i defined as above is given by

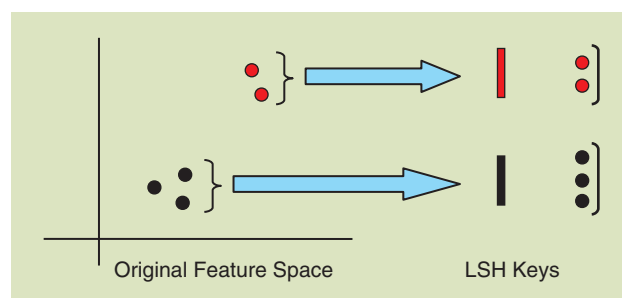
$$L_i(x) = \begin{cases} 1 & \text{if } \mathbf{r}_i^T \mathbf{x} > 0, \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

LSH is inherently not privacy preserving due to its locality sensitive property. It is possible to reconstruct the input vector by observing a sufficient number of LSH keys obtained from the same vector. To satisfy the privacy constraint, a cryptographic hash function $H[\cdot]$ is applied to the LSH keys. Cryptographic hash functions, such as SHA-256 and MD5 are orders of magnitude faster to compute compared to homomorphic encryption.

PRIVACY-PRESERVING SPEECH PROCESSING THROUGH STRING COMPARISON

The private string comparison framework lends itself very well to biometric tasks where audio-length-invariant supervector representations of the audio may be derived. In applications such as speaker verification and speaker identification, the user can convert the test speech sample into supervectors, apply the LSH transformation, and finally apply a cryptographic hash function and submit the output to the system. The system can simply compare the hashes provided by the user to the hashes computed over the enrollment data and accept or reject the user by comparing the number of matches to a precalibrated

**LSH IS A WIDELY USED
TECHNIQUE FOR PERFORMING
EFFICIENT APPROXIMATE
NEAREST-NEIGHBOR SEARCH.**



[FIG1] LSH.

threshold. Due to the irreversibility of the hashes, the system is not able to recover the original speech sample, and we are able to maintain the privacy of the speech input. As an illustrative example, we present the string comparison technique applied to the privacy-preserving speaker verification problem below and refer the reader to [10] for other applications.

PRIVACY-PRESERVING SPEAKER VERIFICATION AS STRING COMPARISON

As the possible values for LSH keys discussed above lie in a relatively small set by cryptographic standards, 256^k for k -bit Euclidean LSH and 2^k for k -bit cosine LSH, it is possible for the server to obtain $L(s)$ from $H[L(s)]$ by applying brute-force search. To make this attack infeasible, the domain of the hash function $H[\cdot]$ is increased by concatenating the LSH key with a long random string q_i (e.g., 80-bit in length) that is unique to the user i , which is called the salt, as summarized in Figure 2. Requiring the user to keep the salt private and unique to each system also gives the additional advantage of rendering cryptographically hashed enrollment data useless to an adversary. With this modification, LSH-based privacy-preserving speaker verification is achieved using the enrollment and authentication protocols described below.

ENROLLMENT PROTOCOL

Each user has a set of enrollment utterances $\{x_1, \dots, x_n\}$. The users also obtain the UBM and the l LSH functions $\{L_1(\cdot), \dots, L_l(\cdot)\}$, each of length k -bit from the system. Each user i generates the random 80-bit salt string q_i .

For each enrollment utterance x_j , user i :

- performs adaptation of x_j with the UBM to obtain supervector s_j .
- applies the l LSH functions to s_j to obtain the keys $\{L_1(s_j), \dots, L_l(s_j)\}$.

REQUIRING THE USER TO KEEP THE SALT PRIVATE AND UNIQUE TO EACH SYSTEM ALSO GIVES THE ADDITIONAL ADVANTAGE OF RENDERING CRYPTOGRAPHICALLY HASHED ENROLLMENT DATA USELESS TO AN ADVERSARY.

- applies the cryptographic hash function salted with q_i to each of these keys to obtain $\{H[L_1(s_j) \| q_i], \dots, H[L_l(s_j) \| q_i]\}$, and sends them to the system.

AUTHENTICATION PROTOCOL

For a test utterance x' , user i :

- performs adaptation of x' with the UBM to obtain supervector s'
- applies the l LSH functions to s' to obtain the keys $\{L_1(s'), \dots, L_l(s')\}$
- applies the cryptographic hash function salted with q_i to each of these keys to obtain $\{H[L_1(s') \| q_i], \dots, H[L_l(s') \| q_i]\}$ and sends it to the system
- the system computes the number of matches between the hashed keys for the test utterance and the corresponding hashes of enrollment utterances. If this number exceeds a threshold, it accepts the user.

$$\text{match} = \sum_{i \in \text{enrollment}} \sum_{j=1}^l I(H[L_j(s_i) \| q_i] = H[L_j(s') \| q_i]),$$

if $\text{match} > \text{threshold}$: accept.

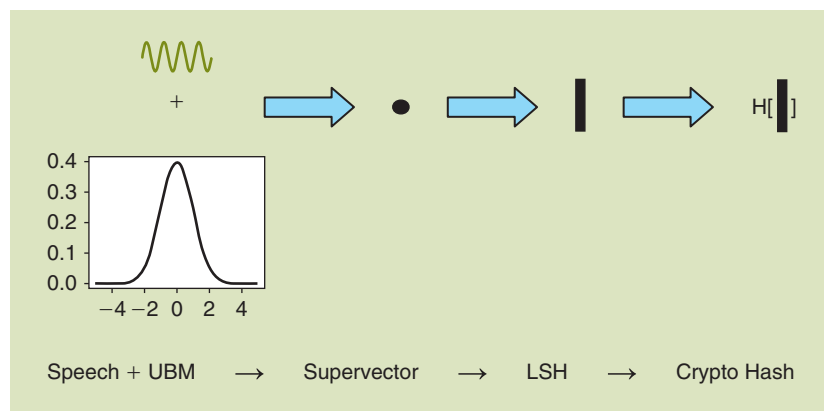
The system never observes any LSH key before a salted cryptographic hash function is applied to it. Apart from the salt, the user does not need to store any speech data on its device. The enrollment and verification protocols, therefore, satisfy the privacy constraints discussed above.

EXPERIMENTS

Experiments examining the accuracy and performance of LSH-based schemes have been reported using the YOHO data set [42] which comprises a collection of short utterances, each a sequence of three two-digit numbers, produced by 138 speakers. There are 96 enrollment utterances and 40 test utterances from each speaker. MFCC features augmented by differences and double differences are chosen as feature vectors in these experiments [10]. A UBM with 64 Gaussian mixture components is trained on a random subset of the enrollment data belonging to all users. Supervectors are obtained by individually adapting all enrollment and verification utterances to the UBM.

ACCURACY

The lowest EER was achieved by using $l = 200$ instances of LSH functions each of length $k = 20$ for both Euclidean and cosine distances. Table 3 indicates that LSH for Euclidean distance performs better than LSH for cosine distance, while combining the two classifiers gives the best accuracy.



[FIG2] Speaker verification as string comparison.

Furthermore, using imposter data achieves lower EER when using the combined scores for both the distances: one-sided classifiers are clearly not sufficiently accurate. While the actual error rates in this example may appear relatively high, it should be noted that an SVM-based classifier working from supervectors obtained a classification accuracy of 10.8% on the same setup. In other experiments not reported here, using supervectors derived from more detailed GMMs, EERs of approximately 5% have been obtained using the string matching framework.

EXECUTION TIME

Compared to a nonprivate speaker recognition system based on supervectors, the only computational overhead for the privacy-preserving version is in applying the LSH and salted cryptographic hash function. For a $64 \times 39 = 2,496$ -dimensional supervector representing a single utterance, the computation for both Euclidean and cosine LSH involves a multiplication with a random matrix of size $20 \times 2,496$, which requires a fraction of a millisecond. Performing this operation 200 times required 15.8 ms on average [10]. The reported times are for a laptop running 64-bit Ubuntu 11.04 with 2 GHz Intel Core 2 Duo processor and 3-GB RAM.

The Euclidean and cosine LSH keys of length $k = 20$ require $8 \times 20 \text{ bits} = 20 \text{ B}$ and $20 = 1.6 \text{ B}$ for storage, respectively. Using a C++ implementation of SHA-256 cryptographic hashing algorithm based on the OpenSSL libraries [50], hashing 200 instances of each of these keys in total required 28.34 ms on average. Beyond this, the verification protocol only consists of matching the 256-bit long cryptographically hashed keys derived from the test utterance to those obtained from the enrollment data.

CONCLUSIONS

The two frameworks presented in this article both show promise in enabling privacy-preserving speech processing. The cryptographic framework, while more generic, carries the usual computational overhead of cryptography. However, it can be made arbitrarily secure and flexible. The string-matching framework, on the other hand, is much more efficient; however, it is restricted in its applicability and currently results in a degradation of performance.

At this point, both of them can only be viewed as initial forays into the implementation of truly secure speech-processing frameworks, and much work remains. Researchers continue to investigate more efficient protocols, possibly using simpler encryption techniques, which could be orders of magnitude faster than the methods described here. Within the string-matching framework, recent work has shown that significantly greater accuracies can be obtained using nearest-neighbor methods such as those described in [34]. These also show great

[TABLE 3] AVERAGE EER FOR THE TWO ENROLLMENT DATA CONFIGURATIONS AND THREE LSH STRATEGIES: EUCLIDEAN, COSINE, AND COMBINED (EUCLIDEAN AND COSINE).

ENROLLMENT: ONLY SPEAKER		
EUCLIDEAN	COSINE	COMBINED
15.18%	17.35%	13.80%
ENROLLMENT: SPEAKER AND IMPOSTER		
EUCLIDEAN	COSINE	COMBINED
15.16%	18.79%	11.86%

promise in affording greater generalizability than the string-matching solutions described here.

A computationally efficient implementation of a fully homomorphic encryption (FHE) scheme [27] would significantly change the construction of privacy-preserving protocols. This would allow the creation of a noninteractive protocol where the user uploads the encrypted speech sample, and the server can perform all the necessary computations without requiring any further involvement from the user. Such a scheme would not alleviate all the problems of privacy-preserving computation; there would still be a need to deal with key distribution, malicious adversaries, communication overhead, and many other practical problems. Nevertheless, researchers and practitioners of privacy-preserving speech processing are following developments in FHE with interest.

A COMPUTATIONALLY EFFICIENT IMPLEMENTATION OF A FULLY HOMOMORPHIC ENCRYPTION SCHEME WOULD SIGNIFICANTLY CHANGE THE CONSTRUCTION OF PRIVACY-PRESERVING PROTOCOLS.

AUTHORS

Manas A. Pathak (mpathak@adchemy.com) received his Ph.D. degree from Carnegie Mellon University in 2012. He is a research scientist at Adchemy, Inc. His thesis on privacy preserving speech processing was selected to be published as a monograph in the *Springer Best Theses* series. He has authored over 15 papers in various conferences and journals. His research interests include intersection of data privacy, machine learning, and speech processing.

Bhiksha Raj (bhiksha@cs.cmu.edu) received his Ph.D. degree from Carnegie Mellon University, where he is an associate professor in the Language Technologies Institute and also holds affiliate positions in the Electrical and Computer Engineering and Machine Learning Departments. His primary research interests lie in automatic speech recognition, machine learning, and data privacy. He has authored over 150 papers on these topics.

Shantanu Rane (rane@merl.com) received his Ph.D. degree from Stanford University in 2007. He is a principal research scientist at Mitsubishi Electric Research Laboratories in Cambridge, Massachusetts. He is an associate editor of *IEEE Transactions on Information Forensics and Security* and *IEEE Signal Processing Letters*, and he is a member of the IFS Technical Committee. He conducts research in secure signal processing and distributed

source coding and has participated in standardization activity for H.264/AVC video compression and ISO/SC37 biometrics.

Paris Smaragdis (paris@illinois.edu) received his Ph.D. degree from Massachusetts Institute of Technology. He is an assistant professor in the Computer Science and the Electrical and Computer Science Departments at the University of Illinois at Urbana-Champaign. Previously, he was a senior research scientist at Adobe Systems and a research scientist at Mitsubishi Electric Research Labs. He is a Senior Member of the IEEE and a member of the Acoustical Society of America. He is the vice chair of the IEEE Technical Committee for Machine Learning for Signal Processing, the chair of the Steering Committee for Latent Variable Analysis and Signal Separation, and is a member of the IEEE Audio and Acoustic Signal Processing Technical Committee.

REFERENCES

- [1] *Interception and disclosure of wire, oral, or electronic communications prohibited*, US Federal Law, Title 18, Part 1, chapter 19, section 2511, 2012.
- [2] "The end of privacy," *The Economist*, Apr. 1999.
- [3] P. Schwartz and J. R. Reidenberg, *Data Privacy Law: A Study of United States Data Protection*. LEXIS Law, 1996.
- [4] H. Nissenbaum, *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford, CA: Stanford Univ. Press, 2010.
- [5] R. K. Nichols and P. C. Lekkas, *Speech Cryptology*. New York: McGraw-Hill, 2002.
- [6] M. Shashanka and P. Smaragdis, "Secure sound classification: Gaussian mixture models," in *Proc. ICASSP*, vol. 3, Toulouse, France, 2006, p. 3.
- [7] P. Smaragdis and M. Shashanka, "A framework for secure speech recognition," *IEEE Trans. Audio, Speech, Lang. Process.*, vol. 15, no. 4, pp. 1404–1413, 2007.
- [8] M. A. Pathak, S. Rane, W. Sun, and B. Raj, "Privacy preserving probabilistic inference with hidden Markov models," in *Proc. ICASSP*, Prague, Czech Republic, May 2011, pp. 5868–5871.
- [9] M. A. Pathak and B. Raj, "Privacy preserving speaker verification using adapted GMMs," in *Proc. Interspeech*, 2011, pp. 2405–2408.
- [10] M. A. Pathak, "Privacy-preserving machine learning for speech processing," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 2012.
- [11] R. L. Rivest, L. Adleman, and M. L. Dertouzos, "On data banks and privacy homomorphisms," in *Foundations of Secure Computation*. New York: Academic, vol. 4, no. 11, 1978, pp. 169–180.
- [12] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. EUROCRYPT*, vol. 1592, May 1999, pp. 169–180, pp. 223–238.
- [13] J. Benaloh, "Dense probabilistic encryption," in *Proc. Workshop Selected Areas of Cryptography*, Kingston, ON, Canada, May 1994, pp. 120–128.
- [14] I. Damgård, M. Geisler, and M. Krøigård, "Homomorphic encryption and secure comparison," *Int. J. Appl. Cryptography*, vol. 1, no. 1, pp. 22–31, 2008.
- [15] A. Yao, "Protocols for secure computations (extended abstract)," in *Proc. IEEE Symp. Foundations Computer Science*, 1982, pp. 160–164.
- [16] M. Rabin, "How to exchange secrets by oblivious transfer," Harvard Univ., Cambridge, MA, Tech. Rep. TR-81, 1981.
- [17] A. Yao, "How to generate and exchange secrets," in *Proc. 27th Annu. Symp. Foundations of Computer Science (SFCS)*, Washington, DC, 1986, pp. 162–167.
- [18] M. A. Pathak and B. Raj, "Privacy preserving speaker verification as password matching," in *ICASSP*, Kyoto, Japan, May 2012, pp. 1849–1852.
- [19] X. Huang, A. Acero, and H.-W. Hon, *Spoken Language Processing*. Englewood Cliffs, NJ: Prentice Hall, 2001.
- [20] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Trans. Acoustics, Speech, Signal Process.*, vol. 28, no. 4, pp. 357–366, 1980.
- [21] S. Imai, "Cepstral analysis synthesis on the mel frequency scale," in *Proc. IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Boston, MA, Apr. 1983, vol. 8, pp. 93–96.
- [22] D. Reynolds and R. C. Rose, "Robust text-independent speaker identification using gaussian mixture speaker models," *IEEE Trans. Speech Audio Process.*, vol. 3, no. 1, pp. 72–83, 1995.
- [23] P. Kenny, "New map estimators for speaker recognition," in *Proc. Eurospeech*, Geneva, Switzerland, Sept. 2003, pp. 2961–2964.
- [24] P. Kenny, "Joint factor analysis of speaker and session variability: Theory and algorithms," Centre de Recherche Informatique de Montreal, Tech. Rep. CRIM-06/08-13, 2005.
- [25] T. Petrie, "Probabilistic functions of finite state Markov chains," *Ann. Math. Stat.*, vol. 40, no. 1, pp. 97–115, 1969.
- [26] B. Raj, R. Singh, and T. Virtanen, "The basics of automatic speech recognition," in *Techniques for Noise Robustness in Automatic Speech Recognition*, T. Virtanen, R. Singh, and B. Raj, Eds. Hoboken, NJ: Wiley, 2012.
- [27] C. Aguilar-Melchor, S. Fau, C. Fontaine, G. Gogniat, and R. Sirdey "Recent advances in homomorphic encryption," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 108–117, 2013.
- [28] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comp. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984.
- [29] J. Vaidya, C. Clifton, M. Kantarcioglu, and S. Patterson, "Privacy-preserving decision trees over vertically partitioned data," *ACM Trans. Knowledge Discovery from Data*, vol. 2, no. 3, pp. 1–27, 2008.
- [30] W. Du and M. J. Atallah, "Privacy-preserving cooperative statistical analysis," in *Computer Security Applications Conference (ACSAC)*, 2001, pp. 102–110.
- [31] S. Avidan and M. Butman, "Blind vision," in *Proc. 9th European Conf. Computer Vision (ECCV)*, Graz, Austria, May 2006.
- [32] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *Proc. Int. Conf. Information Security and Cryptology (ICISC)*, pp. 23–25, 2004.
- [33] M. Atallah and J. Li, "Secure outsourcing of sequence comparisons," *Int. J. Inform. Sec.*, vol. 4, no. 4, pp. 277–287, 2005.
- [34] S. Rane and P. Boufounos, "Privacy-preserving nearest neighbor methods," *IEEE Signal Process. Mag.*, vol. 30, no. 2, pp. 18–28, Mar. 2013.
- [35] V. Kolesnikov, A. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *Proc. Cryptology and Network Security (CANS)*, Kanazawa, Japan, Dec. 2009, pp. 1–20.
- [36] F. Kerschbaum, D. Biswas, and S. de Hoogh, "Performance comparison of secure comparison protocols," in *Proc. Int. Workshop on Database and Expert System Applications*, 2009, pp. 133–136.
- [37] B. Goethals, S. Laur, H. Lipmaa, and T. Mielikäinen, "On private scalar product computation for privacy-preserving data mining," in *Proc. Int. Conf. Information Security and Cryptology*, 2004.
- [38] M. Quisquater, L. Genelle, and E. Prouff, "Thwarting higher-order side channel analysis with additive and multiplicative maskings," in *Cryptographic Hardware and Embedded Systems 2011*, Nara, Japan, 2011, pp. 240–255.
- [39] J. Feigenbaum, "Overview of interactive proof systems and zero-knowledge," in *Contemporary Cryptology: The Science of Information Integrity*. Piscataway, NJ: IEEE Press, 1992.
- [40] S. Laur and H. Lipmaa, "Additive conditional disclosure of secrets and applications," *Cryptology ePrint Archive, Report 2005/378*, 2005. [Online]. Available: <http://eprint.iacr.org/>
- [41] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory of Cryptography Conf.*, 2005, pp. 325–341.
- [42] J. P. Campbell, "Testing with the YOHO CD-ROM voice verification corpus," in *Proc. ICASSP*, 1995, pp. 341–344.
- [43] *FIPS 180-3: Secure Hash Standard*, National Institute for Standards and Technology, 2008.
- [44] W. M. Campbell, D. E. Sturm, D. A. Reynolds, and A. Solomonoff, "SVM based speaker verification using a GMM supervector kernel and NAP variability compensation," in *ICASSP*, vol. 1, May 2006, Toulouse, France, p. 1.
- [45] J. Mariéthoz, S. Bengio, and Y. Grandvalet, *Kernel Based Text-Independent Speaker Verification*. Hoboken, NJ: Wiley, 2008.
- [46] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proc. ACM Symp. Theory of Computing*, 1998, pp. 604–613.
- [47] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *Proc. IEEE Int. Conf. Computer Vision*, 2009, pp. 2130–2137.
- [48] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proc. ACM Symp. Computational Geometry*, 2004, pp. 253–262.
- [49] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *Proc. ACM Symp. Theory of Computing*, 2002.
- [50] OpenSSL. (10 May 2010). [Online]. Available: <http://www.openssl.org>